The AMath and DAMath Special Functions

Reference Manual
and
Implementation Notes

Version 2.10

Including Complex Functions

Wolfgang Ehrhardt

October 2015

Contents

Chapter 1

Introduction

The **AMath** package contains Pascal/Delphi open source code for accurate mathematical methods without using multi precision arithmetic. The accurate implementation of special functions needs versions of certain elementary mathematical functions that are more accurate than those supplied by Delphi and (Free-)Pascal, especially sin, cos, and power. 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**, **AMTools**, **AMCmplx**, and the **Special Functions** units. The units and basic test programs can be compiled with the usual Pascal (TP5[1], 5.5, 6; BP 7; VP 2.1; FPC 1.0, 2.0, 2.2, 2.4, 2.6, 3.x) and Delphi (1..7, 9, 10, 12, 17, 18) versions.[2]

The **DAMath** package has mathematical functions and methods for double precision compatible with 64-bit systems without extended precision or 387-FPU. With a few exceptions all **AMath** double precision functions have **DAMath** counterparts, which in most cases are not mentioned separately in this manual.

This manual describes the **AMath** Special and Complex Functions with additional implementation notes and formulas. For the other **AMath** functions and tools see the **AMath** HTML introduction at http://www.wolfgang-ehrhardt.de/amath_functions.html.

The latest version of this document is always available from the author's home page at http://www.wolfgang-ehrhardt.de.

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 MiKTeX V2.4/2.8 and TeXstudio V2.2.

[1] 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.

[2] Delphi versions 11,14,15,16 should work but are not tested; **AMCmplx** needs BP7 or newer.

Chapter 2

AMath and DAMath

2.1 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 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 $\text{power}(x, y) = x^y = \exp(y \ln x)$ 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. **AMath** uses a table size of 512 entries resulting in about four additional bits of accuracy; the tables are calculated with MPArith.[1]

The following less common notations from the **AMath** unit are used in this manual: The Euler constant `EulerGamma` $= \gamma = 0.5772156649...$, the largest finite extended number `MaxExtended` $\sim 2^{16384} = 1.189731\text{E}{+}4932$, the smallest positive normalised extended number `MinExtended` $= 2^{-16382} = 3.362103\text{E}{-}4932$, `ln_MaxExt` $= \ln($`MaxExtended`$)$, `ln_MinExt` $= \ln($`MinExtended`$)$, the machine epsilons for extended `eps_x` $= 2^{-63} = 1.084202\text{E}{-}19$ and double `eps_d` $= 2^{-52} = 2.220446\text{E}{-}16$, further some accuracy improved functions: $\text{expm1}(x) = \exp(x) - 1$, $\text{expx2}(x) = \exp(x \cdot |x|)$, $\text{ln1p}(x) = \ln(1 + x)$, and $\text{powm1}(x, y) = x^y - 1$.

[1] The table with 1024 entries gives nearly full accuracy, but requires about 15 KB.

2.2 DAMath functions and special functions

2.2.1 DAMath functions

The **DAMath** units implement accurate mathematical functions and methods for double precision without using multi precision arithmetic or assembler. The main purpose is to make the **AMath** functions available for 64-bit systems without Extended Precision or 387-FPU, but they can be used with 32-bit systems[2]. The units assume IEEE-754 53-bit double precision (binary64) and rounding to nearest; there are **no** rounding or precision control functions.

The **DAMath** unit provides the double precision accurate mathematical functions: exponential, logarithmic, trigonometric, hyperbolic, inverse circular and hyperbolic functions; and there are the polynomial, vector, statistic operations and floating point functions.

DAMath uses the system functions abs, arctan, frac, int, ln, and trunc (including possible bugs for 32-bit). Because FreePascal (versions $\leq$ 2.6.4; Target OS: Win64 for x64) looses up to 13 of the 53 bits for exp, **DAMath** implements its own exp function. System $\sin(x)$ and $\cos(x)$ are used for $|x| \leq \pi/4$, Payne/Hanek range reduction is performed if $|x| > 2^{30}$.

On Win7/64 the 64-bit **DAMath** one argument elementary transcendental functions and power have peak relative errors < 2 `eps_d` (about 4.4E-16), the RMS values are < 0.6 `eps_d`.

2.2.2 DAMath special functions

The **DAMath** based double precision special functions are derived from the **AMath** implementations, the interfaced functions in the **SpecFunD** unit have the same names as the **AMath** double functions, the descriptions and implementation notes from Chapter 3 are essentially valid etc. The following list shows some internal differences:

- The internal functions are pre-fixed with `sfd_`, the unit names start with `sd`, e.g. sdBessel vs. sfBessel.
- Constants, arguments, and function value ranges are adjusted to the double precision target.
- Hex/extended constructs are replaced by hex/double (partly recalculated).
- Most polynomial and rational approximations are kept, resulting in slightly sub-optimal implementations (optimal double precision approximations are usually very different).
- Chebyshev approximations are easily adjusted to double precision!
- The relative errors of the **DAMath** special functions are usually larger (especially on 64-bit systems) than those of the corresponding **AMath** double functions (which are often correctly rounded due to the internal extended precision calculations).

[2] The compiler versions TP 5.0, 5.5, and 6.0 are no longer supported in **DAMath**.

Chapter 3

Special functions

The **AMath** units **SpecFun**[1] and **SpecFunX** interface special functions for double and extended precision; the following function groups are available:

- Bessel functions and related,
- Elliptic integrals, elliptic and theta functions,
- Gamma function and related,
- Error function and related,
- Exponential integrals and related,
- Zeta functions, polylogarithms, and related,
- Orthogonal polynomials and Legendre functions,
- Hypergeometric functions and related,
- Statistical distributions,
- 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), e.g. erfc vs. erfcx. Generally the extended versions have larger relative errors (measured in the corresponding machine epsilons eps_x or eps_d) than their double counterparts. Note that some functions are very 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 $x <$ eps_x, $x <$ eps_x$^{1/2}$, or $x > 1/$ 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.

[1] and the **DAMath** unit **SpecFunD**

3.1 Bessel functions and related

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

Note that the Bessel functions $J_\nu(x), Y_\nu(x)$ and some other related functions have an infinite number of irrational real roots. Except in special cases the **AMath** functions can have large relative errors for arguments x near these zeros and only absolute accuracy is achievable; generally the absolute errors have magnitudes of a few `eps_x`.

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 $|x| \geq 9$ (cf. 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 $|x| \geq 9$ (cf. Abramowitz and Stegun [1, 9.2.17 – 9.2.31]), the results are computed with two rational approximations from Cephes [7, file j1l.c].

3.1.1.1 $\mathbf{J_0(x)}$

```
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$(0, x)$.

3.1.1.2 $\mathbf{J_1(x)}$

```
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$(1, x)$.

3.1.1.3 $\mathbf{J_n(x)}$

```
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:

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

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

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

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[2] (3.1), the final result is normalized with the value of $J_0(x)$.

3.1.1.4 $\mathbf{Y_0(x)}$

```
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$(0, x)$.

3.1.1.5 $\mathbf{Y_1(x)}$

```
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$(1, x)$.

3.1.1.6 $\mathbf{Y_n(x)}$

```
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]):

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

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

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

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

[2] 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).

3.1.2.1 $\mathbf{I_0(x)}$

```
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.

3.1.2.2 Exponentially scaled $\mathbf{I_{0,e}(x)}$

```
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.

3.1.2.3 $\mathbf{I_1(x)}$

```
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.

3.1.2.4 Exponentially scaled $\mathbf{I_{1,e}(x)}$

```
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.

3.1.2.5 $\mathbf{I_n(x)}$

```
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 e.g. Olver et al. [30, 10.25.2]:

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

For $n = 0, 1$ the basic functions $I_0(x)$ or $I_1(x)$ are returned. For $0 \le x \le 2$ the power series is evaluated, otherwise the backward recurrence is used

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

Starting values are computed from the continued fraction (3.2) with $\nu = n$; the final result is normalized with the value of $I_0(x)$.

3.1.2.6 $\mathbf{K_0(x)}$

```
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 $x \le 2$ the result is bess_k0_small(x), otherwise $e^{-x}K_{0,e}(x)$ is returned.

3.1.2.7 Exponentially scaled $\mathbf{K_{0,e}(x)}$

```
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 $x \le 2$ the common function returns bess_k0_small(x)e^x, otherwise two Chebyshev approximations from Fullerton [20, 14] (file dbsk0e.f) are used.

3.1.2.8 $\mathbf{K_1(x)}$

```
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 $x \le 2$ the result is bess_k1_small(x), otherwise $e^{-x}K_{1,e}(x)$ is returned.

3.1.2.9 Exponentially scaled $\mathbf{K_{1,e}(x)}$

```
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 $x \le 2$ the common function returns bess_k1_small(x)e^x, otherwise two Chebyshev approximations from Fullerton [20, 14] (file dbsk1e.f) are used.

3.1.2.10 $\mathbf{K_n(x)}$

```
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]):

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

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

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

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; ν 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 $\mu = 4\nu^2$:

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

$$\theta_\nu \sim x - (\tfrac{1}{2}\nu + \tfrac{1}{4})\pi + \frac{\mu-1}{2(4x)} + \frac{(\mu-1)(\mu-25)}{6(4x)^3} + \frac{(\mu-1)(\mu^2-114\mu+1073)}{5(4x)^5}\cdots$$

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 $|\nu| <$ MaxLongint. (For $x < 0$ the functions J_ν and Y_ν are in general complex; and Y_ν 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 ν), the asymptotic expansions are used, and for $\nu < 0$ the functions are computed with the reflection formulas [13, 6.7.19]

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

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_ν, J'_ν, Y_ν, and Y'_ν simultaneously. The first continued fraction CF1 ([13, 6.7.2] and [1, 9.1.73]) evaluates

$$\frac{J'_\nu}{J_\nu} = \frac{\nu}{x} - \frac{J_{\nu+1}}{J_\nu} = \frac{\nu}{x} - \frac{1}{2(\nu+1)/x -} \; \frac{1}{2(\nu+2)/x -} \cdots \tag{3.1}$$

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

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

If $x < 2$ Temme's method [51] 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.

3.1.3.1 $\mathbf{J}_\nu(\mathbf{x})$

```
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 ν:

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

see [30, 10.2.2], if $x < 0$ then ν 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.

3.1.3.2 $\mathbf{Y}_\nu(\mathbf{x})$

```
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 ν, 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 ν 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 $|\sin\nu\pi| = 1$ in this case).

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 $|\nu| <$ 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]):

$$\frac{I_{\nu+1}}{I_\nu} = \frac{I'_\nu}{I_\nu} - \frac{\nu}{x} = \frac{1}{2(\nu+1)/x+}\ \frac{1}{2(\nu+2)/x+}\cdots \tag{3.2}$$

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

The forward recurrence for K_ν is used, if $x < 2$ with a corresponding Temme normalisation [52] 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.

3.1.4.1 $\mathbf{I_\nu(x)}$

```
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 ν, $x \geq 0$ if ν is not an integer, defined as [30, 10.25.2]:

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

The common routine `sfc_iv` handles the four special ν 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.

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

```
function bessel_ive(v, x: double): double;
function bessel_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 ν, $x \geq 0$ if ν 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 bessel_ik.

3.1.4.3 $\mathbf{K}_\nu(\mathbf{x})$

```
function bessel_kv(v, x: double): double;
function bessel_kvx(v, x: extended): extended;
```

These functions return $K_\nu(x)$, the modified Bessel function of the 2nd kind, real order ν, $x > 0$, 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[3] 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 bessel_ik.

3.1.4.4 Exponentially scaled $\mathbf{K}_{\nu,\mathbf{e}}(\mathbf{x})$

```
function bessel_kve(v, x: double): double;
function bessel_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 ν, $x > 0$. 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 bessel_ik.

3.1.5 Airy functions

In this section let $z = (2/3)|x|^{3/2}$. For large negative x the Airy functions and the Scorer function $\mathrm{Gi}(x)$ have asymptotic expansions oscillating with $\cos(z + \pi/4)$ or $\sin(z + \pi/4)$, see Abramowitz and Stegun [1, 10.4.60, 10.4.64, 10.4.87]; therefore the phase information becomes totally unreliable for $x < -(2/\mathrm{eps\_x})^{2/3} \approx -0.7\mathrm{E}13$, and the relative error increases strongly for x less than the square root ($\approx -2.6\mathrm{E}6$).

For $x \geq 0$ the Airy functions can be defined by

$$\mathrm{Ai}(x) = \frac{1}{\pi}\sqrt{\frac{x}{3}}\,K_{1/3}(z),$$
$$\mathrm{Bi}(x) = \sqrt{\frac{x}{3}}\,\left(I_{1/3}(z) + I_{-1/3}(z)\right).$$

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 $\mathrm{Ai}(x)$ or $\mathrm{Bi}(x)$ using Maclaurin series for 'small' x. It evaluates the two series expansions from Olver et al. [30, 9.4.1/9.4.3]:

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

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

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

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

[3] and less than MaxLongint

3.1.5.1 Airy Ai(x)

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

These functions calculate the Airy function $\mathrm{Ai}(x)$. Airy_small(x, f_0, f_1) is returned for $-2 \le x \le 1$ with

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

For $x > 1$ the definition formula is evaluated

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

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

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

3.1.5.2 Airy Bi(x)

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

These functions calculate the Airy function $\mathrm{Bi}(x)$. Airy_small(x, f_0, f_1) is returned for $-1 \le x \le 1$ with

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

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

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

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

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

3.1.5.3 Airy Ai'(x)

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

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

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

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

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

3.1.5.4 Airy Bi'(x)

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

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

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

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

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

3.1.5.5 Scaled Airy $\mathrm{Ai_s}(x)$

```
function airy_ais(x: double): double;
function airy_aisx(x: extended): extended;
```

These functions return the scaled Airy function $\mathrm{Ai}_s(x)$, defined as $\mathrm{Ai}_s(x) = e^z \mathrm{Ai}(x)$ for $x > 0$ and $\mathrm{Ai}_s(x) = \mathrm{Ai}(x)$ otherwise. For $x \le 1$ the common function `sfc_airy_ais` uses the definition, if $x \ge 2^{41}$ the asymptotic expression from [1, 10.4.59]

$$\mathrm{Ai}_s(x) \sim \frac{1}{2}\pi^{-1/2} x^{-1/4}$$

is evaluated; otherwise `bessel_ik` is called to the get the scaled Bessel functions.

3.1.5.6 Scaled Airy $\mathrm{Bi_s}(x)$

```
function airy_bis(x: double): double;
function airy_bisx(x: extended): extended;
```

These functions return the scaled Airy function $\mathrm{Bi}_s(x)$, defined as $\mathrm{Bi}_s(x) = e^{-z} \mathrm{Bi}(x)$ for $x > 0$ and $\mathrm{Bi}_s(x) = \mathrm{Bi}(x)$ otherwise. For $x \le 10^{-20}$ the common function `sfc_airy_bis` uses $\mathrm{Bi}(x)$, if $x \ge 2^{41}$ the asymptotic expression from [1, 10.4.63]

$$\mathrm{Bi}_s(x) \sim \pi^{-1/2} x^{-1/4}$$

is evaluated; otherwise `bessel_ik` is called to the get the scaled Bessel functions, where the $K_{1/3}(z)$ contribution is negligible if $x > 12$.

3.1.5.7 Airy Gi(x)

```
function airy_gi(x: double): double;
function airy_gix(x: extended): extended;
```

These functions calculate the inhomogeneous Airy function $\mathrm{Gi}(x)$ (also called Scorer function), cf. [30, 9.12.19]:

$$\mathrm{Gi}(x) = \frac{1}{\pi}\int_0^\infty \sin\left(xt - \frac{1}{3}t^3\right) \mathrm{d}t$$

The common function `sfc_airy_gi` uses Chebyshev approximations from MacLeod's MISCFUN [22, function AIRYGI], and `sfc_airy_bi` instead of the in-line approximations for $\mathrm{Bi}(x)$.

3.1.5.8 Airy Hi(x)

```
function airy_hi(x: double): double;
function airy_hix(x: extended): extended;
```

These functions calculate the inhomogeneous Airy function $\mathrm{Hi}(x)$ (also called Scorer function), cf. [30, 9.12.20]:

$$\mathrm{Hi}(x) = \frac{1}{\pi}\int_0^\infty \exp\left(xt - \frac{1}{3}t^3\right)\mathrm{d}t$$

The common function `sfc_airy_hi` uses Chebyshev approximations from MacLeod's MISCFUN [22, function AIRYHI], and `sfc_airy_bi` instead of the in-line approximations for $\mathrm{Bi}(x)$.

3.1.6 Kelvin functions

The Kelvin functions of general order ν can be defined as

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

see Abramowitz and Stegun [1, 9.9.1 and 9.9.2]. When $\nu = 0$, suffices and brackets are normally 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 $\mathrm{ber}\,x = \mathrm{ber}_0(x)$ for $0 \le x < 20$ with the series expansion [1, 9.9.10]:

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

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

This function computes the Kelvin function $\mathrm{bei}\,x = \mathrm{bei}_0(x)$ for $0 \le x < 20$ with the series expansion [1, 9.9.10]:

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

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

This function computes the Kelvin function $\ker x = \mathrm{ker}_0(x)$ for $0 < x < 3$ with the series expansion [1, 9.9.12]:

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

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

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

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

This function computes the Kelvin function $\operatorname{kei} x = \operatorname{kei}_0(x)$ for $0 < x < 3$ with the series expansion [1, 9.9.12]:

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

```
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]:

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

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

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

ker and kei are always calculated, ber and bei only if cb=true.

```
procedure ker_kei_med(x: extended; var kr, ki: extended);
```

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

3.1.6.1 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 br = ber(x), and bi = bei(x). If $|x| < 20$ the values are computed with the separate small functions ber_sm and bei_sm, and otherwise kelvin_large is used.

3.1.6.2 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 = 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.

3.1.6.3 Kelvin function ber(x)

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

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

3.1.6.4 Kelvin function bei(x)

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

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

3.1.6.5 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.

3.1.6.6 Kelvin function kei(x)

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

These functions return the Kelvin function 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

3.1.7.1 Spherical Bessel function $\mathbf{j_n}(\mathbf{x})$

```
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 $\mathrm{sinc}(x)$ is used, the result is calculated just from the definition [1, 10.1.1]:

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

3.1.7.2 Spherical Bessel function $\mathbf{y_n}(\mathbf{x})$

```
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]:

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

3.1.7.3 Modified spherical Bessel function $\mathbf{i_n(x)}$

```
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]

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

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

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

which can be expressed by i_n, i.e.

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

3.1.7.4 Exponentially scaled $\mathbf{i_{n,e}(x)}$

```
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

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

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

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

3.1.7.5 Modified spherical Bessel function $\mathbf{k_n(x)}$

```
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]:

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

3.1.7.6 Exponentially scaled $\mathbf{k_{n,e}(x)}$

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

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

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

3.1.8 Struve functions

For real ν, x 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])[4]:

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

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

$\mathbf{H}_\nu(x)$ has the asymptotic expansion for large $x > 0$ [1, 12.1.29]:

$$\begin{aligned}\mathbf{H}_\nu(x) &\sim Y_\nu(x) + \frac{1}{\pi} \sum_{k=0}^{\infty} \frac{\Gamma(k+\frac{1}{2})}{\Gamma(\nu+\frac{1}{2}-k)(\frac{1}{2}x)^{2k-\nu+1}} \\ &\sim Y_\nu(x) + \frac{\left(\frac{1}{2}x\right)^{\nu-1}}{\sqrt{\pi}\Gamma(\nu+\frac{3}{2})} \sum_{k=0}^{\infty} (\tfrac{1}{2})_k (\tfrac{1}{2}-\nu)_k \left(-\frac{4}{x^2}\right)^k\end{aligned}$$

and the integral representation for $x > 0$ [1, 12.1.8]:

$$\mathbf{H}_\nu(x) = Y_\nu(x) + \frac{2(\frac{1}{2}x)^k}{\sqrt{\pi}\Gamma(\nu+\frac{1}{2})} \int_0^\infty e^{-xt}(1+t^2)^{\nu-\frac{1}{2}} \mathrm{d}t$$

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

$$\begin{aligned}\mathbf{L}_\nu(x) &\sim I_{-\nu}(x) - \frac{1}{\pi} \sum_{k=0}^{\infty} \frac{(-1)^k \Gamma(k+\frac{1}{2})}{\Gamma(\nu+\frac{1}{2}-k)(\frac{1}{2}x)^{2k-\nu+1}} \\ &\sim I_{-\nu}(x) - \frac{\left(\frac{1}{2}x\right)^{\nu-1}}{\sqrt{\pi}\Gamma(\nu+\frac{3}{2})} \sum_{k=0}^{\infty} (\tfrac{1}{2})_k (\tfrac{1}{2}-\nu)_k \left(\frac{4}{x^2}\right)^k\end{aligned}$$

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

```
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.[5]

3.1.8.2 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.

[4] For the sums with Pochhammer symbol see http://functions.wolfram.com/03.09.06.0027.01 for $\mathbf{H}_\nu(x)$ and http://functions.wolfram.com/03.10.06.0028.01 for $\mathbf{L}_\nu(x)$.

[5] The function in [22] has a third approximation instead of Y_0.

3.1.8.3 Struve function $\mathbf{H}_\nu(\mathbf{x})$

```
function struve_h(v, x: double): double;
function struve_hx(v, x: extended): extended;
```

These functions compute $\mathbf{H}_\nu(x)$, the Struve function of order ν[6]. Negative x are allowed only for integer $\nu = n$ and then the reflection formula $\mathbf{H}_n(-x) = (-1)^{n+1}\mathbf{H}_n(x)$ is applied.

When $\nu = 0, 1$ the basic functions $\mathbf{H}_0(x)$ or $\mathbf{H}_1(x)$ are returned and for negative half-integers $\mathbf{H}_{-n-\frac{1}{2}}(x) = Y_{-n-\frac{1}{2}}(x)$. For $\nu = \frac{1}{2}$ there is the special case [1, 12.1.16]:

$$\mathbf{H}_{\frac{1}{2}}(x) = \left(\frac{2}{\pi x}\right)^{\frac{1}{2}}(1-\cos x) = \frac{\operatorname{vers} x}{\sqrt{x\pi/2}}$$

The order $\nu = \frac{1}{2}$ is a border case: For $\nu > \frac{1}{2}$ the functions $\mathbf{H}_\nu(x)$ have no zero other than $x = 0$, but when $\nu \le \frac{1}{2}$ there are an infinite number of irrational real roots[7]. And near these zeros only absolute accuracy is achievable, i.e. the **AMath** functions may have large relative errors.

For $x \ge 0$ the common function `sfc_struve_h` evaluates the power series for small x and the asymptotic expansion for large x; in the medium range[8] the integral representation is used with a custom version of the double-exponential integration routine from **AMTools**.

3.1.8.4 Modified Struve function $\mathbf{L}_\nu(\mathbf{x})$

```
function struve_l(v, x: double): double;
function struve_lx(v, x: extended): extended;
```

These functions return $\mathbf{L}_\nu(x)$, the modified Struve function of order ν. Negative x are allowed only for integer $\nu = n$ where the formula $\mathbf{L}_n(-x) = (-1)^{n+1}\mathbf{L}_n(x)$ is used.

For negative half-integers there is the relation $\mathbf{L}_{-n-\frac{1}{2}}(x) = I_{n+\frac{1}{2}}(x)$; the special cases $\mathbf{L}_\nu(0) = 0,\ \nu > -1$, and $\nu = \frac{1}{2}$ (see [30, 11.4.7])

$$\mathbf{L}_{\frac{1}{2}}(x) = \left(\frac{2}{\pi x}\right)^{\frac{1}{2}}(\cosh x - 1)$$

are handled separately, otherwise the common function `sfc_struve_l` evaluates the power series for small and medium x and the asymptotic expansion for large x.

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

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

These legacy routines return the Struve function $\mathbf{L}_0(x)$, in the current implementation they just call `sfc_struve_l(0,x)`.

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

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

These legacy routines return the Struve function $\mathbf{L}_1(x)$, in the current implementation they just call `sfc_struve_l(1,x)`.

[6] ν is written as v in the source code.

[7] Asymptotically for large x the functions look like a damped sin : $\mathbf{H}_\nu(x) \sim \sqrt{\frac{2}{\pi x}}\sin(x-(2\nu+1)\pi/4)$; see http://functions.wolfram.com/03.09.06.0041.01 ff.

[8] The three ranges depend on x and ν.

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 Legendre style elliptic integrals

3.2.1.1 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 modulus $|k| < 1$

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

The common function `sfc_EllipticK` returns

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

where the AGM is performed in-line, see 3.10.1 for the algorithm.

3.2.1.2 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 modulus $|k| \leq 1$

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

The common function `sfc_EllipticEC` returns

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

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

3.2.1.3 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 modulus $|k| < 1$, and parameter $\nu \neq 1$

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

The common function `sfc_EllipticPiC` returns

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

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

3.2.1.4 Complete elliptic integral D(k)

```
function comp_ellint_d(k: double): double;
function comp_ellint_dx(k: extended): extended;
```

These functions compute the value of the complete Legendre style elliptic integral $D(k)$ with modulus $|k| < 1$:

$$D(k) = \int_0^{\pi/2} \frac{\sin^2 t \, \mathrm{d}t}{\sqrt{1 - k^2 \sin^2 t}} = \frac{K(k) - E(k)}{k^2}.$$

For $|k| < 2 \cdot 10^{-5}$ two terms of the Maclaurin expansion [30, 19.5.3] are used

$$D(k) = \frac{1}{4}\pi \left(1 + \frac{3}{8}k^2 + \frac{15}{64}k^4 + O(k^6)\right),$$

otherwise the common function `sfc_cel_d` returns

$$D(k) = \mathrm{cel2}(k_c, 0, 1)$$

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

3.2.1.5 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

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

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

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

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

For $|k| < 1$ the argument φ is reduced mod π, i.e. $\varphi = n\pi + \varphi'$ with $|\varphi'| \leq \pi/2$. If $|\varphi'| = \pi/2$, i.e. φ 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:

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

And since F is quasi-periodic[9] the total result is

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

[9] See e.g. [30, 19.2.10]

3.2.1.6 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

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

with $|k\sin\varphi|\le 1$. 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 $R(\beta,k)=(E(\beta,k)-k'^2F(\beta,k))/k$ with $k\sin\beta=\sin\varphi$ can be rewritten with the function

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

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 φ is reduced mod π, i.e. $\varphi=n\pi+\varphi'$ with $|\varphi'|\le\pi/2$. If $|\varphi'|=\pi/2$, i.e. φ 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:

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

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

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

3.2.1.7 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

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

with $|k\sin\varphi|\le 1$. 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:

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

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

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

For $|k|<1$ the argument φ is reduced mod π, i.e. $\varphi=n\pi+\varphi'$ with $|\varphi'|\le\pi/2$. Using the quasi-periodicity of Π 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)]:

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

3.2.1.8 Legendre elliptic integral D(φ, k)

```
function ellint_d(phi,k: double): double;
function ellint_dx(phi,k: extended): extended;
```

These functions compute the incomplete Legendre elliptic integral $D(\varphi, k)$

$$D(\varphi, k) = \int_0^{\varphi} \frac{\sin^2 t \, \mathrm{d}t}{\sqrt{1 - k^2 \sin^2 t}} = \left(F(\varphi, k) - E(\varphi, k)\right)/k^2.$$

with $|k \sin \varphi| \leq 1$. For $|k| > 1$ and $|\varphi| < \pi/2$ the reciprocal-modulus transformations for $F(\varphi, k)$ and $E(\varphi, k)$ from [30, 19.7.4] give

$$D(\varphi, k) = D(\arcsin(k \sin \varphi), 1/k)/k^3.$$

If $|k| < 1$ the argument φ is reduced mod π, i.e. $\varphi = n\pi + \varphi'$ with $|\varphi'| \leq \pi/2$. Using the quasi-periodicity [30, 19.2.10] of $D(\varphi, k)$ the total integral is evaluated with the complete integral $D(k)$ and the Carlson function R_D, see [12, (4.4)]:

$$D(\varphi, k) = 2nD(k) + \frac{1}{3}(\sin \varphi')^3 R_D(\cos^2 \varphi', 1 - k^2 \sin^2 \varphi', 1)$$

3.2.2 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.

3.2.2.1 Degenerate elliptic integral R_C

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

These functions compute Carlson's degenerate elliptic integral R_C for $x \geq 0, y \neq 0$:

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

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

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

R_C is calculated with Carlson's [12] Algorithm 2, cf. Press et al. [13, 6.11], function rc.

3.2.2.2 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

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

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[10], therefore **AMath** scales the error with $s^{4.5}$.

3.2.2.3 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

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

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).

3.2.2.4 Integral of the 2nd kind R_G

```
function ell_rg(x,y,z: double): double;
function ell_rgx(x,y,z: extended): extended;
```

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

$$R_G\,(x,y,z) = \frac{1}{4}\int_0^\infty \frac{1}{\sqrt{t+x}\,\sqrt{t+y}\,\sqrt{t+z}}\left(\frac{x}{t+x}+\frac{y}{t+y}+\frac{z}{t+z}\right)t\,\mathrm{d}t$$

from NIST [30, 19.23.7]. The common function `sfc_ell_rg` handles the special cases from [30, 19.20.4] and the asymptotic approximation [30, 19.27.4]. The connection formula [30, 19.21.10] (assuming the ordering $x \geq z \geq y$ to avoid cancellation)

$$2R_G\,(x,y,z) = zR_F\,(x,y,z) - \tfrac{1}{3}(x-z)(y-z)R_D\,(x,y,z) + \sqrt{xy/z}, \quad z \neq 0$$

is used for $z \geq y > 0$, and for $y = 0$ the relation to the complete Bulirsch function[11]

$$R_G\,(x,0,z) = \tfrac{1}{2}\sqrt{x}\,\mathrm{cel2}\left(\frac{z}{x}, 1, \sqrt{\frac{z}{x}}\right)$$

3.2.2.5 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

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

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.

[10] Also noticed in [19], function ellint_rf.hpp: Boost uses $s^{4.25}$.
[11] This follows from Carlson [67, (56)] $E(k) = 2R_G(0, 1-k^2, 1)$ if $E(k)$ is expressed with cel2.

3.2.3 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).

3.2.3.1 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

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

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

3.2.3.2 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

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

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 $\mathrm{cel}(k_c, 1, a, b)$ for $ab < 0$.

3.2.3.3 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

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

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]. Every linear combination of $K(k)$ and $E(k)$ or $\Pi(\nu, k)$ can be calculated with one call to cel, cf. [11, 1.2.2]:

$$\begin{aligned}\lambda K(k) + \mu E(k) &= \mathrm{cel}(k_c, 1, \lambda+\mu, \lambda+\mu k_c^2),\\ \lambda K(k) + \mu \Pi(\nu, k) &= \mathrm{cel}(k_c, 1-\nu, \lambda+\mu, \lambda(1-\nu)+\mu).\end{aligned}$$

3.2.3.4 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

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

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

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

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

3.2.3.5 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

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

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

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

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

3.2.3.6 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

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

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]:

$$\mathrm{el3}(x, k_c, p) = xR_F(1, 1+k_c^2x^2, 1+x^2) + \tfrac{1}{3}(1-p)x^3 R_J(1, 1+k_c^2x^2, 1+x^2, 1+px^2)$$

3.2.4 Maple style elliptic integrals

3.2.4.1 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

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

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

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

where the AGM is computed in-line.

3.2.4.2 Complete integral of the 1st kind for imaginary modulus

```
function EllipticKim(k: double): double;
function EllipticKimx(k: extended): extended;
```

These functions compute the complete elliptic integral of the first kind

$$\mathrm{EllipticKim}(k) = K(ik) = \int_0^1 \frac{\mathrm{d}t}{\sqrt{(1-t^2)(1+k^2t^2)}}$$

for the imaginary modulus ik with $k \in \mathbb{R}$. The basic formula comes from the imaginary-modulus transformation [30, 19.7.5], which reduces in the complete case to

$$K(ik) = \frac{1}{\sqrt{1+k^2}} K\left(\frac{k}{\sqrt{1+k^2}}\right).$$

It is applied by the common function `sfc_EllipticKim` for $|k| < 1$, for $1 \le |k| < 10^5$ the complementary complete integral of the 1st kind K' is used (in order to avoid inaccuracies for the transformed argument near 1):

$$K(ik) = \frac{1}{\sqrt{1+k^2}} \,\mathrm{EllipticCK}\left(\frac{1}{\sqrt{1+k^2}}\right),$$

and if $|k| \ge 10^5$ the result is calculated with the asymptotic expansion

$$K(ik) = \frac{1}{|k|}\left(\ln(4|k|) + \frac{1}{4k^2}\Big(1 - \ln(4|k|)\Big) + O(k^{-4})\right).$$

3.2.4.3 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

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

with $k \neq 0$. The common function `sfc_EllipticCK` returns

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

3.2.4.4 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

$$\mathrm{EllipticEC}(k) = \int_0^1 \frac{\sqrt{1-k^2t^2}}{\sqrt{1-t^2}}\,\mathrm{d}t$$

for $|k| \le 1$. The common function `sfc_EllipticEC` returns

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

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

3.2.4.5 Complete integral of the 2nd kind for imaginary modulus

```
function EllipticECim(k: double): double;
function EllipticECimx(k: extended): extended;
```

These functions compute the complete elliptic integral of the second kind

$$\mathrm{EllipticECim}(k) = E(ik) = \int_0^1 \frac{\sqrt{1+k^2t^2}}{\sqrt{1-t^2}}\,\mathrm{d}t$$

for the imaginary modulus ik with $k \in \mathbb{R}$. The basic formula comes from the imaginary-modulus transformation [30, 19.7.5], which reduces in the complete case to

$$E(ik) = \sqrt{1+k^2}\,\mathrm{EllipticEC}\left(\frac{1}{\sqrt{1+k^2}}\right),$$

The common function `sfc_EllipticECim` handles the special cases $E(ik) \approx \frac{\pi}{2}$ for $|k| \le 6.5 \cdot 10^{-10}$ and $E(ik) \approx |k|$ for $|k| \ge 1.6 \cdot 10^{10}$, otherwise the result is computed with the complete Bulirsch function of the 2nd kind

$$E(ik) = \mathrm{cel2}\big(\kappa_c, 1, \kappa_c^2\big)/\kappa_c, \quad \kappa_c = \frac{1}{\sqrt{1+k^2}}.$$

3.2.4.6 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

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

The common function `sfc_EllipticCE` returns

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

3.2.4.7 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 with $|k| < 1$, $\nu \neq 1$

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

The common function `sfc_EllipticPiC` returns (Bulirsch[11, 1.2.2]):

$$\mathrm{EllipticPiC}(\nu, k) = \mathrm{cel}(\sqrt{1-k^2}, 1-\nu, 1, 1)$$

3.2.4.8 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 $|k| \neq 0$, $\nu \neq 1$

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

The common function `sfc_EllipticCPi` returns

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

3.2.4.9 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 $|z| \leq 1$ and $|kz| \leq 1$

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

The common function `sfc_EllipticF` returns $\arcsin z$ if $k = 0$, and

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

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

3.2.4.10 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 second kind with $|z| \leq 1$ and $|kz| \leq 1$

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

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

3.2.4.11 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 $|z| \le 1$ and $|kz| \le 1$

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

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].

3.2.5 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 $|k| \le 1$ (cf. Abramowitz and Stegun [30, 17.4.39/17.4.40], where a slightly different notation is used)

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

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

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

Heuman's function Λ_0 is quasi-periodic with

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

If $|\varphi'| = \pi/2$, i.e. φ 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]:

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

3.2.6 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 $|k| \le 1$

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

Zeta is periodic[12] $Z(\varphi + \pi, k) = Z(\varphi, k)$, $Z(\pi/2, k) = 0$, and for $|k| = 1$ we have[13]

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

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

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

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

3.2.8 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 $|k| \le 0.125$ 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[14]

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

[12] Cf. http://functions.wolfram.com/08.07.04.0010.01
[13] Cf. http://functions.wolfram.com/08.07.03.0002.01
[14] Cf. http://functions.wolfram.com/09.53.06.0003.01: **AMath** uses only the most significant terms of the given expansion.

3.2.9 Jacobi amplitude

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

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

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

with the special case $\mathrm{am}(x,0) = x$. When $|k| > 1$, $\mathrm{am}(x,k)$ is periodic with period $4K(1/k)/k$, and if $|k| = 1$, then it is equal to the Gudermannian function $\mathrm{am}(x,\pm1) = \mathrm{gd}(x)$.

The common function `sfc_jam` handles the special cases, and for $|k| > 1$ it calls `sfc_sncndn` and returns[15]

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

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

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

3.2.10 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 $\mathrm{sn}, \mathrm{cn}, \mathrm{dn}$ 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

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

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])

$$\mathrm{sn}(x,k) = \sin(\mathrm{am}(x,k)), \qquad \mathrm{cn}(x,k) = \cos(\mathrm{am}(x,k)),$$

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

3.2.10.1 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 $\mathrm{sn}(x,k)$. The common function `sfc_jacobi_sn` calls `sfc_sncndn` with $m_c = (1-k)(1+k)$.

[15] The calculation with arctan2 is slightly more accurate than arcsin(sn), and because sn and cn are computed simultaneously there is no time penalty.

3.2.10.2 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 $\mathrm{cn}(x,k)$. The common function `sfc_jacobi_cn` calls `sfc_sncndn` with $m_c = (1-k)(1+k)$.

3.2.10.3 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 $\mathrm{dn}(x,k)$. The common function `sfc_jacobi_dn` calls `sfc_sncndn` with $m_c = (1-k)(1+k)$.

3.2.10.4 Jacobi elliptic function nc

```
function jacobi_nc(x,k: double): double;
function jacobi_ncx(x,k: extended): extended;
```

These functions return the Jacobi elliptic function $\mathrm{nc}(x,k)$. The common function `sfc_jacobi_nc` calls `sfc_sncndn` with $m_c = (1-k)(1+k)$ and returns 1/cn.

3.2.10.5 Jacobi elliptic function sc

```
function jacobi_sc(x,k: double): double;
function jacobi_scx(x,k: extended): extended;
```

These functions return the Jacobi elliptic function $\mathrm{sc}(x,k)$. The common function `sfc_jacobi_sc` calls `sfc_sncndn` with $m_c = (1-k)(1+k)$ and returns sn/cn.

3.2.10.6 Jacobi elliptic function dc

```
function jacobi_dc(x,k: double): double;
function jacobi_dcx(x,k: extended): extended;
```

These functions return the Jacobi elliptic function $\mathrm{dc}(x,k)$. The common function `sfc_jacobi_dc` calls `sfc_sncndn` with $m_c = (1-k)(1+k)$ and returns dn/cn.

3.2.10.7 Jacobi elliptic function nd

```
function jacobi_nd(x,k: double): double;
function jacobi_ndx(x,k: extended): extended;
```

These functions return the Jacobi elliptic function $\mathrm{nd}(x,k)$. The common function `sfc_jacobi_nd` calls `sfc_sncndn` with $m_c = (1-k)(1+k)$ and returns 1/dn.

3.2.10.8 Jacobi elliptic function sd

```
function jacobi_sd(x,k: double): double;
function jacobi_sdx(x,k: extended): extended;
```

These functions return the Jacobi elliptic function $\mathrm{sd}(x,k)$. The common function `sfc_jacobi_sd` calls `sfc_sncndn` with $m_c = (1-k)(1+k)$ and returns sn/dn.

3.2.10.9 Jacobi elliptic function cd

```
function jacobi_cd(x,k: double): double;
function jacobi_cdx(x,k: extended): extended;
```

These functions return the Jacobi elliptic function $\mathrm{cd}(x,k)$. The common function `sfc_jacobi_cd` calls `sfc_sncndn` with $m_c = (1-k)(1+k)$ and returns cn/dn.

3.2.10.10 Jacobi elliptic function ns

```
function jacobi_ns(x,k: double): double;
function jacobi_nsx(x,k: extended): extended;
```

These functions return the Jacobi elliptic function $\mathrm{ns}(x,k)$. The common function `sfc_jacobi_ns` calls `sfc_sncndn` with $m_c = (1-k)(1+k)$ and returns 1/sn.

3.2.10.11 Jacobi elliptic function cs

```
function jacobi_cs(x,k: double): double;
function jacobi_csx(x,k: extended): extended;
```

These functions return the Jacobi elliptic function $\mathrm{cs}(x,k)$. The common function `sfc_jacobi_cs` calls `sfc_sncndn` with $m_c = (1-k)(1+k)$ and returns cn/sn.

3.2.10.12 Jacobi elliptic function ds

```
function jacobi_ds(x,k: double): double;
function jacobi_dsx(x,k: extended): extended;
```

These functions return the Jacobi elliptic function $\mathrm{dn}(x,k)$. The common function `sfc_jacobi_ds` calls `sfc_sncndn` with $m_c = (1-k)(1+k)$ and returns dn/sn.

3.2.11 Inverse Jacobi elliptic functions

The inverse Jacobi elliptic functions can be defined like the inverse trigonometric functions: e.g. if $\mathrm{sn}(y,k) = x$ then $y = \mathrm{arcsn}(x,k)$; they are multivalued and the **AMath** functions return their principal values.

The functions can be represented as elliptic integrals [30, §22.15(ii)][16] and they are computed with the incomplete elliptic integral $F(.,k)$ using the table from Abramowitz and Stegun [1, p.596].

The standard range for k is $0 \le k \le 1$. **AMath** forces $k \ge 0$, and $k > 1$ is normally handled with the computation of $F(.,k)$, where the common function `sfc_ellint_1` performs a reciprocal-modulus transformation [30, 19.7.4]; explicit formulas are given in the fourth column of the table below. The third column lists the primary relations to $F(.,k)$ and inverse trigonometric functions; note that most of them are valid only for restricted argument ranges, more complete specifications are given in the subsections of the specific functions.

[16] The complete table is available on-line from http://dlmf.nist.gov/22.15#ii

$\operatorname{arcsn}(x,k)$	$\int_0^x \frac{\mathrm{d}t}{\sqrt{(1-t^2)(1-k^2t^2)}}$	$F\big(\arcsin(x),k\big)$	$\frac{1}{k}\operatorname{arcsn}(kx,\frac{1}{k})$
$\operatorname{arccn}(x,k)$	$\int_x^1 \frac{\mathrm{d}t}{\sqrt{(1-t^2)(k_c^2+k^2t^2)}}$	$F\big(\arccos(x),k\big)$	$\frac{1}{k}\operatorname{arcdn}(x,\frac{1}{k})$
$\operatorname{arcdn}(x,k)$	$\int_x^1 \frac{\mathrm{d}t}{\sqrt{(1-t^2)(t^2-k_c^2)}}$	$F\big(\arcsin(\sqrt{\frac{1-x^2}{k^2}}),k\big)$	$\frac{1}{k}\operatorname{arccn}(x,\frac{1}{k})$
$\operatorname{arccd}(x,k)$	$\int_x^1 \frac{\mathrm{d}t}{\sqrt{(1-t^2)(1-k^2t^2)}}$	$F\big(\arcsin(\sqrt{\frac{1-x^2}{1-k^2x^2}}),k\big)$	$\frac{1}{k}\operatorname{arcdc}(x,\frac{1}{k})$
$\operatorname{arcsd}(x,k)$	$\int_0^x \frac{\mathrm{d}t}{\sqrt{(1-k_c^2t^2)(1+k^2t^2)}}$	$F\big(\arcsin(\frac{x}{\sqrt{1+k^2x^2}}),k\big)$	$\frac{1}{k}\operatorname{arcsc}(kx,\frac{1}{k})$
$\operatorname{arcnd}(x,k)$	$\int_1^x \frac{\mathrm{d}t}{\sqrt{(t^2-1)(1-k_c^2t^2)}}$	$F(\arcsin(\sqrt{\frac{x^2-1}{k^2x^2}}),k)$	$\frac{1}{k}\operatorname{arcnc}(x,\frac{1}{k})$
$\operatorname{arcdc}(x,k)$	$\int_1^x \frac{\mathrm{d}t}{\sqrt{(t^2-1)(t^2-k^2)}}$	$F(\arcsin(\sqrt{\frac{1-x^2}{k^2-x^2}}),k)$	$\frac{1}{k}\operatorname{arccd}(x,\frac{1}{k})$
$\operatorname{arcnc}(x,k)$	$\int_1^x \frac{\mathrm{d}t}{\sqrt{(t^2-1)(k^2+k_c^2t^2)}}$	$F(\operatorname{arcsec}(x),k)$	$\frac{1}{k}\operatorname{arcnd}(x,\frac{1}{k})$
$\operatorname{arcsc}(x,k)$	$\int_0^x \frac{\mathrm{d}t}{\sqrt{(1+t^2)(1+k_c^2t^2)}}$	$F(\arctan(x),k)$	$\frac{1}{k}\operatorname{arcsd}(kx,\frac{1}{k})$
$\operatorname{arcns}(x,k)$	$\int_x^\infty \frac{\mathrm{d}t}{\sqrt{(t^2-1)(t^2-k^2)}}$	$F(\operatorname{arccsc}(x),k)$	$\frac{1}{k}\operatorname{arcns}(\frac{x}{k},\frac{1}{k})$
$\operatorname{arcds}(x,k)$	$\int_x^\infty \frac{\mathrm{d}t}{\sqrt{(t^2+k^2)(t^2-k_c^2)}}$	$F(\arcsin(\frac{1}{\sqrt{k^2+x^2}}),k)$	$\frac{1}{k}\operatorname{arccs}(\frac{x}{k},\frac{1}{k})$
$\operatorname{arccs}(x,k)$	$\int_x^\infty \frac{\mathrm{d}t}{\sqrt{(1+t^2)(t^2+k_c^2)}}$	$F(\operatorname{arccot}(x),k)$	$\frac{1}{k}\operatorname{arcds}(\frac{x}{k},\frac{1}{k})$

3.2.11.1 Inverse Jacobi elliptic function arcsn

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

These functions compute the inverse Jacobi elliptic function $\operatorname{arcsn}(x,k)$ for $|x| \le 1$ and $|kx| \le 1$. The common function `sfc_jacobi_arcsn` handles the special case $\operatorname{arcsn}(x,1) = \operatorname{arctanh}(x)$, otherwise it returns

$$\operatorname{arcsn}(x,k) = \int_0^x \frac{\mathrm{d}t}{\sqrt{(1-t^2)(1-k^2t^2)}} = F\big(\arcsin(x),k\big).$$

3.2.11.2 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 $\operatorname{arccn}(x,k)$ for $|x| < 1$ if $k \le 1$, and $x^2 > 1 - 1/k^2$ if $k > 1$. The common function `sfc_jacobi_arccn` handles the special case $\operatorname{arccn}(x,1) = \operatorname{arcsech}(x)$, otherwise it returns

$$\operatorname{arccn}(x,k) = \int_x^1 \frac{\mathrm{d}t}{\sqrt{(1-t^2)(k_c^2+k^2t^2)}} = F\big(\arcsin\big(\frac{1}{\sqrt{1-x^2}}\big),k\big) = F(\arccos(x),k).$$

3.2.11.3 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 $\operatorname{arcdn}(x,k)$ for $0 \le x \le 1$ and $k^2 + x^2 > 1$ if $|k| < 1$; and $|x| \le 1$ if $|k| > 1$. The common function `sfc_jacobi_arcdn`

handles the special case $\operatorname{arcdn}(x,1) = \operatorname{arcsech}(x)$, for $x \geq 0$ it returns

$$\operatorname{arcdn}(x,k) = \int_x^1 \frac{\mathrm{d}t}{\sqrt{(1-t^2)(t^2-k_c^2)}} = F\left(\arcsin\left(\sqrt{\frac{1-x^2}{k^2}}\right), k\right),$$

and for negative x and $|k| > 1$ the result is

$$\operatorname{arcdn}(x,k) = \frac{2}{k}K\left(\frac{1}{k}\right) - \operatorname{arcdn}(|x|,k).$$

3.2.11.4 Inverse Jacobi elliptic function arccd

```
function jacobi_arccd(x,k: double): double;
function jacobi_arccdx(x,k: extended): extended;
```

These functions compute the inverse Jacobi elliptic function $\operatorname{arccd}(x,k)$ for $|x| \leq 1$ if $|k| < 1$; and $|x| \geq 1$ if $|k| > 1$. The common function `sfc_jacobi_arccd` handles the special cases $\operatorname{arccd}(0,k) = 0$ and $\operatorname{arccd}(x,0) = \arccos(x)$ and the primary definition

$$\operatorname{arccd}(x,k) = \int_x^1 \frac{\mathrm{d}t}{\sqrt{(1-t^2)(1-k^2t^2)}} = F\left(\arcsin\left(\sqrt{\frac{1-x^2}{1-k^2x^2}}\right), k\right)$$

is applied for $1/16 < x < 16$; for other ranges the following relations are used:

$$\operatorname{arccd}(x,k) = K(k) - \operatorname{arcsn}(x,k),$$
$$\operatorname{arccd}(x,k) = \operatorname{arcdc}(1/x,k) = \operatorname{arccd}(1/x,1/k)/k.$$

3.2.11.5 Inverse Jacobi elliptic function arcsd

```
function jacobi_arcsd(x,k: double): double;
function jacobi_arcsdx(x,k: extended): extended;
```

These functions compute the inverse Jacobi elliptic function $\operatorname{arcsd}(x,k)$ for $x \in \mathbb{R}$ if $|k| \geq 1$ and $|x| < 1/\sqrt{1-k^2}$ otherwise. The common function `sfc_jacobi_arcsd` handles the special case $\operatorname{arcsd}(x,1) = \operatorname{arcsinh}(x)$, otherwise it returns

$$\operatorname{arcsd}(x,k) = \int_0^x \frac{\mathrm{d}t}{\sqrt{(1-k_c^2t^2)(1+k^2t^2)}} = F\left(\arcsin\left(\frac{x}{\sqrt{1+k^2x^2}}\right), k\right)$$

if $|k| < 1$. For $|k| > 1$ the reciprocal-modulus transformation gives

$$\operatorname{arcsd}(x,k) = \operatorname{arcsc}(kx,1/k)/k = F\big(\arctan(kx),1/k\big)/k.$$

3.2.11.6 Inverse Jacobi elliptic function arcnd

```
function jacobi_arcnd(x,k: double): double;
function jacobi_arcndx(x,k: extended): extended;
```

These functions compute the inverse Jacobi elliptic function $\operatorname{arcnd}(x,k)$ for $x \geq 1$ and $x^2 \leq k^2/(1-k^2)$ if $|k| < 1$. The common function `sfc_jacobi_arcnd` handles the special case $\operatorname{arcnd}(x,1) = \operatorname{arccosh}(x)$, otherwise it returns

$$\operatorname{arcnd}(x,k) = \int_1^x \frac{\mathrm{d}t}{\sqrt{(t^2-1)(1-k_c^2t^2)}} = F\left(\arcsin\left(\sqrt{\frac{x^2-1}{k^2x^2}}\right), k\right),$$

except if $|k| > 1$ and $x > 2$, where the reciprocal-modulus transformation gives

$$\operatorname{arcnd}(x,k) = \operatorname{arcnc}(x,1/k)/k = F\big(\operatorname{arcsec}(x),1/k\big)/k.$$

3.2.11.7 Inverse Jacobi elliptic function arcdc

```
function jacobi_arcdc(x,k: double): double;
function jacobi_arcdcx(x,k: extended): extended;
```

These functions compute the inverse Jacobi elliptic function $\operatorname{arcdc}(x,k)$ for $|x| \geq 1$ if $|k| < 1$; and $|x| \leq 1$ if $|k| > 1$. The common function `sfc_jacobi_arcdc` handles the special cases $\operatorname{arcdc}(1,k) = 0$ and $\operatorname{arcdc}(x,0) = \operatorname{arcsec}(x)$ and the primary definition

$$\operatorname{arcdc}(x,k) = \int_1^x \frac{\mathrm{d}t}{\sqrt{(t^2-1)(t^2-k^2)}} = F\left(\arcsin\left(\sqrt{\frac{1-x^2}{k^2-x^2}}\right), k\right)$$

is applied for $1/16 < |x| < 16$; for $|x| \leq 1/16$ (and $|k| > 1$)

$$\operatorname{arcdc}(x,k) = \big(K(1/k) - \operatorname{arcsn}(x,1/k)\big)/k$$

is used [17], and for $|x| \geq 16$ the result is $\operatorname{arcdc}(x,k) = \operatorname{arccd}(1/x,k)$.

3.2.11.8 Inverse Jacobi elliptic function arcnc

```
function jacobi_arcnc(x,k: double): double;
function jacobi_arcncx(x,k: extended): extended;
```

These functions compute the inverse Jacobi elliptic function $\operatorname{arcnc}(x,k)$ for $x \geq 1$, and $x^2 \leq k^2/(k^2-1)$ for $|k| > 1$. The common function `sfc_jacobi_arcnc` handles the special case $\operatorname{arcnc}(x,1) = \operatorname{arccosh}(x)$, otherwise it returns

$$\operatorname{arcnc}(x,k) = \int_1^x \frac{\mathrm{d}t}{\sqrt{(t^2-1)(k^2+k_c^2t^2)}} = F\big(\arcsin\big(\sqrt{\frac{x^2-1}{x^2}}\big), k\big) = F(\operatorname{arcsec}(x), k).$$

3.2.11.9 Inverse Jacobi elliptic function arcsc

```
function jacobi_arcsc(x,k: double): double;
function jacobi_arcscx(x,k: extended): extended;
```

These functions compute the inverse Jacobi elliptic function $\operatorname{arcsc}(x,k)$ for $x \in \mathbb{R}$ if $|k| \leq 1$ and $|x| \leq 1/\sqrt{k^2-1}$ if $|k| > 1$. The common function `sfc_jacobi_arcsc` handles the special case $\operatorname{arcsc}(x,1) = \operatorname{arcsinh}(x)$, otherwise it returns

$$\operatorname{arcsc}(x,k) = \int_0^x \frac{\mathrm{d}t}{\sqrt{(1+t^2)(1+k_c^2t^2)}} = F\big(\arcsin\big(\frac{x}{\sqrt{1+x^2}}\big), k\big) = F(\arctan(x), k).$$

3.2.11.10 Inverse Jacobi elliptic function arcns

```
function jacobi_arcns(x,k: double): double;
function jacobi_arcnsx(x,k: extended): extended;
```

These functions compute the inverse Jacobi elliptic function $\operatorname{arcns}(x,k)$ for $|x| \geq 1$ if $|k| < 1$ and $|x| \geq |k|$ if $|k| > 1$. The common function `sfc_jacobi_arcns` handles the special case $\operatorname{arcns}(x,1) = \operatorname{arccoth}(x)$, otherwise it returns

$$\operatorname{arcns}(x,k) = \int_x^\infty \frac{\mathrm{d}t}{\sqrt{(t^2-1)(t^2-k^2)}} = F\left(\arcsin\left(1/x\right), k\right) = F(\operatorname{arccsc}(x), k).$$

[17] see http://functions.wolfram.com/09.40.27.0012.01

3.2.11.11 Inverse Jacobi elliptic function arcds

```
function jacobi_arcds(x,k: double): double;
function jacobi_arcdsx(x,k: extended): extended;
```

These functions compute the inverse Jacobi elliptic function $\operatorname{arcds}(x,k)$ for $x \in \mathbb{R}$ if $|k| > 1$ and $|x| \geq \sqrt{1-k^2}$ if $|k| < 1$. The common function `sfc_jacobi_arcds` handles the special case $\operatorname{arcds}(x,1) = \operatorname{arccsch}(x)$, for $|k| < 1$ the primary definition

$$\operatorname{arcds}(x,k) = \int_x^\infty \frac{\mathrm{d}t}{\sqrt{(t^2+k^2)(t^2-k_c^2)}} = F\left(\arcsin\left(\frac{1}{\sqrt{k^2+x^2}}\right), k\right)$$

with sign extension is used, otherwise the reciprocal-modulus transformation gives

$$\operatorname{arcds}(x,k) = \operatorname{arccs}(x/k, 1/k)/k = F\big(\operatorname{arccot}(x/k), 1/k\big)/k.$$

3.2.11.12 Inverse Jacobi elliptic function arccs

```
function jacobi_arccs(x,k: double): double;
function jacobi_arccsx(x,k: extended): extended;
```

These functions compute the inverse Jacobi elliptic function $\operatorname{arccs}(x,k)$ for $x \in \mathbb{R}$ if $|k| < 1$ and $|x| \geq \sqrt{k^2-1}$ if $|k| > 1$. The common function `sfc_jacobi_arccs` handles the special case $\operatorname{arccs}(x,1) = \operatorname{arccsch}(x)$, otherwise it returns

$$\operatorname{arccs}(x,k) = \int_x^\infty \frac{\mathrm{d}t}{\sqrt{(1+t^2)(t^2+k_c^2)}} = F\left(\arcsin\left(\frac{1}{\sqrt{1+x^2}}\right), k\right) = F(\operatorname{arccot}(x), k).$$

3.2.12 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 Olver et al. [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 2π or π:

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

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[18]

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

and similar for the other θ_n. The transformed series used by **AMath** are:

$$\theta_1(x,q) = 2\left(-\frac{\pi}{p}\right)^{1/2} e^{(x^2+\pi^2/4)/p} \sum_{k=0}^{\infty} (-1)^k e^{k(k+1)y} \sinh\big((2k+1)w\big)$$

$$\theta_2(x,q) = \left(-\frac{\pi}{p}\right)^{1/2} e^{x^2/p}\Big(1 + 2\sum_{k=1}^{\infty} (-1)^k e^{k^2 y} \cosh(2kw)\Big)$$

$$\theta_3(x,q) = \left(-\frac{\pi}{p}\right)^{1/2} e^{x^2/p}\Big(1 + 2\sum_{k=1}^{\infty} e^{k^2 y} \cosh(2kw)\Big)$$

$$\theta_4(x,q) = 2\left(-\frac{\pi}{p}\right)^{1/2} e^{(x^2+\pi^2/4)/p} \sum_{k=0}^{\infty} e^{k(k+1)y} \cosh\big((2k+1)w\big),$$

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.

3.2.13 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 $\theta_1(q) \equiv 0$, 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", the theta q series can be restricted to very small q, theoretically to $0 \le q \le \exp(-\pi) = 0.0432139\ldots$ for $z = 0$; see the example in [30, 20.14] "Methods of Computation" for $\theta_3(0.9)$.

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

$$\theta_2(q) = (-\pi/\ln q)^{1/2}\, \theta_4\big(\exp(\pi^2/\ln q)\big)$$
$$\theta_3(q) = (-\pi/\ln q)^{1/2}\, \theta_3\big(\exp(\pi^2/\ln q)\big)$$
$$\theta_4(q) = (-\pi/\ln q)^{1/2}\, \theta_2\big(\exp(\pi^2/\ln q)\big)$$

Additionally for $q < 0$ the relation $\theta_4(q) = \theta_3(-q)$ is applied. The transformation formula for θ_1 from [30, 20.7.30] implies that $\theta_1'(q)$ has the functional equation

$$\theta_1'(q) = (-\pi/\ln q)^{3/2}\, \theta_1'\big(\exp(\pi^2/\ln q)\big).$$

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

$$\theta_1'^{(s)}(q) = 2q^{1/4} \sum_{n=0}^{\infty} (-1)^n (2n+1) q^{n(n+1)}$$

$$\theta_2^{(s)}(q) = 2q^{1/4} \sum_{n=0}^{\infty} q^{n(n+1)}$$

$$\theta_3^{(s)}(q) = 1 + 2\sum_{n=1}^{\infty} q^{n^2}$$

[18] http://functions.wolfram.com/09.01.06.0042.01

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 $q \gtrsim 0.1$.

3.2.13.1 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 $0 \le q < 1$. The common function `sfc_theta1p` uses the q series to compute

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

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

3.2.13.2 Jacobi theta2(q)

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

These functions return $\theta_2(q) = \theta_2(0,q)$ for $0 \le q < 1$. The common function `sfc_theta2` uses the q series to compute

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

3.2.13.3 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

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

3.2.13.4 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

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

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

3.2.14 Lemniscate functions

```
procedure sincos_lemn(x: double; var sl,cl: double);
procedure sincos_lemnx(x: extended; var sl,cl: extended);
```

The lemniscate functions sl, cl or sin lemn and cos lemn were defined by Gauß and others as inverses of a special elliptic integral, see e.g. Cox [69, section 3.B]:

$$x = \int_0^{\operatorname{sl} x} \frac{\mathrm{d}z}{\sqrt{1-z^4}}, \quad \operatorname{cl} x = \operatorname{sl}\left(\frac{\varpi}{2} - x\right), \quad \frac{\varpi}{2} = \int_0^1 \frac{\mathrm{d}z}{\sqrt{1-z^4}},$$

where $\varpi \approx 2.62205755429211981$ is the transcendental Gauß lemniscate constant. The functions have the real[19] period $2\varpi = 2\pi/\operatorname{agm}(\sqrt{2},1) = \Gamma(\frac{1}{4})^2/\sqrt{2\pi} \approx 5.24411510858423962$, and can be expressed as Jacobi elliptic functions with modulus $k = \frac{1}{2}\sqrt{2}$

$$\sin\operatorname{lemn}(x) = \frac{\sqrt{2}}{2}\operatorname{sd}\left(x\sqrt{2}, \frac{\sqrt{2}}{2}\right), \qquad \cos\operatorname{lemn}(x) = \operatorname{cn}\left(x\sqrt{2}, \frac{\sqrt{2}}{2}\right).$$

Although **AMath** uses a Cody/Waite style range reduction the accuracy rapidly decreases for arguments with full-bit mantissa[20] and magnitudes greater than a few 2ϖ due to rounding/truncations errors from the operations with $\sqrt{2}$ and ϖ.

For small $|x| \le 1.75 \cdot 10^{-5}$ the truncated Maclaurin series are used

$$\operatorname{sl} x = x - \tfrac{1}{10}x^5 + O(x^9), \qquad \operatorname{cl} x = 1 - x^2 + \tfrac{1}{2}x^4 + O(x^6),$$

otherwise the common procedure `sfc_lemn` performs the range reduction, invokes the basic procedure `sfc_sncndn` 3.2.10 for the Jacobi elliptic functions with $m_c = 1/2$, and computes sd from sn, dn.

3.2.14.1 sin lemn (x)

```
function sin_lemn(x: double): double;
function sin_lemnx(x: extended): extended;
```

These functions return the lemniscate sine function $\sin\operatorname{lemn}(x)$ with a call to the common procedure `sfc_lemn`.

3.2.14.2 cos lemn (x)

```
function cos_lemn(x: double): double;
function cos_lemnx(x: extended): extended;
```

These functions return the lemniscate cosine function $\cos\operatorname{lemn}(x)$ with a call to the common procedure `sfc_lemn`.

[19] Substituting $z = iz'$ in the defining integral gives $\operatorname{sl}(ix) = i\operatorname{sl}(x)$, which implies $\operatorname{sl}(x + 2i\varpi) = i\operatorname{sl}(x/i + 2\varpi) = i\operatorname{sl}(x/i) = \operatorname{sl}(x)$, and therefore sl and cl are doubly periodic with periods 2ϖ and $2i\varpi$.

[20] But the results are accurate e.g. for the large value $x = 10^{12}$ with its 28 significant mantissa bits.

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 defined by

$$F(x) = e^{-x^2} \int_0^x e^{t^2} \mathrm{d}t.$$

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 Generalized Dawson integral

```
function dawson2(p,x: double): double;
function dawson2x(p,x: extended): extended;
```

These functions calculate the value of the generalized Dawson integral $F(p, x)$, defined for $p \geq 0$, and $x \geq 0$ by

$$F(p, x) = e^{-x^p} \int_0^x e^{t^p} \mathrm{d}t.$$

For p=0 the common function `sfc_gendaw` returns $F(0, x) = x$, for $p = 1$ the result is $F(1, x) = -\mathrm{expm1}(-x)$, otherwise the implementation is based on D. Dijkstra's paper [48]. In this section let $z = x^p$ and $a = 1/p$. For most (p, x) values the integral is computed with the continued fraction [48, 2.6]

$$F(p, x) = \frac{ax}{a + z -} \; \frac{1z}{a + 1 + z -} \; \frac{2z}{a + 2 + z -} \cdots \frac{nz}{a + n + z -} \cdots,$$

for very large z one term of the asymptotic expansion [48, 2.5]

$$F(p, x) \sim ax \sum_{n=0}^{\infty} (1 - a)_n z^{-n-1} \sim \frac{a}{x^{p-1}}, \qquad z = x^p \to \infty$$

is used, and for very small z one term of the Taylor expansion [48, 2.4] gives $F(p, x) = x$. For $x = 1$ and very large or small p values, $F(p, 1)$ is approximated based on expressions with Kummer's confluent hypergeometric M function:

$$F(p, 1) = e^{-1} M(a, a + 1, 1) = M(1, a + 1, -1)$$

3.3.3 Error function erf

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

These functions compute the error function

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

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

3.3.4 Generalized error function erfg

```
function erfg(p,x: double): double;
function erfgx(p,x: extended): extended;
```

These functions compute the generalized error function[21] defined for $p, x \geq 0$ by

$$\mathrm{erfg}(p,x) = \int_0^x e^{-t^p}\,\mathrm{d}t.$$

The function expint3(x) in 3.3.13 is the special case erfg($3,x$). Change of integration variable gives a relation to the non-normalised lower incomplete gamma function

$$\mathrm{erfg}(p,x) = \int_0^x e^{-t^p}\,\mathrm{d}t = \frac{1}{p}\int_0^{x^p} e^{-z} z^{\frac{1}{p}-1}\mathrm{d}z = \frac{1}{p}\,\gamma\left(\frac{1}{p}, x^p\right),$$

valid for $p > 0$. When $p = 0$ the common function `sfc_erfg` returns $e^{-1}x$ and for $p = 1$ the result is $1 - e^{-x} = -\,\mathrm{expm1}(-x)$. For large x^p values $\Gamma(1/p)/p = \Gamma(1 + 1/p)$ is returned. Otherwise erfg(p,x) is calculated with the `sfc_igammal` function; but a copy of the Temme/Gautschi code for the Taylor expansion of $P(a,x)$ is used to avoid the rounding/truncation errors introduced by scaling/rescaling with $\Gamma(1/p)$ if $(p < 1)$ `and` $(1/p > x^p + 0.25)$.

3.3.5 Complementary error function erfc

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

These functions compute the complementary error function

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

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

3.3.6 Exponentially scaled complementary error function

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

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

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

[21] Note that there are at least three other definitions of the generalized error function with different scaling factors in front of the integral: $\Gamma(p)$ in the French and $\Gamma(p)/\sqrt{\pi}$ in the English Wikipedia article, $p/\Gamma(1/p)$ in laser beam shaping articles etc. **AMath** uses the plain definition with factor 1 as implied in the NIST handbook [30, 7.16].

3.3.7 Scaled repeated integrals of erfc

```
function inerfc(n: integer; x: double): double;
function inerfcx(n: integer; x: extended): extended;
```

These functions compute the scaled repeated integrals of complementary error function, defined for $n \geq -1$ using the awkward but standard notation [1, 7.2.1]

$$i^n \operatorname{erfc} x = \int_x^\infty i^{n-1} \operatorname{erfc} t \, \mathrm{d}t = \frac{2}{\sqrt{\pi}} \int_x^\infty \frac{(t-x)^n}{n!} e^{-t^2} \, \mathrm{d}t, \quad (n = 0, 1, 2, \ldots)$$

$$i^{-1} \operatorname{erfc} x = \frac{2}{\sqrt{\pi}} e^{-x^2}, \quad i^0 \operatorname{erfc} x = \operatorname{erfc} x.$$

Following Gautschi [49] the scaled value $e^{x^2} i^n \operatorname{erfc} x$ is returned for $x > 0$. The common function `sfc_inerfc` handles the special cases $n = 0, -1$; for $x > 0$ and $n > 2965$ the result is zero accurate to extended precision, and for $x = 0$ it is (cf.[1, 7.2.7]):

$$i^n \operatorname{erfc} 0 = \frac{1}{2^n \Gamma(\frac{n}{2} + 1)}.$$

When $x > 8.5 + 3.5/n$ the scaled repeated integral is approximated by the asymptotic expansion (derived from [1, 7.2.14]):

$$e^{x^2} i^n \operatorname{erfc} x \sim \frac{2}{\sqrt{\pi}} \sum_{m=0}^{\infty} \frac{(-1)^m (2m+n)!}{n! m! (2x)^{2m+n+1}}$$

In all other parameter cases the integral is computed with the recurrence formula

$$i^n \operatorname{erfc} x = -\frac{x}{n} i^{n-1} \operatorname{erfc} x + \frac{1}{2n} i^{n-2} \operatorname{erfc} x,$$

see [1, 7.2.5]. For $x < 0$ it is stable in the forward direction. In the remaining cases for $x > 0$ it is normally used in the backward direction. If $x > 0.5$ the starting values are taken from the continued fraction (if it converges fast enough) given by Olver et al. [30, 7.18.13]

$$\frac{i^n \operatorname{erfc} x}{i^{n-1} \operatorname{erfc} x} = \frac{1/2}{x+} \, \frac{(n+1)/2}{x+} \, \frac{(n+2)/2}{x+} \cdots,$$

otherwise a Miller-type backward recursion algorithm is attempted with the starting index $m = 10 + (\sqrt{n} + 23/x)^2 > n$; if this m is too large, the forward recurrence is used (and may yield suboptimal accuracy).

3.3.8 Imaginary error function erfi

```
function erfi(x: double): double;
function erfix(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.

3.3.9 Inverse error functions

3.3.9.1 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.

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

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

3.3.9.2 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.

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

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

3.3.10 Probability functions

The following subsections describe the implemented erf related Gaussian probability functions P, Q, Z from Abramowitz and Stegun [1, 26.2].

3.3.10.1 P(x)

```
function erf_p(x: double): double;
function erf_px(x: extended): extended;
```

These functions return the integral $P(x)$

$$P(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2} \mathrm{d}t = \int_{-\infty}^{x} Z(t) \mathrm{d}t = \frac{1}{2}\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right) = \frac{1}{2}\,\text{erfc}\left(-\frac{x}{\sqrt{2}}\right).$$

3.3.10.2 Q(x)

```
function erf_q(x: double): double;
function erf_qx(x: extended): extended;
```

These functions return the integral $Q(x)$

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_{x}^{\infty} e^{-t^2/2} \mathrm{d}t = \int_{x}^{\infty} Z(t) \mathrm{d}t = P(-x).$$

3.3.10.3 Z(x)

```
function erf_z(x: double): double;
function erf_zx(x: extended): extended;
```

These functions compute the Gaussian density function $Z(x)$ using the special accuracy improved **AMath** function `expmx2h`:

$$Z(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} = \texttt{expmx2h}(x)/\sqrt{2\pi}$$

3.3.11 Fresnel integrals

```
procedure Fresnel(x: double; var s,c: double);
function  FresnelC(x: double): double;
function  FresnelS(x: double): double;
procedure Fresnelx(x: extended; var s,c: extended);
function  FresnelCx(x: extended): extended;
function  FresnelSx(x: extended): extended;
```

These procedures and functions return the Fresnel integrals

$$C(x) = \int_0^x \cos(\tfrac{1}{2}\pi t^2)\mathrm{d}t,$$
$$S(x) = \int_0^x \sin(\tfrac{1}{2}\pi t^2)\mathrm{d}t.$$

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

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

In the medium range $1.5 \le |x| < 1.562\text{e}6$ a continued fraction algorithm is used; cf. Press et al. [13] (Ch. 6.9, function frenel).[22] For larger $|x|$ the values are given by the asymptotic expressions:

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

3.3.12 Goodwin-Staton integral

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

These functions return the Goodwin-Staton integral

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

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

[22] The continued fraction evaluates $C(x)+iS(x)$ using the complex error function, the complex arithmetic is performed in-line in procedure `fresnel_cfrac`.

3.3.13 Expint3

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

These functions return for $x > 0$ the integral $\int_0^x e^{-t^3}\,\mathrm{d}t$. The implementation is based on the MISCFUN [22] routine EXP3; it uses two Chebyshev approximations. The function is a special case of the generalized error function erfg (see 3.3.4)

$$\mathrm{expint3}(x) = \mathrm{erfg}(3, x) = \frac{1}{3}\gamma\big(\frac{1}{3}, x^3\big),$$

it is kept as a separate **AMath** routine for historical reasons and because it is more efficient than $\mathrm{erfg}(3, x)$.

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 $x > 0$

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

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

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

3.4.2 Cosine integral Ci

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

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

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

and $\mathrm{Ci}(x) = \mathrm{Ci}(-x)$ 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 $0 < x \le 4$ and for $x > 4$ the function is given by

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

Where the auxiliary functions f and g

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

are computed with five Chebyshev approximations (for details see procedure `auxfg` in unit sfExpInt), si(x) is the shifted sine integral ssi (3.4.15).

3.4.3 Entire cosine integral Cin

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

These functions return the entire cosine integral defined by

$$\operatorname{Cin}(x) = \int_0^x \frac{1-\cos(t)}{t}\,\mathrm{d}t.$$

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

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

and for $x < 0$ the result is Cin($-x$); with the auxiliary functions f and g from 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 defined by

$$\operatorname{Cinh}(x) = \int_0^x \frac{\cosh(t)-1}{t}\,\mathrm{d}t,$$

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

3.4.5 Exponential integral E_1

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

These functions return the exponential integral $\mathrm{E}_1(x)$ for $x \neq 0$

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

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 $E_1(x) = -\operatorname{Ei}(-x)$.

3.4.6 Exponential integral Ei

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

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

$$\mathrm{Ei}(x) = -\ \mathrm{PV}\!\int_{-x}^{\infty} \frac{e^{-t}}{t}\,\mathrm{d}t = \mathrm{PV}\!\int_{-\infty}^{x} \frac{e^{t}}{t}\,\mathrm{d}t.$$

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

3.4.7 Inverse of the exponential integral Ei

```
function ei_inv(x: double): double;
function ei_invx(x: extended): extended;
```

These functions return the functional inverse of the exponential integral, i.e. $\mathrm{Ei}(\mathrm{Ei}^{-1}(x)) = x$ for $x \in \mathbb{R}$[23]. Ei^{-1} can be expressed with li^{-1} using the definition of li :

$$\mathrm{li}(x) = \mathrm{Ei}(\ln x) \quad\Longrightarrow\quad \mathrm{Ei}(x) = \mathrm{li}(e^x) \quad\Longrightarrow\quad \mathrm{Ei}^{-1}(x) = \ln \mathrm{li}^{-1}(x)$$

For $x < -43.1$ the result is $e^{x-\gamma}$ accurate to extended precision, if $x \approx 0$ the linear Taylor approximation is used, otherwise the common function `sfc_ei_inv` returns $\ln \mathrm{li}^{-1}(x)$, where for $x < -2$ a subsequent Newton iteration is performed to give better accuracy.

3.4.8 Entire exponential integral Ein

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

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

$$\mathrm{Ein}(x) = \int_0^x \frac{1 - e^{-t}}{t}\,\mathrm{d}t.$$

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

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

otherwise the function is computed as

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

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

[23] In the current implementation the maximum argument x is $\approx$ `MaxExtended`/ $\ln$(`MaxExtended`), actually $0.10477 \cdot 10^{4929}$ for the extended and $2.53631 \cdot 10^{305}$ for the double version.

3.4.9 Exponential integrals $\mathbf{E_n}$

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

These functions return the exponential integrals of integer order

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

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

$$E_n(0) = \frac{1}{n-1} \qquad (n > 1),$$
$$E_0(x) = \frac{e^{-x}}{x}.$$

These special cases and $\mathrm{E}_1(x)$ are handled separately in the common function `sfc_en`. If $n > 10^4$ the result is calculated with the asymptotic expansion 3.3 for the generalized integral E_p. For $x > 1$ the continued fraction [1, 5.1.22]

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

is used, cf. 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]

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

3.4.10 Generalized exponential integrals $\mathbf{E_p}$

```
function gei(p,x: double): double;
function geix(p,x: extended): extended;
```

These functions return the generalized exponential integrals of real order $p \in \mathbb{R}$

$$E_p(x) = x^{p-1}\int_x^\infty \frac{e^{-t}}{t^p}\,\mathrm{d}t \;= \int_1^\infty \frac{e^{-xt}}{t^p}\,\mathrm{d}t\,,$$

with $x > 0$ for $p \leq 1$, and $x \geq 0$ for $p > 1$. If p is zero or a positive integer, the result is computed with the integer order function `sfc_en`. For large $p \gg 1$ and with $\lambda = x/p$ there is the asymptotic expansion from Olver et al. [30, 8.20(ii)]:

$$E_p(\lambda p) \sim \frac{e^{-\lambda p}}{(\lambda+1)p}\sum_{k=0}^{\infty}\frac{A_k(\lambda)}{(\lambda+1)^{2k}}\,\frac{1}{p^k} \tag{3.3}$$

with $A_0(\lambda) = A_1(\lambda) = 1$; and for $k > 1$ the $A_k(\lambda)$ are polynomials of degree $k-1$:

$$A_2(\lambda) = 1 - 2\lambda, \quad A_3(\lambda) = 1 - 8\lambda + 6\lambda^2, \quad A_4(\lambda) = 1 - 22\lambda + 58\lambda^2 - 24\lambda^3, \ \ldots$$

The terms up to $k = 4$ are enough to give full precision for $x \geq 0$ and $p \geq 10^4$. If $p > 0$ and $x > 11390.2$, then $E_p(x) \approx 0$ to extended precision, otherwise the relations to the incomplete gamma functions are used (for large parameters in logarithmic form)

$$E_p(x) = x^{p-1}\Gamma(1-p,x) = x^{p-1}Q(1-p,x)\,\Gamma(1-p),$$

where the Legendre continued fraction 3.4 is used directly if possible in order to avoid the scaling and rescaling with large $\Gamma(1-p)$ and x^{p-1} terms.

3.4.11 Logarithmic integral li

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

These functions return the logarithmic integral $\mathrm{li}(x)$ defined for $x \geq 0, x \neq 1$ as

$$\mathrm{li}(x) = \mathrm{PV}\int_0^x \frac{1}{\ln(t)}\,\mathrm{d}t \qquad (x \neq 1).$$

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

3.4.12 Inverse of the logarithmic integral

```
function li_inv(x: double): double;
function li_invx(x: extended): extended;
```

These functions return $\mathrm{li}^{-1}(x)$ the functional inverse of the logarithmic integral, i.e. $\mathrm{li}(\mathrm{li}^{-1}(x)) = x$ for $x \in \mathbb{R}$. The common function `sfc_ali` computes $\mathrm{li}^{-1}(x)$ as the zero of the function $f(z) = \mathrm{li}(z) - x$ using Halley iterations

$$\begin{aligned} z_{n+1} &= z_n - \frac{f(z_n)}{f'(z_n)}\left[1 - \frac{f(z_n)}{f'(z_n)}\frac{f''(z_n)}{2f'(z_n)}\right]^{-1} \\ &= z_n - \Delta_n \ln(z_n)\left[1 + \frac{\Delta_n}{2z_n}\right]^{-1} \end{aligned}$$

where $\Delta_n = f(z_n) = \mathrm{li}(z_n) - x$. At most 3 iterations[24] are needed for all x with the following starting values z_0

$$z_0 = z_0(x) = \begin{cases} x\ln x & x > 3.5 \\ 1 + x & 0.75 < x \leq 3.5 \\ 1.45137 + 0.37251 \cdot x & -0.5 < x \leq 0.75 \\ 1 + e^{x-\gamma} & -43.8 < x \leq -0.5 \end{cases}$$

For $x \leq -43.8$ the result is 1 accurate to extended precision.

3.4.13 Hyperbolic sine integral Shi

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

These functions return the hyperbolic sine integral

$$\mathrm{Shi}(x) = \int_0^x \frac{\sinh(t)}{t}\,\mathrm{d}t.$$

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

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

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

[24] Iteration is terminated if the magnitude of the correction term is $< z_{n+1}(\texttt{eps\_x}/2)^{1/2}$, because due to the cubic convergence the next change could be neglected.

3.4.14 Sine integral Si

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

These functions return the sine integral

$$\mathrm{Si}(x) = \int_0^x \frac{\sin(t)}{t}\,\mathrm{d}t.$$

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

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

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

3.4.15 Shifted sine integral si

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

These functions return the shifted sine integral

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

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 $x_0 = 1.92644766\ldots$ is the smallest positive root of si.

3.5 Gamma function and related

The Pascal units **sfGamma** and **sfGamma2** implement the common code for the gamma and related functions.

3.5.1 Gamma functions

At first some important 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 $|x| \le 13$; 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. 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 $|x| \le 1/32$. It uses two polynomial approximations, cf. Cephes [7] gammal.[25]

```
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 $x \ge 8$

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

and is computed with a polynomial approximation (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.

3.5.1.1 Gamma function Γ(x)

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

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

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

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

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

[25] Label `small` in file ldouble/gammal.c

3.5.1.2 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 $|x| <$ 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.

3.5.1.3 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:

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

For $x > 8$ the common function sfc_gstar returns $\exp(\text{sfc\_lngcorr}(x))$, in the range $1 \le x \le 8$ two Chebyshev expansions computed with Maple are used[26], and if $x < 1$ Temme's recursion formula is applied to make $x > 1$:

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

3.5.1.4 Logarithm of the gamma function

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

These functions compute $\ln|\Gamma(x)|$ for $x \neq 0, -1, -2, \ldots$[27]. 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 $\text{lngamma\_small}(x, x-1, x-2, \text{false})$ is returned.

3.5.1.5 Inverse of lnΓ

```
function lngamma_inv(y: double): double;
function lngamma_invx(y: extended): extended;
```

These functions calculate the inverse function of $\ln\Gamma$, i.e. they return $x = \texttt{lngamma\_inv}(y)$ with $\ln\Gamma(x) = y$ for $y \ge -0.12142 > y_m$ (the minimum of $\ln\Gamma(x)$ for positive arguments), the result is greater than $x_m = 1.46163\ldots$ (the positive zero of the ψ function). The common function `sfc_ilng` computes the zero $x > x_m$ of the function $f(x) = \ln\Gamma(y) - x$ using Newton's method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{\ln\Gamma(x_n) - y}{\psi(x_n)},$$

where the starting values x_0 are taken from four different cases.[28]

[26] The rational approximations from [26] are not suitable for extended precision.
[27] signgamma can be used if the sign of $\Gamma(x)$ is needed.
[28] E.g. for $y \ge 10$ from a simplified inverted Stirling approximation.

3.5.1.6 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$(x+1, x, x-1, \text{true})$ is returned.

3.5.1.7 Reciprocal gamma function

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

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

3.5.1.8 Sign of gamma function

```
function signgamma(x: double): double;
function signgammax(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[29].

3.5.1.9 Logarithm and sign of gamma function

```
function lngammas(x: double; var s: integer): double;
function lngammasx(x: extended; var s: integer): extended;
```

These functions return $\ln|\Gamma(x)|$ using lngamma(x) and s is the sign of $\Gamma(x)$.

3.5.2 Incomplete gamma functions

The incomplete gamma functions are defined as

$$\gamma(a,x) = \int_0^x t^{a-1}e^{-t}\,\mathrm{d}t \qquad a \neq 0, -1, -2, \ldots,$$
$$\Gamma(a,x) = \int_x^\infty t^{a-1}e^{-t}\,\mathrm{d}t,$$

or in their normalised forms

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

An important internal routine for the incomplete gamma code is:

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

This function returns $e^{-x}x^a/\Gamma(a)$, 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[30] from Boost[19] file gamma.hpp, function regularised_gamma_prefix.

[29] The sign is meaningless for $x = 0$ or negative integers.
[30] Not for TP 5.0!

3.5.2.1 Normalised incomplete gamma functions

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

These procedures simultaneously compute the normalised incomplete gamma functions

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

for $a \geq 0$ and $x \geq 0$. The implementation of the common function sfc_incgamma[31] 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.[32]

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

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

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

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 upper incomplete gamma function $Q(a,x)$

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

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 $a = 0$.

3.5.2.2 Non-normalised incomplete gamma functions

```
function igamma(a,x: double): double;
function igammax(a,x: extended): extended;
```

These functions return the non-normalised upper incomplete gamma function

$$\Gamma(a,x) = \int_x^\infty t^{a-1} e^{-t}\,\mathrm{d}t$$

for $x \geq 0$. Special cases are $\Gamma(a,0) = \Gamma(a)$ and $\Gamma(0,x) = E_1(x)$.

If $a < 0$, the parameter ranges for the calculation are as follows: For $x > 1/4$ or $a \leq -10000$ the Legendre continued fraction from Abramowitz/Stegun [1, 6.5.31]

$$\Gamma(a,x) = e^{-x} x^a \left(\frac{1}{x+} \frac{1-a}{1+} \frac{1}{x+} \frac{2-a}{1+} \frac{2}{x+} \cdots \right) \tag{3.4}$$

is computed using the Temme/Gautschi form [26, procedure qfraction]. If $a \geq -1/2$ the Taylor expansion for $Q(a,x)$ is used, and otherwise the recurrence relation from Olver et al. [30, 8.8.2]: $\Gamma(a+1,x) = a\Gamma(a,x) + x^a e^{-x}$.

For $a > 0$ the result is just $\Gamma(a,x) = Q(a,x)\Gamma(a)$, if $\Gamma(a)$ will not overflow. For larger a values the continued fraction is used if $e^{-x}x^a$ will not overflow, otherwise the logarithmic form $\ln\Gamma(a,x) = \ln Q(a,x) + \ln\Gamma(a)$ is applied.

[31] Actually in sfc_incgamma_ex where the prefix calculation is optional.
[32] Temme's pqasymp with it's ≈ 10 digit accuracy is not suitable for extended precision.

```
function igammal(a,x: double): double;
function igammalx(a,x: extended): extended;
```

These functions return the non-normalised lower incomplete gamma function $\gamma(a,x)$ for $x \geq 0$ and $a \neq 0, -1, -2, \ldots$

$$\gamma(a,x) = \int_0^x t^{a-1} e^{-t}\,\mathrm{d}t$$

The result is computed from the relations $\gamma(a,x) = \Gamma(a) - \Gamma(a,x)$ or $\gamma(a,x) = P(a,x)\Gamma(a)$ (or it's logarithmic form) except if $x < a - 1/4$ or $x < 4$ and $\Gamma(a)$ overflows: In this case the Taylor expansion for $P(a,x)$ from [26, (5.5)] is used. At the zeros of $\gamma(a,x)$ for $a < 0$ only absolute accuracy is achievable, if it is less than half of the precision debug mode warnings are available.

3.5.2.3 Tricomi's entire incomplete gamma function

```
function igammat(a,x: double): double;
function igammatx(a,x: extended): extended;
```

These functions return Tricomi's entire incomplete gamma function γ^*, defined for $a > 0$ by the integral [30, 8.2.7]

$$\gamma^*(a,x) = \frac{1}{\Gamma(a)} \int_0^1 t^{a-1} e^{-tx}\,\mathrm{d}t, \qquad a > 0,$$

and by analytic continuation with Kummer's confluent hypergeometric M function as an entire function of both arguments for all a, x

$$\gamma^*(a,x) = e^{-x}\,\frac{M(1,a+1,x)}{\Gamma(a+1)} = e^{-x} \sum_{n=0}^{\infty} \frac{x^n}{\Gamma(a+n+1)}.$$

Special cases are $\gamma^*(0,x) = 1$, $\gamma^*(a,0) = 1/\Gamma(a+1)$, and $\gamma^*(-n,x) = x^n$, if $-n$ is a negative integer. Otherwise there are the following relations to other incomplete gamma functions (see Olver et. al [30, 8.2.6]):

$$\gamma^*(a,x) = \frac{x^{-a}}{\Gamma(a)}\,\gamma(a,x) = x^{-a} P(a,x).$$

The common function `sfc_git` handles the special cases, and uses the full hypergeometric function `sfc_1f1` if $x < 0$.

Some of the hard cases for $x > 0$ (where separate $\Gamma(a)$ terms would over/underflow) are handled with code snippets from SLATEC [14, 20, files dgamit.f, d9lgic.f]. When $a > x+1$ or $a < 0, x < 1$ the result is computed with a simple Kummer series, and if $a < 0, x \geq 1$ the function is evaluated as

$$\gamma^*(a,x) = \frac{\Gamma(a) - \Gamma(a,x)}{\Gamma(a)} x^{-a}.$$

At the non-trivial zeros of γ^* (e.g. if the above numerator vanishes, similar for the other parameter cases) only absolute accuracy is achievable; if it is less than half of the extended precision, `sfc_git` can emit debug mode warnings.

3.5.2.4 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 ≥ 0 for success, and < 0 for input errors or iterations failures:

$\geq$ 0	iteration count
= -2	if $a \leq 0$
= -4	if $p < 0$, $q < 0$, or $\lvert p + q - 1 \rvert >$ eps_d
= -6	if 10 iterations were performed
= -7	iteration failed
= -8	x is calculated with unknown accuracy

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 $|p+q-1| \leq$ 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[33].

```
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 lower incomplete gamma function, i.e. they calculate x with $P(a,x) = p$ with $a > 0$, $0 \leq p < 1$ using `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 upper incomplete gamma function, i.e. they calculate x with $Q(a,x) = q$ with $a > 0$, $0 < q \leq 1$, using `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[34] from the Boost[19] file beta.hpp, function ibeta_power_terms.

[33] Together with the Boost[19] interpretation from igamma_inverse.hpp
[34] Not for TP 5.0!

3.5.3.1 Beta function B(x,y)

```
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. If both $x, y > 0$ then the common function `sfc_beta` uses the Lanczos expression from Boost [19, beta.hpp], otherwise when $x \le 0$ or $y \le 0$ and $x + y$ is not an non-positive integer the result is computed as[35]

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

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

3.5.3.2 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 $x, y \neq 0, -1, -2, \ldots$ Since the function is symmetric, let $x \le y$. The implementation is based on the SLATEC [14] routine dlbeta.f. Let $s = x + y$. If $x \ge 8$ then

$$\ln|B(x,y)| = -0.5 \ln y + \ln\sqrt{2\pi} + c_1 + (x - 0.5)\ln(x/s) + y\operatorname{ln1p}(-x/s)$$

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

$$\ln|B(x,y)| = \ln|\Gamma(x)| + c_2 + x - x\ln(s) + (y - 0.5)\operatorname{ln1p}(-x/s)$$

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

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

3.5.3.3 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 $0 \le x \le 1$:

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

There are some special cases

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

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 $bx \le 1$ and $x \le 0.95$ (or the corresponding symmetry transformed relations) the result is computed with a hypergeometric series, cf. NIST handbook [30, 8.17.7], otherwise I_x is evaluated with a continued fraction expansion.[36] 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.

[35] $\operatorname{sign} B(x,y) = \operatorname{sign}\Gamma(x)\operatorname{sign}\Gamma(y)/\operatorname{sign}\Gamma(x+y)$

[36] Actually the two almost identical CF from [7] are merged into a single **AMath** routine.

3.5.3.4 Non-normalised incomplete beta function

```
function beta3(a, b, x: double): double;
function beta3x(a, b, x: extended): extended;
```

These functions return the non-normalised incomplete beta function $B_x(a,b)$, defined for $a>0$, $b>0$, and $0\le x\le 1$ by

$$B_x(a,b)=\int_0^x t^{a-1}(1-t)^{b-1}\,\mathrm{d}t.$$

There are some special cases

$$B_0(a,b)=0,\qquad B_1(a,b)=B(a,b),\qquad B_x(a,1)=\frac{x^a}{a},\qquad B_x(1,b)=\frac{1-(1-x)^b}{b},$$

and the relation $B_{1-x}(a,b)=B(a,b)-B_x(b,a)$, which is used if $x>a/(a+b)$. For $a>0$, $b>0$ the common function `sfc_nnbeta` uses essentially the same routines as `sfc_ibeta`, and $B_x(1,b)$ is evaluated as $-\mathrm{pow1pm1}(-x,b)/b$.

When $a\le 0$ or $b\le 0$, the Gauss hypergeometric function F function is applied: If $a\ne 0$ is no negative integer, the result is[37]

$$B_x(a,b)=\frac{x^a}{a}F(a,1-b,a+1,x),\qquad -a\notin\mathbb{N},$$

else if $b\ne 0$ is no negative integer, then[38]

$$B_x(a,b)=B(a,b)-\frac{(1-x)^b x^a}{b}F(1,a+b,b+1,1-x),\qquad -b\notin\mathbb{N}.$$

3.5.3.5 Inverse normalised incomplete beta function

```
function ibeta_inv(a, b, y: double): double;
function ibeta_invx(a, b, y: extended): extended;
```

These functions calculate the inverse of normalised incomplete beta function for $a,b>0$ and $0\le y\le 1$, i.e. they return $x=$ `ibeta_inv`(a,b,y) with $I_x(a,b)=y$. The common function `sfc_ibeta_inv` is based on Cephes [7, ldouble/incbil.c], the result is computed with Newton iterations or bisections using the starting value $\sqrt{2}\ \mathrm{erfc}^{-1}(2y)$.

3.5.4 Factorials, Pochhammer symbol, binomial coefficient

3.5.4.1 Factorial

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

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

[37] http://functions.wolfram.com/06.19.26.0005.01
[38] http://functions.wolfram.com/06.19.26.0007.01

3.5.4.2 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 ∞. 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[39] with integer $k \geq 0$:

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

3.5.4.3 Logarithm of factorials

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

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

3.5.4.4 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 $k \leq 4000$[40] and a rough estimate shows that there will be no overflow the

[39] Γ is called explicitly for odd $n > 25$.
[40] k is replaced by $n - k$ if it is $> n/2$.

value is computed as a product of quotients as in the definition. Otherwise the `lnbeta` function is used to return

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

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)

3.5.4.5 Pochhammer symbol

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

These functions return the Pochhammer symbol $(a)_x = \Gamma(a+x)/\Gamma(a)$. 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". By convention $(a)_0 = 1$.

If a or $a+x$ are negative integers or zero 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 Γ reflection formula

$$(a)_x = (-1)^x \frac{\Gamma(1-a)}{\Gamma(1-a-x)},$$

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.

3.5.4.6 Relative Pochhammer symbol

```
function poch1(a,x: double): double;
function poch1x(a,x: extended): extended;
```

These functions compute the relative Pochhammer symbol defined as

$$\text{poch1}(a, x) = \frac{(a)_x - 1}{x},$$

accurate even for small x. The common function `sfc_poch1` is a translation of the routine dpoch1.f by Fullerton [14, 20].

If $|x|$ is small, cancellation errors are avoided by using an expansion by Fields and Luke with generalized Bernoulli polynomials. For $x = 0$ the value $\psi(a)$ is returned, otherwise the result is calculated from the definition.

3.5.5 Ratio of gamma functions

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

These functions compute $\Gamma(x)/\Gamma(x+d)$, accurate even for $|d| \ll |x|$. If $x > 0$ and $x+d > 0$ the Pascal code uses Lanczos sums and is based on the corresponding Boost [19] function in gamma.hpp, the case $x < 0$ and $x + d < 0$ is reduced to the former with a reflection formula. If x and $x + d$ have different signs, the result is computed with calls to `lngamma`.

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

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

3.5.6 Psi and polygamma functions

3.5.6.1 Digamma function $\psi(x)$

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

These functions return the digamma or ψ function, which is defined as

$$\psi(x) = \mathrm{d}(\ln\Gamma(x))/\mathrm{d}x = \Gamma'(x)/\Gamma(x), \qquad x \neq 0, -1, -2, \ldots$$

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] is used to make $x > 12$:

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

There are two special cases: First, if $|x - x_0| < 0.2$, where $x_0 = 1.4616321\ldots$ is the positive zero of ψ, 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 in reversed order with [1, 6.3.2]:

$$\psi(n) = -\gamma + \sum_{k=1}^{n-1} k^{-1} \tag{3.5}$$

3.5.6.2 Trigamma function $\psi'(x)$

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

These functions return the trigamma function $\psi'(x)$, $x \neq 0, -1, -2, \ldots$ 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

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

3.5.6.3 Tetragamma function $\psi''(\mathbf{x})$

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

These functions return the tetragamma function $\psi''(x)$, $x \neq 0, -1, -2, \ldots$ 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

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

3.5.6.4 Pentagamma function $\psi'''(\mathbf{x})$

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

These functions return the pentagamma function $\psi'''(x)$, $x \neq 0, -1, -2, \ldots$ 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

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

3.5.6.5 Polygamma function $\psi^{(\mathbf{n})}(\mathbf{x})$

```
function polygamma(n: integer; x: double): double;
function polygammax(n: integer; x: extended): extended;
```

These functions return the polygamma function $\psi^{(n)}(x)$, i.e. the n^{th} derivative of the ψ function with $n \geq 0$. x must be $\neq 0, -1, -2, \ldots$ 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]

$$\psi^{(n)}(x) = (-1)^{n+1} n! \sum_{k=0}^{\infty} (x + k)^{-n-1} = (-1)^{n+1} n!\, \zeta(n + 1, x). \tag{3.6}$$

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:

$$\psi^{(n)}(x) \sim (-1)^{n-1} \left(\frac{(n-1)!}{x^n} + \frac{n!}{2x^{n+1}} + \sum_{k=1}^{\infty} B_{2k} \frac{(2k + n - 1)!}{(2k)! x^{2k+n}} \right)$$

In extreme parameter cases a term in the asymptotic formula can cause overflow (e.g. for polygamma$(5000, 200) \approx -0.149687668E4819$) or there is no convergence within reasonable limits (e.g. for polygamma$(4000, 200) \approx -0.69367177835736E3467$); in these situations the series expansion (3.6) is evaluated in the form

$$\psi^{(n)}(x) = (-1)^{n+1} \frac{\Gamma(n + 1)}{x^{n+1}} \sum_{k=0}^{\infty} \frac{1}{(1 + k/x)^{n+1}}.$$

For $x < 0$ the polygamma reflection formula [1, 6.4.7] is used

$$\psi^{(n)}(1 - x) + (-1)^{n+1} \psi^{(n)}(x) = (-1)^n \pi \frac{\mathrm{d}^n}{\mathrm{d}x^n} \cot(\pi x) \tag{3.7}$$

with pre-computed coefficients of the polynomials $p_n(\cot(\pi x))$ from the n^{th} derivative of $\cot(\pi x)$ if $n \le 12$ and for $n \ge 70$ with a series for the RHS of 3.7

$$(-1)^n \left(\frac{1}{x} + \sum_{k=1}^{\infty} \left(\frac{1}{x+k} + \frac{1}{x-k} \right) \right)^{(n)} = n! \left(\frac{1}{x^{n+1}} + \sum_{k=1}^{\infty} \left(\frac{1}{(x+k)^{n+1}} + \frac{1}{(x-k)^{n+1}} \right) \right)$$

derived from Euler's series for $\pi \cot(\pi x)$. For $12 < n < 70$ the polynomials are used with dynamically computed coefficients, if the presumed convergence of the series is slow.

3.5.6.6 Inverse digamma function $\psi^{-1}(y)$

```
function psi_inv(y: double): double;
function psi_invx(y: extended): extended;
```

These routines return the functional inverse $\psi^{-1}(y)$ of the digamma function ψ for $y \le \ln(\text{MaxExtended})$. The common function `sfc_ipsi` computes the result x using Newton's root finding method for $f(x) = \psi(x) - y$ with the trigamma function ψ'

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{\psi(x_n) - y}{\psi'(x_n)} .$$

The starting values x_0 are taken from three different cases.

3.5.6.7 Bateman function G(x)

```
function BatemanG(x: double): double;
function BatemanGx(x: extended): extended;
```

These functions return Bateman's $G(x)$ defined in Erdélyi et al. [50, 1.8(1)] by

$$G(x) = \psi\left(\frac{x}{2} + \frac{1}{2}\right) - \psi\left(\frac{x}{2}\right), \qquad x \ne 0, -1, -2, \ldots$$

The common function `sfc_batemang` handles small or large x with asymptotic expressions

$$G(x) = \frac{2}{x} - 2\ln 2 + \frac{\pi^2}{6}\, x + O(x^2), \qquad x \to 0$$
$$G(x) = \frac{1}{x} + \frac{1}{2x^2} - \frac{1}{4x^4} + O(x^{-6}), \qquad x \to +\infty$$

(computed with Maple) and negative x with the reflection formula [50, 1.8(8)]

$$G(1-x) = 2\pi \csc(\pi x) - G(x).$$

For $x \ge 20$ the simple hypergeometric sum with positive terms

$$G(x) = {}_2F_1\left(1, 1; 1+x; \frac{1}{2}\right) \frac{1}{x}$$

is evaluated, it is obtained from Erdélyi's equation [50, 1.8(6)]

$$G(x) = {}_2F_1\Big(1, x; 1+x; -1\Big) \frac{2}{x}$$

by a linear transformation, cf. [1, 15.3.4]. For arguments $0 < x < 20$ the recurrence formula from [50, 1.8(7)]

$$G(x+1) = \frac{2}{x} - G(x), \qquad G(x) = \frac{2}{x(x+1)} + G(x+2)$$

is used with double steps (i.e. the second equation) to make $x \ge 20$.

3.6 Zeta functions, polylogarithms, and related

The units **sfZeta** and **sfZeta2** implement 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 η/ζ function group. In this section let $\eta_\epsilon = 10^{-9}$, for $|s| \le \eta_\epsilon$ the eta functions are linear (accurate to extended precision).

```
function etam1pos(s: extended): extended;
```

This function returns the Dirichlet $\eta(s) - 1$ function for $s \ge -\eta_\epsilon$

$$\eta(s) - 1 = -\sum_{k=2}^{\infty} \frac{(-1)^k}{k^s}.$$

If $|s| \le \eta_\epsilon$ 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^{24} are used with P. Borwein's [36] convergence acceleration technique.[41]

```
function zetap(s,sc: extended): extended;
```

This function returns the Riemann zeta function for $s > 0, s \ne 1$. 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 functions

3.6.1.1 Riemann ζ(s) function

```
function zeta(s: double): double;
function zetax(s: extended): extended;
```

These functions calculate the Riemann zeta function $\zeta(s)$ for $s \ne 1$, defined by

$$\zeta(s) = \sum_{k=1}^{\infty} \frac{1}{k^s}, \quad s > 1,$$

and by analytic continuation for $s \ne 1$. When $s = 0$ the result is $-1/2$, and for $s > 0$ the common function `sfc_zeta` returns zetap$(s, 1 - s)$. If $s < 0$ the reflection formula [30, 25.4.2] is used:

$$\zeta(s) = 2(2\pi)^{s-1} \sin(\tfrac{1}{2}\pi s)\Gamma(1 - s)\zeta(1 - s)$$

3.6.1.2 Riemann ζ(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 $n \ne 1$. For $n > 63$ the result is 1, for $0 \le n \le 63$ the value is taken from table, otherwise the Bernoulli numbers are used: $\zeta(n) = B_{1-n}/(n - 1)$ for $n < 0$.

[41] The coefficients are from Algorithm 2, but re-calculated, scaled, and shifted with Maple; **DAMath** uses powers up to k^{21} for $s < 17$.

3.6.1.3 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.

3.6.1.4 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 separate routines because $\zeta(s) \to 1$ 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 (or its real part for $x < 1$)

$$P(x) = \sum_{p \text{ prime}} p^{-x}, \quad x > \tfrac{1}{5}.$$

In the implemented range P has singularities at $1/3, 1/2, 1$ and the imaginary part $\Im P(x)$ is $\frac{\pi}{6}$ for $\frac{1}{5} < x < \frac{1}{3}$, $\frac{\pi}{2}$ for $\frac{1}{3} < x < \frac{1}{2}$, and π for $\frac{1}{2} < x < 1$.[42]

For $x \geq 27$ the common function `sfc_pz` uses the first significant terms of the sum up to 7^{-x}, otherwise the calculation is based on

$$P(x) = \sum_{n=1}^{\infty} \frac{\mu(n)}{n} \ln(\zeta(nx)),$$

where μ is the Möbius function[43]. Actually $P(x)$ is computed with Cohen's speed-up[44]

$$P(x) = \sum_{p \leq A} p^{-x} + \sum_{n \geq 1} \frac{\mu(n)}{n} \ln(\zeta_{p>A}(nx))$$

$$\zeta_{p>A}(s) = \zeta(s) \prod_{p \leq A} (1 - p^{-s})$$

Since $\zeta(nx)$ rapidly approaches 1, the sum is evaluated with the accuracy improved functions ln1p and zetam1 (if $nx > 1$) in order to minimise rounding and truncation errors.

[42] $P(x)$ has an infinite number of singularities at $x = \frac{1}{n}$ where n is a square-free integer (i.e. $nx = 1$ and $\mu(n) \neq 0$); this is a reason why $P(x)$ cannot be continued to $x \leq 0$, and computationally an increasing number of terms in the series is needed (**AMath** uses up to 95 for $x \approx \frac{1}{5}$).

[43] $\mu(n) = 1$ 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.

[44] See e.g. http://www.math.u-bordeaux.fr/~cohen/hardylw.dvi, section 2.1; **AMath** implements the speed-up with $A = 7$.

3.6.3 Dirichlet eta functions

3.6.3.1 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[45] for η 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 $1 + \text{etam1pos}(s)$.

3.6.3.2 Dirichlet eta function for integer arguments

```
function etaint(n: integer): double;
function etaintx(n: integer): extended;
```

These functions return the Dirichlet eta function $\eta(n)$ for integer arguments. For $n > 64$ the result is 1, for $0 \leq n \leq 64$ the value is taken from table, otherwise for $n < 0$ the Bernoulli numbers are used: $\eta(n) = (2^{1-n} - 1)B_{1-n}/(1-n)$

3.6.3.3 Dirichlet $\eta(s) - 1$

```
function etam1(s: double): double;
function etam1x(s: extended): extended;
```

These functions return the Dirichlet function $\eta(s) - 1$, they are provided as separate routines because $\eta(s) \to 1$ 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 $\text{etam1pos}(s)$ 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

$$\beta(s) = 2^{-s}\Phi(-1, s, \tfrac{1}{2})$$

[45] It can be derived easily from the reflection formula for ζ.

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 Γ function will overflow and ∞ is returned, and if s is close to 0 then

$$\beta(s) \approx \frac{1}{2} + s\beta'(0), \qquad \beta'(0) = \ln \frac{\Gamma^2(1/4)}{2\pi\sqrt{2}} = 0.39159439270683677647\ldots$$

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

$$\lambda(s) = (1 - 2^{-s})\zeta(s) = -\text{exp2m1}(-s)\zeta(s).$$

3.6.6 Fermi-Dirac integrals

3.6.6.1 Fermi-Dirac integrals of integer order

```
function fermi_dirac(n: integer; x: double): double;
function fermi_diracx(n: integer; x: extended): extended;
```

These functions return the complete Fermi-Dirac integrals $F_n(x)$ of integer order. They are defined for real orders $s > -1$ by (see Olver et al. [30, 25.12.14])

$$F_s(x) = \frac{1}{\Gamma(s+1)} \int_0^{\infty} \frac{t^s}{e^{t-x}+1}\, \mathrm{d}t$$

and by analytic continuation for $s \le -1$ using polylogarithms, cf. [30, 25.12.16]:

$$F_s(x) = -\text{Li}_{s+1}(-e^x)$$

The common function `sfc_fermi_dirac` handles some special cases and basically uses `sfc_polylog` with the argument $-\exp(x)$ except in two non-trivial cases:

If $x >$ `ln_MaxExt`, i.e. $-\exp(x)$ would overflow, the integral $F_s(x)$ is computed for $0 < s \ll x$ with the asymptotic expansion from Goano [47, (9)][46]

$$F_s(x) \sim \cos(\pi s)\, F_s(-x) + \frac{x^{s+1}}{\Gamma(s+2)} \left(1 + 2s(s+2) \sum_{k=0}^{\infty} \frac{(1-s)_{2k}}{x^{2k+2}} \eta(2k+2)\right) \tag{3.8}$$

And if $-e^x < -1$, the code for the inversion formula (3.13) from `sfc_polylog` is repeated with an externally supplied third parameter to avoid the inaccuracies caused by computing $\ln e^x$.

[46] $\cos(\pi s)F_s(-x) = 0$ if s is a half-integer or (accurate to extended precision) if $x >$ `ln_MaxExt`.

3.6.6.2 Fermi-Dirac integral $\mathbf{F_{-1/2}(x)}$

```
function fermi_dirac_m05(x: double): double;
function fermi_dirac_m05x(x: extended): extended;
```

These functions return the complete Fermi-Dirac integral $F_{-1/2}(x)$. If $x \leq 0$ the common function returns $-$`sfc_polylogr`$(1/2, -\exp(x))$, in the range $0 < x \leq 40$ three Chebyshev expansion (computed with Maple V) are used, and for $x > 40$ the result is calculated with the asymptotic expansion 3.8.

3.6.6.3 Fermi-Dirac integral $\mathbf{F_{1/2}(x)}$

```
function fermi_dirac_p05(x: double): double;
function fermi_dirac_p05x(x: extended): extended;
```

These functions return the complete Fermi-Dirac integral $F_{1/2}(x)$. If $x \leq 0$ the common function returns $-$`sfc_polylogr`$(3/2, -\exp(x))$, in the range $0 < x \leq 36$ three Chebyshev expansion (computed with Maple V) are used, and for $x > 36$ the result is calculated with the asymptotic expansion 3.8.

3.6.6.4 Fermi-Dirac integral $\mathbf{F_{3/2}(x)}$

```
function fermi_dirac_p15(x: double): double;
function fermi_dirac_p15x(x: extended): extended;
```

These functions return the complete Fermi-Dirac integral $F_{3/2}(x)$. If $x \leq 0$ the common function returns $-$`sfc_polylogr`$(5/2, -\exp(x))$, in the range $0 < x \leq 36$ three Chebyshev expansion (computed with Maple V) are used, and for $x > 36$ the result is calculated with the asymptotic expansion 3.8.

3.6.6.5 Fermi-Dirac integral $\mathbf{F_{5/2}(x)}$

```
function fermi_dirac_p25(x: double): double;
function fermi_dirac_p25x(x: extended): extended;
```

These functions return the complete Fermi-Dirac integral $F_{5/2}(x)$. If $x \leq 0$ the common function returns $-$`sfc_polylogr`$(7/2, -\exp(x))$, in the range $0 < x \leq 36$ three Chebyshev expansion (computed with Maple V) are used, and for $x > 36$ the result is calculated with the asymptotic expansion 3.8.

3.6.7 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

$$\zeta(s,a) = \sum_{k=0}^{\infty} \frac{1}{(k+a)^s} \quad (s > 1,\ a \neq 0, -1, -2, \cdots),$$

and by continuation to $s \neq 1$. **Note**: The current **AMath** implementation restricts the arguments to $s \neq 1$ and $a > 0$. If $a = 1$ then $\zeta(s)$ is returned, and if $s = 0$ the result is $\frac{1}{2} - a$.

The common function `sfc_zetah`[47] uses Euler-Maclaurin summation[48] for $s > -8$ and s no negative integer:

$$\zeta(s,a) = \sum_{k=0}^{n} \frac{1}{(a+k)^s} + \frac{(a+n)^{1-s}}{s-1} + \sum_{k=0}^{\infty} \frac{B_{k+1}}{(k+1)!} \frac{(s+k-1)_k}{(a+n)^{s+k}}$$

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 $|s| < 1/2$ 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.

For negative integers $s = -1, -2, \ldots, -8$, the result is computed with the Bernoulli polynomials [30, 25.11.14]:

$$\zeta(-n,a) = -\frac{B_{n+1}(a)}{n+1}, \quad n = 1, 2, \ldots$$

where for $n = 1, \ldots, 9$ the simple explicit representation [30, 24.2.5] is adequate

$$B_n(x) = \sum_{k=0}^{n} \binom{n}{k} B_k x^{n-k}.$$

For negative s and a close to 1 the expansion [30, 25.11.10] is computed, but this may fail either because $\zeta(1)$ should be evaluated or the convergence is too slow. If $s < -8$ and $0 < a \le 1$ the Hurwitz formula from Erdély et al. [50, Vol.I, 1.10 (6)] is applied in the following form

$$\zeta(s,a) = \frac{2\Gamma(1-s)}{(2\pi)^{1-s}} \sum_{n=1}^{\infty} \frac{\sin\left(\left(2na + \frac{s}{2}\right)\pi\right)}{n^{1-s}}, \tag{3.9}$$

other arguments $a > 1$ are reduced to the range $0 < a \le 1$ with [30, 25.11.4]:

$$\zeta(s, a+m) = \zeta(s,a) - \sum_{n=0}^{m-1} \frac{1}{(n+a)^s}, \qquad m = 1, 2, 3, \ldots$$

3.6.8 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 $s \ge 0, |x| \le 1$ by[49]

$$\chi_s(x) = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)^s}.$$

The function can be expressed as

$$\chi_s(x) = 2^{-s} x\, \Phi(x^2, s, \tfrac{1}{2}) = \tfrac{1}{2}\left(\mathrm{Li}_s(x) - \mathrm{Li}_s(-x)\right)$$

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 Li_s relation is used (with elementary functions, see 3.10 and 3.11), otherwise the result is computed with Lerch's transcendent.

[47] Initially based on Cephes [7, zeta.c], newer versions are expanded and rewritten.
[48] See e.g. http://functions.wolfram.com/10.02.06.0020.01
[49] Additionally $x \ne 1$ if $s \le 1$.

3.6.9 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]

$$\Phi(z,s,a)=\sum_{n=0}^{\infty}\frac{z^n}{(a+n)^s},\quad a\neq 0,-1,-2,\ldots,|z|<1;\quad \Re s>1,|z|=1.$$

and by continuation for other z values. The current **AMath** implementation is restricted to real arguments $|z|\le 1,\ s\ge -1,\ a>0$.[50]

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[51].

The asymptotic expansion for large $a>0$ implements the formula given in **Theorem 1** by Ferreira and López [41], which can be specialised for the **AMath** context as (it is used only for $s>0$)

$$\Phi(z,s,a)=\frac{1}{1-z}\frac{1}{a^s}+\sum_{n=1}^{N-1}\frac{(-1)^n\mathrm{Li}_{-n}(z)}{n!}\frac{(s)_n}{a^{n+s}}+R_N(z,s,a),\quad N=1,2,3,\ldots$$

with the error term $R_N(z,s,a)=O(a^{-N-s})$ as $a\to\infty$.

For $z=1,s\neq 1$ 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[52].

3.6.10 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

$$\mathrm{Li}_n(x)=\sum_{k=1}^{\infty}\frac{x^k}{k^n},\quad n\in\mathbb{Z},\ |x|<1,$$

or it's analytic continuation[53]; for $n>0$ there is the argument restriction $x\le 1$. [54]

[50] For $s<-1$ there are severe convergence/cancellation problems.

[51] If $|z|\le 0.5$ the framework of lerchphi.c essentially performs a sophisticated direct summation, this part is skipped by the driver function in **AMath**.

[52] 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 $a>2/\mathrm{eps\_x}$!

[53] The function can also be defined for complex orders n.

[54] For $n=2,3$ the dilog and trilog functions drop this restriction and compute $\Re\,\mathrm{Li}_n(x)$ for $x>1$.

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:

$$\mathrm{Li}_0(x) = \frac{x}{1-x} \tag{3.10}$$

$$\mathrm{Li}_1(x) = -\ln(1-x) \tag{3.11}$$

$$\mathrm{Li}_2(x) = \mathrm{dilog}(x) \tag{3.12}$$

and for fixed x:

$$\mathrm{Li}_n(1) = \zeta(n)$$
$$\mathrm{Li}_n(-1) = -\eta(n)$$

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[55] is implemented:

$$\mathrm{Li}_n(-x) + (-1)^n \mathrm{Li}_n(-x^{-1}) = -\frac{\ln^n(x)}{n!} - 2\sum_{k=1}^{\lfloor n/2 \rfloor} \frac{\ln^{n-2k}(x)}{(n-2k)!}\,\eta(2k) \tag{3.13}$$

For $-1 < x \le -0.875$ the recursive formula [31, 1.2]

$$\mathrm{Li}_n(x) + \mathrm{Li}_n(-x) = 2^{1-n}\mathrm{Li}_n(x^2)$$

is used and for $0.5 \le x < 1$:

$$\mathrm{Li}_n(x) = \sum_{\substack{m=0 \\ m-n\neq 1}}^{\infty} \frac{\zeta(n-m)}{m!}\ln^m(x) + \frac{\ln^{n-1}(x)}{(n-1)!}\Big(H_{n-1} - \ln(-\ln x)\Big).$$

All the above formulas are valid only for $n > 0$. For negative n and $x \neq 1$ the polylogarithms are rational-polynomial functions:

$$\mathrm{Li}_{-n}(x) = \frac{1}{(1-x)^{n+1}} \sum_{m=1}^{n} \left(\sum_{k=1}^{m} (-1)^{k+1} \binom{n+1}{k-1} (m-k+1)^n \right) x^m \quad (n > 0)$$

In `sfc_polylog` the numerator polynomials for $-10 < n < 0$ are hard-coded; otherwise for $|x| \le 0.25$ the function is computed with the series definition. The cases $x > 4$ and $x < -11.5$ are transformed to the previous with

$$\mathrm{Li}_{-n}(x) = (-1)^{n-1}\mathrm{Li}_{-n}\left(x^{-1}\right), \quad (n > 0),$$

and the other arguments are handled with Crandall's relation [31, 1.5]:

$$\mathrm{Li}_n(x) = (-n)!(-\ln x)^{n-1} - \sum_{k=0}^{\infty} \frac{B_{k-n+1}}{k!(k-n+1)}\ln^k(x), \quad (n \le 0),$$

where in-line complex arithmetic is used if $x < 0$.

[55] See Cephes[7], Crandall [31, 1.3], or http://functions.wolfram.com/10.08.17.0060.01. I use $-\eta(2k)$ instead of $\mathrm{Li}_{2k}(-1)$.

3.6.11 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 $s \geq -1$

$$\mathrm{Li}_s(x) = \sum_{k=1}^{\infty} \frac{x^k}{k^s}, \quad s \geq -1,\ |x| \leq 1,$$

for $s \leq 1$ there is the additional argument restriction $x \neq 1$. The common function handles the special cases $\mathrm{Li}_s(0) = 0$, $\mathrm{Li}_s(1) = \zeta(s)$, $\mathrm{Li}_s(-1) = -\eta(s)$, 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 (cf. [30, 25.14.3]):

$$\mathrm{Li}_s(x) = x\,\Phi(x, s, 1)$$

3.6.12 Dilogarithm function

```
function dilog(x: double): double;
function dilogx(x: extended): extended;
```

These functions return the dilogarithm function

$$\mathrm{dilog}(x) = \Re\,\mathrm{Li}_2(x) = -\,\Re \int_0^x \frac{\ln(1-t)}{t}\,\mathrm{d}t.$$

For $x \leq 1$ the relation $\mathrm{dilog}(x) = \mathrm{polylog}(2, x)$ 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 $\mathrm{dilog}(x) = \mathrm{Li}_2(1-x)$ and then call $\mathrm{Li}_2(x)$ 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).

3.6.13 Trilogarithm function

```
function trilog(x: double): double;
function trilogx(x: extended): extended;
```

These functions return the trilogarithm function $\mathrm{trilog}(x) = \Re\,\mathrm{Li}_3(x)$. For $x \leq 1$ the relation $\mathrm{trilog}(x) = \mathrm{polylog}(3, x)$ is valid, but trilog is implemented even for $x > 1$.

For $x > 1$ the common function `sfc_trilog` uses the real part of the inversion formula from Crandall [31, 1.3] with $n = 3$:

$$\Re\,\mathrm{Li}_3(x) = \Re\,\mathrm{Li}_3(x^{-1}) + \frac{1}{3}\pi^2 \ln x - \frac{1}{6}(\ln x)^3,$$

and sfc_polylog(3, x) when $x \leq 1$, where one recursive call is handled in-line for $x < -1$ evaluating (3.13) with $n = 3$.

3.6.14 Clausen function $\mathrm{Cl}_2(x)$

```
function cl2(x: double): double;
function cl2x(x: extended): extended;
```

These functions return the Clausen function

$$\mathrm{Cl}_2(x) = \Im\mathrm{Li}_2(e^{ix}) = -\int_0^x \ln|2\sin(\frac{t}{2})|\,\mathrm{d}t.$$

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 $z = x \mod 2\pi$. The first for $z \in (-\pi/2, \pi/2)$ is based on [22, (3)] or [1, 27.8.2]:

$$\mathrm{Cl}_2(x) = -x\ln|x| + x - \sum_{k=1}^{\infty} \frac{(-1)^k B_{2k} x^{2k+1}}{2k(2k+1)(2k)!},$$

and the second for the function $\mathrm{Cl}_2(x+\pi)$ uses

$$\mathrm{Cl}_2(x) = \Im\mathrm{Li}_2(e^{ix}) = -\frac{i}{2}\Big(\mathrm{Li}_2(e^{ix}) - \mathrm{Li}_2(e^{-ix})\Big).$$

The calculations are done using Maple[56] and **MPArith**. 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_cl2` returns zero.

3.6.15 Inverse-tangent integral

```
function ti2(x: double): double;
function ti2x(x: extended): extended;
```

These functions return the inverse-tangent integral

$$\mathrm{Ti}_2(x) = \int_0^x \frac{\tan^{-1}t}{t}\,\mathrm{d}t.$$

MISCFUN [22] (function atnint) and GSL [21] (function specfunc/atanint.c) give Chebyshev approximations, but both are not suitable for extended precision.

For $0 \le x \le 1$ the integral is computed with a Chebyshev approximation calculated with Maple[57], for $x > 1$ the relation [35, 8.6]

$$\mathrm{Ti}_2(x) = \mathrm{Ti}_2\left(\frac{1}{x}\right) + \frac{\pi}{2}\ln x$$

is used, and for $x < 0$ the result is $\mathrm{Ti}_2(x) = -\mathrm{Ti}_2(-x)$.

[56] Note that Maple's polylog(2,x) must be used for $\mathrm{Li}_2(x)$.
[57] `chebyshev(int(arctan(t)/t, t=0..x)/x, x=-1..1, 0.5E-20);`

3.6.16 Lobachevsky's log-cos integral

```
function lobachevsky_c(x: double): double;
function lobachevsky_cx(x: extended): extended;
```

These functions return the Lobachevsky's log-cos integral $L(x)$

$$L(x) = -\int_0^x \ln|\cos(t)|\,\mathrm{d}t.$$

The common function `sfc_llci` is based on MacLeod's MISCFUN [22] routine LOBACH; it uses two Chebyshev approximations. Since $L(\pi) = \pi \ln 2 \neq 0$ the function is monotone increasing and quasi-periodic:

$$L(x + n\pi) = L(x) + n\pi \ln 2$$

That is why, contrary to MISCFUN, the **AMATH** implementation does not raise an error for large x (where the reduction mod π is unreliable) but returns $L(x) = x \ln 2$ for $|x| > 1/\texttt{eps\_x}$. And there is a fix to avoid taking logarithms of very small negative values after range reduction (e.g. for the extended value of $\pi/2$).

3.6.17 Lobachevsky's log-sin integral

```
function lobachevsky_s(x: double): double;
function lobachevsky_sx(x: extended): extended;
```

These functions return the Lobachevsky's log-sin integral $\Lambda(x)$

$$\Lambda(x) = -\int_0^x \ln|2\sin(t)|\,\mathrm{d}t,$$

where sometimes Λ is replaced by the Cyrillic capital letter El. The common function `sfc_llsi` uses the relation to the Clausen function[58] and simply computes

$$\Lambda(x) = \frac{1}{2}\mathrm{Cl}_2(2x).$$

3.6.18 Harmonic number function H_x

```
function harmonic(x: double): double;
function harmonicx(x: extended): extended;
```

These functions return the harmonic number function defined by

$$H_x = \psi(x+1) + \gamma, \qquad x \neq -1, -2, \ldots$$

The common function `sfc_harmonic` evaluates a Chebyshev expansion calculated with Maple for $|x| \leq \frac{1}{4}$ and for integer $n \leq 12$ the harmonic sum

$$H_n = \sum_{k=1}^{n} k^{-1}$$

is used to avoid the subtraction and addition of γ because ψ has the similar sum 3.5. Otherwise the result is computed with the given definition formula.

[58] See e.g. http://mathworld.wolfram.com/LobachevskysFunction.html

3.6.19 Generalized harmonic number function $\mathbf{H_x^{(r)}}$

```
function harmonic2(x,r: double): double;
function harmonic2(x,r: double): double;
```

These functions return the generalized harmonic number function defined by[59]

$$H_x^{(r)} = \zeta(r) - \zeta(r, x+1) \qquad r \neq 1,$$

and $H_x^{(1)} = H_x$ for $r = 1$. The common function `sfc_harmonic2` requires $x > -1$ and handles some special cases: x small integers, $r = 0, 1$. For negative integers $r = -n$ the Bernoulli polynomials and numbers are used[60]

$$H_x^{(-n)} = \frac{(-1)^n B_{n+1} + B_{n+1}(x+1)}{n+1}, \quad n \in \mathbb{N}.$$

Argument values $-1 < x \leq 20$ values are shifted with recursion formulas to $|x| \leq \frac{1}{2}$. Here and for all remaining cases the result is computed either with the definition formula or the series expansion[61]

$$H_x^{(r)} = \sum_{j=1}^{\infty} \frac{(-1)^{j-1}(r)_j \zeta(j+r) x^j}{j!}, \quad |x| < 1$$

[59] http://functions.wolfram.com/06.17.02.0001.01

[60] http://functions.wolfram.com/06.17.27.0005.01

[61] http://functions.wolfram.com/06.17.06.0002.01. Note that there is the restriction $\Re r > 1$ in this formula, but it is not present in another (06.17.06.0018.01), and numerical evidence shows that it converges for $r < 1$.

3.7 Orthogonal polynomials and Legendre functions

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. $\int_a^b w(x)p_n(x)p_m(x)\mathrm{d}x = 0$ 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.

The (orthogonal) Legendre polynomials are special cases of the general Legendre functions, **AMath** implements some integer order classes of these functions including spherical and toroidal harmonics.

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]:

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_{n+1}(x) &= 2x\,T_n(x) - T_{n-1}(x) \end{aligned}$$

If $n > 64$ the following trigonometric and hyperbolic identities [1, 22.3.15]:

$$\begin{aligned} T_n(x) &= \cos(n \arccos(x)) && |x| < 1 \\ T_n(x) &= \cosh(n \operatorname{arccosh}(x)) && |x| > 1 \end{aligned}$$

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 polynomials 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]:

$$\begin{aligned} U_0(x) &= 1 \\ U_1(x) &= 2x \\ U_{n+1}(x) &= 2x\,U_n(x) - U_{n-1}(x) \end{aligned}$$

If $n > 64$ the trigonometric and hyperbolic identities [1, 22.3.16]:

$$\begin{aligned} U_n(x) &= \frac{\sin((n+1)\arccos(x))}{\sin(\arccos(x))} && |x| < 1 \\ U_n(x) &= \frac{\sinh((n+1)\operatorname{arccosh}(x))}{\sinh(\operatorname{arccosh}(x))} && |x| > 1 \end{aligned}$$

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 Chebyshev polynomials of the third kind

```
function chebyshev_v(n: integer; x: double): double;
function chebyshev_vx(n: integer; x: extended): extended;
```

These functions return $V_n(x)$, the Chebyshev polynomial of the third kind of degree $n \geq 0$. The V_n are orthogonal on the interval $(-1, 1)$, with respect to the weight function $w(x) = (1+x)^{1/2}(1-x)^{-1/2}$.[62] Normally the common function `sfc_chebyshev_v` evaluates $V_n(x)$ with the standard recurrence formulas:

$$\begin{aligned} V_0(x) &= 1 \\ V_1(x) &= 2x - 1 \\ V_{n+1}(x) &= 2x\,V_n(x) - V_{n-1}(x) \end{aligned}$$

but if $n > 512$ or $n > 16$ and $||x| - 1| < 10^{-5}$ the trigonometric and hyperbolic forms

$$\begin{aligned} V_n(x) &= \frac{\cos((n+\frac{1}{2})\arccos(x))}{\cos(\frac{1}{2}\arccos(x))} && |x| < 1 \\ V_n(x) &= \frac{\cosh((n+\frac{1}{2})\operatorname{arccosh}(x))}{\cosh(\frac{1}{2}\operatorname{arccosh}(x))} && x > 1 \\ V_n(x) &= \frac{\sinh((n+\frac{1}{2})\operatorname{arccosh}(-x))}{\sinh(\frac{1}{2}\operatorname{arccosh}(-x))}(-1)^n = (-1)^n W_n(-x) && x < -1 \end{aligned}$$

are used, and the special cases $x = -1, 0, 1$ are handled separately.

3.7.4 Chebyshev polynomials of the fourth kind

```
function chebyshev_w(n: integer; x: double): double;
function chebyshev_wx(n: integer; x: extended): extended;
```

These functions return $W_n(x)$, the Chebyshev polynomial of the fourth kind of degree $n \geq 0$. The W_n are orthogonal on the interval $(-1, 1)$, with respect to the weight function $w(x) = (1-x)^{1/2}(1+x)^{-1/2}$. Normally the common function `sfc_chebyshev_w` evaluates $W_n(x)$ with the standard recurrence formulas:

$$\begin{aligned} W_0(x) &= 1 \\ W_1(x) &= 2x + 1 \\ W_{n+1}(x) &= 2x\,W_n(x) - W_{n-1}(x) \end{aligned}$$

but if $n > 512$ or $n > 16$ and $||x| - 1| < 10^{-5}$ the trigonometric and hyperbolic forms

$$\begin{aligned} W_n(x) &= \frac{\sin((n+\frac{1}{2})\arccos(x))}{\sin(\frac{1}{2}\arccos(x))} && |x| < 1 \\ W_n(x) &= \frac{\sinh((n+\frac{1}{2})\operatorname{arccosh}(x))}{\sinh(\frac{1}{2}\operatorname{arccosh}(x))} && x > 1 \\ W_n(x) &= \frac{\cosh((n+\frac{1}{2})\operatorname{arccosh}(-x))}{\cosh(\frac{1}{2}\operatorname{arccosh}(-x))}(-1)^n = (-1)^n V_n(-x) && x < -1 \end{aligned}$$

are used, and the special cases $x = -1, 0, 1$ are handled separately.

[62] This definition is used in almost all published sources, unfortunately the NIST handbook [30, 18.3] exchanges V_n and W_n. **AMath** uses the definitions introduced by W.Gautschi[58] and J.C.Mason[59].

3.7.5 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 parameter 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]:

$$\begin{aligned} C_0^{(a)}(x) &= 1 \\ C_1^{(a)}(x) &= 2ax \\ nC_n^{(a)}(x) &= 2(n + a - 1)xC_{n-1}^{(a)}(x) - (n + 2a - 2)C_{n-2}^{(a)}(x) \end{aligned}$$

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.

3.7.6 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]:

$$\begin{aligned} H_0(x) &= 1 \\ H_1(x) &= 2x \\ H_n(x) &= 2xH_{n-1}(x) - 2(n - 1)H_{n-2}(x). \end{aligned}$$

3.7.7 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 are the somewhat complicated recurrence relations from [30] (18.9.1) and (18.9.2):

$$\begin{aligned} P_{n+1}^{(a,b)} &= (A_n x + B_n) P_n^{(a,b)} - C_n P_{n-1}^{(a,b)} \\ A_n &= \frac{(2n+a+b+1)(2n+a+b+2)}{2(n+1)(n+a+b+1)} \\ B_n &= \frac{(a^2-b^2)(2n+a+b+1)}{2(n+1)(n+a+b+1)(2n+a+b)} \\ C_n &= \frac{(n+a)(n+b)(2n+a+b+2)}{(n+1)(n+a+b+1)(2n+a+b)} \end{aligned}$$

3.7.8 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

$$\int_0^1 R_n^m(r) R_{n'}^m(r) r \mathrm{d}r = \frac{1}{2(n+1)} \delta_{nn'}.$$

The common function `sfc_zernike_r` evaluates the R_n^m using special cases of the Jacobi polynomials[63]

$$R_n^m(r) = (-1)^{(n-m)/2}\, r^m\, P_{(n-m)/2}^{(m,0)}(1-2r^2).$$

3.7.9 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[64], see Abramowitz and Stegun[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:

$$\begin{aligned} L_0^{(a)}(x) &= 1 \\ L_1^{(a)}(x) &= -x + 1 + a \\ n L_n^{(a)}(x) &= (2n + a - 1 - x) L_{n-1}^{(a)}(x) - (n + a - 1) L_{n-2}^{(a)}(x) \end{aligned}$$

[63] See e.g. http://mathworld.wolfram.com/ZernikePolynomial.html, (6)
[64] A simple M function summation is coded in-line, `sfc_1f1` from unit **sfHyperG** is not used.

3.7.10 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` simply calls `sfc_laguerre(n,0,x)`.

3.7.11 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

$$L_n^m(x) = (-1)^m \frac{\mathrm{d}^m}{\mathrm{d}x^m} L_{n+m}(x).$$

The L_n^m are computed using the generalized Laguerre polynomials

$$L_n^m(x) = L_n^{(m)}(x).$$

3.7.12 Legendre polynomials / functions

```
function legendre_p(l: integer; x: double): double;
function legendre_px(l: integer; x: extended): extended;
```

These functions return $P_l(x)$, the Legendre functions of the first kind, also called Legendre polynomials if $l \geq 0$ and $|x| \leq 1$. The Legendre polynomials are orthogonal on the interval $(-1, 1)$ with $w(x) = 1$. If $l \geq 0$ the common function `sfc_legendre_p` uses the recurrence relation for varying degree from [1, 8.5.3]:

$$\begin{aligned} P_0(x) &= 1 \\ P_1(x) &= x \\ (l+1)P_{l+1}(x) &= (2l+1)xP_l(x) - lP_{l-1}(x), \end{aligned}$$

and for negative l the result is $P_l(x) = P_{-l-1}(x)$, see [1, 8.2.1].

3.7.13 Associated Legendre polynomials / functions

```
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[65] 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 for $m > 0, |x| < 1$ by the Rodrigues formula[66]

$$P_l^m(x) = (-1)^m (1-x^2)^{m/2} \frac{\mathrm{d}^m}{\mathrm{d}x^m} P_l(x).$$

[65] True polynomials only for even $|m|$.

[66] A similar definition for $x > 1$ is given in [30, 14.7.11]. For $x < -1$ the **AMath** function use the convention $P_l^m(x) = (-1)^l P_l^m(|x|)$

Note that the factor $(-1)^m$ is not always included in the literature, the **AMath** 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 [30, 14.9.3/5 and 14.9.11/13]:

$$\begin{aligned} P^m_{-l-1}(x) &= P_l{}^m(x), \\ P_l^{-m}(x) &= (-1)^m \frac{(l-m)!}{(l+m)!} P_l^m(x), \quad |x| \le 1, \end{aligned}$$

the factor $(-1)^m$ is omitted if $|x| > 1$. 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[67], which is based on the recurrence formula for varying degree [1, 8.5.3]

$$(l-m+1)P^m_{l+1}(x) = (2l+1)P_l{}^m(x) - (l+m)P^m_{l-1}(x)$$

and uses some tweaks from [19] (file legendre.hpp) and [13] Ch.6.8.

3.7.14 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 $l \ge 0$ and $|x| \ne 1$. For $|x| < 1$ the common function `sfc_legendre_q` uses the forward recurrence formulas[68]

$$\begin{aligned} Q_0(x) &= \frac{1}{2} \ln \frac{1+x}{1-x}, \\ Q_1(x) &= \frac{x}{2} \ln \frac{1+x}{1-x} - 1, \\ (k+1)Q_{k+1}(x) &= (2k+1)xQ_k(x) - kQ_{k-1}(x). \end{aligned}$$

For $|x| > 1$ the forward recurrence is not stable and the backward recursion is performed

$$\begin{aligned} Q_{k-1}(x) &= \Big((2k+1)xQ_k(x) - (k+1)Q_{k+1}(x)\Big)/k, \\ Q_1(x) &= \frac{x}{2} \ln \frac{x+1}{x-1} - 1, \\ Q_0(x) &= \frac{1}{2} \ln \frac{x+1}{x-1}, \end{aligned}$$

with the starting values $Q_{l+2} = t$, $Q_{l+1} = f(l,x)t$, where t is an arbitrary tiny value[69], and $f(l,x)$ is the continued fraction from [30, 14.14.3/4]

$$l\frac{Q_l(x)}{Q_{l-1}(x)} = \frac{x_0}{y_0 -} \frac{x_1}{y_1 -} \frac{x_2}{y_2 -} \cdots \quad \text{with} \quad x_k = (l+k)^2,\ y_k = (2l+2k+1)x.$$

The result of the recurrence is normalised with the true value $Q_0(x)$.[70] For x values close to 1, the continued fraction may not converge fast enough, in this case if $1 < x \le 1.05$ the hypergeometric representation [45, 7.3.7] of $Q_l(x)$ is used with the linear transformation from [1, 15.3.10].

[67] Also called by the spherical harmonic functions
[68] See Abramowitz and Stegun[1] (8.4.2) and (8.4.4) for the cases $l = 0, 1$
[69] Actually the square root of the smallest positive extended value
[70] Or $Q_1(x)$, dependent on the ratio of the last two values

3.7.15 Associated Legendre functions of the second kind

```
function legendre_qlm(l,m: integer; x: double): double;
function legendre_qlmx(l,m: integer; x: extended): extended;
```

These functions return $Q_l^m(x)$, the associated Legendre functions of the second kind with $l \geq 0$, $l + m \geq 0$ and $|x| \neq 1$. Like the $P_l^m(x)$ they can be defined by the Rodrigues formulas

$$Q_l^m(x) = (-1)^m (1-x^2)^{m/2} \frac{\mathrm{d}^m}{\mathrm{d}x^m} Q_l(x), \qquad |x| < 1,$$
$$Q_l^m(x) = \qquad (x^2-1)^{m/2} \frac{\mathrm{d}^m}{\mathrm{d}x^m} Q_l(x), \qquad |x| > 1.$$

Negative m are mapped to positive values with [30, 14.9.4]:

$$Q_l^{-m}(x) = (-1)^m \frac{(l-m)!}{(l+m)!} Q_l^m(x), \quad |x| < 1.$$

The factor $(-1)^m$ is omitted if $|x| > 1$, see [30, 14.9.14] and [1, 8.2.6]. If $m = 0$ the result is $Q_l(x)$. For $m > 0$ the recurrence relations [30, 14.10.1 and 14.10.6]

$$Q_l^{m+2}(x) + 2(m+1)x(1-x^2)^{-1/2} Q_l^{m+1}(x) + (l-m)(l+m+1) Q_l^m(x) = 0 \quad |x| < 1,$$
$$Q_l^{m+2}(x) + 2(m+1)x(x^2-1)^{-1/2} Q_l^{m+1}(x) - (l-m)(l+m+1) Q_l^m(x) = 0 \quad |x| > 1,$$

are used in the forward direction, where the $Q_l^1(x)$ values are computed from the $Q_l(x)$ with [30, 14.10.2 and 14.10.7].

3.7.16 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 π in θ, 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 θ, ϕ, 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):

$$\mathrm{yr} = \Re\, Y_{lm}(\theta, \phi) = A\cos(m\phi),$$
$$\mathrm{yi} = \Im\, Y_{lm}(\theta, \phi) = A\sin(m\phi).$$

3.7.17 Some toroidal harmonic functions

```
function toroidal_qlm(l, m: integer; x: double): double;
function toroidal_qlmx(l, m: integer; x: extended): extended;
```

These functions return the toroidal harmonics $Q^m_{l-1/2}(x)$ with $l = 0, 1$; $x > 1$. Negative orders m are mapped to positive values with [1, 8.2.6]. The common function `sfc_thq` uses the forward recurrence (stable if $x > 1$) for the general Legendre functions $Q^\mu_\nu(x)$ from Olver et al. [30, 14.10.6]

$$Q^{m+2}_\nu(x) + \frac{2(m+1)x}{(x^2-1)^{1/2}}\, Q^{m+1}_\nu(x) - (\nu - m)(\nu + m + 1)\, Q^m_\nu(x) = 0$$

for $\nu = \pm 1/2$. The basic functions $Q^{0,1}_{\pm 1/2}$ can be expressed with the complete elliptic integrals of the first and second kind, but only

$$Q^0_{-1/2}(x) = \sqrt{\frac{2}{x+1}}\, K\left(\sqrt{\frac{2}{x+1}}\right),$$
$$Q^1_{-1/2}(x) = \frac{-1}{\sqrt{2(x-1)}}\, E\left(\sqrt{\frac{2}{x+1}}\right)$$

can be evaluated accurately as given. $Q^0_{1/2}$ and $Q^1_{1/2}$ are combinations of E and K and the textbook versions

$$Q^0_{1/2}(x) = x\sqrt{\frac{2}{x+1}}\, K\left(\sqrt{\frac{2}{x+1}}\right) - \sqrt{2(x-1)}\, E\left(\sqrt{\frac{2}{x+1}}\right),$$
$$Q^1_{1/2}(x) = \frac{-x}{\sqrt{2(x-1)}}\, E\left(\sqrt{\frac{2}{x+1}}\right) + \sqrt{\frac{x-1}{2}}\, K\left(\sqrt{\frac{2}{x+1}}\right),$$

suffer from cancellations. The **AMath** functions use algebraic pre-computations and Bulirsch's general complete elliptic integral cel (3.2.3.3) for $x < 8$:

$$Q^0_{1/2}(x) = \sqrt{\frac{2}{x+1}}\, \mathrm{cel}\left(\sqrt{\frac{x-1}{x+1}}, 1, -1, 1\right),$$
$$Q^1_{1/2}(x) = \mathrm{cel}\left(\sqrt{\frac{x-1}{x+1}}, 1, \frac{-1}{\sqrt{2(x-1)}}, \frac{\sqrt{(x-1)/2}}{x+1}\right),$$

and truncated hypergeometric series for $x \geq 8$ from [1, 8.1.3]:

$$Q^m_\nu(x) = (-1)^m \frac{\pi^{\frac{1}{2}}\Gamma(\nu+m+1)(x^2-1)^{\frac{m}{2}}}{2^{\nu+1}\Gamma(\nu+\frac{3}{2})x^{\nu+m+1}}\, {}_2F_1\left(\frac{\nu+m+2}{2}, \frac{\nu+m+1}{2}; \nu+\tfrac{3}{2}; \frac{1}{x^2}\right)$$

```
function toroidal_plm(l, m: integer; x: double): double;
function toroidal_plmx(l, m: integer; x: extended): extended;
```

These functions return the toroidal harmonics $P^m_{l-1/2}(x)$ with $l, m = 0, 1$; and $x \geq 1$. From the formulas with elliptic integrals given in the literature only [1, 8.13.5]

$$P^0_{1/2}(x) = \frac{2}{\pi}\left(x + \sqrt{x^2-1}\right)^{\frac{1}{2}} E\left(\sqrt{\frac{2(x^2-1)^{1/2}}{x + (x^2-1)^{1/2}}}\right),$$

is directly usable, the others are transformed to expressions with Bulirsch style functions, e.g. the formula [1, 8.13.1]:

$$P^0_{-1/2}(x) = \frac{2}{\pi}\sqrt{\frac{2}{x+1}}\,K\left(\sqrt{\frac{x-1}{x+1}}\right),$$
$$= \frac{2}{\pi}\sqrt{\frac{2}{x+1}}\,\mathrm{cel1}\left(\sqrt{\frac{2}{x+1}}\right).$$

Alternate forms for $P^1_{\pm 1/2}(x)$ with elliptic integrals can be obtained from Wolfram Alpha[33][71], but the results are too complicated and not suitable for numerical computations. Therefore **AMath** uses the equivalent Bulirsch style expressions:

$$P^1_{1/2}(x) = \frac{-1}{\pi}\sqrt{\frac{x+1}{x-1}}\,\mathrm{cel}\left(k_c, 1, 1-\frac{x}{k_c^2}, 1-x\right),$$
$$P^1_{-1/2}(x) = \frac{-1}{\pi}\sqrt{\frac{x+1}{x-1}}\,\mathrm{cel2}\left(k_c, 1-\frac{1}{k_c^2}, 0\right),$$

where $k_c = \sqrt{(x+1)/2}$ is the complementary modulus. For large $x \geq 2\times 10^{19}$ the following approximations are implemented:

$$P^0_{1/2}(x) \sim \frac{4}{\pi}\sqrt{x/2},$$
$$P^1_{1/2}(x) \sim \frac{2}{\pi}\sqrt{x/2},$$
$$P^1_{-1/2}(x) \sim \frac{2\gamma + \ln(2) + \psi(1/2) + \psi(-1/2) - \ln x}{2\pi\sqrt{x/2}}$$

[71] By entering e.g. LegendreP(-1/2,1,3,x) at the input prompt.

3.8 Hypergeometric functions and related

This section the describes the **AMath** implementations of the Gauss hypergeometric function and the confluent hypergeometric functions. Many special functions (Legendre, modified Bessel, incomplete gamma, ...) can be obtained by selecting certain parameter and argument values, see e.g. the tables in Abramowitz and Stegun [1, 13.6]. Obviously it is very difficult to implement such general functions over a large (let alone complete) parameter range, and the **AMath** implementations are far from being complete.[72]

The Pascal unit **sfHyperG** contains the common code for the hypergeometric functions. There was a long development time before the first release due to the complicated structure/combination of the different algorithms and methods, see e.g. Pearson's dissertation [43], Muller's review [44], or Forrey's work [54]. The release was forced by the need of using ${}_2F_1$ for the non-normalised incomplete beta function $B_x(a,b)$ with $a \le 0$ or $b \le 0$.

3.8.1 Gauss hypergeometric function F(a,b;c;x)

```
function hyperg_2F1(a,b,c,x: double): double;
function hyperg_2F1x(a,b,c,x: extended): extended;
```

These functions return the Gauss hypergeometric function $F(a,b;c;x)$, defined for $|x| < 1$ by the series (the first two equations show other common notations)

$$F(a,b;c;x) = {}_2F_1(a,b;c;x) = F\left(\begin{matrix}a,b\\c\end{matrix};x\right) = \sum_{k=0}^{\infty}\frac{(a)_k(b)_k}{(c)_k k!}x^k$$
$$= 1 + \frac{ab}{c}x + \frac{a(a+1)b(b+1)}{c(c+1)2!}x^2 + \cdots,$$

and by analytic continuation to other x. Except for special cases it is required that $-c \notin \mathbb{N}$. For $x > 1$ the function is generally complex and not implemented in **AMath**; but if a or b is a non-positive integer, then $F(a,b;c;x)$ becomes a polynomial in x and there is no restriction on x. Special values are

$$F(0,b;c;x) = F(a,0;c;x) = F(a,b;c;0) = 1$$

and if $c-a-b > 0$

$$F(a,b;c;1) = \frac{\Gamma(c)\Gamma(c-a-b)}{\Gamma(c-a)\Gamma(c-b)}.$$

In **AMath** the analytic continuation is done using one or two linear transformations: For all $x < 1$ there are (see [1, 15.3.3-5]):

$$F(a,b;c;x) = (1-x)^{c-a-b}F(c-a,c-b;c;x)$$
$$= (1-x)^{-a}F\left(a,c-b;c;\frac{x}{x-1}\right)$$
$$= (1-x)^{-b}F\left(c-a,b;c;\frac{x}{x-1}\right),$$

and for $0 < x < 1$ if c and $c-a-b$ are no integers [1, 15.3.6]

$$F(a,b;c;x) = \frac{\Gamma(c)\Gamma(c-a-b)}{\Gamma(c-a)\Gamma(c-b)}F(a,b;a+b-c+1;1-x)$$
$$+ (1-x)^{c-a-b}\frac{\Gamma(c)\Gamma(a+b-c)}{\Gamma(c-a)\Gamma(c-b)}F(c-a,c-b;c-a-b+1;1-x).$$

[72] The implementation of the Tricomi function $U(a,b,x)$ is still in beta state and the Whittaker functions are coded just from the definitions without special overflow/underflow considerations. The same applies to the parabolic cylinder and Hermite functions, which additionally suffer from some cancellation for negative x.

For $c = a + b \pm m, (m = 0, 1, \ldots)$ the (complicated) formulas from Abramowitz and Stegun [1, 15.3.10-12] are implemented. If $a = -m$ is a negative integer[73] the limiting cases for the (polynomial) transformations are [30, 15.8.6/7]:

$$\begin{aligned} F\left(\begin{matrix}-m, b\\ c\end{matrix}; x\right) &= \frac{(b)_m}{(c)_m}(-x)^m F\left(\begin{matrix}-m, 1-c-m\\ 1-b-m\end{matrix}; \frac{1}{x}\right) && x \le -2 \\ &= \frac{(b)_m}{(c)_m}(1-x)^m F\left(\begin{matrix}-m, c-b\\ 1-b-m\end{matrix}; \frac{1}{1-x}\right) && -2 < x \le -1/2 \\ &= \frac{(c-b)_m}{(c)_m} F\left(\begin{matrix}-m, b\\ b-c-m+1\end{matrix}; 1-x\right) && 1/2 \le x < 1 \\ &= \frac{(c-b)_m}{(c)_m} x^m F\left(\begin{matrix}-m, 1-c-m\\ b-c-m+1\end{matrix}; 1-\frac{1}{x}\right) && x \ge 1 \end{aligned}$$

The common function[74] `sfc_2f1` applies these transformations and computes the series by a compensated summation method (using in-line **AMath** TwoSum) with error estimate. If the estimated errors are too large, a secondary method or transformation is used if available. For $x < 0$ an additional secondary method is Luke's rational approximation for $F(a, b; c; -x)$ from [42, Ch.13]; this is a powerful but somewhat obscure method which assumes $c \ne a$ and $c \ne b$, error estimation is not reliable and it is used only if the other methods failed.

3.8.2 Regularized Gauss hypergeometric function

```
function hyperg_2F1r(a,b,c,x: double): double;
function hyperg_2F1rx(a,b,c,x: extended): extended;
```

These functions return the regularized Gauss hypergeometric function $\mathbf{F}(a, b; c; x)$ for unrestricted c, defined by [30, 15.1.2]

$$\mathbf{F}(a, b; c; x) = \frac{1}{\Gamma(c)} F(a, b; c; x), \qquad (c \ne 0, -1, -2, \ldots)$$

and by the corresponding limit if $c = 0, -1, -2, \ldots = -m$, with the value [1, 15.1.2]

$$\mathbf{F}(a, b; -m; x) = \frac{(a)_{m+1}(b)_{m+1}}{(m+1)!}\, x^{m+1}\, F(a+m+1, a+m+1; m+2; x)$$

If c is a non-positive integer, the common function `sfc_2f1r` computes the four prefix terms in a single combined loop (if the Pochhammer symbols are non-zero), otherwise the result is evaluated from the definition.

3.8.3 Kummer's confluent hypergeometric function M(a,b,x)

```
function hyperg_1F1(a,b,x: double): double;
function hyperg_1F1x(a,b,x: extended): extended;
```

These functions return Kummer's confluent hypergeometric function $M(a, b, x)$, defined by the series (the first two equations show other common notations)

$$\begin{aligned} M(a, b, x) &= {}_1F_1(a, b, x) = F\left(\begin{matrix}a\\ b\end{matrix}; x\right) = \sum_{k=0}^{\infty} \frac{(a)_k}{(b)_k} \frac{x^k}{k!} \\ &= 1 + \frac{a}{b}x + \frac{a(a+1)}{b(b+1)} \frac{x^2}{2!} + \cdots, \qquad b \ne 0, -1, -2, \ldots \end{aligned}$$

[73] or b with a and b swapped, or both and $a \ge b$

[74] Originally based on the Cephes[7] functions in hyp2f1.c

The Kummer function $M(a,b,x)$ and Tricomi's function $U(a,b,x)$ are linear independent solutions of the confluent hypergeometric differential equation

$$xy'' + (b-x)y' - ay = 0.$$

If $a = -n$ is a negative integer, M is a generalized Laguerre polynomial, see [1, 13.6.9]:

$$M(-n,b,x) = \frac{n!}{(b)_n} L_n^{(b-1)}(x)$$

The common function `sfc_1f1`[75] handles this and the two other important special cases $b = a$ or $b = 2a$, which are e^x and related to the modified Bessel function I_ν. A central part is the function `h1f1_sum` that computes the defining series by a compensated summation method (using in-line **AMath** TwoSum) with error estimate. For large $x \gg 1$ the asymptotic expansion from [1, 13.5] or [44, 2.16]

$$M(a,b,x) \sim e^x x^{a-b} \frac{\Gamma(b)}{\Gamma(a)} \sum_{k=0}^{\infty} \frac{(b-a)_k (1-a)_k}{k!} x^{-k}$$

is used, either as a sum or as a continued fraction (see Muller [44, Method 2.C]). If x and a have different signs and b is no negative integer, there is Tricomi's Bessel function expansion see Tricomi [53, 1.6(9)], Abramowitz and Stegun [1, 13.3.7], or Pearson [43, 3.20/21].

And finally, if all other methods were unsuccessful, another Luke rational approximation ([42, Ch. 15]) is available for $x < 0$, cf. [44, Method 3].

3.8.4 Regularized Kummer function

```
function hyperg_1F1r(a,b,x: double): double;
function hyperg_1F1rx(a,b,x: extended): extended;
```

These functions return the regularized Kummer function $\mathbf{M}(a,b,x)$ for unrestricted b, defined by [30, 13.2.3]

$$\mathbf{M}(a,b,x) = \frac{1}{\Gamma(b)} M(a,b,x), \qquad (b \neq 0, -1, -2, \ldots)$$

and by the corresponding limit if $b = -n = 0, -1, -2, \ldots$ with the value [30, 13.2.5]:

$$\mathbf{M}(a,-n,x) = \frac{(a)_{n+1}}{(n+1)!}\, x^{n+1}\, M(a+n+1, n+2, x)$$

If b is a non-positive integer, the common function `sfc_1f1r` computes the three prefix terms in a single combined loop (if the Pochhammer symbol is non-zero), otherwise the result is evaluated from the definition.

3.8.5 Tricomi's confluent hypergeometric function U(a,b,x)

```
function hyperg_u(a,b,x: double): double;
function hyperg_ux(a,b,x: extended): extended;
```

These functions return Tricomi's confluent hypergeometric function $U(a,b,x)$[76] given for $x > 0$ and $b \neq 0, \pm 1, \pm 2, \ldots$ by Lebedev [45, 9.10.3]:

$$U(a,b,x) = \frac{\Gamma(1-b)}{\Gamma(1+a-b)}\, M(a,b,x) + \frac{\Gamma(b-1)}{\Gamma(a)}\, x^{1-b}\, M(1+a-b, 2-b, x).$$

[75] Originally based on the Cephes[7] functions in hyperg.c

[76] Also called *confluent hypergeometric function of the second kind.*

If $b = n + 1,\ n = 0, 1, 2, \ldots$ is an integer and $a \neq 0, -1, -2, \ldots$, Lebedev's calculations lead to a complicated series expansion [45, 9.10.6] including $\ln x$ and ψ[77], and for negative integers $b = 1 - n, n = 1, 2, 3, \ldots$ there is [45, 9.10.9]:

$$U(a, 1 - n, x) = x^n U(a + n, n + 1, x).$$

The current common function `sfc_chu` is based on Temme's paper [55] with the ALGOL60 procedures `chu` and `uabx` and on Pascal translations of Fullerton's Fortran functions `dchu` and `d9chu` from [20, 14], together with a few special cases:

When $a = -m,\ m = 0, 1, 2, \ldots$ Tricomi [53, 2.4(6)] obtains the relations

$$U(-m, b, x) = (-1)^m (b)_m M(-m, b, x) = (-1)^m m! L_m^{(b-1)}(x). \tag{3.14}$$

The first equation can be used[78] to compute $U(a, b, x)$ for all non-positive integers a, the second gives the special cases $U(0, b, x) = 1$ and $U(-1, b, x) = x - b$, cf. 3.7.9. Finally there are two relations from Olver et al. [30, 13.2.8 and 13.6.10]:

$$U(a, a + n + 1, x) = x^{-a} \sum_{k=0}^{n} \binom{n}{k} (a)_k x^{-k}, \quad n = 0, 1, 2, \ldots \tag{3.15}$$

$$U(a, 2a, x) = \frac{x^{\frac{1}{2}-a}}{\sqrt{\pi}} e^{\frac{x}{2}} K_{a-\frac{1}{2}}\left(\frac{x}{2}\right)$$

If no special case is applicable, the Luke rational approximation from `d9chu` is attempted if $x > 10/9 \max(|a|, 1) \max(|1 + a - b|, 1)$.

Temme's [55] procedures (`chu` assumes $a \geq 0$) are based on recurrence formulas for

$$u_k = (a)_k U(a + k, b, x), \qquad v_k = (a)_k U'(a + k, b, x), \qquad k = 0, 1, 2, \ldots,$$

together with a Miller algorithm or an asymptotic expansion with modified Bessel functions K_ν for small x. In **AMath** they are implemented in a simplified form[79] and used for general parameters and arguments, except in two additional cases:

For non-integer values $a < 0$ a double recursion scheme on descending a and ascending b values is implemented with recurrence relations from Abramowitz and Stegun [1, 13.4] and starting values for $U(a_1, b_1, x)$ and $U'(a_1, b_1, x)$ computed with the Temme algorithm for $a_1 = a - \lfloor a \rfloor$ and $b_1 = (b - \lfloor b \rfloor) + 1$.

Temme's `chu` underflows for a values of order `MAXGAM` (the Gamma function overflow threshold) or larger, here the Pascal translation of the SLATEC function `dchu` is used.

If $x < 0$ then for real U values a must be an integer and additionally $a < 0$ or $1 + a - b$ a negative integer; in these cases $U(a, b, x)$ is computed with 3.14 or 3.15.

3.8.6 Confluent hypergeometric limit function ${}_0\mathbf{F}_1(\mathbf{b}, \mathbf{x})$

```
function hyperg_0F1(b,x: double): double;
function hyperg_0F1x(b,x: extended): extended;
```

These functions return the confluent hypergeometric limit function ${}_0F_1(b, x)$, defined for $b \neq 0, -1, -2, -3, \ldots$ by the limit and series:

$${}_0F_1(b, x) = {}_0F_1(-; b; x) = \lim_{a \to \infty} {}_1F_1\left(a; b; \frac{x}{a}\right) = \sum_{k=0}^{\infty} \frac{x^k}{(b)_k k!}$$

[77] It was used in earlier version and is superseded by the Temme code

[78] This was done in the first alpha release of `sfc_chu`, the current implementation uses the Temme code for negative integers a, which empirically gives good results although there are no strict stability conditions if $a < 0$.

[79] The original allows computing an array of functions values $u_k, k > 0$

The common function `sfc_0f1` just calls the general routine `gen_0f1`[80], which handles the special case ${}_0F_1(b,0)=1$ and uses relations to Bessel functions[81] ,[82] for the regular calculation:

$$
\begin{aligned}
{}_0F_1(b,x) &= \Gamma(b)\,(+x)^{\frac{1-b}{2}}\,I_{b-1}\left(2\sqrt{+x}\right), \qquad x>0,\\
{}_0F_1(b,x) &= \Gamma(b)\,(-x)^{\frac{1-b}{2}}\,J_{b-1}\left(2\sqrt{-x}\right), \qquad x<0.
\end{aligned}
$$

A continued fraction[83] from the Wolfram function site [33] is evaluated with the modified Lentz method for $|x|\le 0.125$ or if the regular computation fails due to overflow or underflow of the separate parts:

$$
{}_0F_1(b,x) = 1+\cfrac{x/b}{1+\cfrac{-\frac{x}{2(1+b)}}{1+\frac{x}{2(1+b)}+\cfrac{-\frac{x}{3(2+b)}}{1+\frac{x}{3(2+b)}+\cfrac{-\frac{x}{4(3+b)}}{1+\frac{x}{4(3+b)}+\cfrac{-\frac{x}{5(4+b)}}{1+\frac{x}{5(4+b)}+\cdots}}}}}
$$

3.8.7 Regularized confluent hypergeometric limit function

```
function hyperg_0F1r(b,x: double): double;
function hyperg_0F1rx(b,x: extended): extended;
```

These functions return the regularized confluent hypergeometric limit function ${}_0\tilde{F}_1$ defined by

$$
{}_0\tilde{F}_1(b,x) = \frac{1}{\Gamma(b)}\,{}_0F_1(b,x)
$$

and by the corresponding limit if $b=-n=0,-1,-2,\ldots$ with the value[84]

$$
{}_0\tilde{F}_1(-n,x) = x^{n+1}\,{}_0\tilde{F}_1(n+2,x), \qquad n\in\mathbb{N}.
$$

The common function `sfc_0f1r` handles the case when b is a negative integer, otherwise the function `gen_0f1` is called with `reg=true`.

3.8.8 Whittaker M function

```
function WhittakerM(k,m,x: double): double;
function WhittakerMx(k,m,x: extended): extended;
```

These functions return the Whittaker function $M_{\kappa,\mu}(x)$, defined by [30, 13.14.2]

$$
M_{\kappa,\mu}(x) = e^{-\frac{1}{2}x}\,x^{\frac{1}{2}+\mu}\,M\left(\tfrac{1}{2}+\mu-\kappa,1+2\mu,x\right)
$$

with $2\mu\ne -1,-2,\ldots$ (normally, but it is defined if the Kummer M function is a polynomial e.g. $M_{1/2,-3}(x)$). If $x<0$ then $\mu+1/2$ must be an integer.

The **AMath** functions use k,m instead of κ,μ and the common function `sfc_whitm` just evaluates the definition with the Kummer M function. Note that, as a consequence, there are valid parameter/arguments where this approach fails due to intermediate overflow of the Kummer function. (This applies especially to the **DAMath** function `sfd_whitm` without the internal intermediate extended range, for example: `WhittakerM`$(2,-1/4,1000)\approx 9.062112\times 10^{210}$.)

[80] Also called by the regularized version, selected by the boolean parameter `reg`.
[81] http://functions.wolfram.com/07.17.27.0003.01
[82] http://functions.wolfram.com/07.17.27.0002.01
[83] http://functions.wolfram.com/07.17.10.0001.01
[84] http://functions.wolfram.com/07.18.17.0006.01

3.8.9 Whittaker W function

```
function WhittakerW(k,m,x: double): double;
function WhittakerWx(k,m,x: extended): extended;
```

These functions return the Whittaker function $W_{\kappa,\mu}(x)$, defined by [30, 13.14.3]

$$W_{\kappa,\mu}(x) = e^{-\frac{1}{2}x} x^{\frac{1}{2}+\mu}\, U\left(\tfrac{1}{2}+\mu-\kappa, 1+2\mu, x\right)$$

The common function `sfc_whitw` requires $x \geq 0$ and computes the result just from the definition with the Tricomi U function (there are similar notes and range issues as for the $M_{\kappa,\mu}$ function).

3.8.10 Parabolic cylinder functions

For large values of the parameters ν, a or x the parabolic cylinder functions become very large or very small. The **AMath** functions are more or less direct applications of the confluent hypergeometric functions without scaling or asymptotic expansions, and they are therefore usable for moderate ν, a, x values only, and $V(a,x)$ is even more restricted to half-integers $2a \in \mathbb{Z}$.[85]

3.8.10.1 Parabolic cylinder function $\mathbf{D}_\nu(\mathbf{x})$

```
function CylinderD(v, x: double): double;
function CylinderDx(v, x: extended): extended;
```

These functions compute Whittaker's parabolic cylinder function $D_\nu(x)$ (note that ν is replaced by `v` in the Pascal code), defined for $x \geq 0$ in Tricomi [53, 4.8(1)] as

$$D_\nu(x) = 2^{\frac{\nu}{2}}\, e^{-\frac{x^2}{4}}\, U\left(-\frac{\nu}{2}, \frac{1}{2}, \frac{x^2}{2}\right).$$

If $x < 0$ and ν is no positive integer the common function `sfc_pcfd` uses [86]

$$D_\nu(-x) = D_\nu(x) - \frac{2^{\frac{\nu+1}{2}}\nu\,\Gamma\left(\frac{\nu}{2}\right)\sin\left(\frac{\pi\nu}{2}\right)}{\sqrt{\pi}}\, x e^{-\frac{x^2}{4}}\, {}_1F_1\left(\frac{1-\nu}{2}, \frac{3}{2}, \frac{x^2}{2}\right),$$

and for $\nu = n \in \mathbb{N}$ the relation $D_n(-x) = (-1)^n D_n(x)$ is applied.[87]

3.8.10.2 Parabolic cylinder function $\mathbf{U}(\mathbf{a},\mathbf{x})$

```
function CylinderU(a, x: double): double;
function CylinderUx(a, x: extended): extended;
```

These functions compute the parabolic cylinder function $U(a,x)$, which is a standard function in some newer texts. It is defined with the Whittaker function, see Abramowitz and Stegun [1, 19.3.7] or Olver et al. [30, 12.1]

$$U(a,x) = D_{-a-\frac{1}{2}}(x)$$

The common function `sfc_pcfu` is just a wrapper around `sfc_pcfd`.

[85] For unrestricted and scaled implementations of U and V see e.g. A. Gil, J. Segura, N.M. Temme, Computing the real parabolic cylinder functions U(a,x), V(a,x); Modelling, Analysis and Simulation, E0612, 2006, technical report http://oai.cwi.nl/oai/asset/14654/14654D.pdf

[86] http://functions.wolfram.com/07.41.16.0006.01

[87] http://functions.wolfram.com/07.41.16.0007.01

3.8.10.3 Parabolic cylinder function V(a, x)

```
function CylinderV(a, x: double): double;
function CylinderVx(a, x: extended): extended;
```

These functions compute the parabolic cylinder function $V(a,x)$, which is the second standard function in some newer texts, defined in Abramowitz and Stegun [1, 19.3.8] as

$$V(a,x) = \frac{1}{\pi}\Gamma\left(\tfrac{1}{2}+a\right)\left(\sin\pi a \cdot D_{-a-\frac{1}{2}}(x) + D_{-a-\frac{1}{2}}(-x)\right). \tag{3.16}$$

The current **AMath** implementation is restricted to $2a \in \mathbb{Z}$. For $a \geq 0, x \geq 0$ it is based on the three-term recurrence relation [1, 19.6.8]

$$V(a+1,x) = xV(a,x) + (a-\tfrac{1}{2})V(a-1,x).$$

It is stable in the increasing a direction and the starting values are computed from Abramowitz and Stegun [1, 19.15] and the relation [1, 9.6.2] between the modified Bessel functions I_ν and K_ν:

$$\begin{aligned}
V(\tfrac{1}{2},x) &= \sqrt{\frac{2}{\pi}}e^{x^2/4} \\
V(\tfrac{3}{2},x) &= \sqrt{\frac{2}{\pi}}xe^{x^2/4} \\
V(0,x) &= \frac{\sqrt{x}}{2}\left(I_{\frac{1}{4}}\left(\tfrac{1}{4}x^2\right) + I_{-\frac{1}{4}}\left(\tfrac{1}{4}x^2\right)\right) \\
&= \frac{\sqrt{x}}{2}\left(2I_{\frac{1}{4}}\left(\tfrac{1}{4}x^2\right) + \frac{\sqrt{2}}{\pi}K_{\frac{1}{4}}\left(\tfrac{1}{4}x^2\right)\right) \\
V(1,x) &= \frac{\sqrt{x}}{2}\left(I_{\frac{1}{4}}\left(\tfrac{1}{4}x^2\right) + I_{-\frac{1}{4}}\left(\tfrac{1}{4}x^2\right) + I_{\frac{3}{4}}\left(\tfrac{1}{4}x^2\right) + I_{-\frac{3}{4}}\left(\tfrac{1}{4}x^2\right)\right) \\
&= \frac{\sqrt{x}}{2}\left(2\left(I_{\frac{1}{4}}\left(\tfrac{1}{4}x^2\right) + I_{\frac{3}{4}}\left(\tfrac{1}{4}x^2\right)\right) + \frac{\sqrt{2}}{\pi}\left(K_{\frac{1}{4}}\left(\tfrac{1}{4}x^2\right) + K_{\frac{3}{4}}\left(\tfrac{1}{4}x^2\right)\right)\right)
\end{aligned}$$

The pairs $I_\nu\left(\frac{1}{4}x^2\right)$ and $K_\nu\left(\frac{1}{4}x^2\right)$ are evaluated simultaneously[88] with the `bessel_ik` function, except for very small arguments where linear approximations are used.

For $a \geq 0, x < 0$ reflections formulas are applied, they follow from the definition formula 3.16 and $\sin n\pi = 0,\ \sin(n+\frac{1}{2})\pi = (-1)^n$:

$$\begin{aligned}
V(n,x) &= \frac{1}{\pi}\Gamma\left(n+\tfrac{1}{2}\right)D_{-n-\frac{1}{2}}(-x) \\
V(n+\tfrac{1}{2},x) &= \frac{1}{\pi}\Gamma(n+1)\left((-1)^n D_{-n-1}(x) + D_{-n-1}(-x)\right) \\
&= (-1)^n\frac{1}{\pi}\Gamma(n+1)\left((-1)^n D_{-n-1}(-x) + D_{-n-1}(x)\right) \\
&= (-1)^n V(n+\tfrac{1}{2},-x)
\end{aligned}$$

For $a < 0$ the decomposition into odd/even functions [30, 12.4.2, 12.7.12/13] is used[89]

$$\begin{aligned}
V(a,x) &= V(a,0)\cdot u_1(a,x) + V'(a,0)\cdot u_2(a,x), \\
u_1(a,x) &= e^{-\frac{1}{4}x^2}M(\tfrac{1}{2}a+\tfrac{1}{4},\tfrac{1}{2},\tfrac{1}{2}x^2), \\
u_2(a,x) &= xe^{-\frac{1}{4}x^2}M(\tfrac{1}{2}a+\tfrac{3}{4},\tfrac{3}{2},\tfrac{1}{2}x^2)
\end{aligned}$$

[88] This is the reason for introducing the K_ν into the formulas.
[89] This decomposition is valid for all a, x but for certain $a > 0$ catastrophic cancellation occurs.

with the initial values at $x = 0$ from [1, 19.36]

$$V(a,0) = \frac{2^{\frac{1}{2}a+\frac{1}{2}} \sin \pi(\frac{3}{4} - \frac{1}{2}a)}{\Gamma(\frac{3}{4} - \frac{1}{2}a)} \quad \text{and} \quad V'(a,0) = \frac{2^{\frac{1}{2}a+\frac{3}{4}} \sin \pi(\frac{1}{4} - \frac{1}{2}a)}{\Gamma(\frac{1}{4} - \frac{1}{2}a)}.$$

3.8.10.4 Hermite function $\mathbf{H}_\nu(\mathbf{x})$

```
function HermiteH(v, x: double): double;
function HermiteHx(v, x: extended): extended;
```

These functions return $H_\nu(x)$, the Hermite function of degree ν (note that ν is replaced by `v` in the Pascal code), defined for $x \geq 0$ in Lebedev [45, Sec. 10.2]:

$$H_\nu(x) = 2^\nu \, U\left(-\frac{\nu}{2}, \frac{1}{2}, x^2\right),$$

i.e. the Hermite functions are actually exponentially scaled parabolic cylinder functions[90]

$$H_\nu(x) = 2^{\frac{\nu}{2}} \, e^{\frac{x^2}{4}} D_\nu(\sqrt{2}\, x).$$

For $\nu = n \in \mathbb{N}$ the function is the standard Hermite polynomial H_n from 3.7.6. When $x < 0$ the common function `sfc_pcfhh` uses the reflection formula[91]

$$H_\nu(x) = H_\nu(-x) - 4x \, \frac{2^\nu \sqrt{\pi}}{\Gamma\left(-\frac{\nu}{2}\right)} \, {}_1F_1\left(\frac{1-\nu}{2}, \frac{3}{2}, x^2\right).$$

[90] http://functions.wolfram.com/07.01.27.0004.01
[91] Derived from http://functions.wolfram.com/07.01.16.0012.01

3.9 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 $[A, B]$ or (A, B), the *cumulative distribution function* (CDF)

$$F(x) = \int_{-\infty}^{x} f(t)\mathrm{d}t = \int_{A}^{x} f(t)\mathrm{d}t, \qquad F(x) = 1 \text{ for } x \geq B,$$

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.[92] If regarded as functions of continuous arguments, the CDFs of discrete distributions are step functions and have no mathematical inverses and therefore **AMath** implements no discrete ICDFs.[93]

There are two additional references for this section, they are not used in actual implementation but provide formulas and background information (e.g. other parameters of the distributions like mean, median, mode, variance, skewness etc.):

- NIST/SEMATECH e-Handbook of Statistical Methods, 2006 ed., Section 1.3.6 Probability Distributions http://www.itl.nist.gov/div898/handbook/eda/section3/eda36.htm
- Wikipedia, the free encyclopedia. List of probability distributions with links to many specific probability distributions: http://en.wikipedia.org/wiki/List_of_probability_distributions

3.9.1 Erlang, geometric, and Pascal distributions

Being special cases of other distributions they are not implemented separately: The Erlang distribution is a gamma distribution (3.9.9) with shape $a \in \mathbb{N}$, the geometric and Pascal distributions are negative binomial distribution (3.9.21) with $r = 1$ and $r \in \mathbb{N}$ respectively.

3.9.2 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 $a > 0$, $b > 0$, and the support interval $(0, 1)$:

$$\begin{aligned} \text{PDF:} \quad & f(x) = x^{a-1}(1-x)^{b-1}/B(a,b) \\ \text{CDF:} \quad & F(x) = I_x(a,b) = \text{ibeta}(a,b,x) \\ \text{ICDF:} \quad & F^{-1}(y) = \text{ibeta\_inv}(a,b,y) \end{aligned}$$

[92] The **AMath** functions have the suffixes '_pdf', '_pmf', '_cdf', and '_inv' (plus the usual 'x' for the extended versions).

[93] As can be seen by the closed-form CDFs, estimates can be computed for the Poisson and negative binomial distribution using normalised inverse incomplete gamma or beta functions. But even with this approach, the mapping of the obtained value to the discrete domain is not always obvious.

3.9.3 Binomial distribution

```
function binomial_pmf(p: double; n, k: longint): double;
function binomial_cdf(p: double; n, k: longint): double;
function binomial_pmfx(p: extended; n, k: longint): extended;
function binomial_cdfx(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 $0 \leq p \leq 1$.

$$\text{PMF:} \quad f(k) = \binom{n}{k} p^k (1-p)^{n-k} = \text{beta\_pdf}(k+1, n-k+1, p)/(n+1)$$

$$\text{CDF:} \quad F(k) = I_{1-p}(n-k, k+1) = \text{ibeta}(n-k, k+1, 1-p)$$

3.9.4 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), $b > 0$ (scale), and the support interval $(-\infty, +\infty)$:

$$\text{PDF:} \quad f(x) = \frac{1}{\pi b(1 + ((x-a)/b)^2)}$$

$$\text{CDF:} \quad F(x) = \frac{1}{2} + \frac{1}{\pi} \arctan\left(\frac{x-a}{b}\right)$$

The inverse cumulative distribution function is

$$F^{-1}(y) = \begin{cases} a - b/\tan(\pi y) & y < 0.5, \\ a & y = 0.5, \\ a - b/\tan(\pi(1-y)) & y > 0.5\ . \end{cases}$$

3.9.5 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 $(0, +\infty)$:

$$\text{PDF:} \quad f(x) = \frac{(\frac{1}{2}x)^{\nu/2-1} \exp(-\frac{1}{2}x)}{2\Gamma(\nu/2)} = \text{sfc\_igprefix}(\tfrac{1}{2}\nu, \tfrac{1}{2}x)/x$$

$$\text{CDF:} \quad F(x) = P(\tfrac{1}{2}\nu, \tfrac{1}{2}x) = \text{igammap}(\tfrac{1}{2}\nu, \tfrac{1}{2}x)$$

$$\text{ICDF:} \quad F^{-1}(p) = 2 \cdot \text{igammap\_inv}(\tfrac{1}{2}\nu, p)$$

3.9.6 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, rate $\alpha > 0$, and the support interval $(a, +\infty)$:

$$\begin{aligned} \text{PDF:} \quad & f(x) = \alpha \exp(-\alpha(x-a)) \\ \text{CDF:} \quad & F(x) = 1 - \exp(-\alpha(x-a)) = \operatorname{expm1}(-\alpha(x-a)) \\ \text{ICDF:} \quad & F^{-1}(y) = a - \operatorname{ln1p}(-y)/\alpha \end{aligned}$$

3.9.7 Extreme Value Type I distribution

```
function evt1_pdf(a, b, x: double): double;
function evt1_cdf(a, b, x: double): double;
function evt1_inv(a, b, y: double): double;
function evt1_pdfx(a, b, x: extended): extended;
function evt1_cdfx(a, b, x: extended): extended;
function evt1_invx(a, b, y: extended): extended;
```

These functions return PDF, CDF, and ICDF of the Extreme Value Type I distribution with location a, scale $b > 0$, and the support interval $(-\infty, +\infty)$: [94]

$$\begin{aligned} \text{PDF:} \quad & f(x) = \frac{e^{-(x-a)/b}}{b} e^{-e^{-(x-a)/b}} \\ \text{CDF:} \quad & F(x) = e^{-e^{-(x-a)/b}} \\ \text{ICDF:} \quad & F^{-1}(y) = a - b \ln(-\ln y) \end{aligned}$$

3.9.8 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 $(0, +\infty)$.

$$\begin{aligned} \text{PDF:} \quad & f(x) = \frac{x^{\frac{1}{2}\nu_1 - 1}}{B(\frac{1}{2}\nu_1, \frac{1}{2}\nu_2)} \left(\frac{\nu_1}{\nu_2}\right)^{\frac{1}{2}\nu_1} \left(1 + \frac{\nu_1}{\nu_2} x\right)^{-\frac{1}{2}(\nu_1+\nu_2)} \\ \text{CDF:} \quad & F(x) = I_{\frac{\nu_1 x}{\nu_1 x + \nu_2}}\left(\tfrac{1}{2}\nu_1, \tfrac{1}{2}\nu_2\right) = \operatorname{ibeta}\left(\tfrac{1}{2}\nu_1, \tfrac{1}{2}\nu_2, \frac{\nu_1 x}{\nu_1 x + \nu_2}\right) \end{aligned}$$

[94] "The Extreme Value Type I distribution has two forms. One is based on the smallest extreme and the other is based on the largest extreme. We call these the minimum and maximum cases, respectively." (http://www.itl.nist.gov/div898/handbook/eda/section3/eda366g.htm) See also [66, Section 3.11.4.1]; **AMath** implements the maximum case.

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.

$$F^{-1}(y) = \frac{\nu_2 x}{\nu_1(1-x)} \quad \text{with} \quad x = \text{beta\_inv}(\tfrac{1}{2}\nu_1, \tfrac{1}{2}\nu_2, y),$$
$$F^{-1}(y) = \frac{\nu_2(1-x)}{\nu_1 x} \quad \text{with} \quad x = \text{beta\_inv}(\tfrac{1}{2}\nu_1, \tfrac{1}{2}\nu_2, 1-y).$$

3.9.9 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 $a > 0$, scale $b > 0$, and the support interval $(0, +\infty)$. Note that a gamma distribution with shape $a \in \mathbb{N}$ is called *Erlang* distribution.

$$\text{PDF:} \quad f(x) = \frac{x^{a-1}e^{-x/b}}{\Gamma(a)b^a}$$
$$\text{CDF:} \quad F(x) = P(a, x/b) = \text{igammap}(a, x/b)$$
$$\text{ICDF:} \quad F^{-1}(p) = b \cdot \text{igammap\_inv}(a, p)$$

3.9.10 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 μ, standard deviation $\sigma > 0$, and the support interval $(-\infty, +\infty)$:

$$\text{PDF:} \quad f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$
$$\text{CDF:} \quad F(x) = \frac{1}{2}\left(1 + \text{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right) = \text{normstd\_cdf}\left(\frac{x-\mu}{\sigma}\right)$$
$$\text{ICDF:} \quad F^{-1}(y) = \sigma \cdot \text{normstd\_inv}(y) + \mu$$

3.9.11 Hypergeometric distribution

```
function hypergeo_pmf(n1,n2,n,k: longint): double;
function hypergeo_cdf(n1,n2,n,k: longint): double;
function hypergeo_pmfx(n1,n2,n,k: longint): extended;
function hypergeo_cdfx(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$_1$ objects and n_2 type$_2$ objects there are exactly k type$_1$ objects[95]:

$$\text{PMF}: \quad f(k) = \frac{\binom{n_1}{k}\binom{n_2}{n-k}}{\binom{n_1+n_2}{n}} \quad (n, n_1, n_2 \geq 0;\ n \leq n_1 + n_2).$$

$f(k)$ 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:

$$f(k+1) = \frac{(n_1 - k)(n - k)}{(k+1)(n_2 - n + k + 1)} f(k)$$
$$f(k-1) = \frac{k(n_2 - n + k)}{(n_1 - k + 1)(n - k + 1)} f(k)$$

3.9.12 Inverse gamma distribution

```
function invgamma_pdf(a, b, x: double): double;
function invgamma_cdf(a, b, x: double): double;
function invgamma_inv(a, b, y: double): double;
function invgamma_pdfx(a, b, x: extended): extended;
function invgamma_cdfx(a, b, x: extended): extended;
function invgamma_invx(a, b, y: extended): extended;
```

These functions return PDF, CDF, and ICDF of the inverse gamma distribution with shape $a > 0$, scale $b > 0$, and the support interval $(0, +\infty)$.

$$\text{PDF}: \quad f(x) = \left(\frac{b}{x}\right)^a \frac{e^{-\frac{b}{x}}}{x\Gamma(x)} = \text{sfc\_igprefix}(a, b/x)/x$$
$$\text{CDF}: \quad F(x) = Q(a, b/x) = \text{igammaq}(a, b/x)$$
$$\text{ICDF}: \quad F^{-1}(y) = \frac{b}{Q^{-1}(a, y)} = \frac{b}{\text{igammaq\_inv}(a, y)}$$

3.9.13 Kumaraswamy distribution

```
function kumaraswamy_pdf(a, b, x: double): double;
function kumaraswamy_cdf(a, b, x: double): double;
function kumaraswamy_inv(a, b, y: double): double;
function kumaraswamy_pdfx(a, b, x: extended): extended;
function kumaraswamy_cdfx(a, b, x: extended): extended;
function kumaraswamy_invx(a, b, y: extended): extended;
```

These functions return PDF, CDF, and ICDF of the Kumaraswamy distribution with shape parameters $a > 0$, $b > 0$, and the support interval $(0, 1)$:

$$\text{PDF}: \quad f(x) = abx^{a-1}\left(1 - x^a\right)^{b-1}$$
$$\text{CDF}: \quad F(x) = 1 - \left(1 - x^a\right)^b$$
$$\text{ICDF}: \quad F^{-1}(y) = \left(1 - (1-y)^{\frac{1}{b}}\right)^{\frac{1}{a}}$$

[95] There are different definitions; the **AMath** version is compatible with Maple and R.

3.9.14 Laplace distribution

```
function laplace_pdf(a, b, x: double): double;
function laplace_cdf(a, b, x: double): double;
function laplace_inv(a, b, y: double): double;
function laplace_pdfx(a, b, x: extended): extended;
function laplace_cdfx(a, b, x: extended): extended;
function laplace_invx(a, b, y: extended): extended;
```

These functions return PDF, CDF, and ICDF of the Laplace distribution with location a, scale $b > 0$, and the support interval $(-\infty, +\infty)$:

$$\text{PDF:} \quad f(x) = \exp(-|x-a|/b)/(2b)$$

$$\text{CDF:} \quad F(x) = \begin{cases} \frac{1}{2} - \frac{1}{2}\operatorname{expm1}(-\frac{x-a}{b}) & x \geq a \\ \frac{1}{2}\exp(-\frac{x-a}{b}) & x < a \end{cases}$$

$$\text{ICDF:} \quad F^{-1}(y) = \begin{cases} a + b\ln(2y) & y < 0.5 \\ a - b\ln(2(1-y)) & y \geq 0.5 \end{cases}$$

3.9.15 Lévy distribution

```
function levy_pdf(a, b, x: double): double;
function levy_cdf(a, b, x: double): double;
function levy_inv(a, b, y: double): double;
function levy_pdfx(a, b, x: extended): extended;
function levy_cdfx(a, b, x: extended): extended;
function levy_invx(a, b, y: extended): extended;
```

These functions return PDF, CDF, and ICDF of the Lévy distribution with location a and scale $b > 0$ and the support interval $(a, +\infty)$:

$$\text{PDF:} \quad f(x) = \sqrt{\frac{b}{2\pi}}\,\frac{e^{-\frac{b}{2(x-a)}}}{(x-a)^{3/2}}$$

$$\text{CDF:} \quad F(x) = \operatorname{erfc}\left(\sqrt{\frac{b}{2(x-a)}}\right)$$

$$\text{ICDF:} \quad F^{-1}(y) = a + \frac{b}{2\operatorname{erfc}^{-1}(y)^2}$$

3.9.16 Logarithmic (series) distribution

```
function logseries_pmf(a: double; k: longint): double;
function logseries_cdf(a: double; k: longint): double;
function logseries_pmfx(a: extended; k: longint): extended;
function logseries_cdfx(a: extended; k: longint): extended;
```

These functions return PMF and CDF of the (discrete) logarithmic (series) distribution with shape parameter $0 < a < 1$ and $k > 0$ (where Φ is Lerch's transcendent 3.6.9)[96]:

$$\text{PMF:} \quad f(k) = -\frac{a^k}{k\ln(1-a)}$$

$$\text{CDF:} \quad F(k) = \sum_{j=1}^{k} f(j) = 1 + \frac{a^k\left(\Phi(a,1,k) - \frac{1}{k}\right)}{\ln(1-a)}$$

[96] Numerically the closed form with Φ is more effective and accurate for large k or $a \approx 1$.

3.9.17 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_pdfx(a, b, x: extended): extended;
function logistic_cdfx(a, b, x: extended): extended;
function logistic_invx(a, b, y: extended): extended;
```

These functions return PDF, CDF, and ICDF of the logistic distribution with parameters a (location), $b > 0$ (scale), and the support interval $(-\infty, +\infty)$:

$$\begin{aligned}
\text{PDF:} \quad f(x) &= \frac{1}{b} \frac{\exp(-\frac{x-a}{b})}{\left(1 + \exp(-\frac{x-a}{b})\right)^2} \\
\text{CDF:} \quad F(x) &= \frac{1}{1 + \exp(\frac{a-x}{b})} \\
\text{ICDF:} \quad F^{-1}(y) &= a - b \ln\big((1-y)/y\big)
\end{aligned}$$

3.9.18 Log-normal 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 $(0, +\infty)$:

$$\begin{aligned}
\text{PDF:} \quad f(x) &= \frac{1}{bx\sqrt{2\pi}} \exp\left(-\frac{(\ln x - a)^2}{2b^2}\right) \\
\text{CDF:} \quad F(x) &= \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{\ln x - a}{b\sqrt{2}}\right)\right) \\
\text{ICDF:} \quad F^{-1}(y) &= \exp\big(a + b \cdot \text{normstd_inv}(y)\big)
\end{aligned}$$

3.9.19 Maxwell distribution

```
function maxwell_pdf(b, x: double): double;
function maxwell_cdf(b, x: double): double;
function maxwell_inv(b, y: double): double;
function maxwell_pdfx(b, x: extended): extended;
function maxwell_cdfx(b, x: extended): extended;
function maxwell_invx(b, y: extended): extended;
```

These functions return PDF, CDF, and ICDF of the Maxwell distribution with scale $b > 0$ and the support interval $(0, +\infty)$:

$$\text{PDF:} \quad f(x) = \sqrt{\frac{2}{\pi}} \left(\frac{x^2}{b^3}\right) \exp\left(-\frac{x^2}{2b^2}\right)$$
$$\text{CDF:} \quad F(x) = P\left(\frac{3}{2}, \frac{x^2}{2b^2}\right)$$
$$\text{ICDF:} \quad F^{-1}(y) = b\sqrt{2P^{-1}\left(\frac{3}{2}, y\right)}$$

where P and P^{-1} are the normalised lower incomplete gamma function and its inverse.

3.9.20 Moyal distribution

```
function moyal_pdf(a, b, x: double): double;
function moyal_cdf(a, b, x: double): double;
function moyal_inv(a, b, y: double): double;
function moyal_pdfx(a, b, x: extended): extended;
function moyal_cdfx(a, b, x: extended): extended;
function moyal_invx(a, b, y: extended): extended;
```

These functions return PDF, CDF, and ICDF of the Moyal distribution with location a and scale $b > 0$ and the support interval $(-\infty, +\infty)$:

$$\text{PDF:} \quad f(x) = \frac{e^{-\frac{x-a}{2b} - \frac{1}{2}e^{-\frac{x-a}{b}}}}{\sqrt{2\pi}b}$$
$$\text{CDF:} \quad F(x) = \text{erfc}\left(\frac{e^{-\frac{x-a}{2b}}}{\sqrt{2}}\right)$$
$$\text{ICDF:} \quad F^{-1}(y) = a - b\ln\left(2\text{erfc\_inv}(y)^2\right)$$

3.9.21 Negative binomial distribution

```
function negbinom_pmf(p,r: double; k: longint): double;
function negbinom_cdf(p,r: double; k: longint): double;
function negbinom_pmfx(p,r: extended; k: longint): extended;
function negbinom_cdfx(p,r: extended; k: longint): extended;
```

These functions return PMF and CDF of the (discrete) negative binomial distribution[97] with target for number of successful trials $r > 0$ and success probability $0 \le p \le 1$:

$$\text{PMF:} \quad f(k) = \frac{\Gamma(k+r)}{k!\Gamma(r)} p^r (1-p)^k = \frac{p}{r+k}\text{beta\_pdf}(r, k+1, p)$$
$$\text{CDF:} \quad F(k) = I_{1-p}(r, k+1) = \text{ibeta}(r, k+1, 1-p)$$

Note that if $r = n$ is a positive integer the name *Pascal* distribution is used, and for $r = 1$ it is called *geometric* distribution.

[97] There are different definitions for the negative binomial distribution; the **AMath** version is compatible with Maple, R, GSL, Boost.

3.9.22 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 $k > 0$, shape $a > 0$, and the support interval $(k, +\infty)$:

$$\begin{aligned} \text{PDF:} \quad & f(x) = \frac{a}{x}\left(\frac{k}{x}\right)^a \\ \text{CDF:} \quad & F(x) = 1 - \left(\frac{k}{x}\right)^a = -\operatorname{powm1}(k/x, a) \\ \text{ICDF:} \quad & F^{-1}(y) = k(1-y)^{-1/a} \end{aligned}$$

3.9.23 Poisson distribution

```
function poisson_pmf(mu: double; k: longint): double;
function poisson_cdf(mu: double; k: longint): double;
function poisson_pmfx(mu: extended; k: longint): extended;
function poisson_cdfx(mu: extended; k: longint): extended;
```

These functions return PMF and CDF of the Poisson distribution with mean $\mu \geq 0$.

$$\begin{aligned} \text{PMF:} \quad & f(k) = \frac{\mu^k}{k!}e^{-\mu} = \operatorname{sfc\_igprefix}(1+k, \mu) \\ \text{CDF:} \quad & F(k) = e^{-\mu}\sum_{i=0}^{k}\frac{\mu^i}{i!} = \operatorname{igammaq}(1+k, \mu) \end{aligned}$$

3.9.24 Rayleigh distribution

```
function rayleigh_pdf(b, x: double): double;
function rayleigh_cdf(b, x: double): double;
function rayleigh_inv(b, y: double): double;
function rayleigh_pdfx(b, x: extended): extended;
function rayleigh_cdfx(b, x: extended): extended;
function rayleigh_invx(b, y: extended): extended;
```

These functions return PDF, CDF, and ICDF of the Rayleigh distribution with scale $b > 0$ and the support interval $(0, +\infty)$:[98]

$$\begin{aligned} \text{PDF:} \quad & f(x) = \frac{x}{b^2}\exp\left(-\frac{x^2}{2b^2}\right) \\ \text{CDF:} \quad & F(x) = 1 - \exp\left(-\frac{x^2}{2b^2}\right) = -\operatorname{expm1}\left(-\frac{x^2}{2b^2}\right) \\ \text{ICDF:} \quad & F^{-1}(y) = b\sqrt{-2\operatorname{ln1p}(-y)} \end{aligned}$$

[98] Note that the Rayleigh distribution actually is a Weibull distribution with shape=2 and scale=$b\sqrt{2}$, separate functions are implemented to give better accuracy.

3.9.25 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_pdfx(nu: longint; x: extended): extended;
function t_cdfx(nu: longint; t: extended): extended;
function t_invx(nu: longint; p: extended): extended;
```

These functions return PDF, CDF, and ICDF of Student's t-distribution with $\nu \in \mathbb{N}$ degrees of freedom and the support interval $(-\infty, +\infty)$:

$$\text{PDF:} \quad f(x) = \frac{1}{\sqrt{\nu} B(\frac{1}{2}, \frac{1}{2}\nu)} \left(1 + \frac{x^2}{\nu}\right)^{-\frac{1}{2}(\nu+1)}$$

$$\text{CDF:} \quad F(t) = \begin{cases} I_z(\frac{1}{2}\nu, \frac{1}{2}) & t \le 0, \quad \text{with} \quad z = \frac{\nu}{\nu + t^2} \\ 1 - I_z(\frac{1}{2}\nu, \frac{1}{2}) & t > 0 \end{cases}$$

The actual implementation of the CDF is based on Cephes [7, stdtrl.c] and uses the normalised incomplete beta function I_z (with transformed arguments, cf. [1, 26.7.1]) only for $t < -1.6$ or $\nu > 20000$; otherwise it is computed with the series expansions from [1, 26.7.3/4]. The ICDF uses beta_inv and distinguishes between several cases.

3.9.26 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_pdfx(a, b, c, x: extended): extended;
function triangular_cdfx(a, b, c, x: extended): extended;
function triangular_invx(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, $a \le c \le b$.

$$\text{PDF:} \quad f(x) = \begin{cases} 0 & x < a \\ \frac{2(x-a)}{(b-a)(c-a)} & a \le x < c \\ \frac{2}{b-a} & x = c \\ \frac{2(b-x)}{(b-a)(b-c)} & c < x \le b \\ 0 & x > b \end{cases}$$

$$\text{CDF:} \quad F(x) = \begin{cases} 0 & x < a \\ \frac{(x-a)^2}{(b-a)(c-a)} & a \le x < c \\ \frac{c-a}{b-a} & x = c \\ 1 - \frac{(b-x)^2}{(b-a)(b-c)} & c < x \le b \\ 1 & x > b \end{cases}$$

and with $t = (c-a)/(b-a)$

$$\text{ICDF:} \quad F^{-1}(y) = \begin{cases} a + \sqrt{(b-a)(c-a)y} & y < t \\ c & y = t \\ b - \sqrt{(b-a)(b-c)(1-y)} & y > t \end{cases}$$

3.9.27 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 $(-\infty, +\infty)$:

$$\begin{aligned}
\text{PDF:} \quad & f(x) = \frac{1}{\sqrt{2\pi}} \exp(-\tfrac{1}{2}x^2) = \text{erf_z}(x) \\
\text{CDF:} \quad & F(x) = \Phi(x) = \frac{1}{2}\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right) = \frac{1}{2}\,\text{erfc}\left(-\frac{x}{\sqrt{2}}\right) = \text{erf_p}(x) \\
\text{ICDF:} \quad & F^{-1}(y) = -\sqrt{2}\,\text{erfc_inv}(2y)
\end{aligned}$$

3.9.28 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 $a < b$:

$$\begin{aligned}
\text{PDF:} \quad & f(x) = \frac{1}{b-a} \\
\text{CDF:} \quad & F(x) = \frac{x-a}{b-a} \\
\text{ICDF:} \quad & F^{-1}(y) = a + y(b-a)
\end{aligned}$$

3.9.29 Wald or inverse Gaussian distribution

```
function wald_pdf(mu, b, x: double): double;
function wald_cdf(mu, b, x: double): double;
function wald_inv(mu, b, y: double): double;
function wald_pdfx(mu, b, x: extended): extended;
function wald_cdfx(mu, b, x: extended): extended;
function wald_invx(mu, b, y: extended): extended;
```

These functions return PDF, CDF, and ICDF of the Wald or inverse Gaussian distribution with mean $\mu > 0$, scale $b > 0$, and the support interval $(0, +\infty)$, cf. Rinne [66, 3.11.10]:

$$\begin{aligned}
\text{PDF:} \quad & f(x) = \sqrt{\frac{b}{2\pi x^3}} \exp\left(-\frac{b(x-\mu)^2}{2x\mu^2}\right) \\
\text{CDF:} \quad & F(x) = \Phi\left[\sqrt{\frac{b}{x}}\left(\frac{x}{\mu} - 1\right)\right] + e^{2b/\mu} \cdot \Phi\left[-\sqrt{\frac{b}{x}}\left(\frac{x}{\mu} + 1\right)\right]
\end{aligned}$$

Here $\Phi(x) = \text{erf\_p}(x)$ is the CDF of the standard normal distribution.

There is no known closed form for the ICDF: It is computed with Newton iterations where the starting values are from Boost[19, file inverse_gaussian.hpp] or the R package [39, SuppDists V1.1-9.1]. [99]

3.9.30 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 $a > 0$, scale $b > 0$, and the support interval $(0, +\infty)$.

$$\begin{aligned}
\text{PDF:} \quad & f(x) = \frac{a}{x}\left(\frac{x}{b}\right)^a \exp(-(x/b)^a) \\
\text{CDF:} \quad & F(x) = 1 - \exp(-(x/b)^a) = -\text{expm1}(-(x/b)^a) \\
\text{ICDF:} \quad & F^{-1}(y) = b\left(\ln\frac{1}{1-y}\right)^{1/a} = b\big(-\text{ln1p}(-y)\big)^{1/a}
\end{aligned}$$

3.9.31 Zipf distribution

```
function zipf_pmf(r: double; k: longint): double;
function zipf_cdf(r: double; k: longint): double;
function zipf_pmfx(r: extended; k: longint): extended;
function zipf_cdfx(r: extended; k: longint): extended;
```

These functions return PMF and CDF of the (discrete) Zipf[100] distribution with the positive parameter r and $k > 0$[101]

$$\begin{aligned}
\text{PMF:} \quad & f(k) = \frac{k^{-(r+1)}}{\zeta(r+1)}, \\
\text{CDF:} \quad & F(k) = \frac{H_k^{(r+1)}}{\zeta(r+1)} = 1 - \frac{\zeta(r+1, k+1)}{\zeta(r+1)},
\end{aligned}$$

where $\zeta(r+1)$ is evaluated with the accuracy improved function `zeta1p`.

[99] The R function is sub-optimal for very small y, and both codes have difficulties (e.g. for some μ, b with $y \approx 1$). In these cases Newton iteration is repeated with the mode as starting value. This is from G. Giner and G.K. Smyth: *A monotonically convergent Newton iteration for the quantiles of any unimodal distribution, with application to the inverse Gaussian distribution*, Rev. July 11 2014 as http://www.statsci.org/smyth/pubs/qinvgaussPreprint.pdf. The listed simple R example code has problems itself, e.g. negative results for $F^{-1}(1, 2, 0.1)$ or more than 500 iterations for $F^{-1}(1, 1, 10^{-200})$, and is used only as a fall-back option.

[100] Sometimes called Zeta or discrete Pareto distribution

[101] See e.g. http://mathworld.wolfram.com/ZipfDistribution.html

3.10 Other special functions

AMAth has several other special functions which do not fit into the previous categories, some of them are implemented in the `sfMisc` unit.

3.10.1 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 $a_0 = \max(|x|, |y|)$ and $b_0 = \min(|x|, |y|)$ the AGM is calculated using the recurrence formulas

$$a_{n+1} = \frac{1}{2}(a_n + b_n)$$
$$b_{n+1} = \sqrt{a_n b_n}$$

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 ϵ is less than eps_x$^{1/2}$. 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.

3.10.2 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

$$\frac{t}{e^t - 1} = \sum_{n=0}^{\infty} B_n \frac{t^n}{n!}, \qquad |t| < 2\pi.$$

If $n < 0$ or if $n > 2$ 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[102] is 4, and the maximum relative error of 4.5 eps_x occurs for $n = 878$.

[102] Note that the recursion can go in both directions.

3.10.3 Bernoulli polynomials

```
function bernpoly(n: integer; x: double): double;
function bernpolyx(n: integer; x: extended): extended;
```

These functions return the Bernoulli polynomials $B_n(x)$ of degree $n \geq 0$, defined by the generating function [30, 24.2.3]

$$\frac{te^{xt}}{e^t - 1} = \sum_{n=0}^{\infty} B_n(x)\frac{t^n}{n!},$$

or the simple explicit representation [30, 24.2.5]

$$B_n(x) = \sum_{k=0}^{n} \binom{n}{k} B_k x^{n-k}.$$

The common function `sfc_bernpoly` handles the special cases

$$B_0(x) = 1, \quad B_1(x) = x - \tfrac{1}{2}, \quad B_n(0) = (-1)^n B_n(1) = B_n, \quad B_n(\tfrac{1}{2}) = (2^{1-n} - 1)B_n,$$

if $|x - 1| > 0.5$ the symmetry relation $B_n(x) = (-1)^n B_n(1 - x)$ is applied, and for large x with $|x| > 12000n$[103] the first two non-zero terms of the expansion[104]

$$\begin{aligned} B_n(x) &= \sum_{k=0}^{n} \binom{n}{k} B_k \left(\frac{1}{2}\right) \left(x - \frac{1}{2}\right)^{n-k} \\ &= \left(x - \frac{1}{2}\right)^n - \frac{n(n-1)}{24}\left(x - \frac{1}{2}\right)^{n-2} + \frac{7}{240}\binom{n}{4}\left(x - \frac{1}{2}\right)^{n-4} + \cdots \end{aligned}$$

are sufficient. Otherwise if $n \leq 10$ the simple sum is evaluated; and for small $|x| < \frac{1}{8n}$ it is computed backwards terminating if two subsequent non-zero terms do not change the sum. In all other cases the relations to the Hurwitz zeta function [30, 25.11.14]

$$B_n(x) = \begin{cases} -n\zeta(1 - n, x) & x \geq 0, \\ (-1)^{n+1} n\zeta(1 - n, 1 - x) & x < 0. \end{cases}$$

are used and for most n, x this will end in applying the Hurwitz formula 3.9.

3.10.4 Catalan function C(x)

```
function catalan(x: double): double;
function catalanx(x: extended): extended;
```

These functions return the Catalan function $C(x)$ defined as the generalization of the Catalan numbers C_n :

$$C_n = \frac{1}{n+1}\binom{2n}{n}, \quad C(x) = \frac{1}{x+1}\binom{2x}{x}$$

Rewriting the binomial coefficient and applying the Γ duplication formula we get

$$C(x) = \frac{\Gamma(2x+1)}{(x+1)\Gamma(x+1)^2} = \frac{2^{2x}\Gamma(x+\frac{1}{2})}{x(x+1)\sqrt{\pi}\Gamma(x)} = \frac{2^{2x}\Gamma(x+\frac{1}{2})}{\sqrt{\pi}\Gamma(x+2)}$$

The common function evaluates the Γ quotient with `gamma_delta_ratio`$(x + 0.5, 1.5)$ and rounds the result for integer $x \leq 35$.[105]

[103] $7/240/12000^4/4! \approx 0.58607 \cdot 10^{-19} \approx 0.5\text{eps\_x}$

[104] This follows from [30, 24.4.12] with the substitutions $h := x - \frac{1}{2}$, $x := \frac{1}{2}$ and $B_{2k+1}(\frac{1}{2}) = 0$.

[105] The extended numbers C_n are exactly computed up to $C_{35} = 3116285494907301262$, the maximum relative error of $4.337 \cdot 10^{-19}$ occurs first for $n = 664$, and the maximum index is $n = 8202$.

3.10.5 Debye functions

```
function debye(n: integer; x: double): double;
function debyex(n: integer; x: extended): extended;
```

These functions return the Debye functions

$$\mathrm{D_n}(x) = \frac{n}{x^n}\int_0^x \frac{t^n}{e^t-1}\,\mathrm{d}t \qquad (n>0,\ x\ge 0).$$

The calculation is based on section 27.1 of Abramowitz and Stegun[1]. For $x \le 4$ the formula [1, 27.1.1]

$$\int_0^x \frac{t^n}{e^t-1}\,\mathrm{d}t = x^n\left(\frac{1}{n} - \frac{x}{2(n+1)} + \sum_{k=1}^{\infty}\frac{B_{2k}x^{2k}}{(2k+n)(2k)!}\right)$$

is used, where the B_{2k} are the Bernoulli numbers 3.10.2. The series converges for $x < 2\pi$. If $x > 4$ the partial sums of [1, 27.1.2]

$$\int_x^\infty \frac{t^n}{e^t-1}\,\mathrm{d}t = \sum_{k=1}^{\infty} e^{-kx}\left(\frac{x^n}{k} + \frac{nx^{n-1}}{k^2} + \frac{n(n-1)x^{n-2}}{k^3} + \cdots + \frac{n!}{k^{n+1}}\right)$$

are subtracted from the complete integral [1, 27.1.3]

$$\int_0^\infty \frac{t^n}{e^t-1}\,\mathrm{d}t = n!\zeta(n+1).$$

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[106] for $x > 4$ and $n > 6$.

3.10.6 Euler numbers

```
function euler(n: integer): double;
function eulerx(n: integer): extended;
```

These functions return the Euler numbers E_n, which are defined by [30, 24.2.6]

$$\operatorname{sech}(t) = \frac{2e^t}{e^{2t}+1} = \sum_{n=0}^{\infty} E_n\frac{t^n}{n!}, \qquad |t| < \tfrac{1}{2}\pi.$$

If $n < 0$ or if n is odd, the result is 0, otherwise the common function `sfc_euler` simply combines the Bernoulli numbers, the Dirichlet β, and the Riemann ζ function:[107]

$$E_n = -\frac{4^n\beta(n+1)}{\zeta(n)}\,\frac{2B_n}{\pi}$$

The **AMath** maximum relative error 4.5 eps_x is inherited from the Bernoulli numbers, the maximum index is $n = 1866$ for extended (and $n = 186$ for double).[108]

[106] A customised version of the **AMTools** procedure `intde` based on [38]

[107] The formula follows from the Fourier series [30, 24.8.1 and 24.8.4] and $E_n = 2^n E_n(\frac{1}{2})$. Note that $\beta(n+1)$ and $\zeta(n)$ rapidly converge to 1, and for $n \le 26$ the values are taken from a table. In **DAMath** the table includes E_{30} and the Dirichlet β part is not needed.

[108] The simpler relations $E_n = 4^{n+1}\zeta(-n, \frac{1}{4})$ or even $E_n = 2\beta(-n)$ have relative errors of about 600 eps_x for $n \approx 1700$ (resulting from the use of $\Gamma(n+1)$ and scaled powers π^{n+1}).

3.10.7 Fibonacci polynomials

```
function fibpoly(n: integer; x: double): double;
function fibpolyx(n: integer; x: extended): extended;
```

These functions return $F_n(x)$, the Fibonacci polynomial of index $n \in \mathbb{Z}$, defined by

$$\begin{aligned} F_0(x) &= 0, \\ F_1(x) &= 1, \\ F_n(x) &= xF_{n-1}(x) + F_{n-2}(x), \quad n \geq 2, \end{aligned}$$

and $F_{-n}(x) = (-1)^{n-1}F_n(x)$ for negative indices. With $f_n = F_n(x)$ and the matrices

$$M_n = \begin{pmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{pmatrix}, \quad X = \begin{pmatrix} x & 1 \\ 1 & 0 \end{pmatrix},$$

the recursive definition implies for the product M_nX

$$\begin{pmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{pmatrix} \begin{pmatrix} x & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} xf_{n+1} + f_{n+1} & f_{n+1} \\ xf_n + f_{n-1} & f_n \end{pmatrix} = \begin{pmatrix} f_{n+2} & f_{n+1} \\ f_{n+1} & f_n \end{pmatrix} = M_{n+1},$$

and therefore $f_n = F_n(x) = X_{1,1}^{n-1}$. The common function `sfc_fpoly` computes $F_n(x)$ for $x > 0$ from the recurrence formulas if $1 < n < 32$, and uses a fast matrix powering algorithm (with $O(\log(n))$ multiplications) if $n \geq 32$. For $x < 0$ the parity relation $F_n(-x) = (-1)^{n-1}F_n(x)$ is applied[109]; $F_n(0)$ is 1 for odd n, zero otherwise. Note that the ordinary Fibonacci numbers $F_n = F_n(1)$ can be calculated up to $n = 23601$ with the extended function ($n = 1476$ for double)[110].

3.10.8 Integral of cos powers

```
function cosint(n: integer; x: double): double;
function cosintx(n: integer; x: extended): extended;
```

These functions return for $n \geq 0$ the integral of the nth cos power:

$$\mathrm{IC}_n(x) = \int_0^x \cos^n t \, \mathrm{d}t.$$

The recurrence relation for the integrals from Abramowitz and Stegun [1, 4.3.127]

$$\mathrm{IC}_n(x) = \frac{\sin x \cos^{n-1} x}{n} + \frac{n-1}{n} \int_0^x \cos^{n-2} t \, \mathrm{d}t = \frac{n-1}{n} \mathrm{IC}_{n-2}(x) + \frac{1}{n} \sin x \cos^{n-1} x$$

is stable for all x and n; but it is used for $|x| \leq \pi$ only. The common function `sfc_cosint` handles the special cases $\mathrm{IC}_0(x) = x$, $\mathrm{IC}_1(x) = \sin(x)$, and $\mathrm{IC}_n(0) = 0$. For odd n the integral is periodic and the result is computed as $\mathrm{IC}_n(x \bmod 2\pi)$. For even $n = 2m$ the function is quasi-periodic

$$\mathrm{IC}_{2m}(k\pi + x) = \mathrm{IC}_{2m}(x) + 2k \cdot \mathrm{IC}_{2m}\left(\frac{\pi}{2}\right), \qquad \mathrm{IC}_n\left(\frac{\pi}{2}\right) = \frac{\sqrt{\pi}}{2} \frac{\Gamma\left(\frac{n}{2} + \frac{1}{2}\right)}{\Gamma\left(\frac{n}{2} + 1\right)}.$$

[109] http://functions.wolfram.com/05.12.04.0002.01

[110] The F_n are exactly computed up to $F_{102} = 92737269219307899917 6$, which is greater than 2^{69}; the maximum relative error of $9.6494 \cdot 10^{-18}$ occurs first for $n = 17561$.

3.10.9 Integral of sin powers

```
function sinint(n: integer; x: double): double;
function sinintx(n: integer; x: extended): extended;
```

These functions return for $n \geq 0$ the integral of the nth sin power:

$$\mathrm{IS}_n(x) = \int_0^x \sin^n t \, \mathrm{d}t$$

Unfortunately the recurrence formula from Abramowitz and Stegun [1, 4.3.127] is not stable; therefore if $|x| \leq 1$ the series[111]

$$\mathrm{IS}_n(x) = \sum_{k=0}^{\infty} \frac{(2k-1)!!}{(2k)!!} \frac{\sin^{2k+n+1}(x)}{2k+n+1} \qquad \left(|x| < \frac{\pi}{2}\right)$$

is evaluated and for $1 < |x| \leq \pi$ the relation

$$\mathrm{IS}_n(x) = \mathrm{IC}_n\left(\frac{\pi}{2}\right) - \mathrm{IC}_n\left(\frac{\pi}{2} - x\right)$$

is applied. The common function `sfc_sinint` handles the special cases $\mathrm{IS}_0(x) = x$, $\mathrm{IS}_1(x) = \operatorname{vers} x$, and $\mathrm{IS}_n(0) = 0$. For odd n the integral is periodic and the result is computed as $\mathrm{IS}_n(x \bmod 2\pi)$. For even $n = 2m$ the function is quasi-periodic

$$\mathrm{IS}_{2m}(k\pi + x) = \mathrm{IS}_{2m}(x) + 2k \cdot \mathrm{IS}_{2m}\left(\frac{\pi}{2}\right), \quad \mathrm{IS}_n\left(\frac{\pi}{2}\right) = \mathrm{IC}_n\left(\frac{\pi}{2}\right) = \frac{\sqrt{\pi}}{2} \frac{\Gamma\left(\frac{n}{2} + \frac{1}{2}\right)}{\Gamma\left(\frac{n}{2} + 1\right)}.$$

3.10.10 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

$$W(x)\, e^{W(x)} = x.$$

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[112], and for $x \geq 3$ from the asymptotic series [23, 4.19] or [24, 40].

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[113] for $x < -0.125$, and otherwise with the asymptotic series near zero [24, 40].

[111] Derived by using the binomial series for $(1 - \sin^2 x)^{-\frac{1}{2}}$ with $|x| < \frac{\pi}{2}$ and term-wise integration of

$$\sin^n x = \frac{\sin^n x \cos x}{\sqrt{1 - \sin^2 x}} = \sin^n x \cos x \sum_{k=0}^{\infty} \frac{(2k-1)!!}{(2k)!!} \sin^{2k} x = \sum_{k=0}^{\infty} \frac{(2k-1)!!}{(2k)!!} \sin^{2k+n}(x) \cos x$$

[112] Computed with Maple V: `pade(LambertW(x), x, [3,2])`

[113] `pade(LambertW(-1,x),x=-0.2,[2,2])`

3.10.11 Lucas polynomials

```
function lucpoly(n: integer; x: double): double;
function lucpolyx(n: integer; x: extended): extended;
```

These functions return $L_n(x)$, the Lucas polynomial of index $n \in \mathbb{Z}$, defined by

$$\begin{aligned} L_0(x) &= 2, \\ L_1(x) &= x, \\ L_n(x) &= xL_{n-1}(x) + L_{n-2}(x), \quad n \geq 2, \end{aligned}$$

and $L_{-n}(x) = (-1)^n L_n(x)$ for negative indices. The Lucas polynomials are closely related to the Fibonacci polynomials for all n, x :

$$L_n(x) = xF_n(x) + 2F_{n-1}(x) = F_{n+1}(x) + F_{n-1}(x)$$

The common function `sfc_lpoly` computes $L_n(x)$ with the recurrence formulas if $n < 32$, and from $F_n(x), F_{n-1}(x)$ using the Fibonacci matrix algorithm if $n \geq 32$.

3.10.12 Riemann prime counting function R(x)

```
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.

3.10.13 Solutions of Kepler's equation

```
function kepler(M,e: double): double;
function keplerx(M,e: extended): extended;
```

These functions return the solutions (eccentric anomaly x) of Kepler's equation from the mean anomaly M and the eccentricity $e \geq 0$, more precisely the solutions x of

$$\begin{aligned} M &= x - e\sin x, && e < 1 \\ M &= x + x^3/3, && e = 1 \quad \text{(Barker's equation)} \\ M &= e\sinh x - x, && e > 1 \end{aligned}$$

The common function `sfc_kepler` computes the solutions for $M > 0$ with Halley iterations if $e \neq 1$ and uses a robust direct formula for the cubic equation when $e = 1$[114]; for negative M the sign of x is adjusted.

[114] See T. Fukushima, Theoretical Astrometry, Draft March 19, 2003, especially Ch. 8.8 Analytic Solution of Barker's Equation, formula (8.209), http://chiron.mtk.nao.ac.jp/~toshio/education/main.pdf

Chapter 4

Complex functions

The units `AMCmplx` and `DAMCmplx` provide **AMath/DAMath** based complex arithmetic, basic and transcendental functions. The `complex` data type is a record with real and imaginary parts (using the base type extended in `AMCmplx` and double in `DAMCmplx`). Most routines are procedures with `const` input record(s) and a `var` output record; therefore the units are usable with BP7 or newer (only a restriction with `AMCmplx`, since **DAMath** already assumes BP7+).

4.1 Complex arithmetic and basic functions

Unless indicated otherwise, this section assumes $z = x + iy$ and $w = u + iv$ for functions with argument z and result w. In other cases the explicit real and imaginary parts like $\Re x, \Im y, \ldots$ are used.

4.1.1 Internal square root

```
procedure cx_sqrt(a,b: extended; var u,v: extended);
```

This procedure returns $u + iv = \sqrt{a + ib}$ based on Abramowitz and Stegun [1, 3.7.27] and Numerical Recipes [13, 5.4.6/7]. For the trivial case $a = b = 0$ the result is $u = v = 0$. Otherwise with $x = |a|$ and $y = |b|$ two helper variables r, t are computed:

$$t = x, \quad r = \frac{1}{2}\left(1 + \sqrt{1 + \left(\frac{y}{x}\right)^2}\right) \qquad x \geq y$$

$$t = y, \quad r = \frac{1}{2}\left(\frac{x}{y} + \sqrt{1 + \left(\frac{x}{y}\right)^2}\right) \qquad x < y$$

Then another square root is taken using overflow protected cases

$$t = \begin{cases} \sqrt{tr} & t \leq \texttt{MaxExtended}/1.25 \\ \sqrt{t}\sqrt{r} & \text{otherwise} \end{cases}$$

The returned answer depends on the signs of a, b. If $a \geq 0$ it is

$$u = t, \quad v = \frac{b}{2t},$$

otherwise if $b < 0$ then t is negated; with the final value of t the result is

$$u = \frac{b}{2t}, \quad v = t.$$

4.1.2 abs(z)

```
function cabs(const z: complex): extended;
```

These functions compute the complex absolute value or modulus of z: $|z| = \sqrt{x^2 + y^2}$ using the **AMath** function hypot(x, y), which is accurate and avoids over/underflows.

4.1.3 add(x,y)

```
procedure cadd(const x,y: complex; var z: complex);
```

These procedures compute the complex sum $z = x + y$, they just use $\Re z = \Re x + \Re y$ and $\Im z = \Im x + \Im y$.

4.1.4 arg(z)

```
function carg(const z: complex): extended;
```

These functions return the principal value of the argument or phase angle of z. The angle is in $[-\pi, \pi]$ with a branch cut along the negative real axis; the result is simply evaluated as

$$\arg z = \arctan2(\Im z, \Re z).$$

4.1.5 cis(x)

```
procedure ccis(x: extended; var z: complex);
```

These procedures simply return $z = e^{ix} = \cos x + i \sin x$.

4.1.6 conj(z)

```
procedure cconj(const z: complex; var w: complex);
```

These procedures compute the complex conjugate $w = \bar{z}$ or $w = z^* = \Re z - i\Im z$.

4.1.7 div(x,y)

```
procedure cdiv(const x,y: complex; var z: complex);
```

These procedures perform the complex division $z = x/y$. Because the schoolbook definition may suffer from loss of accuracy and spurious over/underflows, the **AMath** procedure uses Smith's method, see Knuth [32, Exercise 4.2.1.16] or Numerical Recipes [13, 5.4.5]. With $x = a + ib, y = c + id$ the quotient is computed as

$$z = \frac{a+ib}{c+id} = \begin{cases} \frac{(a+b(d/c))+i(b-a(d/c))}{c+d(d/c)} & |c| \geq |d| \\ \frac{(a(c/d)+b)+i(b(c/d)-a)}{c(c/d)+d} & |c| < |d| \end{cases}$$

Note that there are still some rare cases with real/imaginary parts near `MaxExtended` or `MinExtended` where this method fails. This is especially an issue for **DAMath** using 64-bit or SSE code without intermediate extended precision. Therefore a modified version of the robust and optionally scaled division algorithm by Baudin and Smith [68] is implemented and controlled via conditional defines.[1]

[1] Currently scaled robust division is enabled as default in **DAMath** and disabled in **AMath**.

4.1.8 inv(z)

```
procedure cinv(const z: complex; var w: complex);
```

These procedures compute the complex inverse $w = 1/z$ using a call to `rdivc`$(1, z, w)$.

4.1.9 mul(x,y)

```
procedure cmul(const x,y: complex; var z: complex);
```

These procedures compute the complex product $z = x \cdot y$. The result is computed as $\Re z = \Re x \cdot \Re y - \Im x \cdot \Im y$ and $\Im z = \Re x \cdot \Im y + \Im x \cdot \Re y$.

4.1.10 neg(z)

```
procedure cneg(const z: complex; var w: complex);
```

These procedures compute the negative $w = -z$ of z using $\Re w = -\Re z$ and $\Im w = -\Im z$.

4.1.11 polar(z,r,theta)

```
procedure cpolar(const z: complex; var r,theta: extended);
```

These procedures compute the polar form $z = re^{i\theta}$ with $r = |z|$ and $\theta = \arg z$.

4.1.12 poly(z,a,n)

```
procedure cpoly(const z: complex; const a: array of complex;
                n: integer; var w: complex);
```

These procedures evaluate the polynomial function $w = a_0 + a_1 z + a_2 z^2 + ... + a_{n-1} z^{n-1}$ with complex coefficients $a_k \in \mathbb{C}$ using Horner's scheme.

4.1.13 polyr(z,a,n)

```
procedure cpolyr(const z: complex; const a: array of extended;
                 n: integer; var w: complex);
```

These procedures evaluate the polynomial function $w = a_0 + a_1 z + a_2 z^2 + ... + a_{n-1} z^{n-1}$ with real coefficients $a_k \in \mathbb{R}$ using the scheme described in Knuth [32, 4.6.4 (3)]:

$$\begin{aligned} u_1 &= a_{n-1}, & v_1 &= a_{n-2}, & r &= \Re z, \quad s = |z|^2 \\ u_j &= v_{j-1} + r u_{j-1}, & v_j &= a_{n-j-1} - s u_{j-1}, & 1 &< j < n, \quad n > 2. \end{aligned}$$

Then the result is $w = z u_{n-1} + v_{n-1}$.

4.1.14 rdivc(x,y)

```
procedure rdivc(x: extended; const y: complex; var z: complex);
```

These procedures compute the quotient $z = x/y$ with real x using a stripped-down version of the complete division routine `cdiv`.

4.1.15 set(x,y)

```
procedure cset(var z: complex; x,y: extended);
```

These procedures simply set the real and imaginary part of $z = x + iy$.

4.1.16 sgn(z)

```
function csgn(const z: complex): integer;
```

These functions return the sign of z, that is $\operatorname{sign} \Re z$ if $\Re z \neq 0$ and $\operatorname{sign} \Im z$ otherwise.

4.1.17 sqr(z)

```
procedure csqr(const z: complex; var w: complex);
```

These procedures return the square $w = u + iv = z^2 = (x + iy)^2$ of the argument z; the result is computed as $u = (x - y)(x + y),\ v = 2xy$.

4.1.18 sqrt(z)

```
procedure csqrt(const z: complex; var w: complex);
```

These procedures return the complex principal square root $w = \sqrt{z}$, with a branch cut along the negative real axis. The result w is normalised to have a real part $\Re w \geq 0$, it is computed using the internal procedure `cx_sqrt(z.re,z.im,w.re,w.im)` from 4.1.1.

4.1.19 sqrt1mz2(z)

```
procedure csqrt1mz2(const z: complex; var w: complex);
```

These procedures return the complex principal square root $w = \sqrt{1 - z^2}$. For very large z (when z^2 would overflow) we have $w = \pm iz$, where the sign is chosen to make $\Re w \geq 0$; otherwise w is computed from the definition using `csqrt`.

4.1.20 sub(x,y)

```
procedure csub(const x,y: complex; var z: complex);
```

These procedures return the complex difference $z = x - y$, they simply compute $\Re z = \Re x - \Re y$ and $\Im z = \Im x - \Im y$.

4.2 Complex transcendental functions

Please note that some of the exponential, trigonometric, or hyperbolic functions may overflow or return INFs or NaNs for inputs with real or imaginary parts of order `ln_MaxExt` or greater, this will be handled uniformly in future versions.

Unless indicated otherwise, this section assumes $z = x + iy$ and $w = u + iv$ for functions with argument z and result w, and that $\mathbb{I}$ denotes the imaginary axis.

```
procedure coshsinhmult(y,a,b: extended; var u,v: extended);
```

This internal procedure computes $u = a \cosh y,\ v = b \sinh y$ with $|a|, |b| \leq 1$ and avoids spurious overflows for some y with $|y| > \ln(\texttt{MaxExtended})$.

4.2.1 agm(x,y)

```
procedure cagm(const x,y: complex; var w: complex);
```

These procedures return the general *optimal* arithmetic-geometric mean for two arguments $w = \mathrm{AGM}(x, y) = \mathrm{AGM}(y, x)$. If $x = 0$ or $y = 0$ then $w = 0$, otherwise the result is computed as $w = x\,\mathrm{AGM}(1, y/x)$, assuming without loss of generality that $|x| \geq |y|$. If y/x would underflow, w is evaluated recursively as $w = \mathrm{AGM}(a, b)$ with a, b from a single first 'manual' AGM step

$$a = \frac{x+y}{2}, \quad b = \pm\sqrt{xy},$$

where the sign of the square root is selected to make $|a - b| \leq |a + b|$.

4.2.2 agm1(z)

```
procedure cagm1(const z: complex; var w: complex);
```

These procedures return the *optimal* arithmetic-geometric mean $w = \mathrm{AGM}(1, z)$ (with the special cases $\mathrm{AGM}(1, 0) = \mathrm{AGM}(1, -1) = 0$) using the recursion formulas

$$\begin{aligned} a_{n+1} &= \frac{1}{2}\Big(a_n + b_n\Big), \\ b_{n+1} &= \sqrt{a_n b_n}. \end{aligned}$$

Optimal means, that $|w|$ is maximal over all possible AGM sequences, which can be obtained by choosing one of the two sqrt branches, cf. Cox' *right choice* in [69]. The starting values are computed according to Pari/GP[2] and have real parts $\Re a_1, \Re b_1 \geq 0$, and b_{n+1} is always the principal root. Analogous to the real AGM 3.10.1, the sequences a_n, b_n converge quadratically to a common limit w.

4.2.3 arccos(z)

```
procedure carccos(const z: complex; var w: complex);
```

These procedures return the principal value of the complex inverse circular cosine $w = \arccos z$, with branch cuts $\mathbb{R}\backslash[-1, 1]$. The basic formula for the principal value is given by Kahan [61, Table 1] and [30, 4.23.23]

$$w = \arccos z = -2i \ln\left(\sqrt{\frac{1+z}{2}} + i\sqrt{\frac{1-z}{2}}\right).$$

If $|x|, |y| <$ `MaxExtended`$/4$, then w is computed with the derived formulas from Kahan [61, CACOS] and the fact that $\Im(\sqrt{1+z^*}) = -\Im(\sqrt{1+z})$

$$\begin{aligned} u &= 2\arctan\left(\frac{\Re(\sqrt{1-z}\,)}{\Re(\sqrt{1+z}\,)}\right), \\ v &= \operatorname{arcsinh}\Big(\Im(\sqrt{1+z^*}\sqrt{1-z}\,)\Big), \end{aligned}$$

and otherwise the limiting form is used:

$$\begin{aligned} u &= \arctan2(y, x), \\ v &= \ln\big(\mathrm{hypot}(x/2, y/2)\big) + 2\ln 2. \end{aligned}$$

[2] See the Pari/GP[62] source code and the discussion in the pari-dev thread 'Complex AGM': http://pari.math.u-bordeaux.fr/archives/pari-dev-1202/msg00045.html or http://comments.gmane.org/gmane.comp.mathematics.pari.devel/3543

4.2.4 arccosh(z)

```
procedure carccosh(const z: complex; var w: complex);
```

These procedures return the principal value of the complex inverse hyperbolic cosine $w = \operatorname{arccosh} z$, with a branch cut $x < 1$ on the real axis. The basic formula for the principal value is given by Kahan [61, Table 1] and [30, 4.37.21]

$$w = \operatorname{arccosh} z = 2 \ln \left(\sqrt{\frac{z+1}{2}} + \sqrt{\frac{z-1}{2}} \right)$$

If $|x|, |y| <$ `MaxExtended`$/4$, then w is computed with the derived formulas from Kahan [61, CACOSH] and the fact that $\Im(\sqrt{z^* - 1}) = -\Im(\sqrt{z-1})$

$$\begin{aligned} u &= \operatorname{arcsinh} \left(\Re\big(\sqrt{z^* - 1}\sqrt{z+1}\,\big) \right), \\ v &= 2 \arctan \left(\frac{\Im(\sqrt{z-1}\,)}{\Re(\sqrt{z+1}\,)} \right), \end{aligned}$$

and otherwise the limiting form is used:

$$\begin{aligned} u &= \ln \big(\operatorname{hypot}(x/2, y/2) \big) + 2 \ln 2, \\ v &= \operatorname{arctan2}(y, x). \end{aligned}$$

4.2.5 arccot(z)

```
procedure carccot(const z: complex; var w: complex);
```

These procedures return the principal value of the complex inverse circular cotangent $w = \operatorname{arccot} z$, with a branch cut $[-i, i]$ on the imaginary axis. This is the complex form of the **AMath** function `arccot` (the sign symmetric inverse circular cotangent), the result is computed using the relation [1, 4.4.8]

$$w = \operatorname{arccot} z = i \operatorname{arccoth}(iz).$$

4.2.6 arccotc(z)

```
procedure carccotc(const z: complex; var w: complex);
```

These procedures return the principal value of the complex inverse circular cotangent $w = \operatorname{arccotc} z$ with branch cuts $\mathbb{I} \backslash [-i, i]$. This is the complex form of the **AMath** function arccotc (the continuous inverse circular cotangent). The case $z \approx 0$ is handled separately, otherwise the result is computed using the relation

$$w = \operatorname{arccotc} z = \begin{cases} \frac{\pi}{2} - \arctan z, & \Re z \le 0, \\ \operatorname{arccot} z, & \Re z > 0. \end{cases}$$

4.2.7 arccoth(z)

```
procedure carccoth(const z: complex; var w: complex);
```

These procedures return the principal value of the complex inverse hyperbolic cotangent $w = \operatorname{arccoth} z$, with a branch cut $[-1, 1]$ on the real axis. The special case $\Re z = 0$ is

evaluated with the the real arctanh function, otherwise the result is computed using the relation [1, 4.6.6]

$$w = \operatorname{arccoth} z = \operatorname{arctanh}(1/z),$$

except near the branch points ± 1 where the logarithmic form from Corless et al. [63, Section 5.4] is used

$$\operatorname{arccoth} z = \frac{1}{2}\big(\ln(-1-z) - \ln(1-z)\big).$$

4.2.8 arccothc(z)

```
procedure carccothc(const z: complex; var w: complex);
```

These procedures return the principal value of the complex inverse hyperbolic cotangent $w = \operatorname{arccoth} z$, with branch cuts $\mathbb{R}\backslash[-1,1]$, this is the hyperbolic version of arccotc. The result is computed using the relation

$$w = \operatorname{arccothc} z = \begin{cases} \operatorname{arctanh} z + i\frac{\pi}{2}, & \Im z \geq 0, \\ \operatorname{arccoth} z, & \Im z < 0. \end{cases}$$

4.2.9 arccsc(z)

```
procedure carccsc(const z: complex; var w: complex);
```

These procedures return the principal value of the complex inverse circular cosecant $w = \operatorname{arccsc} z$, with a branch cut $[-1,1]$ on the real axis. The result is computed using the relation [1, 4.4.6] $w = \operatorname{arccsc} z = \arcsin(1/z)$, except near the branch points ± 1 where the roots in `carcsin` are evaluated numerically more stable as

$$\sqrt{1 \pm \frac{1}{z}} = \sqrt{\frac{z \pm 1}{z}}\,.$$

4.2.10 arccsch(z)

```
procedure carccsch(const z: complex; var w: complex);
```

These procedures return the principal value of the complex inverse hyperbolic cosecant $w = \operatorname{arccsch} z$, with a branch cut $[-i, i]$ on the imaginary axis. The result is computed using the relation [1, 4.6.4]

$$w = \operatorname{arccsch} z = i \operatorname{arccsc}(iz).$$

4.2.11 arcsec(z)

```
procedure carcsec(const z: complex; var w: complex);
```

These procedures return the principal value of the complex inverse circular secant $w = \operatorname{arcsec} z$, with a branch cut $[-1,1]$ on the real axis. The result is computed using the relation [1, 4.4.7] $w = \operatorname{arcsec} z = \arccos(1/z)$, except near the branch points ± 1 where the roots in `carccos` are evaluated numerically more stable as

$$\sqrt{1 \pm \frac{1}{z}} = \sqrt{\frac{z \pm 1}{z}}\,.$$

4.2.12 arcsech(z)

```
procedure carcsech(const z: complex; var w: complex);
```

These procedures return the principal value of the complex inverse hyperbolic secant $w = \operatorname{arcsech} z$, with with branch cuts $\mathbb{R}\backslash[0,1]$. The result is computed using the relation [1, 4.6.5] $w = \operatorname{arcsech} z = \operatorname{arccosh}(1/z)$, except near the points ± 1 where the roots in `carccosh` are evaluated as

$$\sqrt{\frac{1}{z} \pm 1} = \sqrt{\frac{1 \pm z}{z}}\,.$$

4.2.13 arcsin(z)

```
procedure carcsin(const z: complex; var w: complex);
```

These procedures return the principal value of the complex inverse circular sine $w = \arcsin z$, with branch cuts $\mathbb{R}\backslash[-1,1]$. The basic formula for the principal value is given in [30, 4.23.19]

$$w = \arcsin z = -i \ln\left(\sqrt{1-z^2} + iz\right).$$

If $|x|, |y| <$ `MaxExtended`$/4$, then w is computed with the derived formulas from Kahan [61, CASIN] and the fact that $\Im(\sqrt{1-z^*}) = -\Im(\sqrt{1-z})$

$$u = \arctan\left(\frac{\Re z}{\Re\big(\sqrt{1-z}\,\sqrt{1+z}\,\big)}\right),$$
$$v = \operatorname{arcsinh}\Big(\Im(\sqrt{1-z^*}\sqrt{1-z}\,)\Big),$$

and otherwise a limiting form is used (cf. `carccos`).

4.2.14 arcsinh(z)

```
procedure carcsinh(const z: complex; var w: complex);
```

These procedures return the principal value of the complex inverse hyperbolic sine $w = \operatorname{arcsinh} z$, with the branch cuts $\mathbb{I}\backslash[-i,i]$. The result is computed using the relation [1, 4.6.14] to the circular function: $w = \operatorname{arcsinh} z = -i \arcsin(iz)$

4.2.15 arctan(z)

```
procedure carctan(const z: complex; var w: complex);
```

These procedures return the principal value of the complex inverse circular tangent $w = \arctan z$, with the branch cuts $\mathbb{I}\backslash[-i,i]$. The result is computed using the relation [1, 4.4.22] to the hyperbolic function: $w = \arctan z = -i \operatorname{arctanh}(iz)$

4.2.16 arctanh(z)

```
procedure carctanh(const z: complex; var w: complex);
```

These procedures return the principal value of the complex inverse hyperbolic tangent $w = \operatorname{arctanh} z$, with the branch cuts $\mathbb{R}\backslash[-1,1]$. The basic formula for the principal value is from Kahan [61, Table 1]):

$$w = \operatorname{arctanh} z = \frac{1}{2}\big(\ln(1+z) - \ln(1-z)\big).$$

`carctanh` uses equivalent accuracy improved expressions derived in Kahan [61, procedure CATANH] or Boost[19, file math/complex/atanh.hpp]

$$u = \Re w = \frac{1}{4}\,\mathrm{sign}(\Re z)\,\mathrm{ln1p}\left(\frac{4x}{(1-x)^2} + y^2\right),$$
$$v = \Im w = \frac{1}{2}\,\mathrm{sign}(\Im z)\,\arctan2\left(2y, 1 - x^2 - y^2\right)$$

with $x = |\Re z|, y = |\Im z|$ if there will be no overflow or underflow, i.e. for

$$3.67 \cdot 10^{-2466} \approx 2\sqrt{\texttt{MinExtended}} < x, y < \frac{1}{2}\sqrt{\texttt{MaxExtended}} \approx 5.45 \cdot 10^{2465}.$$

Otherwise the limiting forms of u, v for large or small values of x, y are computed (see Boost [19], Kahan [61] introduces a tiny constant ρ and his pseudo-code is simpler, but in practice it is less accurate).

4.2.17 cbrt(z)

```
procedure ccbrt(const z: complex; var w: complex);
```

These procedures return the complex principal cube root $w = \sqrt[3]{z}$. The result is computed by invoking `cnroot(z,3,w)`.

4.2.18 cos(z)

```
procedure ccos(const z: complex; var w: complex);
```

These procedures return the complex circular cosine $w = \cos z$. The result is computed with the internal procedure `coshsinhmult` using the relation [1, 4.3.56]

$$w = \cos z = \cos(x + iy) = \cos x \cosh y - i \sin x \sinh y.$$

4.2.19 cosh(z)

```
procedure ccosh(const z: complex; var w: complex);
```

These procedures return the complex hyperbolic cosine $w = \cosh z$. The result is computed with the internal procedure `coshsinhmult` using the relation [1, 4.5.50]

$$w = \cosh z = \cosh(x + iy) = \cos y \cosh x + i \sin y \sinh x.$$

4.2.20 cot(z)

```
procedure ccot(const z: complex; var w: complex);
```

These procedures return the complex circular cotangent $w = \cot z$. The result is computed using the relation [1, 4.3.54]

$$w = \cot z = i \coth(iz).$$

4.2.21 coth(z)

```
procedure ccoth(const z: complex; var w: complex);
```

These procedures compute the complex hyperbolic cotangent $w = \coth z$. For $x = 0$ or $y = 0$ there are the special cases

$$\coth(x + i0) = \coth x, \quad \coth(iy) = -i \cot y,$$

otherwise Abramowitz and Stegun[1, 4.5.52] give the basic formula:

$$w = \coth z = u + iv = \coth(x + iy) = \frac{\sinh 2x - i \sin 2y}{\cosh 2x - \cos 2y}.$$

If $|x| > 23$ then $\cosh 2x - \cos 2y \approx \cosh 2x$ and $\sinh 2|x| \approx \cosh 2x \approx \frac{1}{2}e^{2|x|}$ accurate to extended precision and therefore

$$\coth z = \operatorname{sign}(x) - 2i \sin(2y) e^{-2|x|}.$$

`ccoth` avoids inaccuracies and computes the denominator for small $|x| < \frac{1}{2}$ as

$$t = \cosh 2x - \cos 2y = (\cosh 2x - 1 + 1) + (\operatorname{vers} 2y - 1) = \operatorname{coshm1}(2x) + \operatorname{vers} 2y \ ,$$

and returns the real and imaginary parts of the result as

$$u = \frac{\sinh 2x}{t} \text{ and } v = -\frac{\sin 2y}{t}.$$

4.2.22 csc(z)

```
procedure ccsc(const z: complex; var w: complex);
```

These procedures return the complex circular cosecant $w = \csc z$. The result is computed using the relation [1, 4.3.4]:

$$\csc z = (\sin z)^{-1}$$

4.2.23 csch(z)

```
procedure ccsch(const z: complex; var w: complex);
```

These procedures return the complex hyperbolic cosecant $w = \operatorname{csch} z$. The result is computed using the relation [1, 4.5.4]:

$$\operatorname{csch} z = (\sinh z)^{-1}$$

4.2.24 dilog(z)

```
procedure cdilog(const z: complex; var w: complex);
```

These procedures return the principal value of the complex dilogarithm function

$$w = \operatorname{dilog}(z) = \operatorname{Li}_2(z) = -\int_0^z \frac{\ln(1-t)}{t}\, \mathrm{d}t,$$

with the branch cut on the real axis from 1 to $+\infty$. The **AMath** implementation is based on the formulas given by L.C. Maximon [35]. For $|z| \le \frac{1}{2}$ the Taylor series [35, (3.1)]

$$\operatorname{Li}_2(z) = \sum_{n=1}^{\infty} \frac{z^n}{n^2}$$

is used, and for $|z| \geq 2$ the transformation [35, (3.2)]

$$\mathrm{Li}_2(z) = -\mathrm{Li}_2\left(\frac{1}{z}\right) - \frac{\pi^2}{6} - \frac{1}{2}\ln^2(-z).$$

In the range $\frac{1}{2} < |z| < 2$ the series [35, (4.3)] related to the Debye function

$$\mathrm{Li}_2(z) = \sum_{n=0}^{\infty} B_n \frac{(-\ln(1-z))^{n+1}}{(n+1)!} = -\frac{1}{4}\ln(1-z) + \sum_{n=0}^{\infty} B_{2n} \frac{(-\ln(1-z))^{2n+1}}{(2n+1)!}$$

is evaluated; it converges for $|\ln(1-z)| < 2\pi$ but diverges near $z = 1$. Therefore another transformation ([35, (3.3)]) is applied if $|1-z| \leq \frac{1}{2}$:

$$\mathrm{Li}_2(z) = -\mathrm{Li}_2(1-z) + \frac{\pi^2}{6} - \ln(1-z)\ln(z)$$

The function `cdilog` additionally handles the special cases $z = 1$ and z on the branch cut.

4.2.25 ellck(z)

```
procedure cellck(const k: complex; var w: complex);
```

These procedures return the complex complementary complete elliptic integral of the 1st kind $w = K'(k)$ for $k \neq 0$. The result is computed with the AGM

$$K'(k) = \frac{\pi}{2\mathrm{AGM}\,(1,k)}$$

4.2.26 elle(z)

```
procedure celle(const k: complex; var w: complex);
```

These procedures return the complex complete elliptic integral of the 2nd kind $w = E(k)$. The result is obtained with a wrapper call to `cellke`.

4.2.27 ellk(z)

```
procedure cellk(const k: complex; var w: complex);
```

These procedures return the complex complete elliptic integral of the 1st kind $w = K(k)$ with $k^2 \neq 1$. The result is computed using the relation to the AGM[3]

$$K(k) = \frac{\pi}{2\mathrm{AGM}\,(1,k')} = \frac{\pi}{2\mathrm{AGM}\left(1,\sqrt{1-k^2}\right)} = \frac{\pi}{2\mathrm{AGM1}\,(\texttt{sqrt1mz2}(k))}$$

4.2.28 ellke(z)

```
procedure cellke(const k: complex; var kk, ek: complex);
```

These procedures simultaneously compute the complex complete elliptic integrals of the 1st kind $K(k)$ and the 2nd kind $E(k)$. The AGM scheme is used with the starting values $a_0 = \sqrt{1-k^2}$, $b_0 = 1$. When the iteration is finished the results are (see [67, p.11]):

$$K(k) = \frac{\pi}{a_n + b_n}, \quad E(k) = \left(\left(\frac{a_0+b_0}{2}\right)^2 - \sum_{m=1}^{n} 2^{m-2}(a_m - b_m)^2\right) K(k)$$

[3] See [30, 19.8.5] or http://functions.wolfram.com/08.02.26.0133.01

4.2.29 exp(z)

```
procedure cexp(const z: complex; var w: complex);
```

These procedures return the complex exponential function $w = \exp z$. The result is computed using Euler's formula [1, 4.3.47]

$$w = \exp(x + iy) = e^x\big(\cos y + i \sin y\big)$$

while avoiding spurious overflows for some $x > \ln(\texttt{MaxExtended})$.

4.2.30 exp2(z)

```
procedure cexp2(const z: complex; var w: complex);
```

These procedures return $w = 2^z = \exp(z \ln 2)$.

4.2.31 exp10(z)

```
procedure cexp2(const z: complex; var w: complex);
```

These procedures return $w = 10^z = \exp(z \ln 10)$.

4.2.32 expm1(z)

```
procedure cexpm1(const z: complex; var w: complex);
```

These procedures return $w = \exp z - 1$; like the real function the accuracy is improved near $z = 0$. There are four different methods depending on $a = \max(|x|, |y|)$: If $a \geq 1$, then w is computed just from the definition and if $a < a_0 = \texttt{eps\_x}/8$ the result is just $w = z$. In the range $a_0^{\frac{1}{4}} \leq a < 1$ we have with $h = \texttt{expm1}(x) = e^x - 1$:

$$\begin{aligned}\exp(z) - 1 = e^x e^{iy} - 1 &= e^x(\cos y + i \sin y) - 1 = e^x \cos y - 1 + ie^x \sin y \\ &= e^x(1 - \operatorname{vers} y) - 1 + ie^x \sin y = e^x - 1 \ - e^x \operatorname{vers} y + ie^x \sin y \\ &= h - (1 + h) \operatorname{vers} y + i(1 + h) \sin y\end{aligned}$$

Otherwise the truncated Taylor series is used with backwards summation, where the number k of terms comes from a heuristic based on Stirling's formula for $k!$

$$\exp(z) - 1 = \left(\left(\left(\left(\frac{z}{k} + 1\right)\frac{z}{k-1} + 1\right)\frac{z}{k-2} + 1\right)\cdots + 1\right)\frac{z}{2} + z.$$

As for `cln1p` there are $z = x + iy$ with relative errors of order $\texttt{eps\_x}/x$ for $\Re w$ if $x = \frac{1}{2}y^2$. For these x, y the Taylor series at $y = 0$ starts with

$$\exp\left(\tfrac{1}{2}y^2 + iy\right) - 1 = iy + \tfrac{1}{3}iy^3 - \tfrac{1}{12}y^4 + \tfrac{1}{20}iy^5 - \tfrac{1}{45}y^6 + O(y^7),$$

i.e. the real 2nd order term vanishes and finite precision may produce the 'large' relative errors for $\Re w$. But note that in most of these cases the complete relative error $|(f - w)/f|$ with $f = e^z - 1$ remains small because $|\Im f| \gg |\Re f|$.

4.2.33 gamma(z)

```
procedure cgamma(const z: complex; var w: complex);
```

These procedures calculate the complex Gamma function $w = \Gamma(z)$ using `clngamma`

$$w = \Gamma(z) = \exp(\text{lnGamma}(z)).$$

Note that errors of lnGamma are amplified or $\Gamma(z)$ may overflow, and therefore lnGamma(z) should be used instead of $\Gamma(z)$ as long as possible in actual computations.

4.2.34 ln(z)

```
procedure cln(const z: complex; var w: complex);
```

These procedures return the principal value of the complex natural logarithm $w = \ln z$, with a branch cut along the negative real axis using the basic formula from [1, 4.1.2]:

$$w = \ln z = \ln|z| + i \arg z$$

A modification of Kahan's [61] procedure CLOGS is used to improve the accuracy near $z = 1$: With $x = \max(|\Re z|, |\Im z|)$ and $y = \min(|\Re z|, |\Im z|)$ the real part of w is computed as

$$\Re w = \tfrac{1}{2}\,\text{ln1p}\left((x-1)(x+1) + y^2\right).$$

4.2.35 ln1p(z)

```
procedure cln1p(const z: complex; var w: complex);
```

These procedures return the principal branch of $w = \ln(1+z)$, with a branch cut $x < -1$ along the negative real axis. Like the real function the accuracy is improved near $z = 0$. If $\max(|x|, |y|) > 0.75$ or $x < -0.5$, the real part is computed analogous to the standard logarithm function

$$\Re w = \ln(|1+z|) = \ln(\text{hypot}(1+x, y))$$

and otherwise we have with $a = |z|^2$

$$\Re w = \ln(|1+z|) = \tfrac{1}{2}\ln((1+x)^2 + y^2) = \tfrac{1}{2}\ln(1 + 2x + x^2 + y^2) = \tfrac{1}{2}\text{ln1p}(2x + a).$$

Note that there can be inaccuracies for the real part $\Re w$ if $x < 0$ and $2x + a \approx 0$ or $x = -\frac{1}{2}y^2$. In this case the Taylor series at $y = 0$ is

$$\ln(1 - \tfrac{1}{2}y^2 + iy) = iy + \tfrac{1}{6}iy^3 + \tfrac{1}{4}y^4 - \tfrac{1}{20}iy^5 - \tfrac{1}{56}iy^7 - \tfrac{1}{64}y^8 + O(y^9).$$

As can be seen, the quadratic and 6th order real terms are missing and the first non-zero real term is $\frac{1}{4}y^4$; the relative error of $\Re w$ can be of order `eps_x`$/x$, e.g. for $y =$ `1E-5`, $x =$ `-0.5E-10` it is `1.442E-9`. But note that in most of these cases the complete relative error $|(f - w)/f|$ with $f = \ln(1+z)$ remains small because $|\Im f| \gg |\Re f|$. When $x \geq 0$ there is no cancellation and the error of $\Re w$ is small.

In all cases the imaginary part of the result is computed with the standard formula

$$\Im w = \arg(1+z) = \arctan2(y, 1+x).$$

4.2.36 lnGamma(z)

```
procedure clngamma(const z: complex; var w: complex);
```

These procedures return $w = \text{lnGamma}(z)$, the principal branch of the log-Gamma function. lnGamma is analytic on the complex plane without the non-positive integers and is defined as the analytic continuation of $\ln(\Gamma(z))$ from the real positive axis, the negative real axis is the branch cut. The result is computed for $\Re z > 0$ with the logarithmic version of the Lanczos formula (cf. Press et al. [13, 6.1.5])[4]

$$\Gamma(z+1) = \sqrt{2\pi}\,\left(z+g+\tfrac{1}{2}\right)^{z+\frac{1}{2}}\,e^{-z-g-\frac{1}{2}}\left(c_0 + \sum_{k=1}^{N}\frac{c_k}{z+k} + \epsilon(z)\right),$$

except near the roots $z = 1$ or $z = 2$ of $\ln\Gamma(z)$ where the absolute error of the Lanczos formula is inadequate; therefore near $z = 0, 1, 2$ the recurrence formula $\Gamma(z+1) = z\Gamma(z)$ and the power series from Abramowitz and Stegun [1, 6.1.33] are used

$$\ln\Gamma(1+z) = -\ln(1+z) + z(1-\gamma) + \sum_{n=2}^{\infty}(-1)^n\big(\zeta(n)-1\big)\frac{z^n}{n}\cdot$$

For $\Re z \le 0$ the logarithmic version of the reflection formula $\Gamma(z)\Gamma(1-z) = \pi\csc\pi z$ is applied with the addition of a certain multiple of 2π to $\Im w$ for the correct branch[5].

4.2.37 log10(z)

```
procedure clog10(const z: complex; var w: complex);
```

These procedures return the principal branch of the base 10 logarithm of z, the result is simply $w = \ln z/\ln 10$.

4.2.38 logbase(b,z)

```
procedure clogbase(const b,z: complex; var w: complex);
```

These procedures return the principal branch $w = \ln z/\ln b$ of the base b logarithm of z using the principal value of $\ln b$.

4.2.39 nroot(z,n)

```
procedure cnroot(const z: complex; n: integer; var w: complex);
```

These procedures return the complex principal n^{th} root $w = z^{\frac{1}{n}} = \sqrt[n]{z}$. The result is computed using the standard formula [1, 3.7.28]

$$z^{\frac{1}{n}} = \sqrt[n]{r}\exp\left(\frac{i\theta}{n}\right) = \sqrt[n]{r}\left(\cos\frac{\theta}{n} + i\sin\frac{\theta}{n}\right)$$

with $r = |z|$ and $\theta = \arg z$.

[4] The actual values for g, N, c_k are taken from Paul Godfrey's list of coefficients, available as http://my.fit.edu/~gabdo/gammacoeff.txt

[5] Cf. http://functions.wolfram.com/06.11.16.0002.01

4.2.40 nroot1(z,n)

```
procedure cnroot1(n: integer; var z: complex);
```

These procedures return the principal n^{th} root of unity $z = e^{\frac{2\pi i}{n}}$.

4.2.41 pow(z,a)

```
procedure cpow(const z,a: complex; var w: complex);
```

These procedures return the principal value of the complex power function $w = z^a$, with the branch cut (for z) along the negative real axis. The result is computed using the standard formula and two special cases (see [30, 4.2(iv)] and Kahan [61, Table 1]):

$$w = z^a = \exp(a \ln z),\ z^0 = 1,\ 0^a = 0 \ \text{ if } \ \Re a > 0\cdot$$

Note that general complex exponential function $\texttt{pow}(a, z) = a^z = \exp(z \ln a)$ as a function of z is **not** multi-valued after choosing a value for $\ln a$.

4.2.42 powx(z,x)

```
procedure cpowx(const z: complex; x: extended; var w: complex);
```

These procedures return the principal value of the complex power function for real exponents (branch cut for z along the negative real axis):

$$w = z^x = |z|^x e^{ix \arg z}$$

4.2.43 psi(z)

```
procedure cpsi(const z: complex; var w: complex);
```

These procedures compute the complex digamma function $w = \psi(z)$ for $z \neq 0, -1, -2, \ldots$ similar to the real case. If $\Re z \geq x_a$ the asymptotic expansion from Abramowitz and Stegun [1, 6.3.18] is used[6]

$$w = \psi(z) \sim \ln z - \frac{1}{2z} - \sum_{n=1}^{\infty} \frac{B_{2n}}{2nz^{2n}},$$

and for $0 \leq \Re z < x_a$ the recurrence formula [1, 6.3.5] is applied to make $\Re z \geq x_a$

$$\psi(z+1) = \psi(z) + \frac{1}{z}\cdot$$

Arguments with $\Re z < 0$ are transformed to $\Re z > 0$ with the reflection formula [1, 6.3.7]

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

where the special case $z = -m + iy,\ y \neq 0$ with negative integer real part is evaluated as

$$\cot\big(\pi(-m+iy)\big) = \cot(i\pi y) = -i \coth(\pi y).$$

[6] In **AMath** $x_a = 13$ and the sum is for $n = 1 \ldots 8$, **DAMath** uses $x_a = 12, n = 1 \ldots 7$.

4.2.44 sec(z)

```
procedure csec(const z: complex; var w: complex);
```

These procedures return the complex circular secant $w = \sec z$. The result is computed using the relation [1, 4.3.5]: $\sec z = (\cos z)^{-1}$.

4.2.45 sech(z)

```
procedure csech(const z: complex; var w: complex);
```

These procedures return the complex hyperbolic secant $w = \operatorname{sech} z$. The result is computed using the relation [1, 4.5.5]: $\operatorname{sech} z = (\cosh z)^{-1}$.

4.2.46 sin(z)

```
procedure csin(const z: complex; var w: complex);
```

These procedures return the complex circular sine $w = \sin z$. The result is computed with the internal procedure `coshsinhmult` using the relation [1, 4.3.55]

$$w = \sin z = \sin(x + iy) = \sin x \cosh y + i \cos x \sinh y.$$

4.2.47 sinh(z)

```
procedure csinh(const z: complex; var w: complex);
```

These procedures return the complex hyperbolic sine $w = \sinh z$. The result is computed with the internal procedure `coshsinhmult` using the relation [1, 4.5.49]

$$w = \sinh z = \sinh(x + iy) = \cos y \sinh x + i \sin y \cosh x.$$

4.2.48 surd(z,n)

```
procedure csurd(const z: complex; n: integer; var w: complex);
```

These procedures return the complex n^{th} root $w = z^{\frac{1}{n}}$ with $\arg(w)$ closest to $\arg(z)$, e.g. $\operatorname{surd}(-8, 3) = -2$ or $\operatorname{surd}(i, 5) = i$ compared to the `cnroot` results $\sqrt[3]{-8} = 1 + i\sqrt{3}$ and $\sqrt[5]{i} = \cos\frac{\pi}{10} + i \sin\frac{\pi}{10}$.

4.2.49 tan(z)

```
procedure ctan(const z: complex; var w: complex);
```

These procedures return the complex circular tangent $w = \tan z$. The result is computed using the relation [1, 4.3.51]

$$w = \tan z = -i \tanh(iz).$$

4.2.50 tanh(z)

```
procedure ctanh(const z: complex; var w: complex);
```

These procedures compute the complex hyperbolic tangent $w = \tanh z$. For $x = 0$ or $y = 0$ there are the special cases

$$\tanh(x + i0) = \tanh x, \quad \tanh(iy) = i \tan y,$$

otherwise Abramowitz and Stegun[1, 4.5.51] give the basic formula:

$$w = \tanh z = u + iv = \tanh(x + iy) = \frac{\sinh 2x + i \sin 2y}{\cosh 2x + \cos 2y}.$$

If $|x| > 23$ then $\cosh 2x + \cos 2y \approx \cosh 2x$ and $\sinh 2|x| \approx \cosh 2x \approx \frac{1}{2}e^{2|x|}$ accurate to extended precision and therefore

$$\tanh z = \operatorname{sign}(x) + 2i \sin(2y) e^{-2|x|},$$

otherwise `ctanh` avoids inaccuracies with $h = e^{2x} - 1 = \operatorname{expm1}(2x)$ and computes [7]

$$\begin{aligned}
\cosh 2x + \cos 2y &= \frac{e^{2x} + e^{-2x}}{2} + \left(\cos^2 y - \sin^2 y\right) \\
&= 1 + \frac{\left(e^{2x} - 1\right)^2}{2e^{2x}} + \cos^2 y - \sin^2 y \\
&= \frac{h^2}{2(h+1)} + 2\cos^2 y = \frac{h^2 + 4(h+1)\cos^2 y}{2(h+1)}
\end{aligned}$$

$$\sinh 2x = \frac{e^{2x} - e^{-2x}}{2} = \frac{1}{2}\left(e^{2x} - 1 + \frac{e^{2x} - 1}{e^{2x}}\right) = \frac{1}{2}\left(h + \frac{h}{h+1}\right) = \frac{h^2 + 2h}{2(h+1)}$$

and returns the real and imaginary parts of the result as

$$u = \frac{\sinh 2x}{\cosh 2x + \cos 2y} = \frac{h^2 + 2h}{h^2 + 4(h+1)\cos^2 y},$$

$$v = \frac{\sin 2y}{\cosh 2x + \cos 2y} = \frac{2 \cos y \sin y}{\cosh 2x + \cos 2y} = \frac{4(h+1)\cos y \sin y}{h^2 + 4(h+1)\cos^2 y}.$$

[7] Other methods to accurately compute $\cosh 2x + \cos 2y$ without cancellation are given by Kahan [61, procedure CTANH] or the identity $\cosh 2x + \cos 2y = 2\left(\sinh^2 x + \cos^2 y\right)$, which is used in the **MPArith** complex `mpc_tanh` function.

Appendix A

Licenses

This chapter contains the licenses or information of 3rd party libraries used in **AMath.**

The software packages **AMath/DAMath** described in this manual and **MPArith** are distributed under the zlib-license:

A.1 Boost

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 Cephes

```
Cephes Mathematical Library V2.8, (June 2000)
Copyright 1984, 1991, 1998 by Stephen L. Moshier
Author        : Stephen L. Moshier
Original-site : http://www.moshier.net/#Cephes
Alternate-site: http://netlib.org/cephes
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/[1]

[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

[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/OgRuOi05.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. Available from http://www.digizeitschriften.de/en/dms/toc/?PPN=PPN362160546_0007

[11] R. Bulirsch, Numerical Calculation of Elliptic Integrals and Elliptic Functions, part III. Numerische Mathematik 13, 305-315, 1969. Available from http://www.digizeitschriften.de/en/dms/toc/?PPN=PPN362160546_0013

[12] B.C. Carlson, Computing Elliptic Integrals by Duplication. Numerische Mathematik 33, 1-16, 1979. Available from http://www.digizeitschriften.de/en/dms/toc/?PPN=PPN362160546_00033

[13] W.H. Press et al., Numerical Recipes in C, 2nd ed., Cambridge, 1992. http://www.nrbook.com/a/bookcpdf.html

[1] The bibliography contains the general references for the complete **AMath** package, therefore there are some items which are unrelated the special functions units.

[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. Available from http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=AD0639052

[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 as http://oai.cwi.nl/oai/asset/10080/10080A.pdf

[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.[2]

[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 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), 210-224. Available from http://dx.doi.org/10.1016/j.jmaa.2004.05.040

[42] Y.L. Luke, Algorithms for the Computation of Mathematical Functions, Academic Press, 1977

[43] J. Pearson, Computation of Hypergeometric Functions, Master's thesis, University of Oxford, 2009. Available as http://people.maths.ox.ac.uk/porterm/research/pearson_final.pdf

[2] Richard E. Crandall died December 20, 2012 and his former web page at Reed College is now closed. An archived copy of his note is http://wolfgang-ehrhardt.de/Polylog.pdf (retrieved 2010-03-15).

[44] K.E. Muller, Computing the confluent hypergeometric function M(a,b,x), Numerische Mathematik 90, 179-196, 2001; doi:10.1007/s002110100285

[45] N.N. Lebedev, Special Functions and Their Applications, Dover, New York, 1972

[46] S. Graillat, Ph. Langlois, N. Louvet, Compensated Horner Scheme, Research Report No RR2005-04, 2005, Université de Perpignan. Available from http://www-pequan.lip6.fr/~graillat/papers/rr2005-04.pdf or http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.81.2979

[47] M. Goano, Algorithm 745: Computation of the Complete and Incomplete Fermi-Dirac Integral. ACM TOMS, Vol.21, No.3, 1995, pp.221-232. Fortran source available from http://netlib.org/toms/745

[48] D. Dijkstra, A Continued Fraction Expansion for a Generalization of Dawson's Integral, Mathematics of Computation, Vol.31, 1977, pp.503-510. Available as http://doc.utwente.nl/75001/1/Dijkstra77continued.pdf

[49] W. Gautschi, Evaluation of the Repeated Integrals of the Coerror Function, ACM TOMS, Vol.3, No.3, 1977, pp.240-252. Fortran source available from http://netlib.org/toms/521

[50] A. Erdélyi et al., Higher Transcendental Functions Vol. I-III, California Institute of Technology - Bateman Manuscript Project, 1953-1955 McGraw-Hill Inc., reprinted by Krieger Inc. 1981. Available via http://en.wikipedia.org/wiki/Bateman_Manuscript_Project

[51] 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://oai.cwi.nl/oai/asset/10710/10710A.pdf

[52] N.M. Temme, On the Numerical Evaluation of the Modified Bessel Function of the Third Kind, 2nd edition. Preprint, available as http://oai.cwi.nl/oai/asset/7885/7885A.pdf

[53] F.G. Tricomi, Fonctions hypergéométriques confluentes. Mémorial des sciences mathématiques, 140 (1960), p. 1-86. Available from http://www.numdam.org/item?id=MSM_1960__140__1_0

[54] R.C. Forrey, Computing the hypergeometric function, J. Comp. Phys. 137, 79-100 (1997). PDF document and Fortran code available from http://physics.bk.psu.edu/pub/papers/hyper.pdf, and http://physics.bk.psu.edu/codes.html.

[55] N.M. Temme, The Numerical Computation of the Confluent Hypergeometric Function U(a,b,z), Numerische Mathematik, 41, p.63-82, 1983. Available from http://www.digizeitschriften.de/en/dms/toc/?PPN=PPN362160546_0041 or http://oai.cwi.nl/oai/asset/10717/10717A.pdf.

[56] E.J. Weniger, Nonlinear sequence transformations for the acceleration of convergence and the summation of divergent series, 1989, Comput. Phys. Rep. 10, pp.189-371; available as http://arxiv.org/pdf/math/0306302v1.pdf

[57] T. Fessler, W.F. Ford, D.A. Smith, HURRY: An Acceleration Algorithm for Scalar Sequences and Series, ACM TOMS, Vol.9, No.3, 1983, pp.346-354. Fortran source available from http://netlib.org/toms/602

[58] W. Gautschi, On mean convergence of extended Lagrange interpolation, J. Comput. Appl. Math. 43, 1992, pp.19-35. Available as http://www.cs.purdue.edu/homes/wxg/selected_works/section_03/132.pdf

[59] J.C. Mason, Chebyshev polynomials of second, third and fourth kinds. J. Comput. Appl. Math. 49, 1993, 169-178; doi:10.1016/0377-0427(93)90148-5

[60] W. Kahan, To Solve a Real Cubic Equation (Lecture Notes for a Numerical Analysis Course), 1986. Available as
http://www.dtic.mil/dtic/tr/fulltext/u2/a206859.pdf

[61] W. Kahan, Branch Cuts for Complex Elementary Functions, or Much Ado About Nothing's Sign Bit, in: The State of Art in Numerical Analysis, ed. by A. Iserles and M.J.D. Powell, 1987, pp. 165-211. Available as
http://people.freebsd.org/~das/kahan86branch.pdf

[62] PARI/GP: Open Source Number Theory-oriented Computer Algebra System, available from http://pari.math.u-bordeaux.fr/

[63] R.M. Corless, J.H. Davenport, D.J. Jeffrey, and S.M. Watt, "According to Abramowitz and Stegun" or arccoth needn't be uncouth. SIGSAM BULLETIN: Communications on Computer Algebra, 34(2), June 2000. Available as
http://www.sigsam.org/bulletin/articles/132/paper12.pdf
or from the Ontario Research Centre for Computer Algebra
http://www.orcca.on.ca/TechReports/2000/TR-00-17.html

[64] T.J. Dekker, A Floating-point technique for extending the available precision. Numerische Mathematik, 18, 224-242, 1971. Available from
http://www.digizeitschriften.de/en/dms/toc/?PPN=PPN362160546_0018

[65] S. Linnainmaa, Software for Doubled-Precision Floating-Point Computations. ACM TOMS 7 (1981), pp. 272-283. doi: 10.1145/355958.355960

[66] H. Rinne, Taschenbuch der Statistik, 4.Auflage, Harri Deutsch, Frankfurt 2008

[67] B.C. Carlson, Numerical computation of real or complex elliptic integrals, 1994,
http://arxiv.org/pdf/math/9409227v1.pdf

[68] M. Baudin, R.L. Smith, A Robust Complex Division in Scilab, 2013,
http://arxiv.org/pdf/1210.4539.pdf

[69] D.A. Cox, The arithmetic-geometric mean of Gauss, Enseign. Math. **30** (1984), 275–330; available from ETH Zürich: http://dx.doi.org/10.5169/seals-53831