Upcoming p-adic functionality in FLINT

Sebastian Pancratz

p-adic Sage Days, San Diego, 19–23 February 2012

Overview

- Motivation
- Design decisions
- Field of p-adic numbers $\mathbf{Q}_p$
  - Elements of $\mathbf{Q}_p$
  - Addition, multiplication, inversion, square root, exponential, logarithm, Teichmüller lift
- Polynomials over $\mathbf{Q}_p$
- Unramified extensions $\mathbf{Q}_q$
  - Elements of $\mathbf{Q}_q$
  - Addition, multiplication, inversion, Teichmüller lift, Frobenius
- Summary of timings

Motivation

Motivation for the implementation.

- I need p-adic arithmetic for my own research code in point counting, which is largely based on FLINT.

Purpose of the talk.

- Present the already implemented functionality;
- Offer comparisons between Sage, Magma, and FLINT;
- Ask for feedback.

Design decisions

Comparison with Laurent series over $\mathbf{F}_p$.

A Laurent series consists of the data $(m, n, (a_m, \ldots, a_n))$ giving

$$\sum_{i=m}^{n} a_i X^i$$

Given $f(X)$ and $g(X)$, we can compute their sum modulo X^N as

$$f(X) + g(X) = \sum_{i=\min\{m_f, m_g\}}^{\min\{\max\{n_f, n_g\}, N-1\}} (a_i + b_i) X^i$$

As coefficients are readily available, it is reasonable for operations to treat inputs as exact and require only the output precision N.

Design decisions

Decision.

- Each p-adic operation treats the input as exact data and requires the desired output precision as a separate argument.

Rationale.

- A number is *just* a number.
- The intrinsic difficulty in p-adic arithmetic stems from the precision loss, which depends on the particular operation.
- Note that it would be straightforward to implement various precision models on top of this.

Elements of $\mathbf{Q}_p$

Consider two numbers,

$$x = 3 + 2 \times 5 + 1 \times 5^2 + 4 \times 5^3$$
$$y = 1 + 1 \times 5 + 4 \times 5^2 + 2 \times 5^3 + 3 \times 5^4$$

We can compute their sum modulo 5^2,

$$x + y = (3 + 1) + (2 + 1)5$$

without looking at higher order digits. But this is *not* what is happening in practical implementations. The p-adic digits are not readily available, and for $p \ll 2^{64}$ this is certainly not desirable anyway.

Elements of $\mathbf{Q}_p$

Instead, an element $x \neq 0$ is typically stored as $x = p^v u$ with $v = \mathrm{ord}_p(x) \in \mathbf{Z}$ and $u \in \mathbf{Z}$ with $p \nmid u$. In FLINT, we choose

```
typedef struct {
    fmpz u;
    long v;
} padic_struct;
```

Remark

- Improved maintainability by having *one* data type; no special case depending on the size of p or p^N;
- Eventually, $p = 2$ should have a special case.
- One *could* consider a different implementation performing basic arithmetic to base p^k with k s.t. such that p^k fits in a word. This would allow replacing mod p^N operations by mod p^k operations (with a precomputed word-sized inverse) in many algorithms.

Benchmarks for $\mathbf{Q}_p$

We present some timings for arithmetic in $\mathbf{Q}_p \bmod p^N$ where $p = 17$, $N = 2^i$, $i = 0, \ldots, 10$, comparing the three systems Magma (V2.17-13), Sage (4.8 incl. #4821) and FLINT (2.3) on a machine with Intel Xeon CPUs running at 2.93GHz.

To avoid worrying about taking the same random sequences of elements, we instead fix elements $a = 3^{3N}$, $b = 5^{2N}$, $c = 17^2 b$, and $d = 1 - c$ modulo p^N.

We consider the following operations:

- Addition
- Multiplication
- Inversion
- Square root
- Teichmüller lift
- Exponential
- Logarithm

Hensel lifting

Theorem

Let $g \in \mathbf{Z}_q[X]$ and assume that $x_0 \in \mathbf{Z}_q$ satisfies

$$\mathrm{ord}_p(g(x_0))) = m + n, \quad \mathrm{ord}_p(g'(x_0)) = m,$$

for some $0 \leq m < n$. There exists a unique root $x \in \mathbf{Z}_q$ of g satisfying $x \equiv x_0$ modulo p^n.

Algorithm

- Compute sequence $e_k = N, e_{k-1} = \lceil e_k/2 \rceil, \ldots, e_0$ until $1 \leq e_0 \leq n$.
- For $i = 0, \ldots, k-1$, compute

$$x_{i+1} = x_i - \frac{g(x_i)}{g'(x_i)} \pmod{p^{e_{i+1}}}.$$

Hensel lifting

Remark

In the above formulation, Hensel lifting requires a nested lifting process to compute the p-adic inverse of $g'(x_i)$ in each step. This can be replaced by a single parallel Hensel lift:

- Compute sequence $e_k = N, e_{k-1} = \lceil e_k/2 \rceil, \ldots, e_0$ until $1 \leq e_0 \leq n$.
- Set $y_0 = g'(x_0)^{-1} \bmod p$.
- For $i = 0, \ldots, k-1$, compute

$$
\begin{aligned}
x_{i+1} &= x_i - g(x_i) y_i && \pmod{p^{e_{i+1}}}, \\
y_{i+1} &= y_i \big(2 - y_i g'(x_{i+1}) \big) && \pmod{p^{e_{i+1}}}.
\end{aligned}
$$

Addition

Signature

```
void padic_add(z, x, y, ctx)
```

Contract

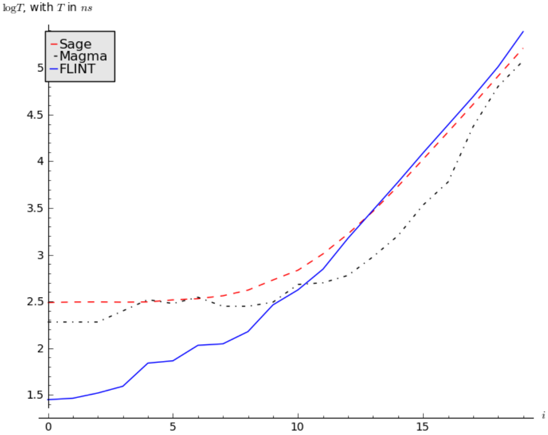
Assumes that x and y are reduced modulo p^N and returns z in reduced form, too.

Algorithm

Avoids expensive modulo operation, replacing this by one comparison and at most one subtraction.

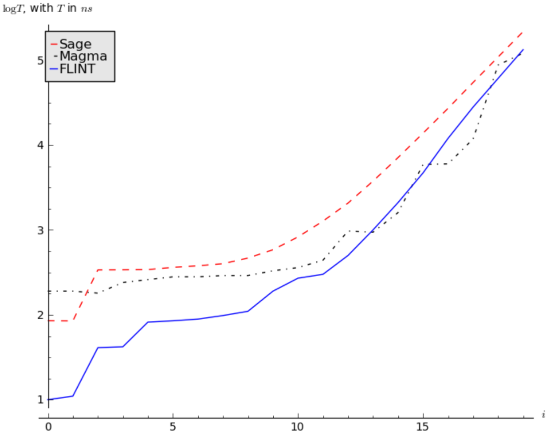
Addition (equal valuation)

Computes $a + b \bmod p^N$.



Addition (distinct valuation)

Computes $a + c \bmod p^N$.



Multiplication

Signature

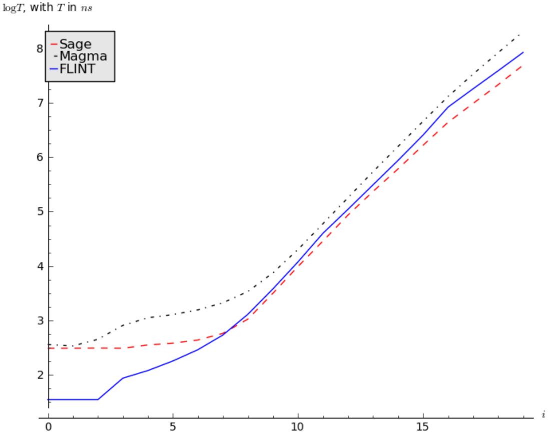
```
void padic_mul(z, x, y, ctx)
```

Contract

Makes no assumptions on x and y, returns z reduced modulo p^N.

Multiplication

Computes $ab \bmod p^N$.



Inversion

Signature

```
void padic_inv(z, x, ctx)
```

Contract

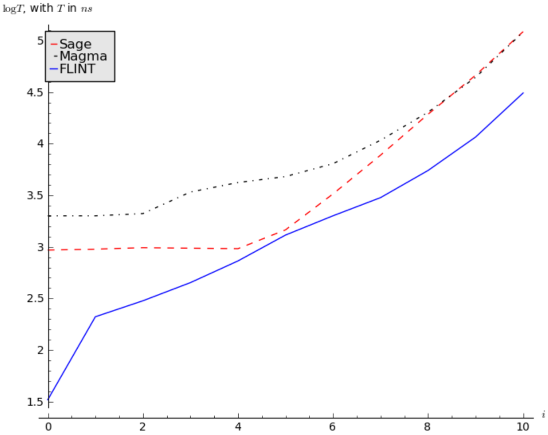
Makes no assumptions on $x \neq 0$, returns z reduced modulo p^N.

Algorithm

Hensel lifting on $g(X) = xX - 1$, starting from an inverse in $\mathbf{F}_p$ and using the update formula $z' = z + z(1 - xz)$.

Inversion

Computes $a^{-1} \bmod p^N$ to the required precision N.



Square root

Signature

```
int padic_sqrt(z, x, ctx)
```

Contract

Returns whether x has a square root, and if this is the case sets z to a square root modulo p^N.

Recall that non-zero $x = p^v u$ has a square root if and only if v is even and u has a square root modulo 8 or p where $p = 2$ or $p > 2$, respectively.

Algorithm

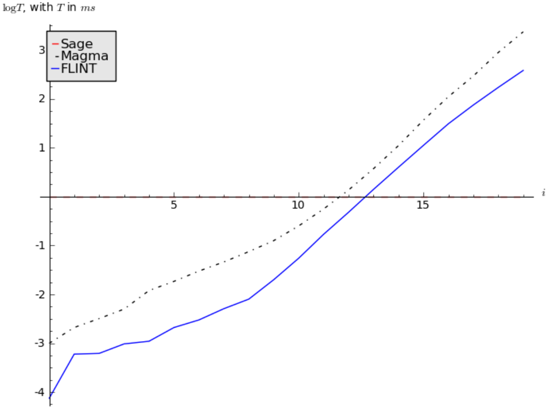
- Compute $x^{-1/2} \bmod p^N$ using Hensel lifting on $g(X) = x^2 X - 1$, starting modulo p and using the division-free update formula

$$z' = z - z\big(xz^2 - 1\big)/2.$$

- Set $z = x x^{-1/2} \bmod p^N$.

Square root

Computes a square root of a to the required precision N.



Teichmüller lift

Signature

```
void padic_teichmuller(z, x, ctx)
```

Contract

Assumes only that $\text{ord}_p(x) = 0$, returns the unique z such that $z \equiv x \pmod{p}$ and $z \equiv x \pmod{p}$ and $z^p - z = 0$, reduced modulo p^N.

Algorithm

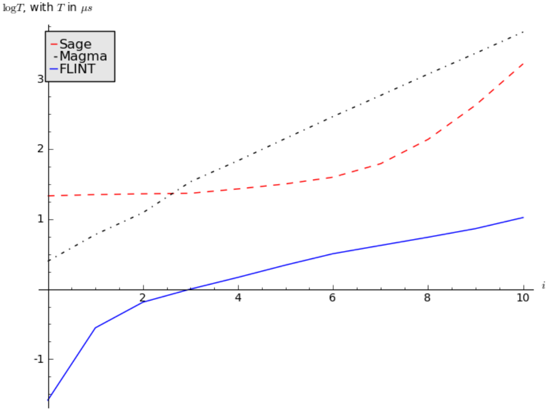
Hensel lifting on $g(X) = X^p - X$, starting from $z_0 = x \bmod p$.

Improvements

- Hensel lifting without inverses.
- At the first step, we want $z_0 = x \bmod p$ and $y_0 = \left((p-1)x^{p-2}\right)^{-1} \bmod p$, so $y_0 = p - z_0$ without inversion.

Teichmüller lift

Computes the Teichmüller lift of $a \bmod p^N$ to the required precision N.



Exponential

Signature

```
int padic_exp(z, x, ctx)
```

Contract

Returns whether $\exp_p(x)$ converges, that is, $\operatorname{ord}_p(x) \geq 2$ or $\operatorname{ord}_p(x) \geq 1$ as $p = 2$ or $p > 2$, respectively, and computes z reduced modulo p^N.

Algorithm

Evaluates the truncated series

$$\exp_p(x) = \sum_{i=0}^{m-1} \frac{x^i}{i!}$$

over $\mathbf{Z}_p$ by multiplying through by $(m-1)!$, hence requiring only one p-adic inversion. We can choose $m = \lceil \big((p-1)N - 1\big) / \big((p-1)v - 1\big) \rceil$.

Exponential

Improvements

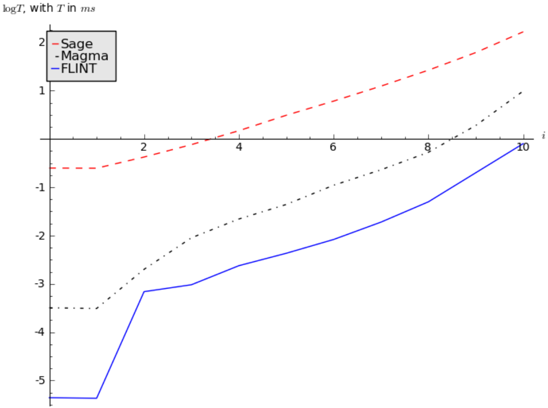
- Rectangular splitting algorithm, starting from the expression

$$\exp_p(x) = \sum_{j=0}^{\lceil m/B \rceil - 1} \left(\sum_{i=0}^{B-1} \frac{x^i}{(i+Bj)!} \right) x^{Bj}$$

  where $B = \lfloor \sqrt{m} \rfloor$.
- Asymptotic improvements possible, e.g. using a binary splitting algorithm, which recursively considers half the coefficients of the series.

Exponential

Computes the exponential of c to the required precision N.



Logarithm

Signature

```
int padic_log(z, x, ctx)
```

Contract

Assumes that $\log_p(x)$ converges, that is, $\mathrm{ord}_p(x-1) \geq 2$ or $\mathrm{ord}_p(x-1) \geq 1$ as $p = 2$ or $p > 2$, respectively, and returns z reduced modulo p^N.

Algorithm

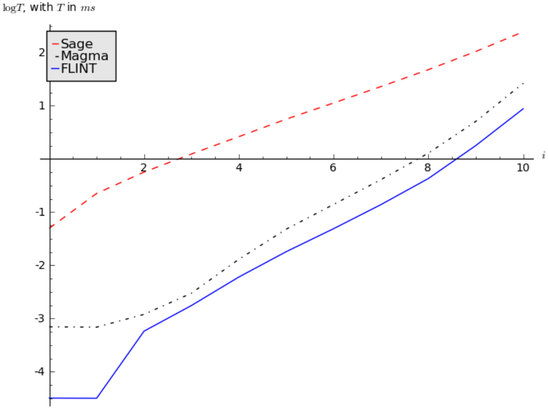
Evaluates the truncated series

$$\log_p(x) = \sum_{i=1}^{m} (-1)^{i-1} \frac{(x-1)^i}{i}$$

over $\mathbf{Z}_p$ by inverting i at each step using a precomputed Hensel lifting structure.

Logarithm

Computes the logarithm of $d = 1 - c$ to the required precision N.



Polynomials over $\mathbf{Q}_p$

We represent a non-zero polynomial $f(X) \in \mathbf{Q}_p[X]$ as

$$f(X) = p^v\bigl(a_0 + a_1 X + \cdots + a_n X^n\bigr)$$

where $a_0, \ldots, a_n \in \mathbf{Z}$ and, for at least one i, p does not divide a_i.

Remark

- Allows for transfer of many problems over $\mathbf{Q}_p$ to $\mathbf{Z}/(p^N)$, where fast implementations are available.
- Similar to the approach chosen over $\mathbf{Q}$ in FLINT (and Sage), see trac ticket #4000.

Functions for $\mathbf{Q}_p[X]$

- Conversions to polynomials over $\mathbf{Z}$ and $\mathbf{Q}$
- Coefficient manipulation
- Addition, subtraction, negation
- Scalar multiplication
- Multiplication
- Powers
- Series inversion
- Derivative
- Evaluation
- Composition

Unramified extensions $\mathbf{Q}_q$

We represent an unramified extension of $\mathbf{Q}_p$ as

$$\mathbf{Q}_q \cong \mathbf{Q}_p[X]/(f(X))$$

where $f(X) \bmod p$ is separable, storing $f(X)$ in a data structure for sparse polynomials.

This allows for the reduction of a degree n polynomial modulo $f(X)$ in linear time $\mathcal{O}(n)$.

Benchmarks for $\mathbf{Q}_q$

We present some timings for arithmetic in $\mathbf{Q}_q \bmod p^N$ where $q = 5^{251}$ and $N = 2^i$, $i = 0, \dots, 10$, comparing the three systems Magma (V2.17-13), Sage (4.8 incl. #4821) and FLINT (2.3) on a machine with Intel Xeon CPUs running at 2.93GHz.

To avoid worrying about taking the same random sequences of elements, we instead fix elements $a = (X+1)^N$, $b = (X^2+2)^N$, and $c = 5^2 b$ modulo p^N.

We consider the following operations:

- Addition
- Multiplication
- Inversion
- Teichmüller lift
- Frobenius

Addition

Signature

```
void qadic_add(z, x, y, ctx)
```

Contract

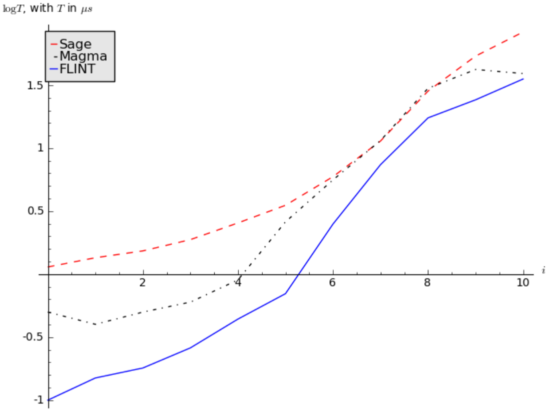
Sets $z = x + y \bmod p^N$, assuming both x and y are reduced modulo p^N.

Algorithm

Avoids expensive modulo operation on the coefficients, replacing this by one comparison and at most one subtraction per coefficient.

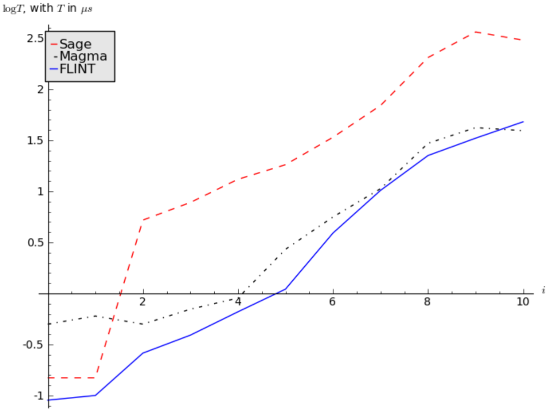
Addition (equal valuation)

Computes the sum $a + b$ to the required precision N.



Addition (distinct valuation)

Computes the sum $a + b$ to the required precision N.



Multiplication

Signature

```
void qadic_mul(z, x, y, ctx)
```

Contract

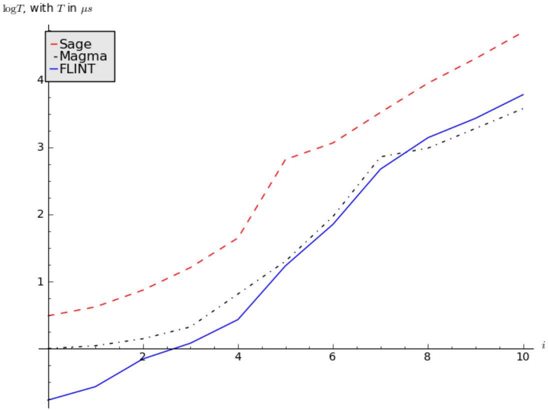
Sets $z = xy \bmod p^N$, without assuming that x, y are reduced modulo p^N.

Algorithm

First compute the product of the polynomials, then reduce the result modulo p^N and $f(X)$.

Multiplication

Computes the product ab to the required precision N.



Inversion

Signature

```
void qadic_inv(z, x, ctx)
```

Contract

Sets z to the inverse of $x \neq 0$ modulo p^N.

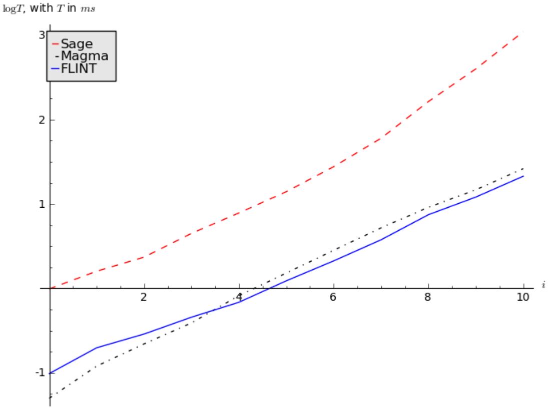
Algorithm

Hensel lifting on $g(X) = xX + 1$, using the update formula $z' = z + z(1 - xz)$; the starting point z_0 is the inverse of x in $\mathbf{F}_p[X]/(f(X))$ computed by a version of Euclid's extended algorithm only updating one cofactor[1].

[1]Using Euclid's extended algorithm to compute d, s, t such that $d = \gcd(a, b) = sa + tb$, one improvement is to only update s during the procedure and then construct $t = (d - sa)/b$. Here, we can omit the last step as we do not need the cofactor of $f(X)$.

Inversion

Computes the inverse of a to the required precision N.



Teichmüller lift

Signature

```
void qadic_teichmuller(z, x, ctx)
```

Contract

Assumes only that $\operatorname{ord}_p(x) = 0$, returns the unique q such that $z^q - z = 0$ reduced modulo p^N.

Algorithm

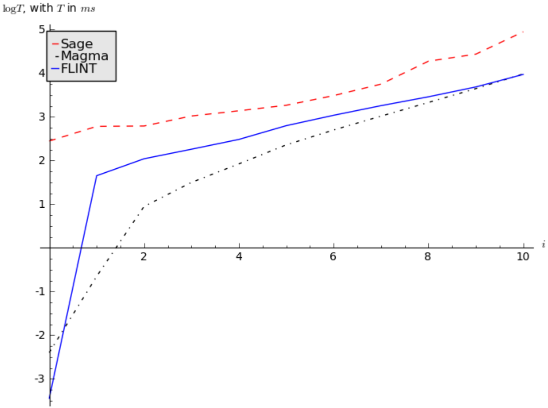
Hensel lifting on $g(X) = X^q - X$, starting from $z_0 = x \bmod p$.

Improvements

Observe that $g'(z_i) = qz_i^{q-1} - 1$ and z_i^{q-1} is close to 1 so $g'(z_i)$ is close to $q - 1$. Thus, we only need to compute an inverse of $q - 1$, which is defined over $\mathbf{Q}_p$.

Teichmüller lift

Computes the Teichmüller lift of a to the required precision N.



Frobenius

Signature

```
void qadic_frobenius(z, x, k, ctx)
```

Contract

Sets z to $\Sigma^k x$ modulo p^N, where $\Sigma \in \mathrm{Gal}(\mathbf{Q}_q/\mathbf{Q}_p) \cong \mathrm{Gal}(\mathbf{F}_q/\mathbf{F}_p)$ is the image of $\sigma\colon \mathbf{F}_q \to \mathbf{F}_q, x \mapsto x^p$.

Algorithm

- Write $\mathbf{Q}_q \cong \mathbf{Q}_p[X]/(f(X))$ and $x = \sum_{i=0}^{d-1} a_i X^i$.
- Compute $\Sigma^k X$ using Hensel lifting on f, starting from $z_0 = X^{p^k}$ in $\mathbf{F}_p[X]/(f(X))$.
- Compute $\Sigma^k x = \sum_{i=0}^{d-1} a_i \left(\Sigma^k X\right)^i$, which is a polynomial composition modulo p^N and $f(X)$.

Frobenius

Improvements

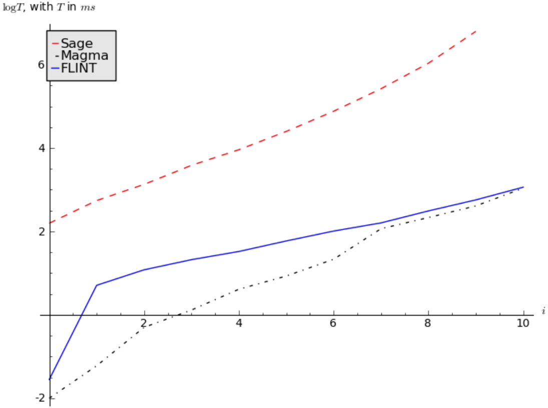
- In a first approach, might use Horner's method to carry out the composition, which uses about d multiplications in $\mathbf{Q}_q$
- Instead, use a rectangular splitting method, starting from the expression

$$x = \sum_{j=0}^{\lceil d/B \rceil - 1} \left(\sum_{i=0}^{B-1} a_{i+Bj} X^i \right) X^{Bj}$$

where $B = \lfloor \sqrt{d} \rfloor$, precomputing $\Sigma^k(X)^i$ for $i = 0, \ldots, B$. This requires about $2\sqrt{d}$ multiplications in $\mathbf{Q}_q$ and extra space for about $d^{3/2}$ elements of $\mathbf{Z}/(p^N)$.

Frobenius

Computes the image of a under the Frobenius homomorphism to the required precision N.



Missing functionality for $\mathbf{Q}_q$

- Exponential
- Logarithm
- Square root
- Norm
- Trace

Summary of timings

	Operation	$T_{\text{Sage}}/T_{\text{FLINT}}$	$T_{\text{Magma}}/T_{\text{FLINT}}$
$\mathbf{Q}_p$	$a+b$	0.67	0.49
	$a+c$	1.63	0.91
	ab	0.58	2.41
	a^{-1}	3.94	3.9
	$\sqrt{a}$		6.17
	Teichmüller(a)	156.19	4670
	$\exp(c)$	206.25	12.25
	$\log(d)$	27.95	3.01
$\mathbf{Q}_q$	$a+b$	2.36	1.1
	$a+c$	6.3	0.82
	ab	8.59	0.62
	a^{-1}	51.47	1.23
	Teichmüller(a)	9.48	1.03
	$\Sigma(a)$	11000	0.72

Codebase

- FLINT,
  http://www.flintlib.org
- Personal development branch for p-adic arithmetic,
  https://github.com/SPancratz/flint2/tree/padic
- Lines of source code,

	padic	padic_poly	padic_poly	qadic
Base	1987	1460	683	920
Test	2321	1380	903	1131