Computation of special functions in mpmath

Fredrik Johansson
fredrik.johansson@gmail.com

Sage Days 23, Leiden, July 2010

http://mpmath.org

mpmath - quick overview

- Arbitrary-precision binary floating-point numbers

mpmath - quick overview

- Arbitrary-precision binary floating-point numbers
- Complex numbers supported essentially everywhere

mpmath - quick overview

- Arbitrary-precision binary floating-point numbers
- Complex numbers supported essentially everywhere
- (Lots of) transcendental functions

mpmath - quick overview

- Arbitrary-precision binary floating-point numbers
- Complex numbers supported essentially everywhere
- (Lots of) transcendental functions
- Partial support for interval and machine (double precision) arithmetic

mpmath - quick overview

- Arbitrary-precision binary floating-point numbers
- Complex numbers supported essentially everywhere
- (Lots of) transcendental functions
- Partial support for interval and machine (double precision) arithmetic
- Standard Sage package

mpmath - quick overview

- Arbitrary-precision binary floating-point numbers
- Complex numbers supported essentially everywhere
- (Lots of) transcendental functions
- Partial support for interval and machine (double precision) arithmetic
- Standard Sage package
- Extensive documentation (see website)

Credits

- Started 2007 as a `math`/`cmath` replacement for SymPy
- My work on mpmath/SymPy was supported by Google Summer of Code (2008)
- My work on special functions in mpmath/Sage is supported by William Stein's NSF grant (2009, 2010)

Contributors:

- Vinzent Steinberg (linear algebra, root-finding)
- Mario Pernici (some elementary and special functions)
- Juan Arias de Reyna (Riemann-Siegel expansion, zeta zeros)
- Case Van Horsen (GMPY support)

Goal

Arbitrary-precision numerical evaluation of any (reasonably nice) formula over the complex numbers, allowing operations of calculus (integrals, derivatives, infinite series, products, sequence limits, ODEs, equation roots, ...).

Example: $-2i \int_{-i}^{+i} \left(\sum_{n=0}^{\infty} t^n\right) dt = \pi$

```
>>> from mpmath import *
>>> mp.dps = 25; mp.pretty = True
>>> -2*j*quad(lambda t:
...   nsum(lambda n: t**n, [0,inf]), [-j,j])
(3.141592653589793238462643 + 0.0j)
>>> +pi
3.141592653589793238462643
```

Example - gamma function

```
>>> gamma(0.5); sqrt(pi)
1.772453850905516027298167
1.772453850905516027298167

>>> gamma(25)
620448401733239439360000.0

>>> gamma(2+3j)
(-0.08239527266561188367387031 +
0.09177428743525931459566742j)
```

Example - gamma function, continued

```
>>> gamma(1e30j)
(-2.37782722900351491454083 8e-6821881769209206873079477801 70 +
2.39397400221776207440413 3e-6821881769209206873079477801 70j)

>>> fp.gamma(0.5)
1.772453850905516 3

>>> iv.dps = 20; iv.pretty = True
>>> iv.gamma(0.5)
[1.77245385090551602729 7386, 1.7724538509055160272990 8]

>>> iv.gamma(2+3j)
([-0.082395272665611883673 93502, -0.08239527266561188367 382914] +
[0.091774287435259314595 60232, 0.0917742874352593145957 082]*j)
```

Implementation layers

Special functions, numerical calculus (Python)

Floating-point arithmetic, utility functions, elementary functions (Python, Cython)

Integer arithmetic (MPIR, Python)

General tools (operations of calculus)

Numerical integration (quadrature)

$$\pi = \int_{-1}^{1} 2\sqrt{1-x^2}\,dx$$

```
>>> quad(lambda x: 2*sqrt(1-x**2), [-1,1])
3.141592653589793238462643
```

$$\pi = \left(\int_{-\infty}^{\infty} e^{-x^2}\,dx\right)^2$$

```
>>> quad(lambda x: exp(-x**2), [-inf,inf])**2
3.141592653589793238462643
```

$$\pi = \int_{-\infty}^{\infty} \frac{e \cos x}{1+x^2}\,dx$$

```
>>> quadosc(lambda x: e*cos(x)/(1+x**2), [-inf,inf], omega=1)
3.141592653589793238462643
```

Methods: doubly exponential (tanh-sinh) quadrature, Gaussian quadrature, summation

Infinite series, products

$$\frac{\pi}{4} = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

```
>>> nsum(lambda n: (-1)**n/(2*n+1), [0,inf]); pi/4
0.785398163397448309615661
0.785398163397448309615661
```

$$\sum_{k=1}^{\infty} k^{-s} = \zeta(s)$$

```
>>> nsum(lambda k: 1/k**2.5, [1,inf], method='e')
1.34148725725091717975677
>>> zeta(2.5)
1.34148725725091717975677
```

Methods: direct summation, convergence acceleration (iterated Richardson extrapolation, Shanks transformation), Euler-Maclaurin summation

Limits

$$\lim_{n\to\infty} \left(1 + \frac{z}{n}\right)^n = e^z$$

```
>>> limit(lambda n: (1+(3+4j)/n)**n, inf)
(-13.128783081462158080327555145 4 - 15.200784463067954562203481023 3j)
>>> exp(3+4j)
(-13.1287830814621580803275551454 - 15.2007844630679545622034810233j)
```

$$\lim_{s\to 1} \zeta(s) - \frac{1}{s-1} = \gamma$$

```
>>> limit(lambda s: zeta(s) - 1/(s-1), 1)
0.577215664901532860606512090082
>>> +euler
0.577215664901532860606512090082
```

Methods: convergence acceleration, direct evaluation

Derivatives, Taylor series

$$\frac{d^n}{dz^n} \sin(z)$$

```
>>> diff(sin, 1, 23); diff(sin, 1, 24)
-0.5403023058681397174009366
0.8414709848078965066525023
```

$$\prod_{k=1}^{\infty} (1 - z^k)^{-1} = \sum_{n=0}^{\infty} p(n) z^n$$

```
>>> f = lambda z: 1/nprod(lambda k: (1-z**k), [1,inf])
>>> taylor(f, 0, 10)
[1.0, 1.0, 2.0, 3.0, 5.0, 7.0, 11.0, 15.0, 22.0, 30.0, 42.0]
```

Methods: finite differences (high precision), Cauchy's formula (numerical integration)

ODEs

$$f(1) = \frac{\pi}{4}; \quad f'(x) = \frac{1}{1+x^2}, f(0) = 0$$

```
>>> f = odefun(lambda x,y: 1/(1+x**2), 0, 0)
>>> f(1)*4
3.14159265358979323846264338
```

$$f(x) = \cos(x); \qquad f'' + f = 0, \quad f(0) = 1, f'(0) = 0$$

```
>>> f = odefun(lambda x,y: [-y[1], y[0]], 0, [1,0])
>>> f(3)
[-0.989992496600445457271572, 0.141120008059867222100744]
>>> [cos(3), sin(3)]
[-0.989992496600445457271572, 0.141120008059867222100744]
```

Methods: Taylor series

Root-finding

Polynomials:

```
>>> for r in polyroots([1,2,4,0,0,1]): print r
...
-0.6859241973588571418291518
(0.3437806321323095437741409 + 0.4853593293545055528378462j)
(0.3437806321323095437741409 - 0.4853593293545055528378462j)
(-1.000818533452880972859565 + 1.766209269481339399167955j)
(-1.000818533452880972859565 - 1.766209269481339399167955j)
```

Arbitrary functions or vector systems:

```
>>> findroot(sin, 3)
3.141592653589793238462643
>>> findroot(zeta, 0.5+14j)
(0.5 + 14.13472514173469379045725j)
>>> findroot(lambda x,y: [cos(x)+y, sin(y)+x], [1,1])
[ 0.6948196907307875655784201]
[-0.7681691567367959774620862]
```

Methods: Newton's method (and variations)

Special functions

Special functions are not only convenient notation for humans. They can typically be evaluated much more efficiently than "general" functions.

```
>>> mp.dps = 100
>>> z = mpf(3.7)
>>> timing(gamma, z)
9.5431123230582668e-05
>>> timing(quad, lambda t: t**(z-1)*exp(-t), [0,inf])
0.4520867875665715
>>> 0.45 / 9.54e-5
4716.9811320754716
```

Function categories – methods of computation

Elementary functions
Taylor series, Newton's method

Hypergeometric functions
Hypergeometric series, asymptotic expansions, extrapolation, numerical integration

Gamma, psi, zeta, polylogarithms, . . .
Euler-Maclaurin summation, asymptotic series, functional equations

Elliptic functions and integrals
Hypergeometric series, theta functions, argument transformations

General methods: Argument reduction, reduction to more general functions, reduction to more specialized functions

Elementary functions

Taylor series are used for exp, cos, sin, log, atan.

All elementary functions have algebraic argument reduction formulas of type $z \to z/2$, so $O(\sqrt{n}M(n))$ complexity is possible.

Complexity can be reduced further using Smith's "concurrent summation" trick (implemented); binary splitting (not implemented, except for constants).

AGM and Newton's method also used in some places.

Essential tricks for performance in Python: fixed-point arithmetic, caching. Future: Cython implementation (see also: fastfunlib).

Most special functions are hypergeometric

- Elementary functions
- Gauss' hypergeometric function
- Kummer's hypergeometric functions
- Whittaker functions
- Meijer G-function
- Error functions
- Complete elliptic integrals
- Incomplete gamma function
- Incomplete beta function
- Exponential integrals
- Logarithmic integral
- Trigonometric integrals
- Hyperbolic integrals
- Fresnel integrals
- Legendre functions
- Toroidal functions
- Conical functions
- Parabolic cylinder functions
- Chebyshev polynomials
- Jacobi polynomials
- Laguerre polynomials
- Hermite polynomials
- Gegenbauer polynomials
- Spherical harmonics
- Clebsch-Gordan coefficients
- Bessel functions
- Spherical Bessel functions
- Hankel functions
- Struve functions
- Kelvin functions
- Airy functions
- Lommel functions
- Coulomb wave functions
- ...

Hypergeometric functions

These functions all solve the *generalized hypergeometric differential equation* with specific parameters and initial conditions (normalizations).

Standardized solution (*generalized hypergeometric series*)

$$ {}_pF_q(a_1, \ldots, a_p; b_1, \ldots, b_q; z) = \sum_{k=0}^{\infty} \frac{(a_1)_k \cdots (a_p)_k}{(b_1)_k \cdots (b_q)_k} \frac{z^k}{k!} $$

where $(a)_k = a(a+1)\cdots(a+k-1)$.
Any hypergeometric function is a linear combination of standard functions:

$$ f(\mathbf{d}, z) = \sum_{k=1}^{K} \mathbf{w}_k \frac{\Gamma(\mathbf{a}_k)}{\Gamma(\mathbf{b}_k)} \, {}_{p_k}F_{q_k}\left(\boldsymbol{\alpha}_k; \boldsymbol{\beta}_k; z_k\right) $$

General framework for hypergeometric functions

Idea: write a routine that evaluates a linear combination (specified quasi-symbolically) of ${}_pF_q$'s, *automatically handling all numerical difficulties*.

General framework for hypergeometric functions

Idea: write a routine that evaluates a linear combination (specified quasi-symbolically) of ${}_pF_q$'s, *automatically handling all numerical difficulties*.

- Accurate evaluation of ${}_pF_q$ (handling slow convergence, cancellation, ...)

General framework for hypergeometric functions

Idea: write a routine that evaluates a linear combination (specified quasi-symbolically) of ${}_pF_q$'s, *automatically handling all numerical difficulties*.

- Accurate evaluation of ${}_pF_q$ (handling slow convergence, cancellation, ...)
- The formula as written may be undefined (gamma function poles, ...). But the singularity is removable and we want the limit value.

General framework for hypergeometric functions

Idea: write a routine that evaluates a linear combination (specified quasi-symbolically) of ${}_pF_q$'s, *automatically handling all numerical difficulties*.

- Accurate evaluation of ${}_pF_q$ (handling slow convergence, cancellation, ...)
- The formula as written may be undefined (gamma function poles, ...). But the singularity is removable and we want the limit value.
- Handle catastrophic cancellation of terms in linear combination.

Example: Bessel function of the first kind

Standardized representation:

$$J_n(z) = \sum_{k=0}^{\infty} \frac{(-1)^k z^{2k+n}}{2^{2k+n} k!(n+k)!} = \frac{1}{\Gamma(n+1)} \left(\frac{z}{2}\right)^n {}_0F_1\left(n+1, -\frac{z^2}{4}\right)$$

```
def J(n, z, **kwargs):
    n, z = convert(n), convert(z)
    return hypercomb(lambda n:
        [([z/2],[n],[],[n+1],[],[n+1],-(z/2)**2)], [n], **kwargs)
```

Automatically handles huge arguments, negative integer n:

```
>>> J(5, 1000000)
-0.000725964384245328505237577997043384891484649290032829
>>> J(-5, 1000000)
0.000725964384245328505237577997043384891484649290032829
>>> besselj(5, 1000000)
-0.000725964384245328505237577997043384891484649290032829
```

Computation of ${}_pF_q$

Can be viewed as a generalized power or exponential function of z, modified by the parameters a_k, b_k.

First case: ${}_1F_0$, ${}_2F_1$, ${}_3F_2, \ldots$. Special cases: natural logarithm, inverse trigonometric functions, various orthogonal polynomials, complete elliptic integrals, polygamma functions.

Computation of ${}_pF_q$

Can be viewed as a generalized power or exponential function of z, modified by the parameters a_k, b_k.

First case: ${}_1F_0$, ${}_2F_1$, ${}_3F_2, \ldots$. Special cases: natural logarithm, inverse trigonometric functions, various orthogonal polynomials, complete elliptic integrals, polygamma functions.

- The power series converges for $|z| < 1$, with a singularity at $z = 1$. Direct summation is efficient for, say $|z| < 0.9$.

Computation of ${}_pF_q$

Can be viewed as a generalized power or exponential function of z, modified by the parameters a_k, b_k.

First case: ${}_1F_0$, ${}_2F_1$, ${}_3F_2, \ldots$. Special cases: natural logarithm, inverse trigonometric functions, various orthogonal polynomials, complete elliptic integrals, polygamma functions.

- The power series converges for $|z| < 1$, with a singularity at $z = 1$. Direct summation is efficient for, say $|z| < 0.9$.
- For $|z| > 1$, we can transform ${}_pF_q(\ldots, z)$ into a linear combination of p terms ${}_pF_q(\ldots, 1/z)$. The transformed parameters are often singular.

Computation of ${}_pF_q$

Can be viewed as a generalized power or exponential function of z, modified by the parameters a_k, b_k.

First case: ${}_1F_0$, ${}_2F_1$, ${}_3F_2, \ldots$. Special cases: natural logarithm, inverse trigonometric functions, various orthogonal polynomials, complete elliptic integrals, polygamma functions.

- The power series converges for $|z| < 1$, with a singularity at $z = 1$. Direct summation is efficient for, say $|z| < 0.9$.
- For $|z| > 1$, we can transform ${}_pF_q(\ldots, z)$ into a linear combination of p terms ${}_pF_q(\ldots, 1/z)$. The transformed parameters are often singular.
- Hard case: z close to the unit circle. Methods: argument transformations, convergence acceleration, specialized formulas.

Computation of ${}_pF_q$ - confluent case

Second case: ${}_0F_0 = \exp$, ${}_0F_1$, ${}_1F_1$, ${}_1F_2$, ${}_2F_2, \ldots$.

Special cases: exponential and trigonometric functions, Bessel functions, error function, exponential integrals, incomplete gamma functions.

The power series has infinite radius of convergence. However, convergence is *very slow* for large $|z|$ (say, $|z| > 100$).

For large $|z|$, we need to use *asymptotic series*, which are known for ${}_pF_q$ of all orders. But for linear combinations, we may also need to use expansions specialized for particular functions.

Computation of ${}_pF_q$ - general issues

Ultimately, the evaluation falls back to truncated hypergeometric series

$$ {}_pF_q(a_1,\ldots,a_p;b_1,\ldots,b_q;z) \approx \sum_{k=0}^{N} \frac{(a_1)_k\cdots(a_p)_k}{(b_1)_k\cdots(b_q)_k}\frac{z^k}{k!}. $$

The series is summed using fixed-point arithmetic (optimized for rational and integer parameters), automatically detecting cancellation.

The current implementation can give wrong results for certain combinations of large parameters. This can be worked around by increasing the precision (how much?).

Divergent hypergeometric series

We can define ${}_pF_q$ when $p > q + 1$ (zero radius of convergence) as asymptotic series, or using Borel regularization

$$ {}_pF_q(a; b; z) = \int_C e^{-t} \, {}_pF_{q+1}(a; b, 1; zt) dt $$

where C goes from 0 to $+\infty$, avoiding the branch cut along $zt \in (1, \infty)$.

For ${}_2F_0$, an exact formula exists. In general, we can use numerical integration.

For asymptotic expansions of convergent series, mpmath falls back to the original series instead of numerically computing the integral.

Example: function defined by a divergent series

$$\mathrm{Ci}(z) \sim \log z - \frac{\log z^2}{2} + \frac{\sin z}{z}\,{}_3F_0\left(\frac{1}{2}, 1, 1; ; -\frac{4}{z^2}\right) - \frac{\cos z}{z^2}\,{}_3F_0\left(1, 1, \frac{3}{2}; ; -\frac{4}{z^2}\right)$$

```
>>> z = mpf(0.5); u = -4/z**2
>>> H1 = hyper([1,1,1.5],[],u)
>>> H2 = hyper([0.5,1,1],[],u)
>>> log(z)-log(z**2)/2-cos(z)/z**2*H1+sin(z)/z*H2
-0.1777840788066129013358103
>>> ci(z)
-0.1777840788066129013358103
```

Example: Meijer G-function

$$G_{p,q}^{m,n}\left(\begin{matrix} a_1,\ldots,a_p \\ b_1,\ldots,b_q \end{matrix} \,\middle|\, z\right) = \frac{1}{2\pi i}\int_L \frac{\prod_{j=1}^{m}\Gamma(b_j - s)\prod_{j=1}^{n}\Gamma(1-a_j+s)}{\prod_{j=m+1}^{q}\Gamma(1-b_j+s)\prod_{j=n+1}^{p}\Gamma(a_j-s)} z^s\, ds$$

Hypergeometric series (two cases):

$$G_{p,q}^{m,n}\left(\begin{matrix} \mathbf{a_p} \\ \mathbf{b_q} \end{matrix} \,\middle|\, z\right) = \sum_{h=1}^{m} \frac{\prod_{j=1}^{m}\Gamma(b_j - b_h)^* \prod_{j=1}^{n}\Gamma(1+b_h-a_j)\, z^{b_h}}{\prod_{j=m+1}^{q}\Gamma(1+b_h-b_j)\prod_{j=n+1}^{p}\Gamma(a_j-b_h)} \times$$

$$\times\, {}_pF_{q-1}\left(\begin{matrix} 1+b_h-\mathbf{a_p} \\ (1+b_h-\mathbf{b_q})^* \end{matrix} \,\middle|\, (-1)^{p-m-n}\, z\right).$$

$$G_{p,q}^{m,n}\left(\begin{matrix} \mathbf{a_p} \\ \mathbf{b_q} \end{matrix} \,\middle|\, z\right) = \sum_{h=1}^{n} \frac{\prod_{j=1}^{n}\Gamma(a_h - a_j)^* \prod_{j=1}^{m}\Gamma(1-a_h+b_j)\, z^{a_h-1}}{\prod_{j=n+1}^{p}\Gamma(1-a_h+a_j)\prod_{j=m+1}^{q}\Gamma(a_h-b_j)} \times$$

$$\times\, {}_qF_{p-1}\left(\begin{matrix} 1-a_h+\mathbf{b_q} \\ (1-a_h+\mathbf{a_p})^* \end{matrix} \,\middle|\, \frac{(-1)^{q-m-n}}{z}\right).$$

Implementation: literal transcription of the two series, along with some conditionals; about 60 lines of code.

Example: Bessel K in terms of Meijer G

Most standard hypergeometric-type functions can be represented by a single Meijer G-function, e.g.:

$$2K_a(2\sqrt{z}) = G_{0,2}^{2,0}\left(\begin{matrix} - \\ \frac{a}{2}, -\frac{a}{2} \end{matrix} \middle| \, z \right)$$

```
>>> a = mpf(3)
>>> b = mpf(1.5)
>>> z = mpf(2.25)
>>> 0.5*meijerg([[],[]], [[a/2,-a/2],[]], (z/2)**2)
0.4105730291497623031944042
>>> besselk(a,z)
0.4105730291497623031944042
```

Meijer G-function: Behind the scenes

```
>>> 0.5*meijerg([[],[]], [[a/2,-a/2],[]], (z/2)**2, verbose=True)
Meijer G m,n,p,q,series = 2 0 0 2 1

ENTERING hypercomb main loop
prec = 96
hextra 0

  Evaluating term 1/2 : 0F1
    powers [1.26563] [1.5]
    gamma [-3.0] []
    hyper [] [4.0]
    z 1.26563
    Value: -2555834231509921839871116l759.9224182109128403373983260847208016

  Evaluating term 2/2 : 0F1
    powers [1.26563] [-1.5]
    gamma [3.0] []
    hyper [] [-2.0]
    z 1.26563
    Value: 2555834231509921839871116l760.7435642692123649437871345064768645

  Cancellation: 95 bits
  Increased precision: 126 bits

...
```

Failure of Meijer G in Mathematica

```
In[1]:= u = MeijerG[{{},{0}}, {{-1/2,-1,-3/2},{}}, 10000]
Out[1]= MeijerG[{{},{0}}, {{-(3/2),-1,-(1/2)},{}}, 10000]

In[2]:= N[u]
Out[2]= 2.13746*10^64

In[3]:= N[u, 20]
Out[3]= 7.5884116228159495633*10^54

In[4]:= N[u, 50]
Out[4]= 8.6169573361936181044600026286852647747386845482
2382*10^26

In[5]:= N[u, 100]
Out[5]= 2.4392576907199563959033246914513222810399956619
69628261302973914717875671298773756157861529245506262*
10^-94
```

Success of Meijer *G* in mpmath

```
>>> mp.dps=5
>>> meijerg([[],[0]],[[-0.5,-1,-1.5],[]],10000)
2.4393e-94
>>> mp.dps=20
>>> meijerg([[],[0]],[[-0.5,-1,-1.5],[]],10000)
2.4392576907199563959e-94
>>> mp.dps=50
>>> meijerg([[],[0]],[[-0.5,-1,-1.5],[]],10000)
2.4392576907199563959033246914340887567143003747173e-94
```

What's happening?

```
>>> mp.dps = 5
>>> meijerg([[],[0]],[[-0.5,-1,-1.5],[]],10000,verbose=True)
Meijer G m,n,p,q,series = 3 0 1 3 1

  [... several pages of output ...]

  Cancellation: 623 bits
  Increased precision: 700 bits
2.4393e-94
```

2D hypergeometric series

Appell functions F1–F4, Kampé de Fériet functions, Horn functions,

$$\sum_{m=0}^{\infty} \sum_{n=0}^{\infty} \frac{P(\mathbf{a}, m, n)}{Q(\mathbf{a}, m, n)} \frac{x^m y^n}{m! n!}$$

where P and Q are products of rising factorials such as $(a_j)_n$ or $(a_j)_{m+n}$.

Method of computation: rewrite as a series of 1D hypergeometric functions

$$\sum_{m=0}^{\infty} c_m \, {}_pF_q(\ldots) \frac{x^m}{m!}.$$

Example: Appell F1

$$F_1(a, b_1, b_2, c, x, y) = \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} \frac{(a)_{m+n}(b_1)_m(b_2)_n}{(c)_{m+n}} \frac{x^m y^n}{m!n!}.$$

```
def appellf1(a,b1,b2,c,x,y):
    return hyper2d({'m+n':[a],'m':[b1],'n':[b2]},
        {'m+n':[c]}, x,y)
```

```
# A known value
>>> appellf1(1,2,3,5,0.5,0.25)
1.54790227030268401933555
>>> 4*hyp2f1(1,2,5,'1/3')/3
1.54790227030268401933555
```

The actual `mpmath.appellf1` does some more of preprocessing.

Gamma, polygamma, zeta, polylogarithms, ...

Applications: glue functions, combinatorics, analytic number theory, . . .

The gamma function

Next to the elementary functions, the gamma function $\Gamma(z)$ is the most important special function (not least for computations).

The gamma function

Next to the elementary functions, the gamma function $\Gamma(z)$ is the most important special function (not least for computations).

- Computation of various combinatorial functions (binomial coefficients, double factorials, ...)

The gamma function

Next to the elementary functions, the gamma function $\Gamma(z)$ is the most important special function (not least for computations).

- Computation of various combinatorial functions (binomial coefficients, double factorials, ...)
- Coefficients of hypergeometric functions

The gamma function

Next to the elementary functions, the gamma function $\Gamma(z)$ is the most important special function (not least for computations).

- Computation of various combinatorial functions (binomial coefficients, double factorials, ...)
- Coefficients of hypergeometric functions
- Functional equations for special functions (e.g. L-functions)

The gamma function

Next to the elementary functions, the gamma function $\Gamma(z)$ is the most important special function (not least for computations).

- Computation of various combinatorial functions (binomial coefficients, double factorials, ...)
- Coefficients of hypergeometric functions
- Functional equations for special functions (e.g. L-functions)
- Important special cases: integer and rational arguments, near poles, large (complex) arguments

Algorithms for the gamma function

$$\log \Gamma(z) \sim \frac{1}{2}\log(2\pi) + \left(z - \frac{1}{2}\right)\log z - z + \sum_{n=1}^{\infty} \frac{B_{2n}}{2n(2n-1)z^{2n-1}}$$

General case: Stirling's series + recurrence ($\Gamma(z) = \Gamma(z+n)/(z)_n$)

Algorithms for the gamma function

$$\log \Gamma(z) \sim \frac{1}{2}\log(2\pi) + \left(z - \frac{1}{2}\right)\log z - z + \sum_{n=1}^{\infty} \frac{B_{2n}}{2n(2n-1)z^{2n-1}}$$

General case: Stirling's series + recurrence ($\Gamma(z) = \Gamma(z+n)/(z)_n$)

- Small integers and half-integers: direct computation, caching

Algorithms for the gamma function

$$\log \Gamma(z) \sim \frac{1}{2}\log(2\pi) + \left(z - \frac{1}{2}\right)\log z - z + \sum_{n=1}^{\infty} \frac{B_{2n}}{2n(2n-1)z^{2n-1}}$$

General case: Stirling's series + recurrence ($\Gamma(z) = \Gamma(z+n)/(z)_n$)

- Small integers and half-integers: direct computation, caching
- Rational numbers: AGM, hypergeometric series (not implemented)

Algorithms for the gamma function

$$\log \Gamma(z) \sim \frac{1}{2}\log(2\pi) + \left(z - \frac{1}{2}\right)\log z - z + \sum_{n=1}^{\infty} \frac{B_{2n}}{2n(2n-1)z^{2n-1}}$$

General case: Stirling's series + recurrence ($\Gamma(z) = \Gamma(z+n)/(z)_n$)

- Small integers and half-integers: direct computation, caching
- Rational numbers: AGM, hypergeometric series (not implemented)
- Small real arguments: Taylor series (involves computing $\zeta(2), \zeta(3), \ldots, \zeta(N)$)

Algorithms for the gamma function

$$\log \Gamma(z) \sim \frac{1}{2}\log(2\pi) + \left(z - \frac{1}{2}\right)\log z - z + \sum_{n=1}^{\infty} \frac{B_{2n}}{2n(2n-1)z^{2n-1}}$$

General case: Stirling's series + recurrence ($\Gamma(z) = \Gamma(z+n)/(z)_n$)

- Small integers and half-integers: direct computation, caching
- Rational numbers: AGM, hypergeometric series (not implemented)
- Small real arguments: Taylor series (involves computing $\zeta(2), \zeta(3), \ldots, \zeta(N)$)
- Log-gamma special-cased

Improvements to gamma?

- Lanczos and Spouge's approximations seem less efficient in practice than Stirling's series. (Or?)
- Possible arithmetic optimizations: reduced-complexity rising factorial and Stirling series.
- It would be nice to have a fast method close to the imaginary axis (Taylor series?).

The Hurwitz zeta function

$$\zeta^{(n)}(s,a) = (-1)^n \sum_{k=0}^{\infty} \frac{\log^n(a+k)}{(a+k)^s}$$

Special cases: Riemann zeta function $\zeta(s) = \zeta(s,1)$, polygamma functions, Dirichlet L-series.

- General case: Euler-Maclaurin summation
- Riemann zeta for real arguments: Borwein's algorithm
- $\zeta(2n)$: also Borwein's algorithm (use ζ to compute large Bernoulli numbers!)
- Riemann zeta for large $\Im(s)$: Riemann-Siegel expansion
- In all cases, the main computational cost is to evaluate the truncated series (sines and cosines for s complex)

Zeta near the critical strip

Riemann-Siegel expansion for large $\Im(s)$, implemented by Juan Arias de Reyna.

```
>>> zeta(0.5+1000000j)
(0.076089069738227100006564 56 + 2.8051021010192989553938 37j)
```

```
>>> zeta(0.5+100000000j)
(-3.362839487530727943146807 + 1.407234559646447885979583j)
>>> zeta(0.5+10000000000j)
(0.356800230856073382539587 9 + 0.286505849095836103292093j)
>>> zeta(2.5+10000000000j)
(0.984153654307166279567326 4 + 0.243417667646574633227435 1j)

>>> zetazero(10)
(0.5 + 49.773832477672302181916 78j)
>>> zetazero(100)
(0.5 + 236.52422966581620580247 55j)
>>> zetazero(100000)
(0.5 + 74920.827498994186793849 2j)
>>> zetazero(1000000000)
(0.5 + 371870203.83702805273405 48j)
```

Elliptic functions

General method: Jacobi theta functions, e.g.

$$\vartheta_1(z,q) = 2q^{1/4} \sum_{n=0}^{\infty} (-1)^n q^{n^2+n} \sin((2n+1)z)$$

```
# Jacobi elliptic functions
>>> sn = ellipfun('sn')
>>> sn(3, 0.5)
0.6300289982420331644946371

# J-function
>>> taylor(lambda q: 1728*q*kleinj(qbar=q), 0, 4,
...     singular=True)
...
[1.0, 744.0, 196884.0, 21493760.0, 864299970.0]
```

Elliptic integrals

An elliptic integral is of the type

$$\int R(t, \sqrt{P(t)})dt$$

where R is a rational function, P a polynomial of degree 3 or 4.
Legendre standard forms:

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

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

$$\Pi(n; \phi, m) = \int_0^{\phi} \frac{dt}{(1 - n \sin^2 t)\sqrt{1 - m \sin^2 t}}$$

Complete elliptic integrals

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

$$K(m) = \frac{\pi}{2}\, {}_2F_1\left(\frac{1}{2}, \frac{1}{2}, 1, m\right) = \frac{\pi}{2\,\mathrm{agm}(1, \sqrt{1-m})}$$

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

$$E(m) = \frac{\pi}{2}\, {}_2F_1\left(\frac{1}{2}, -\frac{1}{2}, 1, m\right)$$

Easily computed efficiently to arbitrary precision.

Incomplete elliptic integrals

"Direct" methods: numerical integration, hypergeometric series of two variables.

```
>>> def E(z,m):
...     return quad(lambda t:
...         sqrt(1-m*sin(t)**2), [0, z])
...
>>> z, m = 0.25, 0.5
>>> E(0.25,0.5)
0.24870810580460582237630 82
>>> ellipe(0.25,0.5)
0.2487081058046058223763082
>>> def E2(z, m):
...     return sin(z)*appellf1(0.5,0.5,-0.5,1.5,
...         sin(z)**2,m*sin(z)**2)
...
>>> E2(0.25,0.5)
0.2487081058046058223763082
```

Problems

- The hypergeometric series is slow near singularities
- Quadrature is slow near singularities
- Both methods are slow at very high precision

A further problem: extending the functions to arbitrary arguments.

Carlson symmetric forms

An alternative to Legendre's canonical forms:

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

$$R_J(x,y,z,p) = \frac{3}{2}\int_0^\infty \frac{dt}{(t+p)\sqrt{(t+x)(t+y)(t+z)}}$$

Plus special cases R_C, R_D, R_G. Symmetric transformations, e.g.:

$$R_F(x,y,z) = 2R_F(x+\lambda, y+\lambda, z+\lambda) = R_F\left(\frac{x+\lambda}{4}, \frac{y+\lambda}{4}, \frac{z+\lambda}{4}\right)$$

where $\lambda = \sqrt{xy} + \sqrt{yz} + \sqrt{zx}$.
Simple structure (continuous except for $x,y,z \in (-\infty, 0)$),
rigorous algorithms provided by Carlson.

Example: elliptic logarithm

$$\mathrm{elog}_{a,b}(z) = \frac{1}{2}\int_z^\infty \frac{dt}{\sqrt{t^3+at^2+bt}} = R_F(z, z+q_+, z+q_-)$$

$$q_\pm = \frac{1}{2}\left(a \pm \sqrt{a^2-4b}\right)$$

```
>>> def elog(z, a, b):
...     r = sqrt(a**2-4*b)
...     return elliprf(z, z+(a+r)/2, z+(a-r)/2)
...
>>> elog(0.3,-5,1)
(0.725631185227299 - 1.06817817835219j)
>>> elog(2+1j,1-1j,-12)
(0.63171902015689 - 0.250181338142278j)
```

Discussion

- There are hundreds of common special functions. Most are redundant and can be reduced to a *handful of very general functions*.
- Using good abstractions can save a lot of work.
- These techniques don't always work in machine precision.
- It is very difficult to do numerical evaluation rigorously without sacrificing power or generality.