Point counting: past, present and future.

Jan Tuitman

KU Leuven

January 25, 2017

Zeta functions

Let X be an algebraic variety over a finite field $\mathbf{F}_q$ with $q = p^n$.

Definition

$$Z(X, T) := \exp\left(\sum_{i=1}^{\infty} |X(\mathbf{F}_{q^i})| \frac{T^i}{i}\right)$$

Zeta functions

Let X be an algebraic variety over a finite field $\mathbf{F}_q$ with $q = p^n$.

Definition

$$Z(X, T) := \exp\left(\sum_{i=1}^{\infty} |X(\mathbf{F}_{q^i})| \frac{T^i}{i}\right)$$

For example:

$$\begin{aligned} Z(\mathbf{P}^1_{\mathbf{F}_q}, T) &= \exp\left(\sum_{i=1}^{\infty} (q^i + 1)\frac{T^i}{i}\right) \\ &= \exp\left(\sum_{i=1}^{\infty} \frac{T^i}{i}\right) \exp\left(\sum_{i=1}^{\infty} \frac{(qT)^i}{i}\right) \\ &= \frac{1}{(1-T)(1-qT)} \end{aligned}$$

Weil conjectures

The main theoretical result on zeta functions is:

Theorem (Weil conjectures)

Suppose that X is smooth proper of dimension d then:

- $Z(X,T) = \prod_{i=0}^{2d} P_i(T)^{(-1)^{i+1}}$ *with* $P_i(T) \in \mathbf{Z}[T]$,
- $Z(X, 1/(q^n T)) = \pm q^{nE/2} T^E Z(T)$,
- $P_i(T) = \prod(1 - \alpha_{ij} T)$ *with* α_{ij} *algebraic numbers of absolute value* $q^{i/2}$.

where E is the Euler characteristic of X.

Weil conjectures

The main theoretical result on zeta functions is:

Theorem (Weil conjectures)

Suppose that X is smooth proper of dimension d then:

- $Z(X,T) = \prod_{i=0}^{2d} P_i(T)^{(-1)^{i+1}}$ *with* $P_i(T) \in \mathbf{Z}[T]$,
- $Z(X, 1/(q^n T)) = \pm q^{nE/2} T^E Z(T)$,
- $P_i(T) = \prod(1 - \alpha_{ij} T)$ *with* α_{ij} *algebraic numbers of absolute value* $q^{i/2}$.

where E is the Euler characteristic of X.

This was proved for curves and Abelian varieties by Weil (1948) and for general X by Dwork, Grothendieck and Deligne (1974).

The proofs construct some cohomology theory such that $P_i(T)$ is the (reverse) characteristic polynomial of the action of the q-th power Frobenius map on $H^i(X)$. For example $H^i(X)$ can be ℓ-adic etale cohomology or rigid cohomology.

Counting points

Since the zeta function can be given by a finite amount of data, one can hope to compute it.

Problem

Compute $Z(X, T)$ efficiently.

Counting points

Since the zeta function can be given by a finite amount of data, one can hope to compute it.

Problem

Compute $Z(X,T)$ efficiently.

Bounds on the the degrees of the numerator and denominator of $Z(X,T)$ are known, so computing $Z(X,T)$ reduces to computing a finite number of $X(\mathbf{F}_{q^i})$.

For example, for a curve of genus g we have to compute up to $X(\mathbf{F}_{q^g})$. Counting naively we need at least q^g operations (usually a bit more). This is too slow to get anywhere for all but the smallest values of q and g.

Let us first give some applications.

Cryptography

For a curve C of genus g we have

$$Z(C,T) = \frac{\chi(T)}{(1-T)(1-qT)},$$

where $\chi(T) = \prod_{i=1}^{2g}(1-\alpha_i)T$ with α_i algebraic integers of absolute value $q^{1/2}$.

Cryptography

For a curve C of genus g we have

$$Z(C,T) = \frac{\chi(T)}{(1-T)(1-qT)},$$

where $\chi(T) = \prod_{i=1}^{2g}(1-\alpha_i)T$ with α_i algebraic integers of absolute value $q^{1/2}$.

One can associate to C a finite Abelian group $J(\mathbf{F}_q)$ of called its Jacobian. The order of this group is $\chi(1)$. The discrete logarithm problem (DLP) on $J(\mathbf{F}_q)$ is:

Problem

given $P, Q \in J(\mathbf{F}_q)$ find (the smallest) $n \in \mathbf{N}$ such that $nP = Q$.

This problem is used in cryptography in Diffie Helmann key exchange. When the order of $J(\mathbf{F}_q)$ only has small prime factors the DLP is easy. So we need to compute $\chi(1)$ and we can do this by computing $Z(X,T)$.

Sato-Tate distributions

Let C be a (smooth) projective curve of genus g defined over $\mathbb{Q}$.

For every prime p let C_p denote the curve over $\mathbf{F}_p$ obtained by reducing (the equations of) C modulo p. Again, for all but a finite number of p:

$$Z(C_p, T) = \frac{L_p(T)}{(1-T)(1-qT)}$$

for some polynomial $L_p(T) \in \mathbb{Z}[T]$ of degree $2g$.

Sato-Tate distributions

Let C be a (smooth) projective curve of genus g defined over $\mathbb{Q}$.

For every prime p let C_p denote the curve over $\mathbf{F}_p$ obtained by reducing (the equations of) C modulo p. Again, for all but a finite number of p:

$$Z(C_p, T) = \frac{L_p(T)}{(1-T)(1-qT)}$$

for some polynomial $L_p(T) \in \mathbb{Z}[T]$ of degree $2g$.

Problem

How is the polynomial $L_p(T/\sqrt{p})$ distributed when p varies?

Sato-Tate distributions

Let C be a (smooth) projective curve of genus g defined over $\mathbb{Q}$.

For every prime p let C_p denote the curve over $\mathbf{F}_p$ obtained by reducing (the equations of) C modulo p. Again, for all but a finite number of p:

$$Z(C_p, T) = \frac{L_p(T)}{(1-T)(1-qT)}$$

for some polynomial $L_p(T) \in \mathbb{Z}[T]$ of degree $2g$.

Problem

How is the polynomial $L_p(T/\sqrt{p})$ distributed when p varies?

Conjectural answer: as the (reverse) characteristic polynomial of a random conjugacy class of a certain compact group. So far only known for $g = 1$.

Andrew Sutherland (with coauthors) computed $L_p(T)$ for C with $g = 2$ and found all predicted distributions!

Elliptic curves

For elliptic curves there are many good point counting algorithms. These algorithms usually use the group structure.

For example:

- Schoof's algorithm
- Baby Step Giant Step
- Canonical lift algorithms

For these algorithms there also exist good implementations in Magma and elsewhere.

We will not go into more details here but move on to more general curves.

Hyperelliptic curves

For hyperelliptic curves of genus g, i.e. curves defined by

$$y^2 = f(x)$$

with $f(x) \in \mathbf{F}_q[x]$ of degree $2g+1$ or $2g+2$, the methods for elliptic curves can be generalised but quickly become unpractical (they are all exponential in g).

Hyperelliptic curves

For hyperelliptic curves of genus g, i.e. curves defined by

$$y^2 = f(x)$$

with $f(x) \in \mathbf{F}_q[x]$ of degree $2g+1$ or $2g+2$, the methods for elliptic curves can be generalised but quickly become unpractical (they are all exponential in g).

Theorem (Kedlaya, 2001)

The zeta function of a hyperelliptic curve of genus g over a finite field $\mathbf{F}_q$ with $q = p^n$ and p odd can be computed in time:

$$O((pg^4n^3)^{1+\epsilon})$$

Hyperelliptic curves

For hyperelliptic curves of genus g, i.e. curves defined by

$$y^2 = f(x)$$

with $f(x) \in \mathbf{F}_q[x]$ of degree $2g+1$ or $2g+2$, the methods for elliptic curves can be generalised but quickly become unpractical (they are all exponential in g).

Theorem (Kedlaya, 2001)

The zeta function of a hyperelliptic curve of genus g over a finite field $\mathbf{F}_q$ with $q = p^n$ and p odd can be computed in time:

$$O((pg^4n^3)^{1+\epsilon})$$

Actually Kedlaya only considered f of degree $2g+1$. Harrison extended it to all hyperelliptic curves in odd characteristic and implemented it in Magma. Vercauteren did the same in even characteristic.

So Magma has good and general implementations of Kedlaya's algorithm.

General curves

What about more general curves?

Some generalisations of Kedlaya's algorithm have been published:

- superelliptic curves, Gaudry-Gürel (2003),
- C_{ab} curves, Denef and Vercauteren (2004),
- nondegenerate curves, Castryck, Denef and Vercauteren (2006),

The class of nondegenerate curves includes all other cases and is the most general kind of curve for which a Kedlaya type algorithm had been worked out before 2014.

General curves

What about more general curves?

Some generalisations of Kedlaya's algorithm have been published:

- superelliptic curves, Gaudry-Gürel (2003),
- C_{ab} curves, Denef and Vercauteren (2004),
- nondegenerate curves, Castryck, Denef and Vercauteren (2006),

The class of nondegenerate curves includes all other cases and is the most general kind of curve for which a Kedlaya type algorithm had been worked out before 2014. However there are two problems:

- Not all curves are nondegenerate. Nondegenerate curves have dim $2g+1$ in the moduli space of curves of genus g which has dim $3g-3$.
- The Castryck, Denef and Vercauteren algorithm is not very practical and was therefore never (really) implemented.

Since 2014 I have developed and implemented another generalisation of Kedlaya's algorithm which does not suffer from these problems.

My algorithm

Let $C/\mathbf{F}_q$ with $q = p^n$ be the smooth projective curve birational to a (singular) plane curve

$$f(x, y) = 0$$

with $f \in \mathbf{F}_q[x, y]$ irreducible and monic in y of degree d_x, d_y in x, y.

My algorithm

Let $C/\mathbf{F}_q$ with $q = p^n$ be the smooth projective curve birational to a (singular) plane curve

$$f(x, y) = 0$$

with $f \in \mathbf{F}_q[x, y]$ irreducible and monic in y of degree d_x, d_y in x, y.

Theorem (T, 2014)

Suppose that we know a 'good' lift $F \in \mathbf{Z}_q[x, y]$ of f to characteristic zero (technical). Then the zeta function of C can be computed in time:

$$O((pd_y^6 d_x^4 n^3)^{1+\epsilon})$$

My algorithm

Let $C/\mathbf{F}_q$ with $q = p^n$ be the smooth projective curve birational to a (singular) plane curve

$$f(x, y) = 0$$

with $f \in \mathbf{F}_q[x, y]$ irreducible and monic in y of degree d_x, d_y in x, y.

Theorem (T, 2014)

Suppose that we know a 'good' lift $F \in \mathbf{Z}_q[x, y]$ of f to characteristic zero (technical). Then the zeta function of C can be computed in time:

$$O((pd_y^6 d_x^4 n^3)^{1+\epsilon})$$

What constitutes a 'good' lift to characteristic zero is rather technical. However, together with Wouter Castryck, we have developed and implemented an algorithm that (usually) finds such a lift for all curves of genus ≤ 5. Moreover, often a lift is known anyway or easy to construct. I have implemented the algorithm completely, it will soon be included in Magma.

How does it work?

There is a way of associating to a curve $C/\mathbf{F}_q$ a cohomology space $H^1_{\text{rig}}(C)$ which is a vector space over a p-adic field, by lifting to characteristic zero and taking (overconvergent) De Rham cohomology.

How does it work?

There is a way of associating to a curve $C/\mathbf{F}_q$ a cohomology space $H^1_{\text{rig}}(C)$ which is a vector space over a p-adic field, by lifting to characteristic zero and taking (overconvergent) De Rham cohomology.

The Frobenius map F on C (raising coordinates to the q-th power) acts on $H^1_{\text{rig}}(C)$ in such a way that

$$Z(C,T) = \frac{\det(1 - FT | H^1_{\text{rig}}(C))}{(1-T)(1-qT)}.$$

We compute the matrix of F on $H^1_{\text{rig}}(C)$ to some p-adic precision that is sufficient to recover $Z(C,T)$ from the Weil conjectures.

How does it work?

There is a way of associating to a curve $C/\mathbf{F}_q$ a cohomology space $H^1_{\text{rig}}(C)$ which is a vector space over a p-adic field, by lifting to characteristic zero and taking (overconvergent) De Rham cohomology.

The Frobenius map F on C (raising coordinates to the q-th power) acts on $H^1_{\text{rig}}(C)$ in such a way that

$$Z(C,T) = \frac{\det(1 - FT | H^1_{\text{rig}}(C))}{(1-T)(1-qT)}.$$

We compute the matrix of F on $H^1_{\text{rig}}(C)$ to some p-adic precision that is sufficient to recover $Z(C,T)$ from the Weil conjectures.

This involves Hensel lifting F to some overconvergent completion of the lift of C, computing a basis for $H^1_{\text{rig}}(C)$ by linear algebra, applying the lift of F to this basis and reducing the resulting 1-forms back to this basis by repeatedly substracting exact forms.

Implementation

I have been working on my implementation on and off for about 2 years now. Until now the code was only available on my webpage and not even really as a package. Over the past two months (my visit here) it has gotten about 3 times as fast and been integrated in Magma (thanks to Allan and especially Steve!).

The main functions are the following:

Implementation

I have been working on my implementation on and off for about 2 years now. Until now the code was only available on my webpage and not even really as a package. Over the past two months (my visit here) it has gotten about 3 times as fast and been integrated in Magma (thanks to Allan and especially Steve!).

The main functions are the following:

```
intrinsic ZetaFunction(f::RngUPolElt, p::RngIntElt : N:=0, exactcoho:=false,
                        W0:=0, Winf:=0) -> FldFunRatUElt
```

Computes the zeta function of the smooth projective curve birational to the plane model defined by f. Here f is a polynomial in $K[x, y]$ where K is a number field such that $\mathcal{O}_K/(p) \cong \mathbf{F}_q$.

Implementation

I have been working on my implementation on and off for about 2 years now. Until now the code was only available on my webpage and not even really as a package. Over the past two months (my visit here) it has gotten about 3 times as fast and been integrated in Magma (thanks to Allan and especially Steve!).

The main functions are the following:

```
intrinsic ZetaFunction(f::RngUPolElt, p::RngIntElt : N:=0, exactcoho:=false,
                        W0:=0, Winf:=0) -> FldFunRatUElt
```

Computes the zeta function of the smooth projective curve birational to the plane model defined by f. Here f is a polynomial in $K[x, y]$ where K is a number field such that $\mathcal{O}_K/(p) \cong \mathbf{F}_q$.

```
intrinsic GonalityPreservingLift(C::Crv[FldFin]) -> RngUPolElt, AlgMatElt, AlgMatElt
```

Computes a 'good' lift f to characteristic zero of a curve of genus $3, 4, 5$ such that the degree in y is a small as possible.

Implementation

I have been working on my implementation on and off for about 2 years now. Until now the code was only available on my webpage and not even really as a package. Over the past two months (my visit here) it has gotten about 3 times as fast and been integrated in Magma (thanks to Allan and especially Steve!).

The main functions are the following:

```
intrinsic ZetaFunction(f::RngUPolElt, p::RngIntElt : N:=0, exactcoho:=false,
                        W0:=0, Winf:=0) -> FldFunRatUElt
```

Computes the zeta function of the smooth projective curve birational to the plane model defined by f. Here f is a polynomial in $K[x, y]$ where K is a number field such that $\mathcal{O}_K/(p) \cong \mathbf{F}_q$.

```
intrinsic GonalityPreservingLift(C::Crv[FldFin]) -> RngUPolElt, AlgMatElt, AlgMatElt
```

Computes a 'good' lift f to characteristic zero of a curve of genus $3, 4, 5$ such that the degree in y is a small as possible.

```
intrinsic ZetaFunction(C::Crv[FldFin] : Al := "Default") -> FldFunRatUElt
```

Combines the two previous functions to compute the zeta function of (almost) any curve of genus at most 5, switching to implementations of Harrison, Vercauteren or naive counting when appropriate. Somewhat unexpectedly my implementation usually runs faster than Harrison's in the case of hyperelliptic curves, even though both algorithms are the same in that case and my implementation has to do some extra work. Mainly due to better bounds on the p-adic precision and a precomputation before doing the cohomological reductions.

Some examples

Example: the modular curve $X_1(17)$

```
> P<x>:=PolynomialRing(RationalField());
> R<y>:=PolynomialRing(P);
> f:=y^4 + (x^3 + x^2 - x + 2)*y^3 + (x^3 - 3*x + 1)*y^2 - (x^4 + 2*x)*y + x^3 + x^2;
> p:=101;
> ZetaFunction(f,p);
(10510100501*T^10 + 6035503258*T^9 + 1900905345*T^8 + 396288448*T^7 + 60231754*T^6 + 6865620*T^5 + 596354*T^4 +
38848*T^3 + 1845*T^2 + 58*T + 1)/(101*T^2 - 102*T + 1)
```

Some examples

Example: the modular curve $X_1(17)$

```
> P<x>:=PolynomialRing(RationalField());
> R<y>:=PolynomialRing(P);
> f:=y^4 + (x^3 + x^2 - x + 2)*y^3 + (x^3 - 3*x + 1)*y^2 - (x^4 + 2*x)*y + x^3 + x^2;
> p:=101;
> ZetaFunction(f,p);
(10510100501*T^10 + 6035503258*T^9 + 1900905345*T^8 + 396288448*T^7 + 60231754*T^6 + 6865620*T^5 + 596354*T^4 +
38848*T^3 + 1845*T^2 + 58*T + 1)/(101*T^2 - 102*T + 1)
```

Example: a generic genus 5 curve

```
> C:=RandomGenus5CurveNonTrigonal(FiniteField(37));
> C;
Curve over GF(37) defined by
19*$.1^2 + 18*$.1*$.2 + 31*$.2^2 + $.1*$.3 + 19*$.2*$.3 + 25*$.3^2 + 8*$.1*$.4 + 17*$.2*$.4 + 29*$.3*$.4 + 19*$.4^2 +
18*$.1*$.5 + 27*$.2*$.5 + 26*$.3*$.5 + 14*$.4*$.5 + 32*$.5^2,
12*$.1^2 + 31*$.1*$.2 + 18*$.2^2 + 11*$.1*$.3 + 24*$.2*$.3 + 21*$.3^2 + 12*$.1*$.4 + 4*$.2*$.4 + 21*$.3*$.4 + 22*$.4^2 +
4*$.1*$.5 + 31*$.2*$.5 + 23*$.3*$.5 + 20*$.4*$.5 + 35*$.5^2,
21*$.1^2 + 35*$.1*$.2 + 17*$.2^2 + 8*$.1*$.3 + 12*$.2*$.3 + 32*$.3^2 + 34*$.1*$.4 + 22*$.2*$.4 + 24*$.3*$.4 + 18*$.4^2 +
19*$.1*$.5 + 10*$.2*$.5 + 19*$.3*$.5 + 10*$.4*$.5
> ZetaFunction(C);
(69343957*T^10 - 5622483*T^9 + 1418284*T^8 + 217671*T^7 - 2997*T^6 + 6604*T^5 - 81*T^4 + 159*T^3 + 28*T^2 - 3*T + 1)
/(37*T^2 - 38*T + 1)
```

Some examples

Example: the modular curve $X_1(17)$

```
> P<x>:=PolynomialRing(RationalField());
> R<y>:=PolynomialRing(P);
> f:=y^4 + (x^3 + x^2 - x + 2)*y^3 + (x^3 - 3*x + 1)*y^2 - (x^4 + 2*x)*y + x^3 + x^2;
> p:=101;
> ZetaFunction(f,p);
(10510100501*T^10 + 6035503258*T^9 + 1900905345*T^8 + 396288448*T^7 + 60231754*T^6 + 6865620*T^5 + 596354*T^4 +
38848*T^3 + 1845*T^2 + 58*T + 1)/(101*T^2 - 102*T + 1)
```

Example: a generic genus 5 curve

```
> C:=RandomGenus5CurveNonTrigonal(FiniteField(37));
> C;
Curve over GF(37) defined by
19*$.1^2 + 18*$.1*$.2 + 31*$.2^2 + $.1*$.3 + 19*$.2*$.3 + 25*$.3^2 + 8*$.1*$.4 + 17*$.2*$.4 + 29*$.3*$.4 + 19*$.4^2 +
18*$.1*$.5 + 27*$.2*$.5 + 26*$.3*$.5 + 14*$.4*$.5 + 32*$.5^2,
12*$.1^2 + 31*$.1*$.2 + 18*$.2^2 + 11*$.1*$.3 + 24*$.2*$.3 + 21*$.3^2 + 12*$.1*$.4 + 4*$.2*$.4 + 21*$.3*$.4 + 22*$.4^2 +
4*$.1*$.5 + 31*$.2*$.5 + 23*$.3*$.5 + 20*$.4*$.5 + 35*$.5^2,
21*$.1^2 + 35*$.1*$.2 + 17*$.2^2 + 8*$.1*$.3 + 12*$.2*$.3 + 32*$.3^2 + 34*$.1*$.4 + 22*$.2*$.4 + 24*$.3*$.4 + 18*$.4^2 +
19*$.1*$.5 + 10*$.2*$.5 + 19*$.3*$.5 + 10*$.4*$.5
> ZetaFunction(C);
(69343957*T^10 - 5622483*T^9 + 1418284*T^8 + 217671*T^7 - 2997*T^6 + 6604*T^5 - 81*T^4 + 159*T^3 + 28*T^2 - 3*T + 1)
/(37*T^2 - 38*T + 1)
```

The fields are chosen very small in these examples, but the same thing works over $GF(p)$ with p about 2^{15} and $GF(q)$ with $q = p^n$ much larger still!

Harvey's $p^{1/2+\epsilon}$ algorithm

Both Kedlaya's original algorithm and our new algorithm are polynomial time in d_x, d_y and n, but exponential in $\log(p)$ hence not polynomial time. Also in practice the characteristic p has to be rather small.

Finding an algorithm for computing zeta functions of curves (even hyperelliptic ones) which is simultaneously polynomial time in d_x, d_y and the field size is a big open problem which we do not expect to be solved any time soon.

Harvey's $p^{1/2+\epsilon}$ algorithm

Both Kedlaya's original algorithm and our new algorithm are polynomial time in d_x, d_y and n, but exponential in $\log(p)$ hence not polynomial time. Also in practice the characteristic p has to be rather small.

Finding an algorithm for computing zeta functions of curves (even hyperelliptic ones) which is simultaneously polynomial time in d_x, d_y and the field size is a big open problem which we do not expect to be solved any time soon.

For hyperelliptic curves David Harvey has improved the dependence of runtime on the characteristic to $p^{1/2+\epsilon}$. Let ω be an exponent for matrix multiplication.

Theorem (Harvey, 2007)

Kedlaya's algorithm can be modified to run in time:

$$O((p^{1/2} g^{\omega+5/2} n^{7/2} + \log(p) g^8 n^5)^{1+\epsilon}).$$

Note that the exponents of g and n are slightly worse than in Kedlaya's algorithm.

Implementation

Harvey's $p^{1/2+\epsilon}$ algorithm for hyperelliptic curves is not included in Magma (as far as I know), but the C code (for the primefield case $q = p$) is on his webpage and does come with SAGE. This is the only case I know of where SAGE beats Magma for point counting.

Implementation

Harvey's $p^{1/2+\epsilon}$ algorithm for hyperelliptic curves is not included in Magma (as far as I know), but the C code (for the primefield case $q = p$) is on his webpage and does come with SAGE. This is the only case I know of where SAGE beats Magma for point counting.

Moritz Minzlaff generalised Harvey's algorithm to superelliptic curves (y^k instead of y^2) and implemented it completely in Magma. However, as far as I know this code does not come with Magma.

The Magma package was released under a GPL license according to his paper, but I could not find it online anymore. The main functions seem to be named `PFrobeniusAction(a,h,N)` and `ZetaFunction(a,h)`.

Harvey's average polynomial time algorithm

Now take a hyperelliptic curve $C : y^2 = f(x)$ with $f \in \mathbf{Z}[x]$ and for a prime of good reduction p let C_p denote its reduction mod p. Let B be a bound on the absolute values of the coefficients of f.

Harvey's average polynomial time algorithm

Now take a hyperelliptic curve $C : y^2 = f(x)$ with $f \in \mathbf{Z}[x]$ and for a prime of good reduction p let C_p denote its reduction mod p. Let B be a bound on the absolute values of the coefficients of f.

Theorem

Kedlaya's algorithm can be modified to return all $Z(C_p, T)$ for $p < N$ in time

$$O(g^{8+\epsilon} N \log^2(N) \log^{1+\epsilon}(BN))$$

Harvey's average polynomial time algorithm

Now take a hyperelliptic curve $C : y^2 = f(x)$ with $f \in \mathbf{Z}[x]$ and for a prime of good reduction p let C_p denote its reduction mod p. Let B be a bound on the absolute values of the coefficients of f.

Theorem

Kedlaya's algorithm can be modified to return all $Z(C_p, T)$ for $p < N$ in time

$$O(g^{8+\epsilon} N \log^2(N) \log^{1+\epsilon}(BN))$$

Since there are roughly $N/\log(N)$ primes (of good reduction) up to N, this is polynomial time per prime. Therefore, the algorithm runs in average polynomial time.

This algorithm has not been implemented directly, but a lower tech version of it (Hasse-Wit matrix) combined with Baby Step Giant Step has been used by Harvey and Sutherland to collect Sato-Tate distributions for curves of genus 2 and 3.

My future plans

First, I want to extend the algorithm to find a 'good' lift to all curves admitting a degree 3 or 4 map to $\mathbf{P}^1$ instead of just curves of genus ≤ 5.

My future plans

First, I want to extend the algorithm to find a 'good' lift to all curves admitting a degree 3 or 4 map to $\mathbf{P}^1$ instead of just curves of genus ≤ 5.

Second, I want to adapt the point counting code so that it can compute Coleman integrals, which are used in the effective Chabauty method for finding points on curves over number fields. This is joint work in progress with Jennifer Balakrishnan, the code is almost finished.

My future plans

First, I want to extend the algorithm to find a 'good' lift to all curves admitting a degree 3 or 4 map to $\mathbf{P}^1$ instead of just curves of genus ≤ 5.

Second, I want to adapt the point counting code so that it can compute Coleman integrals, which are used in the effective Chabauty method for finding points on curves over number fields. This is joint work in progress with Jennifer Balakrishnan, the code is almost finished.

Third, using Harvey's methods, I want to develop and implement adaptations of my algorithm for general curves that will also run in time $p^{1/2+\epsilon}$ and average polynomial time, while only getting slightly worse in terms of the geometry d_x, d_y. This should enormously expand the range of curves for which we can compute Sato-Tate distributions and more generally L-functions.

My future plans

First, I want to extend the algorithm to find a 'good' lift to all curves admitting a degree 3 or 4 map to $\mathbf{P}^1$ instead of just curves of genus ≤ 5.

Second, I want to adapt the point counting code so that it can compute Coleman integrals, which are used in the effective Chabauty method for finding points on curves over number fields. This is joint work in progress with Jennifer Balakrishnan, the code is almost finished.

Third, using Harvey's methods, I want to develop and implement adaptations of my algorithm for general curves that will also run in time $p^{1/2+\epsilon}$ and average polynomial time, while only getting slightly worse in terms of the geometry d_x, d_y. This should enormously expand the range of curves for which we can compute Sato-Tate distributions and more generally L-functions.

A lot of (really technical) work remains to be done.

Smooth projective hypersurfaces

Let X be a projective hypersurface in $\mathbf{P}^{n+1}_{\mathbf{F}_q}$ defined by $P \in \mathbf{F}_q[x_0, \ldots, x_{n+1}]$ homogeneous of degree d and $U = \mathbf{P}^{n+1}_{\mathbf{F}_q} \setminus X$ its complement.

Of particular interest are quartic hypersurfaces in $\mathbf{P}^3_{\mathbf{F}_q}$ since these are K3 surfaces.

Smooth projective hypersurfaces

Let X be a projective hypersurface in $\mathbf{P}^{n+1}_{\mathbf{F}_q}$ defined by $P \in \mathbf{F}_q[x_0, \ldots, x_{n+1}]$ homogeneous of degree d and $U = \mathbf{P}^{n+1}_{\mathbf{F}_q} \setminus X$ its complement.

Of particular interest are quartic hypersurfaces in $\mathbf{P}^3_{\mathbf{F}_q}$ since these are K3 surfaces.

For higher dimension hypersurfaces the situation is quite different from the curve case. All known practical algorithms are p-adic in nature, the most important ones are:

- Direct method,
- Deformation method,
- Harvey's algorithm for arbitrary dimension.

We will give a brief overview and talk about what is in Magma and what could be.

Direct method

Abbott, Kedlaya and Roe (2006): Kedlaya's algorithm for smooth hypersurface $X \subset \mathbf{P}^{n+1}_{\mathbf{F}_q}$ of degree d with $q = p^a$. There is some Magma code on Kedlaya's webpage, but very slow.

Main idea: compute the cohomology $H^{n+1}_{rig}(U)$ of $U = \mathbf{P}^{n+1} \setminus X$ with its action of Frobenius.

Running time should be *about* $O((p^n d^{n^2} a^n)^{1+\epsilon})$.

Direct method

Abbott, Kedlaya and Roe (2006): Kedlaya's algorithm for smooth hypersurface $X \subset \mathbf{P}^{n+1}_{\mathbf{F}_q}$ of degree d with $q = p^a$. There is some Magma code on Kedlaya's webpage, but very slow.

Main idea: compute the cohomology $H^{n+1}_{rig}(U)$ of $U = \mathbf{P}^{n+1} \setminus X$ with its action of Frobenius.

Running time should be *about* $O((p^n d^{n^2} a^n)^{1+\epsilon})$.

This has been and is being improved by Costa, Harvey and Kedlaya (201?). Costa has some implementation in C++, not publicly available yet.

Running time should be *about* $O((p d^{n^2} a^n)^{1+\epsilon})$ and there are also $\sqrt{p}$ and average polynomial time versions.

Can the dependence on the dimension be improved?

Deformation method

Lauder (2004) proposes to put the hypersurface $X \subset \mathbf{P}^{n+1}_{\mathbb{F}_q}$ with $q = p^a$ into a (smooth) family Y/T over some open $T \subset \mathbb{P}^1$ with:

- Y_0 a diagonal hypersurface,
- $Y_1 = X$.

Deformation method

Lauder (2004) proposes to put the hypersurface $X \subset \mathbf{P}^{n+1}_{\mathbb{F}_q}$ with $q = p^a$ into a (smooth) family Y/T over some open $T \subset \mathbb{P}^1$ with:

- Y_0 a diagonal hypersurface,
- $Y_1 = X$.

The relative cohomology $H^n_{rig}(Y/T)$ is an overconvergent F-isocrystal:

- a p-adic differential equation (the Gauss-Manin connection),
- a Frobenius structure with matrix $\Phi(t)$.

The Frobenius structure is horizontal w.r.t the connection, so $\Phi(t)$ satisfies a p-adic differential equation.

Deformation method

Lauder (2004) proposes to put the hypersurface $X \subset \mathbf{P}^{n+1}_{\mathbb{F}_q}$ with $q = p^a$ into a (smooth) family Y/T over some open $T \subset \mathbb{P}^1$ with:

- Y_0 a diagonal hypersurface,
- $Y_1 = X$.

The relative cohomology $H^n_{rig}(Y/T)$ is an overconvergent F-isocrystal:

- a p-adic differential equation (the Gauss-Manin connection),
- a Frobenius structure with matrix $\Phi(t)$.

The Frobenius structure is horizontal w.r.t the connection, so $\Phi(t)$ satisfies a p-adic differential equation.

Main idea: compute $\Phi(0)$ (easy, Y_0 is diagonal) then solve the differential equation for $\Phi(t)$, finally find $\Phi(1)$ and deduce $Z(X, T)$.

Complexity

Pancratz and me (2013) improved this in terms of complexity (a bit) and in practice (a lot). Let ω be an exponent for matrix multiplication and e the basis of the natural logarithm.

Theorem

The (simplified) time complexity of the deformation method is:

$$O\left((pd^{n(\omega+4)}e^{n(\omega+1)}a^3)^{1+\epsilon}\right).$$

Note that this is polynomial in d^n instead of d^{n^2} and has a^3 instead of a^n.

Pancratz has implemented this in C using FLINT (for families defined over $\mathbf{Q}$). The code can be found on his GitHub and my webpage.

Harvey's algorithm

Harvey (2014) gave $p^{1/2+\epsilon}$ and average polynomial time algorithms for any scheme X of finite type over $\mathbf{Z}$ (no smoothness assumptions).

Harvey's algorithm

Harvey (2014) gave $p^{1/2+\epsilon}$ and average polynomial time algorithms for any scheme X of finite type over $\mathbf{Z}$ (no smoothness assumptions).

Theorem

Let $X/\mathbf{Z}$ be a scheme of finite type and X_p the reduction of X modulo p.

- *$Z(X_p, T)$ can be computed in time $O(p \log^{1+\epsilon}(p))$.*
- *$Z(X_p, T)$ can be computed in time $O(p^{1/2} \log^{2+\epsilon})$.*
- *$Z(X_p, T)$ can be computed for all $p < N$ in time $O(N \log^{3+\epsilon}(N))$.*

Harvey's algorithm

Harvey (2014) gave $p^{1/2+\epsilon}$ and average polynomial time algorithms for any scheme X of finite type over $\mathbf{Z}$ (no smoothness assumptions).

Theorem

Let $X/\mathbf{Z}$ be a scheme of finite type and X_p the reduction of X modulo p.

- *$Z(X_p, T)$ can be computed in time $O(p \log^{1+\epsilon}(p))$.*
- *$Z(X_p, T)$ can be computed in time $O(p^{1/2} \log^{2+\epsilon})$.*
- *$Z(X_p, T)$ can be computed for all $p < N$ in time $O(N \log^{3+\epsilon}(N))$.*

To be able to work in this generality, Harvey avoids the use of cohomology. This might also have some disadvantages, the matrices get very large! Moreover, the running time is only polynomial in d^{n^2} again.

Not clear how practical, Costa is implementing it for curves now. Elsenhans did an implementation of a very special type of K3 surface (degree 2) in Magma recently: `WeilPolynomialOfDegree2K3Surface`.

My future plans

The deformation method is better in terms of d^n and a, but not $p^{1/2+\epsilon}$ or average polynomial time yet.

However, again it should be possible to use Harvey's methods to adapt the deformation method so that it runs in $p^{1/2+\epsilon}$ and average polynomial time as well, while remaining polynomial in d^n and a (with slightly worse exponents).

This seems to be possible and I am currently writing it up.

I want to do a Magma implementation of the deformation method (both the original and the improved version).

A lot of (really technical) work remains to be done.