Numerical Algorithms in Pari/GP

Henri Cohen

November 5, 2015

IMB, Université de Bordeaux

Goal of Talk I

Most algorithms in number theory are arithmetic in nature, and in particular involve only exact quantities: e.g., number field computations (prime ideals, class group, etc...), counting points of varieties, etc...

However one also needs a large number of numerical algorithms which involve inexact (or approximate) quantities: e.g., summation of infinite series, numerical integration, extrapolation, higher transcendental functions, and at a higher level, computation of L-functions and related algorithms such as inverse Mellin transforms, generalized incomplete gamma functions, etc...

Goal of Talk I

Most algorithms in number theory are arithmetic in nature, and in particular involve only exact quantities: e.g., number field computations (prime ideals, class group, etc...), counting points of varieties, etc...

However one also needs a large number of numerical algorithms which involve inexact (or approximate) quantities: e.g., summation of infinite series, numerical integration, extrapolation, higher transcendental functions, and at a higher level, computation of L-functions and related algorithms such as inverse Mellin transforms, generalized incomplete gamma functions, etc...

Goal of Talk II

Goal of talk: give an overview of these methods, many of them new, all available in the `Pari/GP` package. At least three reasons:

1. Important to know these algorithms exist at all and very efficient, so as to be able to use them when necessary.
2. Many algorithms involve interesting ideas.
3. Some of the algorithms lead to unsolved algorithmic and mathematical problems. In fact, some algorithms are not rigorously proved, but are so useful as to be essential.

Goal of Talk II

Goal of talk: give an overview of these methods, many of them new, all available in the `Pari/GP` package. At least three reasons:

1. Important to know these algorithms exist at all and very efficient, so as to be able to use them when necessary.
2. Many algorithms involve interesting ideas.
3. Some of the algorithms lead to unsolved algorithmic and mathematical problems. In fact, some algorithms are not rigorously proved, but are so useful as to be essential.

Goal of Talk II

Goal of talk: give an overview of these methods, many of them new, all available in the `Pari/GP` package. At least three reasons:

1. Important to know these algorithms exist at all and very efficient, so as to be able to use them when necessary.
2. Many algorithms involve interesting ideas.
3. Some of the algorithms lead to unsolved algorithmic and mathematical problems. In fact, some algorithms are not rigorously proved, but are so useful as to be essential.

Goal of Talk III

Depending on time, I will present the following:

1. Numerical integration.
2. Numerical summation and extrapolation.
3. Multiple zeta values and multiple polylogs.
4. Inverse Mellin transforms.
5. Computing L-functions.

For all these problems, we want to compute with reasonable but not too small accuracy, say between 30 and 1000 decimal digits. This completely changes the problem compared to standard numerical analysis. In all subjects, I have experimented with many methods, old and new, and usually reached definite conclusions.

Goal of Talk III

Depending on time, I will present the following:

1. Numerical integration.
2. Numerical summation and extrapolation.
3. Multiple zeta values and multiple polylogs.
4. Inverse Mellin transforms.
5. Computing L-functions.

For all these problems, we want to compute with reasonable but not too small accuracy, say between 30 and 1000 decimal digits. This completely changes the problem compared to standard numerical analysis. In all subjects, I have experimented with many methods, old and new, and usually reached definite conclusions.

Numerical Integration I

- First need to isolate possible singularities of the function to integrate and take them into account (e.g.: $\int_0^1 x^{-\alpha} f(x)\,dx$ with $0 < \alpha < 1$ and f regular, make the change of variable $y = x^{1/(1-\alpha)}$ to make the singularity at 0 disappear).
- Must distinguish between compact intervals ($\int_a^b f(x)\,dx$) and infinite intervals (e.g., $\int_0^\infty f(x)\,dx$ or $\int_{-\infty}^\infty f(x)\,dx$). When the compact interval is "long", must also be taken into account (technical).
- For infinite intervals, must distinguish between several types of behavior at infinity: exponential (e.g., e^{-x}), ordinary polynomial type decrease (e.g., $x^{-\alpha}$ with $\alpha \geq 2$), slow polynomial type decrease (same with $1 < \alpha < 2$, which as in (1) can be reduced to ordinary by change of variable), oscillating (or worse) behavior.

Numerical Integration I

- First need to isolate possible singularities of the function to integrate and take them into account (e.g.: $\int_0^1 x^{-\alpha} f(x)\,dx$ with $0 < \alpha < 1$ and f regular, make the change of variable $y = x^{1/(1-\alpha)}$ to make the singularity at 0 disappear).
- Must distinguish between compact intervals ($\int_a^b f(x)\,dx$) and infinite intervals (e.g., $\int_0^\infty f(x)\,dx$ or $\int_{-\infty}^\infty f(x)\,dx$). When the compact interval is "long", must also be taken into account (technical).
- For infinite intervals, must distinguish between several types of behavior at infinity: exponential (e.g., e^{-x}), ordinary polynomial type decrease (e.g., $x^{-\alpha}$ with $\alpha \geq 2$), slow polynomial type decrease (same with $1 < \alpha < 2$, which as in (1) can be reduced to ordinary by change of variable), oscillating (or worse) behavior.

Numerical Integration I

- First need to isolate possible singularities of the function to integrate and take them into account (e.g.: $\int_0^1 x^{-\alpha} f(x)\,dx$ with $0 < \alpha < 1$ and f regular, make the change of variable $y = x^{1/(1-\alpha)}$ to make the singularity at 0 disappear).
- Must distinguish between compact intervals ($\int_a^b f(x)\,dx$) and infinite intervals (e.g., $\int_0^\infty f(x)\,dx$ or $\int_{-\infty}^\infty f(x)\,dx$). When the compact interval is "long", must also be taken into account (technical).
- For infinite intervals, must distinguish between several types of behavior at infinity: exponential (e.g., e^{-x}), ordinary polynomial type decrease (e.g., $x^{-\alpha}$ with $\alpha \geq 2$), slow polynomial type decrease (same with $1 < \alpha < 2$, which as in (1) can be reduced to ordinary by change of variable), oscillating (or worse) behavior.

Numerical Integration II

We have to assume the integrand "sufficiently regular", including in its behavior at infinity. Cannot make this mathematically precise, but no problem in practice. Usual cheat: work with D decimals, then again with $D + 10$ decimals, and check if results agree.

You may have heard of many methods: Simpson, Newton–Cotes, Romberg, Richardson, etc... In fact, two methods are by far the best: the Gauss–Legendre method and generalizations (abbreviated **GLIM**), and Doubly exponential methods (abbreviated **DENIM**). Will describe both in detail.

Numerical Integration II

We have to assume the integrand "sufficiently regular", including in its behavior at infinity. Cannot make this mathematically precise, but no problem in practice. Usual cheat: work with D decimals, then again with $D + 10$ decimals, and check if results agree.

You may have heard of many methods: Simpson, Newton–Cotes, Romberg, Richardson, etc... In fact, two methods are by far the best: the Gauss–Legendre method and generalizations (abbreviated **GLIM**), and Doubly exponential methods (abbreviated **DENIM**). Will describe both in detail.

Integration: Gauss–Legendre I

One of the oldest numerical integration methods, the fastest in multiprecision when the function is very regular. Basic version on $[-1,1]$ with constant weight 1:
The Legendre polynomials $P_n(X)$ can be defined by

$$P_n(X) = \frac{1}{2^n n!}\frac{d^n}{dX^n}((X^2-1)^n)\ ,$$

one of the most classical family of orthogonal polynomials.

Basic Gauss–Legendre says that

$$\int_{-1}^{1} f(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

with a well understood error, where the x_i are the roots of P_n and the weights w_i are given by $w_i = 2/((1-x_i^2)P'(x_i)^2)$.

The x_i can be computed very efficiently using specific methods.

Integration: Gauss–Legendre I

One of the oldest numerical integration methods, the fastest in multiprecision when the function is very regular. Basic version on $[-1,1]$ with constant weight 1:
The Legendre polynomials $P_n(X)$ can be defined by

$$P_n(X) = \frac{1}{2^n n!}\frac{d^n}{dX^n}((X^2-1)^n)\ ,$$

one of the most classical family of orthogonal polynomials.

Basic Gauss–Legendre says that

$$\int_{-1}^{1} f(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

with a well understood error, where the x_i are the roots of P_n and the weights w_i are given by $w_i = 2/((1-x_i^2)P'(x_i)^2)$.

The x_i can be computed very efficiently using specific methods.

Integration: Gauss–Legendre I

One of the oldest numerical integration methods, the fastest in multiprecision when the function is very regular. Basic version on $[-1,1]$ with constant weight 1:
The Legendre polynomials $P_n(X)$ can be defined by

$$P_n(X) = \frac{1}{2^n n!}\frac{d^n}{dX^n}((X^2-1)^n)\ ,$$

one of the most classical family of orthogonal polynomials.

Basic Gauss–Legendre says that

$$\int_{-1}^{1} f(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

with a well understood error, where the x_i are the roots of P_n and the weights w_i are given by $w_i = 2/((1-x_i^2)P'(x_i)^2)$.

The x_i can be computed very efficiently using specific methods.

Integration: Gauss–Legendre II

Method very flexible: for a given measure $\mu(x)\,dx$ on $[a,b]$, we have a formula of the type:

$$\int_a^b f(x)\mu(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i)\ ,$$

where the x_i are roots of suitable orthogonal polynomials and w_i can be computed from them. In more detail:

The measure $\mu(x)dx$ defines a scalar product (possibly not positive definite) $< f,g >= \int_a^b f(x)g(x)\mu(x)\,dx$. In addition, the measure is usually known through its moments $M_k =< 1, x^k >= \int_a^b x^k \mu(x)\,dx$. The standard way to compute x_i and w_i involves a Cholesky decomposition of the matrix of moments. Very slow. Much better method: use continued fractions. Since used elsewhere, we explain in detail.

Integration: Gauss–Legendre II

Method very flexible: for a given measure $\mu(x)\,dx$ on $[a,b]$, we have a formula of the type:

$$\int_a^b f(x)\mu(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i)\ ,$$

where the x_i are roots of suitable orthogonal polynomials and w_i can be computed from them. In more detail:

The measure $\mu(x)dx$ defines a scalar product (possibly not positive definite) $< f,g >= \int_a^b f(x)g(x)\mu(x)\,dx$. In addition, the measure is usually known through its moments $M_k =< 1, x^k >= \int_a^b x^k\mu(x)\,dx$. The standard way to compute x_i and w_i involves a Cholesky decomposition of the matrix of moments. Very slow. Much better method: use continued fractions. Since used elsewhere, we explain in detail.

Integration: Gauss–Legendre III

We define the formal power series

$$\Phi(z) = \sum_{k \geq 0} M_k z^{k+1} = z \int_a^b \frac{\mu(x)\,dx}{1 - xz} .$$

Assuming certain nonvanishing assumptions, we can transform into a formal continued fraction

$$\Phi(z) = c(0)z/(1 + c(1)z/(1 + c(2)z/(1 + \cdots)))$$

(e.g., $c(0) = M_0$, $c(1) = -M_1/M_0$, etc...). In general must allow powers of z, but to simplify exposition assume not.

This can be done very efficiently using a well-known algorithm called the Quotient-Difference (QD) algorithm (NOT by dividing power series).

Integration: Gauss–Legendre III

We define the formal power series

$$\Phi(z) = \sum_{k \geq 0} M_k z^{k+1} = z \int_a^b \frac{\mu(x)\,dx}{1 - xz} .$$

Assuming certain nonvanishing assumptions, we can transform into a formal continued fraction

$$\Phi(z) = c(0)z/(1 + c(1)z/(1 + c(2)z/(1 + \cdots)))$$

(e.g., $c(0) = M_0$, $c(1) = -M_1/M_0$, etc...). In general must allow powers of z, but to simplify exposition assume not.

This can be done very efficiently using a well-known algorithm called the Quotient-Difference (QD) algorithm (NOT by dividing power series).

Integration: Gauss–Legendre IV

Write as usual with continued fractions:

$$p_n(z)/q_n(z) = c(0)z/(1 + c(1)z/(1 + c(2)z/(1 + \ddots + c(n)z))) .$$

Then $\deg(p_{2n-1}) = \deg(q_{2n-1}) = n$ and $p_{2n-1}(0) = 0$. Let N_n (resp., D_n) be the reciprocal polynomial of $p_{2n-1}(z)/z$ (resp., of q_{2n-1})

Assuming μ positive (but works more generally):
Theorem: The D_n are up to normalization the unique family of orthogonal polynomials for the scalar product. The nodes x_i for Gaussian integration are the roots of D_n, and the weights w_i are simply $w_i = N_n(x_i)/D_n'(x_i)$ (D_n has only simple roots).

Integration: Gauss–Legendre IV

Write as usual with continued fractions:

$$p_n(z)/q_n(z) = c(0)z/(1 + c(1)z/(1 + c(2)z/(1 + \ddots + c(n)z))) .$$

Then $\deg(p_{2n-1}) = \deg(q_{2n-1}) = n$ and $p_{2n-1}(0) = 0$. Let N_n (resp., D_n) be the reciprocal polynomial of $p_{2n-1}(z)/z$ (resp., of q_{2n-1})

Assuming μ positive (but works more generally):
Theorem: The D_n are up to normalization the unique family of orthogonal polynomials for the scalar product. The nodes x_i for Gaussian integration are the roots of D_n, and the weights w_i are simply $w_i = N_n(x_i)/D_n'(x_i)$ (D_n has only simple roots).

Integration: Gauss–Legendre V

One more situation where GLIM is useful: integration on infinite intervals of polynomially decreasing functions. For simplicity, assume asymptotic expansion at ∞ of f of the form $f(x) = a_2/x^2 + a_3/x^3 + \cdots$. Then

$$\int_1^\infty f(x)\,dx = \int_0^1 (f(1/x)/x^2)\,dx\ ,$$

and $f(1/x)/x^2 = a_2 + a_3 x + \cdots$ is regular at 0 so GLIM applies: very good method.

Note: if you have heard in undergraduate courses that $\int_0^\infty f(x)\,dx$ can be computed using the Gauss–Laguerre method when $f(x)$ behaves like e^{-x} at infinity, this is a joke. It is mathematically true, sometimes useful for a few decimals, but totally useless in the number-theoretic context. By far the best: **DENIM** (doubly-exponential methods).

Integration: Gauss–Legendre V

One more situation where GLIM is useful: integration on infinite intervals of polynomially decreasing functions. For simplicity, assume asymptotic expansion at ∞ of f of the form $f(x) = a_2/x^2 + a_3/x^3 + \cdots$. Then

$$\int_1^\infty f(x)\,dx = \int_0^1 (f(1/x)/x^2)\,dx\ ,$$

and $f(1/x)/x^2 = a_2 + a_3 x + \cdots$ is regular at 0 so GLIM applies: very good method.

Note: if you have heard in undergraduate courses that $\int_0^\infty f(x)\,dx$ can be computed using the Gauss–Laguerre method when $f(x)$ behaves like e^{-x} at infinity, this is a joke. It is mathematically true, sometimes useful for a few decimals, but totally useless in the number-theoretic context. By far the best: **DENIM** (doubly-exponential methods).

Integration: Doubly–Exponential Methods I

Immediate caveat: applies only to functions meromorphic in a domain containing the interval of integration. In fact rather surprising: even though Gauss–Legendre only needs regularity (C^∞, say), it is less "robust" than DENIM in some sense.

The basic idea is very simple:

- If $F(t)$ tends to 0 doubly-exponentially when $|t| \to \infty$ (like $\exp(-A\exp(B|t|))$ with A, B positive), choose N and h suitably (typically $N = 50$, $h = 0.02$, of course must analyze the method) and just compute Riemann sums:

$$\int_{-\infty}^{\infty} F(t)\,dt \approx h \sum_{m=-N}^{N} F(mh)\ .$$

- For other functions and intervals, make a suitable change of variable.

Integration: Doubly–Exponential Methods I

Immediate caveat: applies only to functions meromorphic in a domain containing the interval of integration. In fact rather surprising: even though Gauss–Legendre only needs regularity (C^∞, say), it is less "robust" than DENIM in some sense.

The basic idea is very simple:

- If $F(t)$ tends to 0 doubly-exponentially when $|t| \to \infty$ (like $\exp(-A\exp(B|t|))$ with A, B positive), choose N and h suitably (typically $N = 50$, $h = 0.02$, of course must analyze the method) and just compute Riemann sums:

$$\int_{-\infty}^{\infty} F(t)\,dt \approx h \sum_{m=-N}^{N} F(mh)\ .$$

- For other functions and intervals, make a suitable change of variable.

Integration: Doubly–Exponential Methods I

Immediate caveat: applies only to functions meromorphic in a domain containing the interval of integration. In fact rather surprising: even though Gauss–Legendre only needs regularity (C^∞, say), it is less "robust" than DENIM in some sense.

The basic idea is very simple:

- If $F(t)$ tends to 0 doubly-exponentially when $|t| \to \infty$ (like $\exp(-A\exp(B|t|))$ with A, B positive), choose N and h suitably (typically $N = 50$, $h = 0.02$, of course must analyze the method) and just compute Riemann sums:

$$\int_{-\infty}^{\infty} F(t)\,dt \approx h \sum_{m=-N}^{N} F(mh)\ .$$

- For other functions and intervals, make a suitable change of variable.

Integration: DENIM II

It is a theorem that in some sense one cannot do better than Riemann sums for such functions. Method invented in 1969 due to Takahashi and Mori.
The whole technology is to find the change of variables, and also possibly change the path of integration in the complex plane.

Main examples:

• For $\int_{-1}^{1} f(x)\,dx$, set $x = \tanh(\sinh((\pi/2)t))$, and for a general compact interval do an affine transformation to reduce to $[-1,1]$.

Integration: DENIM II

It is a theorem that in some sense one cannot do better than Riemann sums for such functions. Method invented in 1969 due to Takahashi and Mori.
The whole technology is to find the change of variables, and also possibly change the path of integration in the complex plane.

Main examples:
• For $\int_{-1}^{1} f(x)\,dx$, set $x = \tanh(\sinh((\pi/2)t))$, and for a general compact interval do an affine transformation to reduce to $[-1, 1]$.

Integration: DENIM III

- For $\int_0^\infty f(x)\,dx$: if f tends to 0 polynomially use $x = \exp(\sinh(t))$, if f tends to 0 exponentially, use $x = \exp(t - \exp(-t))$.
- For $\int_{-\infty}^\infty f(x)\,dx$: if f tends to 0 slowly use $x = \sinh(\sinh(t))$, if exponentially use $x = \sinh(t)$.
- For $\int_0^\infty f(x)\,dx$ where f is oscillating (example $f(x) = \sin(x)/x^2$ on $[1, \infty[$), there are specific methods.

Integration: DENIM III

- For $\int_0^\infty f(x)\,dx$: if f tends to 0 polynomially use $x = \exp(\sinh(t))$, if f tends to 0 exponentially, use $x = \exp(t - \exp(-t))$.
- For $\int_{-\infty}^\infty f(x)\,dx$: if f tends to 0 slowly use $x = \sinh(\sinh(t))$, if exponentially use $x = \sinh(t)$.
- For $\int_0^\infty f(x)\,dx$ where f is oscillating (example $f(x) = \sin(x)/x^2$ on $[1, \infty[$), there are specific methods.

Integration: DENIM III

- For $\int_0^\infty f(x)\,dx$: if f tends to 0 polynomially use $x = \exp(\sinh(t))$, if f tends to 0 exponentially, use $x = \exp(t - \exp(-t))$.
- For $\int_{-\infty}^\infty f(x)\,dx$: if f tends to 0 slowly use $x = \sinh(\sinh(t))$, if exponentially use $x = \sinh(t)$.
- For $\int_0^\infty f(x)\,dx$ where f is oscillating (example $f(x) = \sin(x)/x^2$ on $[1, \infty[$), there are specific methods.

Integration: Examples I

- $$I_1 = \int_0^1 \zeta(x+2)\,dx\ .$$

  At 500 decimal digits, DENIM requires 18.2 seconds, Gauss–Legendre needs 5.6 seconds.

- $$I_2 = \int_1^\infty \log(\Gamma(1+1/x^2))\,dx\ .$$

  Infinite interval, polynomially decreasing function: at 500 D, DENIM requires 10.2s but GLIM needs only 1.3s.

- $$I_3 = \int_1^\infty \Gamma(x+1)/(x+1)^{x+1/2}\,dx\ .$$

  Requires 3.5s using DENIM (GLIM is useless: exponential decrease).

Integration: Examples I

- $$I_1 = \int_0^1 \zeta(x+2)\,dx\ .$$

  At 500 decimal digits, DENIM requires 18.2 seconds, Gauss–Legendre needs 5.6 seconds.

- $$I_2 = \int_1^\infty \log(\Gamma(1+1/x^2))\,dx\ .$$

  Infinite interval, polynomially decreasing function: at 500 D, DENIM requires 10.2s but GLIM needs only 1.3s.

- $$I_3 = \int_1^\infty \Gamma(x+1)/(x+1)^{x+1/2}\,dx\ .$$

  Requires 3.5s using DENIM (GLIM is useless: exponential decrease).

Integration: Examples I

- $$I_1 = \int_0^1 \zeta(x+2)\,dx\ .$$

  At 500 decimal digits, DENIM requires 18.2 seconds, Gauss–Legendre needs 5.6 seconds.

- $$I_2 = \int_1^\infty \log(\Gamma(1+1/x^2))\,dx\ .$$

  Infinite interval, polynomially decreasing function: at 500 D, DENIM requires 10.2s but GLIM needs only 1.3s.

- $$I_3 = \int_1^\infty \Gamma(x+1)/(x+1)^{x+1/2}\,dx\ .$$

  Requires 3.5s using DENIM (GLIM is useless: exponential decrease).

Integration: Examples II

- $$I_4 = \int_0^1 \int_0^1 e^{\sqrt{x^2+y^2+1}}\, dy\, dx\ ,$$

  an example of a double integral. At 115 decimals, DENIM requires 4.0 seconds, while GLIM requires only 0.1 second: thus double integrals can be computed to high accuracy, if possible with Gauss–Legendre.

  Even at 500 D, the time is reasonable for GLIM: 17.6 s, while DENIM takes more than 15 minutes, I did not wait.

Conclusion: Try to use Gauss–Legendre, much faster, otherwise doubly-exponential methods, slower but more robust.

`Pari/GP` commands: **intnumgauss** and **intnum**.

Integration: Examples II

- $$I_4 = \int_0^1 \int_0^1 e^{\sqrt{x^2+y^2+1}}\, dy\, dx\ ,$$

  an example of a double integral. At 115 decimals, DENIM requires 4.0 seconds, while GLIM requires only 0.1 second: thus double integrals can be computed to high accuracy, if possible with Gauss–Legendre.
  Even at 500 D, the time is reasonable for GLIM: 17.6 s, while DENIM takes more than 15 minutes, I did not wait.

Conclusion: Try to use Gauss–Legendre, much faster, otherwise doubly-exponential methods, slower but more robust.

`Pari/GP` commands: **intnumgauss** and **intnum**.

Integration: Examples II

- $$I_4 = \int_0^1 \int_0^1 e^{\sqrt{x^2+y^2+1}}\, dy\, dx\ ,$$

  an example of a double integral. At 115 decimals, DENIM requires 4.0 seconds, while GLIM requires only 0.1 second: thus double integrals can be computed to high accuracy, if possible with Gauss–Legendre.
  Even at 500 D, the time is reasonable for GLIM: 17.6 s, while DENIM takes more than 15 minutes, I did not wait.

Conclusion: Try to use Gauss–Legendre, much faster, otherwise doubly-exponential methods, slower but more robust.

Pari/GP commands: **intnumgauss** and **intnum**.

Integration: Examples II

- $$I_4 = \int_0^1 \int_0^1 e^{\sqrt{x^2+y^2+1}}\, dy\, dx\ ,$$

  an example of a double integral. At 115 decimals, DENIM requires 4.0 seconds, while GLIM requires only 0.1 second: thus double integrals can be computed to high accuracy, if possible with Gauss–Legendre.
  Even at 500 D, the time is reasonable for GLIM: 17.6 s, while DENIM takes more than 15 minutes, I did not wait.

Conclusion: Try to use Gauss–Legendre, much faster, otherwise doubly-exponential methods, slower but more robust.

`Pari/GP` commands: **intnumgauss** and **intnum**.

Numerical Summation I

Again need to assume regularity in some sense, and nice behavior at infinity. As for integration, two methods are by far the best: the Gauss–Monien method and generalizations (abbreviated **GMSM**), and the discrete Euler–MacLaurin (abbreviated **DEMSM**). In addition, specific methods for alternating series.

Note that if the summand tends to zero exponentially, in general summing many terms is sufficient, so we will always assume that the summand tends to 0 polynomially (don't bug me with summands such as $e^{-n/1000}$ which in practice are neither exponential nor polynomial), and not oscillating (such as $\sin(n)/n^{5/2}$, although this can be summed numerically).

Numerical Summation I

Again need to assume regularity in some sense, and nice behavior at infinity. As for integration, two methods are by far the best: the Gauss–Monien method and generalizations (abbreviated **GMSM**), and the discrete Euler–MacLaurin (abbreviated **DEMSM**). In addition, specific methods for alternating series.

Note that if the summand tends to zero exponentially, in general summing many terms is sufficient, so we will always assume that the summand tends to 0 polynomially (don't bug me with summands such as $e^{-n/1000}$ which in practice are neither exponential nor polynomial), and not oscillating (such as $\sin(n)/n^{5/2}$, although this can be summed numerically).

Numerical Summation II

For both GMSM and DEMSM there is an important restriction: the summand must be defined on $\mathbb{R}^+$, not only on the positive integers. Thus cannot use them for $\sum_{n\geq 1}(n-1)!/(n^{n+3/2}e^{-n})$, but can use them for the equivalent sum $\sum_{n\geq 1}\Gamma(n)/(n^{n+3/2}e^{-n})$.

The sumalt method for alternating series does not have this restriction.

Numerical Summation II

For both GMSM and DEMSM there is an important restriction: the summand must be defined on $\mathbb{R}^+$, not only on the positive integers. Thus cannot use them for $\sum_{n\geq 1}(n-1)!/(n^{n+3/2}e^{-n})$, but can use them for the equivalent sum $\sum_{n\geq 1}\Gamma(n)/(n^{n+3/2}e^{-n})$.

The sumalt method for alternating series does not have this restriction.

Gauss–Monien Summation I

Method due to the German physicist Hartmut Monien (Bonn University): in a nutshell, it is Gaussian integration with measure $d\mu(x) = \sum_{m\geq 1} \delta(1/x - m)/m^2$, where δ is the usual Dirac measure centered at 0 of weight 1. It is clear that $\int_0^1 f(x)d\mu(x) = \sum_{m\geq 1} f(1/m)/m^2$.

Thus we follow the general continued fraction method: the kth moment is $\int_0^1 x^k d\mu(x) = \zeta(k+2)$, so we set as before $\Phi(z) = \sum_{k\geq 0} \zeta(k+2)z^{k+1}$, expand formally as a continued fraction using the QD algorithm:

$$\sum_{k\geq 0} \zeta(k+2)z^{k+1} = c(0)z/(1+c(1)z/(1+c(2)z/(1+\cdots)))\ .$$

Gauss–Monien Summation I

Method due to the German physicist Hartmut Monien (Bonn University): in a nutshell, it is Gaussian integration with measure $d\mu(x) = \sum_{m\geq 1} \delta(1/x - m)/m^2$, where δ is the usual Dirac measure centered at 0 of weight 1. It is clear that $\int_0^1 f(x)d\mu(x) = \sum_{m\geq 1} f(1/m)/m^2$.

Thus we follow the general continued fraction method: the kth moment is $\int_0^1 x^k d\mu(x) = \zeta(k+2)$, so we set as before $\Phi(z) = \sum_{k\geq 0} \zeta(k+2)z^{k+1}$, expand formally as a continued fraction using the QD algorithm:

$$\sum_{k\geq 0} \zeta(k+2)z^{k+1} = c(0)z/(1 + c(1)z/(1 + c(2)z/(1 + \cdots))) \; .$$

Gauss–Monien Summation II

We set $N_n(z) = p_{2n-1}(z)$, $D_n(z) = q_{2n-1}(z)$ ($2n-1$st partial quotients of the continued fraction). The latter is the family of orthogonal polynomials for the measure $d\mu$, and we let x_i be the roots of D_n, $w_i = N_n(x_i)/D_n'(x_i)$, and we will have

$$\sum_{m\geq 1} f(1/m)/m^2 \approx \sum_{1\leq i\leq n} w_i f(1/x_i)/x_i^2 \ ,$$

which uppon setting $g(x) = f(1/x)/x^2$ is better written as

$$\sum_{m\geq 1} g(m) \approx \sum_{1\leq i\leq n} w_i g(x_i) \ .$$

Many remarks:

- Because of the above, it is natural to expect that $x_1 \approx 1$, $x_2 \approx 2$, etc... In practice, x_i is not too far from i for $i < n/2$, which reduces considerably the time to compute them as roots of D_n. In fact H. Monien has an even more efficient method to compute them.

Gauss–Monien Summation II

We set $N_n(z) = p_{2n-1}(z)$, $D_n(z) = q_{2n-1}(z)$ ($2n-1$st partial quotients of the continued fraction). The latter is the family of orthogonal polynomials for the measure $d\mu$, and we let x_i be the roots of D_n, $w_i = N_n(x_i)/D_n'(x_i)$, and we will have

$$\sum_{m\geq 1} f(1/m)/m^2 \approx \sum_{1\leq i\leq n} w_i f(1/x_i)/x_i^2 \ ,$$

which uppon setting $g(x) = f(1/x)/x^2$ is better written as

$$\sum_{m\geq 1} g(m) \approx \sum_{1\leq i\leq n} w_i g(x_i) \ .$$

Many remarks:

• Because of the above, it is natural to expect that $x_1 \approx 1$, $x_2 \approx 2$, etc... In practice, x_i is not too far from i for $i < n/2$, which reduces considerably the time to compute them as roots of D_n. In fact H. Monien has an even more efficient method to compute them.

Gauss–Monien Summation III

- As in all Gaussian methods, error estimates exist, but usually much too pessimistic.
- The above assumes implicitly that $g(x)$ has an asymptotic expansion at ∞ of the form $g(x) = a_2/x^2 + a_3/x^3 + \cdots$. It is immediate to generalize to the case $g(x) = \sum_{j\geq 1} a_j/x^{j\alpha+\beta}$ with $\alpha > 0$ whenever the series makes sense, or to any class of functions whose asymptotic behavior at infinity is fixed and known.
- The continued fraction for $\Phi(z) = \sum_{k\geq 0} \zeta(k+2)z^{k+1}$ (which is essentially the logarithmic derivative of the gamma function, but irrelevant) given by the QD algorithm has fascinating properties. It is a consequence of a Riemann–Hilbert problem that the coefficients $c(m)$ of the CF have an asymptotic expansion in $1/n$ with rational coefficients.

Gauss–Monien Summation III

- As in all Gaussian methods, error estimates exist, but usually much too pessimistic.
- The above assumes implicitly that $g(x)$ has an asymptotic expansion at ∞ of the form $g(x) = a_2/x^2 + a_3/x^3 + \cdots$. It is immediate to generalize to the case $g(x) = \sum_{j\geq 1} a_j/x^{j\alpha+\beta}$ with $\alpha > 0$ whenever the series makes sense, or to any class of functions whose asymptotic behavior at infinity is fixed and known.
- The continued fraction for $\Phi(z) = \sum_{k\geq 0} \zeta(k+2)z^{k+1}$ (which is essentially the logarithmic derivative of the gamma function, but irrelevant) given by the QD algorithm has fascinating properties. It is a consequence of a Riemann–Hilbert problem that the coefficients $c(m)$ of the CF have an asymptotic expansion in $1/n$ with rational coefficients.

Gauss–Monien Summation III

- As in all Gaussian methods, error estimates exist, but usually much too pessimistic.
- The above assumes implicitly that $g(x)$ has an asymptotic expansion at ∞ of the form $g(x) = a_2/x^2 + a_3/x^3 + \cdots$. It is immediate to generalize to the case $g(x) = \sum_{j\geq 1} a_j/x^{j\alpha+\beta}$ with $\alpha > 0$ whenever the series makes sense, or to any class of functions whose asymptotic behavior at infinity is fixed and known.
- The continued fraction for $\Phi(z) = \sum_{k\geq 0} \zeta(k+2)z^{k+1}$ (which is essentially the logarithmic derivative of the gamma function, but irrelevant) given by the QD algorithm has fascinating properties. It is a consequence of a Riemann–Hilbert problem that the coefficients $c(m)$ of the CF have an asymptotic expansion in $1/n$ with rational coefficients.

Gauss–Monien Summation IV

- If instead of $\sum_{k\geq 0}\zeta(k+2)z^{k+1}$ we take $\sum_{k\geq 0}(1-2^{-(k+1)})\zeta(k+2)z^{k+1}$ (corresponding to $\sum_{m\geq 1}(-1)^m g(m)$), the coefficients $c(m)$ again have an asymptotic expansion in $1/n$ but nonrational, and I have only been able to determine the first three coefficients in the expansion, the first being the positive solution of $(x/2)\tanh(x/2)=1$.
- Worse, if we take instead $\sum_{k\geq 0}\zeta'(k+2)z^{k+1}$ (corresponding to $\sum_{m\geq 1} g(m)\log(m)$), then one only knows the main term in the asymptotic behavior of $c(m)$, but one does not even know the asymptotics of the "next" term. All this is linked to the estimation of Hankel determinants linked to the zeta function.

Gauss–Monien Summation IV

- If instead of $\sum_{k\geq 0}\zeta(k+2)z^{k+1}$ we take $\sum_{k\geq 0}(1-2^{-(k+1)})\zeta(k+2)z^{k+1}$ (corresponding to $\sum_{m\geq 1}(-1)^m g(m)$), the coefficients $c(m)$ again have an asymptotic expansion in $1/n$ but nonrational, and I have only been able to determine the first three coefficients in the expansion, the first being the positive solution of $(x/2)\tanh(x/2)=1$.
- Worse, if we take instead $\sum_{k\geq 0}\zeta'(k+2)z^{k+1}$ (corresponding to $\sum_{m\geq 1} g(m)\log(m)$), then one only knows the main term in the asymptotic behavior of $c(m)$, but one does not even know the asymptotics of the "next" term. All this is linked to the estimation of Hankel determinants linked to the zeta function.

Discrete Euler–MacLaurin Summation I

A more robust, but slower method is Euler–MacLaurin. Well known: under suitable regularity and convergence assumptions:

$$\sum_{n\geq 1} f(n) = \sum_{1\leq n<N} f(n) + \int_N^\infty f(t)\,dt + f(N)/2 + \sum_{1\leq k\leq p} \frac{B_{2k}}{(2k)!} f^{(2k-1)}(N) + R(N,p)\ ,$$

with a very well controlled error term $R(N,p)$:

If p is chosen appropriately as a function of N, error term of the order of $e^{-2\pi N}$. So for $N = 30$, usually gives more than 80 decimal digits.

Discrete Euler–MacLaurin Summation I

A more robust, but slower method is Euler–MacLaurin. Well known: under suitable regularity and convergence assumptions:

$$\sum_{n\geq 1} f(n) = \sum_{1\leq n<N} f(n) + \int_N^\infty f(t)\,dt + f(N)/2 + \sum_{1\leq k\leq p} \frac{B_{2k}}{(2k)!} f^{(2k-1)}(N) + R(N,p)\ ,$$

with a very well controlled error term $R(N,p)$:

If p is chosen appropriately as a function of N, error term of the order of $e^{-2\pi N}$. So for $N = 30$, usually gives more than 80 decimal digits.

Discrete Euler–MacLaurin Summation II

Computing the integral numerically is usually no problem (see preceding methods). However, computing $f^{(2k-1)}(N)$, especially for k not tiny, is a huge problem. Solution: replace derivatives with discrete differences such as $\Delta(f)(x) = (f(x+\delta) - f(x-\delta))/(2\delta)$.

Easy to iterate (binomial coefficients), replaces Bernoulli numbers B_k by δ-Bernoulli numbers, also easily computed. Convergence is slower than standard Euler–MacLaurin (e^{-cN} with $c = 1$ if $\delta = 1$ for instance instead of $c = 2\pi$), but no need to compute derivatives, so much better than EM in practice. A good choice of δ is $\delta = 1/4$.

Conclusion: If possible, use Gauss–Monien summation, extremely fast. If it fails, discrete Euler–MacLaurin usually works. `Pari/GP` commands: **sumnummonien** and **sumnum**.

Discrete Euler–MacLaurin Summation II

Computing the integral numerically is usually no problem (see preceding methods). However, computing $f^{(2k-1)}(N)$, especially for k not tiny, is a huge problem. Solution: replace derivatives with discrete differences such as $\Delta(f)(x) = (f(x+\delta) - f(x-\delta))/(2\delta)$.

Easy to iterate (binomial coefficients), replaces Bernoulli numbers B_k by δ-Bernoulli numbers, also easily computed. Convergence is slower than standard Euler–MacLaurin (e^{-cN} with $c = 1$ if $\delta = 1$ for instance instead of $c = 2\pi$), but no need to compute derivatives, so much better than EM in practice. A good choice of δ is $\delta = 1/4$.

Conclusion: If possible, use Gauss–Monien summation, extremely fast. If it fails, discrete Euler–MacLaurin usually works. `Pari/GP` commands: **sumnummonien** and **sumnum**.

Discrete Euler–MacLaurin Summation II

Computing the integral numerically is usually no problem (see preceding methods). However, computing $f^{(2k-1)}(N)$, especially for k not tiny, is a huge problem. Solution: replace derivatives with discrete differences such as $\Delta(f)(x) = (f(x+\delta) - f(x-\delta))/(2\delta)$.

Easy to iterate (binomial coefficients), replaces Bernoulli numbers B_k by δ-Bernoulli numbers, also easily computed. Convergence is slower than standard Euler–MacLaurin (e^{-cN} with $c = 1$ if $\delta = 1$ for instance instead of $c = 2\pi$), but no need to compute derivatives, so much better than EM in practice. A good choice of δ is $\delta = 1/4$.

Conclusion: If possible, use Gauss–Monien summation, extremely fast. If it fails, discrete Euler–MacLaurin usually works. `Pari/GP` commands: **sumnummonien** and **sumnum**.

Discrete Euler–MacLaurin Summation II

Computing the integral numerically is usually no problem (see preceding methods). However, computing $f^{(2k-1)}(N)$, especially for k not tiny, is a huge problem. Solution: replace derivatives with discrete differences such as $\Delta(f)(x) = (f(x+\delta) - f(x-\delta))/(2\delta)$.

Easy to iterate (binomial coefficients), replaces Bernoulli numbers B_k by δ-Bernoulli numbers, also easily computed. Convergence is slower than standard Euler–MacLaurin (e^{-cN} with $c = 1$ if $\delta = 1$ for instance instead of $c = 2\pi$), but no need to compute derivatives, so much better than EM in practice. A good choice of δ is $\delta = 1/4$.

Conclusion: If possible, use Gauss–Monien summation, extremely fast. If it fails, discrete Euler–MacLaurin usually works. `Pari/GP` commands: **sumnummonien** and **sumnum**.

Summation of Alternating Series I

Quite old method, put in this form by F. Rodriguez-Villegas, D. Zagier, and myself. Idea: write $f(n) = \int_0^1 x^n w(x)\,dx$ for some measure $w(x)dx$, assumed positive (works in fact in much more general cases). We have $S = \sum_{n\geq 0}(-1)^n f(n) = \int_0^1 (w(x)/(1+x))\,dx$, so for any polynomial P_N such that $P_N(-1) \neq 0$ we have

$$S = \frac{1}{P_N(-1)} \sum_{0\leq j\leq N-1} c_{N,j} u(j) + R_N \ , \text{ with } |R_N| \leq \frac{\sup_{x\in[0,1]} |P_N(x)|}{|P_N(-1)|} S \ ,$$

where

$$\frac{P_N(-1) - P_N(X)}{X+1} = \sum_{0\leq j\leq N-1} c_{N,j} X^j \ .$$

Summation of Alternating Series II

A good choice is the shifted Chebyshev polynomial $P_N(X) = T_N(1-2X)$ for which the relative error $|R_N/S|$ satisfies $|R_N/S| \leq (3+2\sqrt{2})^{-N}$, so it is immediate to determine that $N = 1.31D$. An additional advantage of these polynomials is that the coefficients $c_{N,j}$ can be computed "on the fly" (for certain classes of series, better choices exist).

For $D = 1000$, the program requires between 10 milliseconds and 1 second depending on the complexity of computing the summand.

Note that alternating summation methods can usually correctly compute the "sum" of nonconvergent series such as $\sum_{n\geq 1}(-1)^n \log(n)$ $(= \log(\pi/2)/2)$ or $\sum_{n\geq 1}(-1)^n n$ $(= -1/4)$.

`Pari/GP` command: **sumalt**.

Summation of Alternating Series II

A good choice is the shifted Chebyshev polynomial $P_N(X) = T_N(1-2X)$ for which the relative error $|R_N/S|$ satisfies $|R_N/S| \leq (3+2\sqrt{2})^{-N}$, so it is immediate to determine that $N = 1.31D$. An additional advantage of these polynomials is that the coefficients $c_{N,j}$ can be computed "on the fly" (for certain classes of series, better choices exist). For $D = 1000$, the program requires between 10 milliseconds and 1 second depending on the complexity of computing the summand.

Note that alternating summation methods can usually correctly compute the "sum" of nonconvergent series such as $\sum_{n\geq 1}(-1)^n \log(n)$ $(= \log(\pi/2)/2)$ or $\sum_{n\geq 1}(-1)^n n$ $(= -1/4)$.

`Pari/GP` command: **sumalt**.

Summation of Alternating Series II

A good choice is the shifted Chebyshev polynomial $P_N(X) = T_N(1-2X)$ for which the relative error $|R_N/S|$ satisfies $|R_N/S| \leq (3+2\sqrt{2})^{-N}$, so it is immediate to determine that $N = 1.31D$. An additional advantage of these polynomials is that the coefficients $c_{N,j}$ can be computed "on the fly" (for certain classes of series, better choices exist). For $D = 1000$, the program requires between 10 milliseconds and 1 second depending on the complexity of computing the summand.

Note that alternating summation methods can usually correctly compute the "sum" of nonconvergent series such as $\sum_{n\geq 1}(-1)^n \log(n)$ $(= \log(\pi/2)/2)$ or $\sum_{n\geq 1}(-1)^n n$ $(= -1/4)$.

`Pari/GP` command: **sumalt**.

Summation of Alternating Series II

A good choice is the shifted Chebyshev polynomial $P_N(X) = T_N(1-2X)$ for which the relative error $|R_N/S|$ satisfies $|R_N/S| \leq (3+2\sqrt{2})^{-N}$, so it is immediate to determine that $N = 1.31D$. An additional advantage of these polynomials is that the coefficients $c_{N,j}$ can be computed "on the fly" (for certain classes of series, better choices exist). For $D = 1000$, the program requires between 10 milliseconds and 1 second depending on the complexity of computing the summand.

Note that alternating summation methods can usually correctly compute the "sum" of nonconvergent series such as $\sum_{n\geq 1}(-1)^n \log(n)$ $(= \log(\pi/2)/2)$ or $\sum_{n\geq 1}(-1)^n n$ $(= -1/4)$.

`Pari/GP` command: **sumalt**.

Summation: Examples I

Examples at 500 D:

- $S_1 = \sum_{n\geq 1} \log(\Gamma(1+1/n))/n$. Discrete Euler–MacLaurin `sumnum` requires 15.8 seconds, but `sumnummonien` only requires 0.98 second.
- $S_2 = \sum_{n\geq 1} (\mathrm{Li}_2(1/n) - 1/n)$. Discrete Euler–MacLaurin `sumnum` requires 1.48 seconds, but `sumnummonien` only requires 0.19 second.
- The double sum $S_3 = \sum_{n\geq 1} \sum_{m\geq 1} 1/(m^2n^2+1)$: Discrete Euler–MacLaurin `sumnum` requires 202 seconds, but `sumnummonien` only requires 0.3 second.

Thus, as for double integrals, double sums are computable if they can be computed using the Gaussian method `sumnummonien`.

Summation: Examples I

Examples at 500 D:

- $S_1 = \sum_{n\geq 1} \log(\Gamma(1+1/n))/n$. Discrete Euler–MacLaurin `sumnum` requires 15.8 seconds, but `sumnummonien` only requires 0.98 second.
- $S_2 = \sum_{n\geq 1} (\mathrm{Li}_2(1/n) - 1/n)$. Discrete Euler–MacLaurin `sumnum` requires 1.48 seconds, but `sumnummonien` only requires 0.19 second.
- The double sum $S_3 = \sum_{n\geq 1} \sum_{m\geq 1} 1/(m^2n^2+1)$: Discrete Euler–MacLaurin `sumnum` requires 202 seconds, but `sumnummonien` only requires 0.3 second.

Thus, as for double integrals, double sums are computable if they can be computed using the Gaussian method `sumnummonien`.

Summation: Examples I

Examples at 500 D:

- $S_1 = \sum_{n\geq 1} \log(\Gamma(1+1/n))/n$. Discrete Euler–MacLaurin `sumnum` requires 15.8 seconds, but `sumnummonien` only requires 0.98 second.
- $S_2 = \sum_{n\geq 1} (\mathrm{Li}_2(1/n) - 1/n)$. Discrete Euler–MacLaurin `sumnum` requires 1.48 seconds, but `sumnummonien` only requires 0.19 second.
- The double sum $S_3 = \sum_{n\geq 1} \sum_{m\geq 1} 1/(m^2n^2+1)$: Discrete Euler–MacLaurin `sumnum` requires 202 seconds, but `sumnummonien` only requires 0.3 second.

Thus, as for double integrals, double sums are computable if they can be computed using the Gaussian method `sumnummonien`.

Summation: Examples I

Examples at 500 D:

- $S_1 = \sum_{n\geq 1} \log(\Gamma(1+1/n))/n$. Discrete Euler–MacLaurin `sumnum` requires 15.8 seconds, but `sumnummonien` only requires 0.98 second.
- $S_2 = \sum_{n\geq 1} (\mathrm{Li}_2(1/n) - 1/n)$. Discrete Euler–MacLaurin `sumnum` requires 1.48 seconds, but `sumnummonien` only requires 0.19 second.
- The double sum $S_3 = \sum_{n\geq 1} \sum_{m\geq 1} 1/(m^2n^2+1)$: Discrete Euler–MacLaurin `sumnum` requires 202 seconds, but `sumnummonien` only requires 0.3 second.

Thus, as for double integrals, double sums are computable if they can be computed using the Gaussian method `sumnummonien`.

Summation: Examples II

- The double sum $S_4 = \sum_{n\geq 1} \sum_{m\geq 1} 1/(m^4+n^4)$ cannot be computed correctly with any method presently implemented (when A is large, say $A = 10^{12}$, the asymptotic expansion used for $\sum_{m\geq 1} 1/(m^4+A)$ is completely off the mark).
- The contrived example $S_5 = \sum_{n\geq 1} 1/(n^\pi + n^{1.4} + 1)$ is computed perfectly in 1.58 seconds by `sumnum`, but is totally out of range of `sumnummonien`.
- The geometrically converging sum $S_6 = \sum_{n\geq 2} \zeta'(n)$ is also out of range for `sumnummonien`. It can be computed by `sumnum`, although quite slowly (85 seconds), but since it converges geometrically, simpler and faster simply to sum enough terms (28 seconds).

Summation: Examples II

- The double sum $S_4 = \sum_{n\geq 1} \sum_{m\geq 1} 1/(m^4+n^4)$ cannot be computed correctly with any method presently implemented (when A is large, say $A = 10^{12}$, the asymptotic expansion used for $\sum_{m\geq 1} 1/(m^4+A)$ is completely off the mark).
- The contrived example $S_5 = \sum_{n\geq 1} 1/(n^\pi + n^{1.4} + 1)$ is computed perfectly in 1.58 seconds by `sumnum`, but is totally out of range of `sumnummonien`.
- The geometrically converging sum $S_6 = \sum_{n\geq 2} \zeta'(n)$ is also out of range for `sumnummonien`. It can be computed by `sumnum`, although quite slowly (85 seconds), but since it converges geometrically, simpler and faster simply to sum enough terms (28 seconds).

Summation: Examples II

- The double sum $S_4 = \sum_{n\geq 1} \sum_{m\geq 1} 1/(m^4+n^4)$ cannot be computed correctly with any method presently implemented (when A is large, say $A = 10^{12}$, the asymptotic expansion used for $\sum_{m\geq 1} 1/(m^4+A)$ is completely off the mark).
- The contrived example $S_5 = \sum_{n\geq 1} 1/(n^\pi + n^{1.4} + 1)$ is computed perfectly in 1.58 seconds by `sumnum`, but is totally out of range of `sumnummonien`.
- The geometrically converging sum $S_6 = \sum_{n\geq 2} \zeta'(n)$ is also out of range for `sumnummonien`. It can be computed by `sumnum`, although quite slowly (85 seconds), but since it converges geometrically, simpler and faster simply to sum enough terms (28 seconds).

Extrapolation, Asymptotic Expansions I

Problem: compute $\lim_{n\to\infty} f(n)$, or a number of terms of the asymptotic expansion of f at infinity, assuming regularity (typically $f(n) = a_0 + a_1/n + a_2/n^2 + \cdots$, but can be more general such as $\sum_{j\geq 0} a_j/n^{j\alpha+\beta}$).

Two equivalent methods: Lagrange interpolation (variable $x = 1/n$), or a method popularized by D. Zagier. Lagrange interpolation is straightforward: find a degree N polynomial P_N such that $f(n) = P_N(1/n)$ for $n = 1, 2,...$, or better, for $n = 1000, 1010, ...$, so the limit is close to $P_N(0)$.

Extrapolation, Asymptotic Expansions I

Problem: compute $\lim_{n\to\infty} f(n)$, or a number of terms of the asymptotic expansion of f at infinity, assuming regularity (typically $f(n) = a_0 + a_1/n + a_2/n^2 + \cdots$, but can be more general such as $\sum_{j\geq 0} a_j/n^{j\alpha+\beta}$).

Two equivalent methods: Lagrange interpolation (variable $x = 1/n$), or a method popularized by D. Zagier. Lagrange interpolation is straightforward: find a degree N polynomial P_N such that $f(n) = P_N(1/n)$ for $n = 1, 2,...$, or better, for $n = 1000, 1010, ...$, so the limit is close to $P_N(0)$.

Extrapolation, Asymptotic Expansions II

Zagier's method (which is equivalent) is this: choose some small k (say $k = 10$), and let

$$g(n) = n^k f(n) = a_0 n^k + a_1 n^{k-1} + \cdots + a_k + a_{k+1}/n + \cdots .$$

Then take the kth forward difference $\Delta^k g$ ($\Delta(h)(n) = h(n+1) - h(n)$). Then $(\Delta^k g)(n) = a_0 k! + O(1/n^{k+1})$, since Δ^k of a polynomial of degree less than k vanishes. We can thus recover a_0 with good accuracy.

In practice, apply to $g(n + 1000)$ for instance. Easily generalized to asymptotic expansions, and extremely fast.

`Pari/GP` commands: **limitnum** and **asympnum**.

Extrapolation, Asymptotic Expansions II

Zagier's method (which is equivalent) is this: choose some small k (say $k = 10$), and let

$$g(n) = n^k f(n) = a_0 n^k + a_1 n^{k-1} + \cdots + a_k + a_{k+1}/n + \cdots .$$

Then take the kth forward difference $\Delta^k g$ ($\Delta(h)(n) = h(n+1) - h(n)$). Then $(\Delta^k g)(n) = a_0 k! + O(1/n^{k+1})$, since Δ^k of a polynomial of degree less than k vanishes. We can thus recover a_0 with good accuracy.

In practice, apply to $g(n + 1000)$ for instance. Easily generalized to asymptotic expansions, and extremely fast.

`Pari/GP` commands: **limitnum** and **asympnum**.

Extrapolation, Asymptotic Expansions II

Zagier's method (which is equivalent) is this: choose some small k (say $k = 10$), and let

$$g(n) = n^k f(n) = a_0 n^k + a_1 n^{k-1} + \cdots + a_k + a_{k+1}/n + \cdots .$$

Then take the kth forward difference $\Delta^k g$ ($\Delta(h)(n) = h(n+1) - h(n)$). Then $(\Delta^k g)(n) = a_0 k! + O(1/n^{k+1})$, since Δ^k of a polynomial of degree less than k vanishes. We can thus recover a_0 with good accuracy.

In practice, apply to $g(n + 1000)$ for instance. Easily generalized to asymptotic expansions, and extremely fast.

`Pari/GP` commands: **limitnum** and **asympnum**.

Multiple Zeta Values and Multiple Polylogs I

Recall

$$L(a_1,\dots,a_r;z_1,\dots,z_r)=\sum_{n_1>n_2>\cdots>n_r}\frac{z_1^{n_1}\cdots z_r^{n_r}}{n_1^{a_1}\cdots n_r^{a_r}}\ ,$$

and in particular

$$\zeta(a_1,\dots,a_r)=L(a_1,\dots,a_r;1,\dots,1)\ ,$$

multiple polylogs and multi zeta values (MZV).

A large number of linear and nonlinear relations between these numbers, need to be understood: must compute them sometimes to thousands of decimals. They have been computed in various ways. Two very simple algorithms have been devised by the young Indian mathematician P. Akhilesh. One is quite elementary to prove, and is based on the notion of double tails.

Multiple Zeta Values and Multiple Polylogs I

Recall

$$L(a_1,\dots,a_r;z_1,\dots,z_r)=\sum_{n_1>n_2>\cdots>n_r}\frac{z_1^{n_1}\cdots z_r^{n_r}}{n_1^{a_1}\cdots n_r^{a_r}}\ ,$$

and in particular

$$\zeta(a_1,\dots,a_r)=L(a_1,\dots,a_r;1,\dots,1)\ ,$$

multiple polylogs and multi zeta values (MZV).

A large number of linear and nonlinear relations between these numbers, need to be understood: must compute them sometimes to thousands of decimals. They have been computed in various ways. Two very simple algorithms have been devised by the young Indian mathematician P. Akhilesh. One is quite elementary to prove, and is based on the notion of double tails.

Multiple Zeta Values and Multiple Polylogs II

For simplicity, restrict to MZV's (in which case need $a_1 > 1$ for convergence). Define the double tail

$$\zeta_{m,n}(a_1,\dots,a_r) = \sum_{n_1>n_2>\cdots n_r>n} \frac{1}{\binom{n_1+m}{m} n_1^{a_1}\cdots n_r^{a_r}} .$$

Can easily prove very simple linear recurrence relations when m or n varies, and deduce an extremely fast algorithm to compute all MZV's (more generally all multiple polylogs) up to a given weight $w = a_1 + \cdots + a_r$, and even for a single MZV very efficient (typically 0.1 second at 1000 D).

Note that these recurrence relation use a generalization of a remark by M. Kontsevich that MZV's are simply iterated integrals. Leads to identities such as

$$\sum_{n>m>0} \frac{(-1)^{n+m-1}}{nm^2} = \sum_{n>m>0} \frac{2^{-m}}{n^2 m}$$

perhaps known to Euler.

Multiple Zeta Values and Multiple Polylogs II

For simplicity, restrict to MZV's (in which case need $a_1 > 1$ for convergence). Define the double tail

$$\zeta_{m,n}(a_1,\dots,a_r) = \sum_{n_1>n_2>\cdots n_r>n} \frac{1}{\binom{n_1+m}{m} n_1^{a_1}\cdots n_r^{a_r}} .$$

Can easily prove very simple linear recurrence relations when m or n varies, and deduce an extremely fast algorithm to compute all MZV's (more generally all multiple polylogs) up to a given weight $w = a_1 + \cdots + a_r$, and even for a single MZV very efficient (typically 0.1 second at 1000 D).

Note that these recurrence relation use a generalization of a remark by M. Kontsevich that MZV's are simply iterated integrals. Leads to identities such as

$$\sum_{n>m>0} \frac{(-1)^{n+m-1}}{nm^2} = \sum_{n>m>0} \frac{2^{-m}}{n^2 m}$$

perhaps known to Euler.

Multiple Zeta Values and Multiple Polylogs III

A deeper theorem of Akhilesh gives a nonrecursive formula for individual MZV's (work in progress: generalize to polylogs), ten page proof. This formula is even more efficient than the above, to compute a single MZV.

Warning: For the moment, the formulas for multiple polylogs do not converge when the z_i are close (but not on) the unit circle. Do not know how to repair this for now. On the other hand, MZV's or alternating MZV's ($z_i = \pm 1$) are perfectly OK.

`Pari/GP` commands: **zetamult**, **zetamultall**, **polylogmult**, etc...

Multiple Zeta Values and Multiple Polylogs III

A deeper theorem of Akhilesh gives a nonrecursive formula for individual MZV's (work in progress: generalize to polylogs), ten page proof. This formula is even more efficient than the above, to compute a single MZV.

Warning: For the moment, the formulas for multiple polylogs do not converge when the z_i are close (but not on) the unit circle. Do not know how to repair this for now. On the other hand, MZV's or alternating MZV's ($z_i = \pm 1$) are perfectly OK.

`Pari/GP` commands: **zetamult**, **zetamultall**, **polylogmult**, etc...

Multiple Zeta Values and Multiple Polylogs III

A deeper theorem of Akhilesh gives a nonrecursive formula for individual MZV's (work in progress: generalize to polylogs), ten page proof. This formula is even more efficient than the above, to compute a single MZV.

Warning: For the moment, the formulas for multiple polylogs do not converge when the z_i are close (but not on) the unit circle. Do not know how to repair this for now. On the other hand, MZV's or alternating MZV's ($z_i = \pm 1$) are perfectly OK.

`Pari/GP` commands: **zetamult**, **zetamultall**, **polylogmult**, etc...

Computing Inverse Mellin Transforms I

Recall: if f is a nice function tending to 0 exponentially at infinity, its Mellin transform is $\mathcal{M}(f)(s) = \int_0^\infty t^s f(t)\,dt/t$. Variant of Fourier or Laplace transform. If $g = \mathcal{M}(f)$, to recover f we have the Mellin inversion formula (variant of Fourier inversion)

$$f(t) = \frac{1}{2\pi i}\int_C t^{-s} g(s)\,ds\,,$$

where C is a suitable contour (usually a vertical line to the right of the poles of $g(s)$, but can be other). Note that $g(s)$ usually tends to 0 exponentially fast on C.

Computing such inverse transforms is not so much interesting per se, but is essential for computing L-functions.

At least four methods available: power series expansions, asymptotic expansions and continued fractions, Riemann sums, doubly-exponential integration methods (Gaussian integration impossible since exponential decrease). Will mention the first two.

Computing Inverse Mellin Transforms I

Recall: if f is a nice function tending to 0 exponentially at infinity, its Mellin transform is $\mathcal{M}(f)(s) = \int_0^\infty t^s f(t)\,dt/t$. Variant of Fourier or Laplace transform. If $g = \mathcal{M}(f)$, to recover f we have the Mellin inversion formula (variant of Fourier inversion)

$$f(t) = \frac{1}{2\pi i}\int_C t^{-s} g(s)\,ds\,,$$

where C is a suitable contour (usually a vertical line to the right of the poles of $g(s)$, but can be other). Note that $g(s)$ usually tends to 0 exponentially fast on C.

Computing such inverse transforms is not so much interesting per se, but is essential for computing L-functions.

At least four methods available: power series expansions, asymptotic expansions and continued fractions, Riemann sums, doubly-exponential integration methods (Gaussian integration impossible since exponential decrease). Will mention the first two.

Computing Inverse Mellin Transforms I

Recall: if f is a nice function tending to 0 exponentially at infinity, its Mellin transform is $\mathcal{M}(f)(s) = \int_0^\infty t^s f(t)\,dt/t$. Variant of Fourier or Laplace transform. If $g = \mathcal{M}(f)$, to recover f we have the Mellin inversion formula (variant of Fourier inversion)

$$f(t) = \frac{1}{2\pi i}\int_C t^{-s} g(s)\,ds\,,$$

where C is a suitable contour (usually a vertical line to the right of the poles of $g(s)$, but can be other). Note that $g(s)$ usually tends to 0 exponentially fast on C.

Computing such inverse transforms is not so much interesting per se, but is essential for computing L-functions.

At least four methods available: power series expansions, asymptotic expansions and continued fractions, Riemann sums, doubly-exponential integration methods (Gaussian integration impossible since exponential decrease). Will mention the first two.

Computing Inverse Mellin Transforms II

If we push the line of integration C towards $\Re(s) = -\infty$, we catch the poles of $g(s)$, and obtain a generalized power series expansion for $f(t)$, generalized because it may involve powers of $\log(t)$.

Typical and simplest example: $g(s) = \Gamma(s)$, then $f(t) = e^{-t}$ and we obtain the usual power series of e^{-t}. This is typical: the power series has infinite radius of convergence, so could be used for any t to compute $f(t)$. **But**, as in the case of e^{-t}, you do not want to do this if t is large (even $t = 100$ is large).

Computing Inverse Mellin Transforms II

If we push the line of integration C towards $\Re(s) = -\infty$, we catch the poles of $g(s)$, and obtain a generalized power series expansion for $f(t)$, generalized because it may involve powers of $\log(t)$.

Typical and simplest example: $g(s) = \Gamma(s)$, then $f(t) = e^{-t}$ and we obtain the usual power series of e^{-t}. This is typical: the power series has infinite radius of convergence, so could be used for any t to compute $f(t)$. **But**, as in the case of e^{-t}, you do not want to do this if t is large (even $t = 100$ is large).

Computing Inverse Mellin Transforms III

Solution for t large: push the line of integration towards $+\infty$. We then obtain a power series in $1/t$, which has zero radius of convergence, i.e., is an asymptotic expansion. If t is very large, OK, but if t is medium (such as $t = 100$), not sufficient accuracy.

T. Dokshitser's trick (2002): transform (using the Quotient-Difference algorithm) the asymptotic expansion into a continued fraction. "Miracle": this CF converges for all $t > 1$, and subexponentially ($e^{-Ct^{1/d}}$ for a known d). So all our problems are solved, yes ?

Computing Inverse Mellin Transforms III

Solution for t large: push the line of integration towards $+\infty$. We then obtain a power series in $1/t$, which has zero radius of convergence, i.e., is an asymptotic expansion. If t is very large, OK, but if t is medium (such as $t = 100$), not sufficient accuracy.

T. Dokshitser's trick (2002): transform (using the Quotient-Difference algorithm) the asymptotic expansion into a continued fraction. "Miracle": this CF converges for all $t > 1$, and subexponentially ($e^{-Ct^{1/d}}$ for a known d). So all our problems are solved, yes ?

Computing Inverse Mellin Transforms IV

Unfortunately, no one has any idea how to prove this, even in the simplest cases: for $g(s) = \Gamma(s)$, inverse Mellin is e^{-t}, the asymptotic series is reduced to 1. For $g(s) = \Gamma(s)^2$, inverse Mellin is $2K_0(2\sqrt{t})$ (K-Bessel function), asymptotic series well-known, continued fraction can be proved to converge subexponentially.

But for $g(s) = \Gamma(s)^3$, say? The CF coefficients seem totally random (random size, random sign, etc...), nonetheless the CF converges subexponentially. If anyone has any idea, please tell us.

`Pari/GP` commands: **gammamellininv**, **gammamellininvasymp** (only for $g(s)$ a gamma product).

Computing Inverse Mellin Transforms IV

Unfortunately, no one has any idea how to prove this, even in the simplest cases: for $g(s) = \Gamma(s)$, inverse Mellin is e^{-t}, the asymptotic series is reduced to 1. For $g(s) = \Gamma(s)^2$, inverse Mellin is $2K_0(2\sqrt{t})$ (K-Bessel function), asymptotic series well-known, continued fraction can be proved to converge subexponentially.

But for $g(s) = \Gamma(s)^3$, say? The CF coefficients seem totally random (random size, random sign, etc...), nonetheless the CF converges subexponentially. If anyone has any idea, please tell us.

Pari/GP commands: **gammamellininv**, **gammamellininvasymp** (only for $g(s)$ a gamma product).

Computing Inverse Mellin Transforms IV

Unfortunately, no one has any idea how to prove this, even in the simplest cases: for $g(s) = \Gamma(s)$, inverse Mellin is e^{-t}, the asymptotic series is reduced to 1. For $g(s) = \Gamma(s)^2$, inverse Mellin is $2K_0(2\sqrt{t})$ (K-Bessel function), asymptotic series well-known, continued fraction can be proved to converge subexponentially.

But for $g(s) = \Gamma(s)^3$, say? The CF coefficients seem totally random (random size, random sign, etc...), nonetheless the CF converges subexponentially. If anyone has any idea, please tell us.

`Pari/GP` commands: **gammamellininv**, **gammamellininvasymp** (only for $g(s)$ a gamma product).

Computing L-Functions I

Not surprisingly, most sophisticated among the algorithms mentioned. Want to do numerical work on L-functions (almost always motivic) of (in principle) arbitrary large degree, of course limited by speed and storage considerations.

Many people have worked on this subject, and probably the most general available implementation is that of T. Dokshitser in `magma`, and also M. Rubinstein's lcalc package.

First basic tool need is inverse Mellin transforms of gamma products, see above.

Computing L-Functions I

Not surprisingly, most sophisticated among the algorithms mentioned. Want to do numerical work on L-functions (almost always motivic) of (in principle) arbitrary large degree, of course limited by speed and storage considerations.

Many people have worked on this subject, and probably the most general available implementation is that of T. Dokshitser in `magma`, and also M. Rubinstein's lcalc package.

First basic tool need is inverse Mellin transforms of gamma products, see above.

Computing L-Functions I

Not surprisingly, most sophisticated among the algorithms mentioned. Want to do numerical work on L-functions (almost always motivic) of (in principle) arbitrary large degree, of course limited by speed and storage considerations.

Many people have worked on this subject, and probably the most general available implementation is that of T. Dokshitser in `magma`, and also M. Rubinstein's lcalc package.

First basic tool need is inverse Mellin transforms of gamma products, see above.

Computing L-functions II

Second, the core program, is the computation of the values of the L-functions themselves. There are (at least) two ways to consider this, depending on the goal.

1. Using smoothed versions of the approximate functional equation. Needs generalized incomplete gamma functions, is especially well suited to large scale computations in the critical strip, for instance to check GRH. Extensively developped by M. Rubinstein in a very nice and detailed paper and in different versions of his lcalc program. I refer to his paper for a description.
2. Using Poisson summation directly: this idea is due to A. Booker, and has been extensively developed by P. Molin. Even though it is applicable to rather high values in the critical strip, it is very well suited to the computation of L-values in reasonable ranges. One of its great advantages is that one does not need the approximate functional equation, only inverse Mellin transforms.

Computing L-functions II

Second, the core program, is the computation of the values of the L-functions themselves. There are (at least) two ways to consider this, depending on the goal.

1. Using smoothed versions of the approximate functional equation. Needs generalized incomplete gamma functions, is especially well suited to large scale computations in the critical strip, for instance to check GRH. Extensively developped by M. Rubinstein in a very nice and detailed paper and in different versions of his lcalc program. I refer to his paper for a description.
2. Using Poisson summation directly: this idea is due to A. Booker, and has been extensively developed by P. Molin. Even though it is applicable to rather high values in the critical strip, it is very well suited to the computation of L-values in reasonable ranges. One of its great advantages is that one does not need the approximate functional equation, only inverse Mellin transforms.

Computing L-functions III

Booker–Molin's idea boils down to a very simple formula, directly from Poisson summation: assume $L(s)$ has no poles, let $\Lambda(s) = \gamma(s)L(s)$ be the completed L-function with exponential and gamma factors, and $K(t)$ the inverse Mellin transform of $\gamma(s)$. For all $h > 0$ and $s \in \mathbb{C}$ we have the identity

$$\Lambda(s) = h \sum_{m \in \mathbb{Z}} e^{mhs} \Theta(e^{mh}) - \sum_{k \neq 0} \Lambda(s + 2\pi ik/h) \ ,$$

where $\Theta(t) = \sum_{n \geq 1} a(n)K(nt)$ (trivial modifications if $L(s)$ has poles).

This is nice for three reasons: (1) $K(t)$ decreases exponentially to 0 at a known rate, so $\Theta(t)$ is computed fast. (2) $\Theta(e^{mh})$ can be computed once and for all in a precomputation. (3) Since $\gamma(s)$ tends to 0 exponentially fast on vertical strips, if h is small ($h = 0.1$ is OK, need not be too small) $\Lambda(s + 2\pi ik/h)$ is exponentially small if $k \neq 0$.

Computing L-functions III

Booker–Molin's idea boils down to a very simple formula, directly from Poisson summation: assume $L(s)$ has no poles, let $\Lambda(s) = \gamma(s)L(s)$ be the completed L-function with exponential and gamma factors, and $K(t)$ the inverse Mellin transform of $\gamma(s)$. For all $h > 0$ and $s \in \mathbb{C}$ we have the identity

$$\Lambda(s) = h \sum_{m \in \mathbb{Z}} e^{mhs} \Theta(e^{mh}) - \sum_{k \neq 0} \Lambda(s + 2\pi ik/h) \ ,$$

where $\Theta(t) = \sum_{n \geq 1} a(n)K(nt)$ (trivial modifications if $L(s)$ has poles).

This is nice for three reasons: (1) $K(t)$ decreases exponentially to 0 at a known rate, so $\Theta(t)$ is computed fast. (2) $\Theta(e^{mh})$ can be computed once and for all in a precomputation. (3) Since $\gamma(s)$ tends to 0 exponentially fast on vertical strips, if h is small ($h = 0.1$ is OK, need not be too small) $\Lambda(s + 2\pi ik/h)$ is exponentially small if $k \neq 0$.

Computing L-functions IV

After the core program comes application programs such as computing zeros on the critical line and plotting. But often need guessing programs: the root number and polar parts easy. Finding unknown factors of the conductor or missing Euler factors (when there is an Euler product) more difficult, but can be done quite easily if not too many.

Note that if you only know the gamma factor and the conductor, one can often determine whether or not there are corresponding L-functions, and compute some coefficients: see e.g., work of D. Farmer et al.

Computing L-functions IV

After the core program comes application programs such as computing zeros on the critical line and plotting. But often need guessing programs: the root number and polar parts easy. Finding unknown factors of the conductor or missing Euler factors (when there is an Euler product) more difficult, but can be done quite easily if not too many.

Note that if you only know the gamma factor and the conductor, one can often determine whether or not there are corresponding L-functions, and compute some coefficients: see e.g., work of D. Farmer et al.

Computing L-functions V

Finally, we have implemented a large number of utility programs for the most common type of L-functions: Dirichlet L-functions, L-functions of Hecke characters (of finite order for now), Artin L-functions (last week!), Dedekind zeta functions, L-functions of elliptic curves, of eta quotients, of lattices, and soon of modular forms and of genus 2 curves.

Also specific programs for special values (quadratic characters, modular forms), symmetric square of elliptic curves and modular forms (including special values), Petersson square, etc...

Computing L-functions V

Finally, we have implemented a large number of utility programs for the most common type of L-functions: Dirichlet L-functions, L-functions of Hecke characters (of finite order for now), Artin L-functions (last week!), Dedekind zeta functions, L-functions of elliptic curves, of eta quotients, of lattices, and soon of modular forms and of genus 2 curves.

Also specific programs for special values (quadratic characters, modular forms), symmetric square of elliptic curves and modular forms (including special values), Petersson square, etc...

Computing L-functions VI

For now, at least 23 functions dealing with L-functions, most beginning with the prefix **lfun** are available in `Pari/GP`. Can be downloaded if you have the `GIT` program, otherwise need to wait for next release in January.

Inverse Mellin Transforms I

```
? gammamellininvasymp([0,0,0],9)
%1 = [1, -1/9, 2/81, -14/2187, 8/19683, 440/177147,
      -21520/4782969, 268240/43046721, -2898560/387420489]

? M=gammamellininvasymp([0,0,0],200);
? N=contfracinit(M*1.);
? contfraceval(N,0.2,50)
%4 = 0.97871544142464955957930733267300399652

? contfraceval(N,0.2,100)
%5 = 0.97871544142464955957930733267300399652

? sqrt(16/3)*5^(-2/3)*exp(-3*Pi*5^(2/3))*
contfraceval(N,1/(Pi*5^(2/3)),100)
%6 = 8.3941615239730609987666907762277846680E-13
```

Inverse Mellin Transforms I

```
? gammamellininvasymp([0,0,0],9)
```

$$\%1 = [1, -1/9, 2/81, -14/2187, 8/19683, 440/177147, -21520/4782969, 268240/43046721, -2898560/387420489]$$

```
? M=gammamellininvasymp([0,0,0],200);
? N=contfracinit(M*1.);
? contfraceval(N,0.2,50)
```

$$\%4 = 0.97871544142464955957930733267300399652$$

```
? contfraceval(N,0.2,100)
```

$$\%5 = 0.97871544142464955957930733267300399652$$

```
? sqrt(16/3)*5^(-2/3)*exp(-3*Pi*5^(2/3))*
contfraceval(N,1/(Pi*5^(2/3)),100)
```

$$\%6 = 8.3941615239730609987666907762277846680E-13$$

Inverse Mellin Transforms I

```
? gammamellininvasymp([0,0,0],9)
```

$$\%1 = [1, -1/9, 2/81, -14/2187, 8/19683, 440/177147, -21520/4782969, 268240/43046721, -2898560/387420489]$$

```
? M=gammamellininvasymp([0,0,0],200);
? N=contfracinit(M*1.);
? contfraceval(N,0.2,50)
```

$$\%4 = 0.97871544142464955957930733267300399652$$

```
? contfraceval(N,0.2,100)
```

$$\%5 = 0.97871544142464955957930733267300399652$$

```
? sqrt(16/3)*5^(-2/3)*exp(-3*Pi*5^(2/3))*
contfraceval(N,1/(Pi*5^(2/3)),100)
```

$$\%6 = 8.3941615239730609987666907762277846680E-13$$

Inverse Mellin Transforms I

```
? gammamellininvasymp([0,0,0],9)
```

$$\%1 = [1, -1/9, 2/81, -14/2187, 8/19683, 440/177147, -21520/4782969, 268240/43046721, -2898560/387420489]$$

```
? M=gammamellininvasymp([0,0,0],200);
? N=contfracinit(M*1.);
? contfraceval(N,0.2,50)
```

$$\%4 = 0.97871544142464955957930733267300399652$$

```
? contfraceval(N,0.2,100)
```

$$\%5 = 0.97871544142464955957930733267300399652$$

```
? sqrt(16/3)*5^(-2/3)*exp(-3*Pi*5^(2/3))*
contfraceval(N,1/(Pi*5^(2/3)),100)
```

$$\%6 = 8.3941615239730609987666907762277846680E-13$$

Inverse Mellin Transforms I

```
? gammamellininvasymp([0,0,0],9)
```

$$\%1 = [1, -1/9, 2/81, -14/2187, 8/19683, 440/177147, -21520/4782969, 268240/43046721, -2898560/387420489]$$

```
? M=gammamellininvasymp([0,0,0],200);
? N=contfracinit(M*1.);
? contfraceval(N,0.2,50)
```

$$\%4 = 0.97871544142464955957930733267300399652$$

```
? contfraceval(N,0.2,100)
```

$$\%5 = 0.97871544142464955957930733267300399652$$

```
? sqrt(16/3)*5^(-2/3)*exp(-3*Pi*5^(2/3))*
contfraceval(N,1/(Pi*5^(2/3)),100)
```

$$\%6 = 8.3941615239730609987666907762277846680E-13$$

Inverse Mellin Transforms I

```
? gammamellininvasymp([0,0,0],9)
```

$$\%1 = [1, -1/9, 2/81, -14/2187, 8/19683, 440/177147, -21520/4782969, 268240/43046721, -2898560/387420489]$$

```
? M=gammamellininvasymp([0,0,0],200);
? N=contfracinit(M*1.);
? contfraceval(N,0.2,50)
```

$$\%4 = 0.97871544142464955957930733267300399652$$

```
? contfraceval(N,0.2,100)
```

$$\%5 = 0.97871544142464955957930733267300399652$$

```
? sqrt(16/3)*5^(-2/3)*exp(-3*Pi*5^(2/3))*
contfraceval(N,1/(Pi*5^(2/3)),100)
```

$$\%6 = 8.3941615239730609987666907762277846680E-13$$

Inverse Mellin Transforms I

```
? gammamellininvasymp([0,0,0],9)
```

$$\%1 = [1, -1/9, 2/81, -14/2187, 8/19683, 440/177147, -21520/4782969, 268240/43046721, -2898560/387420489]$$

```
? M=gammamellininvasymp([0,0,0],200);
? N=contfracinit(M*1.);
? contfraceval(N,0.2,50)
```

$$\%4 = 0.97871544142464955957930733267300399652$$

```
? contfraceval(N,0.2,100)
```

$$\%5 = 0.97871544142464955957930733267300399652$$

```
? sqrt(16/3)*5^(-2/3)*exp(-3*Pi*5^(2/3))*
contfraceval(N,1/(Pi*5^(2/3)),100)
```

$$\%6 = 8.3941615239730609987666907762277846680E-13$$

Inverse Mellin Transforms II

```
? G=gammamellininvinit([0,0,0]);
? gammamellininv(G,5)
```

$$\%8 = 8.3941615239730609987666907762277846685E-13$$

```
? gammamellininv([0,0,0],5)
```

$$\%9 = 8.3941615239730609987666907762277846685E-13$$

Inverse Mellin Transforms II

```
? G=gammamellininvinit([0,0,0]);
? gammamellininv(G,5)
```

$$\%8 = 8.3941615239730609987666907762277846685E-13$$

```
? gammamellininv([0,0,0],5)
```

$$\%9 = 8.3941615239730609987666907762277846685E-13$$

Inverse Mellin Transforms II

```
? G=gammamellininvinit([0,0,0]);
? gammamellininv(G,5)
```

$$\%8 = 8.3941615239730609987666907762277846685E-13$$

```
? gammamellininv([0,0,0],5)
```

$$\%9 = 8.3941615239730609987666907762277846685E-13$$

Inverse Mellin Transforms II

```
? G=gammamellininvinit([0,0,0]);
? gammamellininv(G,5)
```

$$\%8 = 8.3941615239730609987666907762277846685E-13$$

```
? gammamellininv([0,0,0],5)
```

$$\%9 = 8.3941615239730609987666907762277846685E-13$$

Creating L-functions I

Two ways: either through a preimplemented constructor (number field, elliptic curve, eta quotient, modular form, Dirichlet and Hecke character, Artin representation, etc...), or explicitly giving the coefficients, gamma factors, weight, conductor, etc...

The explicit way involves the command `lfuncreate` with argument a vector describing the L-function (see below). Most constructors also use this command, except for a few which are specific.

The `lfuncreate` does **not** do any actual L-function computation, only puts the data in a suitable inner format. The actual computations will be done by the commands lfun and lfuninit.

Creating L-functions I

Two ways: either through a preimplemented constructor (number field, elliptic curve, eta quotient, modular form, Dirichlet and Hecke character, Artin representation, etc...), or explicitly giving the coefficients, gamma factors, weight, conductor, etc...

The explicit way involves the command `lfuncreate` with argument a vector describing the L-function (see below). Most constructors also use this command, except for a few which are specific.

The `lfuncreate` does **not** do any actual L-function computation, only puts the data in a suitable inner format. The actual computations will be done by the commands lfun and lfuninit.

Creating L-functions I

Two ways: either through a preimplemented constructor (number field, elliptic curve, eta quotient, modular form, Dirichlet and Hecke character, Artin representation, etc...), or explicitly giving the coefficients, gamma factors, weight, conductor, etc...

The explicit way involves the command `lfuncreate` with argument a vector describing the L-function (see below). Most constructors also use this command, except for a few which are specific.

The `lfuncreate` does **not** do any actual L-function computation, only puts the data in a suitable inner format. The actual computations will be done by the commands lfun and lfuninit.

Creating L-functions II

```
? L1=lfuncreate(1);
```

Riemann zeta.

```
? Lm163=lfuncreate(-163);
```

Quadratic character of discriminant -163.

```
? Lell=lfuncreate(ellinit("11a1"));
```

Elliptic curve $11a1$.

```
? Lnf=lfuncreate(x^3-x-1);
```

Zeta function of the number field defined by the polynomial $x^3 - x - 1$.

```
? Leta12=lfunetaquo(Mat([1,24]));
? Leta2=lfunetaquo([1,2;11,2]);
```

Eta quotients $\Delta(\tau) = \eta(\tau)^{24}$ and $F(\tau) = \eta(\tau)^2\eta(11\tau)^2$. ?

```
Lqf=lfunqf([4,1;1,6]);
```

2-dimensional lattice of discriminant -23.

Creating L-functions II

```
? L1=lfuncreate(1);
```

Riemann zeta.

```
? Lm163=lfuncreate(-163);
```

Quadratic character of discriminant -163.

```
? Lell=lfuncreate(ellinit("11a1"));
```

Elliptic curve $11a1$.

```
? Lnf=lfuncreate(x^3-x-1);
```

Zeta function of the number field defined by the polynomial $x^3 - x - 1$.

```
? Leta12=lfunetaquo(Mat([1,24]));
? Leta2=lfunetaquo([1,2;11,2]);
```

Eta quotients $\Delta(\tau) = \eta(\tau)^{24}$ and $F(\tau) = \eta(\tau)^2\eta(11\tau)^2$. ?

```
Lqf=lfunqf([4,1;1,6]);
```

2-dimensional lattice of discriminant -23.

Creating L-functions II

```
? L1=lfuncreate(1);
```
Riemann zeta.
```
? Lm163=lfuncreate(-163);
```
Quadratic character of discriminant -163.
```
? Lell=lfuncreate(ellinit("11a1"));
```
Elliptic curve $11a1$.
```
? Lnf=lfuncreate(x^3-x-1);
```
Zeta function of the number field defined by the polynomial $x^3 - x - 1$.
```
? Leta12=lfunetaquo(Mat([1,24]));
? Leta2=lfunetaquo([1,2;11,2]);
```
Eta quotients $\Delta(\tau) = \eta(\tau)^{24}$ and $F(\tau) = \eta(\tau)^2\eta(11\tau)^2$. ?
```
Lqf=lfunqf([4,1;1,6]);
```
2-dimensional lattice of discriminant -23.

Creating L-functions II

? `L1=lfuncreate(1);`

Riemann zeta.

? `Lm163=lfuncreate(-163);`

Quadratic character of discriminant -163.

? `Lell=lfuncreate(ellinit("11a1"));`

Elliptic curve $11a1$.

? `Lnf=lfuncreate(x^3-x-1);`

Zeta function of the number field defined by the polynomial $x^3 - x - 1$.

? `Leta12=lfunetaquo(Mat([1,24]));`

? `Leta2=lfunetaquo([1,2;11,2]);`

Eta quotients $\Delta(\tau) = \eta(\tau)^{24}$ and $F(\tau) = \eta(\tau)^2\eta(11\tau)^2$. ? `Lqf=lfunqf([4,1;1,6]);`

2-dimensional lattice of discriminant -23.

Creating L-functions II

? `L1=lfuncreate(1);`

Riemann zeta.

? `Lm163=lfuncreate(-163);`

Quadratic character of discriminant -163.

? `Lell=lfuncreate(ellinit("11a1"));`

Elliptic curve $11a1$.

? `Lnf=lfuncreate(x^3-x-1);`

Zeta function of the number field defined by the polynomial $x^3 - x - 1$.

? `Leta12=lfunetaquo(Mat([1,24]));`

? `Leta2=lfunetaquo([1,2;11,2]);`

Eta quotients $\Delta(\tau) = \eta(\tau)^{24}$ and $F(\tau) = \eta(\tau)^2\eta(11\tau)^2$. ? `Lqf=lfunqf([4,1;1,6]);`

2-dimensional lattice of discriminant -23.

Creating L-functions II

? `L1=lfuncreate(1);`

Riemann zeta.

? `Lm163=lfuncreate(-163);`

Quadratic character of discriminant -163.

? `Lell=lfuncreate(ellinit("11a1"));`

Elliptic curve $11a1$.

? `Lnf=lfuncreate(x^3-x-1);`

Zeta function of the number field defined by the polynomial $x^3 - x - 1$.

? `Leta12=lfunetaquo(Mat([1,24]));`

? `Leta2=lfunetaquo([1,2;11,2]);`

Eta quotients $\Delta(\tau) = \eta(\tau)^{24}$ and $F(\tau) = \eta(\tau)^2\eta(11\tau)^2$. ? `Lqf=lfunqf([4,1;1,6]);`

2-dimensional lattice of discriminant -23.

Creating L-functions III

Create explicitly:

```
e=ellinit("5077a1");a=ellan(e,10000);
L5077=lfuncreate([a,0,[0,1],2,5077,1]);
```

1. a: function giving the Dirichlet coefficients, either a vector as here, or as a closure of the form `n->vector(n,j,a(j))` (**not** as `n->a(n)`, don't ask me why).
2. 0: self-dual (in general, dual L-function).
3. $[0,1]$: gamma factors here $\Gamma_{\mathbb{R}}(s)\Gamma_{\mathbb{R}}(s+1)$.
4. 2, 5077, 1: weight, conductor, root number.
5. If poles, add a suitable seventh component with poles and polar parts.

Creating L-functions III

Create explicitly:

```
e=ellinit("5077a1");a=ellan(e,10000);
L5077=lfuncreate([a,0,[0,1],2,5077,1]);
```

1. a: function giving the Dirichlet coefficients, either a vector as here, or as a closure of the form `n->vector(n,j,a(j))` (**not** as `n->a(n)`, don't ask me why).
2. 0: self-dual (in general, dual L-function).
3. $[0,1]$: gamma factors here $\Gamma_{\mathbb{R}}(s)\Gamma_{\mathbb{R}}(s+1)$.
4. 2, 5077, 1: weight, conductor, root number.
5. If poles, add a suitable seventh component with poles and polar parts.

Creating L-functions III

Create explicitly:

```
e=ellinit("5077a1");a=ellan(e,10000);
L5077=lfuncreate([a,0,[0,1],2,5077,1]);
```

1. a: function giving the Dirichlet coefficients, either a vector as here, or as a closure of the form `n->vector(n,j,a(j))` (**not** as `n->a(n)`, don't ask me why).
2. 0: self-dual (in general, dual L-function).
3. $[0,1]$: gamma factors here $\Gamma_{\mathbb{R}}(s)\Gamma_{\mathbb{R}}(s+1)$.
4. 2, 5077, 1: weight, conductor, root number.
5. If poles, add a suitable seventh component with poles and polar parts.

Creating L-functions III

Create explicitly:

```
e=ellinit("5077a1");a=ellan(e,10000);
L5077=lfuncreate([a,0,[0,1],2,5077,1]);
```

1. a: function giving the Dirichlet coefficients, either a vector as here, or as a closure of the form `n->vector(n,j,a(j))` (**not** as `n->a(n)`, don't ask me why).
2. 0: self-dual (in general, dual L-function).
3. $[0,1]$: gamma factors here $\Gamma_{\mathbb{R}}(s)\Gamma_{\mathbb{R}}(s+1)$.
4. 2, 5077, 1: weight, conductor, root number.
5. If poles, add a suitable seventh component with poles and polar parts.

Creating L-functions III

Create explicitly:

```
e=ellinit("5077a1");a=ellan(e,10000);
L5077=lfuncreate([a,0,[0,1],2,5077,1]);
```

1. a: function giving the Dirichlet coefficients, either a vector as here, or as a closure of the form `n->vector(n,j,a(j))` (**not** as `n->a(n)`, don't ask me why).
2. 0: self-dual (in general, dual L-function).
3. $[0,1]$: gamma factors here $\Gamma_{\mathbb{R}}(s)\Gamma_{\mathbb{R}}(s+1)$.
4. 2, 5077, 1: weight, conductor, root number.
5. If poles, add a suitable seventh component with poles and polar parts.

Creating L-functions III

Create explicitly:

```
e=ellinit("5077a1");a=ellan(e,10000);
L5077=lfuncreate([a,0,[0,1],2,5077,1]);
```

1. a: function giving the Dirichlet coefficients, either a vector as here, or as a closure of the form `n->vector(n,j,a(j))` (**not** as `n->a(n)`, don't ask me why).
2. 0: self-dual (in general, dual L-function).
3. $[0,1]$: gamma factors here $\Gamma_{\mathbb{R}}(s)\Gamma_{\mathbb{R}}(s+1)$.
4. 2, 5077, 1: weight, conductor, root number.
5. If poles, add a suitable seventh component with poles and polar parts.

Computing L-functions I

Once created, the basic functions are `lfuninit` and `lfun`. `lfuninit` does all the hard job, and should be used if several values of $L(s)$ must be computed (otherwise called each time). Must give a rectangular domain where L-values will be computed (but not for `lfun`). Examples:

```
lfun(L1,2)
```

```
1.6449340668482264364724151666460251892
```

(what else?), but if you want to plot:

```
LI=lfuninit(L5077,[1,5,50]);
```

(rectangle $[1-5,1+5]\times[-50,50]$)

```
ploth(t=-50,50,lfunhardy(LI,t));
```

plots the Hardy Z-function on the critical line.

Computing L-functions I

Once created, the basic functions are `lfuninit` and `lfun`. `lfuninit` does all the hard job, and should be used if several values of $L(s)$ must be computed (otherwise called each time). Must give a rectangular domain where L-values will be computed (but not for `lfun`). Examples:

```
lfun(L1,2)
```

$$1.6449340668482264364724151666460251892$$

(what else?), but if you want to plot:

```
LI=lfuninit(L5077,[1,5,50]);
```

(rectangle $[1-5,1+5]\times[-50,50]$)

```
ploth(t=-50,50,lfunhardy(LI,t));
```

plots the Hardy Z-function on the critical line.

Computing L-functions I

Once created, the basic functions are `lfuninit` and `lfun`. `lfuninit` does all the hard job, and should be used if several values of $L(s)$ must be computed (otherwise called each time). Must give a rectangular domain where L-values will be computed (but not for `lfun`). Examples:

```
lfun(L1,2)
```

$$1.6449340668482264364724151666460251892$$

(what else?), but if you want to plot:

```
LI=lfuninit(L5077,[1,5,50]);
```

(rectangle $[1-5,1+5]\times[-50,50]$)

```
ploth(t=-50,50,lfunhardy(LI,t));
```

plots the Hardy Z-function on the critical line.

Computing L-functions II

Also, computes directly derivatives and Taylor expansions:
`lfun(1,1+x+O(x^6))`

$$1.0000000... * x^{-1} + 0.577215... + 0.0728158... * x$$
$$- 0.00484518... * x^2 - 0.00034230... * x^3 + O(x^4)$$

(result edited for clarity but correct to desired number of decimals).

As usual can fill in missing pieces (residues, root number for instance) using `lfunrootres`, or check correctness of functional equation with `lfuncheckfeq`.

Computing L-functions II

Also, computes directly derivatives and Taylor expansions:
`lfun(1,1+x+O(x^6))`

$$1.0000000... * x^{-1} + 0.577215... + 0.0728158... * x$$
$$- 0.00484518... * x^2 - 0.00034230... * x^3 + O(x^4)$$

(result edited for clarity but correct to desired number of decimals).

As usual can fill in missing pieces (residues, root number for instance) using `lfunrootres`, or check correctness of functional equation with `lfuncheckfeq`.

Computing L-functions II

Also, computes directly derivatives and Taylor expansions:
`lfun(1,1+x+O(x^6))`

$$1.0000000... * x^{-1} + 0.577215... + 0.0728158... * x$$
$$- 0.00484518... * x^2 - 0.00034230... * x^3 + O(x^4)$$

(result edited for clarity but correct to desired number of decimals).

As usual can fill in missing pieces (residues, root number for instance) using `lfunrootres`, or check correctness of functional equation with `lfuncheckfeq`.

Computing L-functions III

Can also compute conductor if unknown, either by trials using `lfuncheckfeq`, or analytically: example:

`e=ellinit([0,0,0,-7,3]);`

(elliptic curve $y^2 = x^3 - 7x + 3$), assume I don't know Tate's algorithm.

`lfunconductor(e)` gives me instantly 9032. However, slight cheat, must put conductor component to 0 after `lfuncreate(e)`.

Perhaps better example:

`lfunconductor(857)` asks for the conductor of the quadratic character of conductor 857: answer $[17, 857]$!!!

Indeed, in Granville-Soundararajan's theory of pretentious characters, the character mod 857 *pretends* to be like the one modulo 17. Can be seen on the graph on the real line (not on the critical line).

Computing L-functions III

Can also compute conductor if unknown, either by trials using `lfuncheckfeq`, or analytically: example:

`e=ellinit([0,0,0,-7,3]);`

(elliptic curve $y^2 = x^3 - 7x + 3$), assume I don't know Tate's algorithm.

`lfunconductor(e)` gives me instantly 9032. However, slight cheat, must put conductor component to 0 after `lfuncreate(e)`.

Perhaps better example:

`lfunconductor(857)` asks for the conductor of the quadratic character of conductor 857: answer $[17, 857]$!!!

Indeed, in Granville-Soundararajan's theory of pretentious characters, the character mod 857 *pretends* to be like the one modulo 17. Can be seen on the graph on the real line (not on the critical line).

Computing L-functions III

Can also compute conductor if unknown, either by trials using `lfuncheckfeq`, or analytically: example:

`e=ellinit([0,0,0,-7,3]);`

(elliptic curve $y^2 = x^3 - 7x + 3$), assume I don't know Tate's algorithm.

`lfunconductor(e)` gives me instantly 9032. However, slight cheat, must put conductor component to 0 after `lfuncreate(e)`.

Perhaps better example:

`lfunconductor(857)` asks for the conductor of the quadratic character of conductor 857: answer $[17, 857]$!!!

Indeed, in Granville-Soundararajan's theory of pretentious characters, the character mod 857 *pretends* to be like the one modulo 17. Can be seen on the graph on the real line (not on the critical line).

Computing L-functions III

Can also compute conductor if unknown, either by trials using `lfuncheckfeq`, or analytically: example:

`e=ellinit([0,0,0,-7,3]);`

(elliptic curve $y^2 = x^3 - 7x + 3$), assume I don't know Tate's algorithm.

`lfunconductor(e)` gives me instantly 9032. However, slight cheat, must put conductor component to 0 after `lfuncreate(e)`.

Perhaps better example:

`lfunconductor(857)` asks for the conductor of the quadratic character of conductor 857: answer $[17, 857]$!!!

Indeed, in Granville-Soundararajan's theory of pretentious characters, the character mod 857 *pretends* to be like the one modulo 17. Can be seen on the graph on the real line (not on the critical line).

Thank you for your attention !