Counting points on smooth plane quartics

David Harvey

University of New South Wales

Number Theory Down Under, University of Newcastle
25th October 2014

(joint work with Andrew V. Sutherland, MIT)

Zeta functions of curves

$p =$ a prime

$C =$ a curve of genus g over $\mathbf{F}_p$

The zeta function of C is the generating function

$$Z(T) := \exp\left(\sum_{k \geq 1} \frac{\#C(\mathbf{F}_{p^k})}{k} T^k\right).$$

Zeta functions of curves

Example: let $p = 7$ and let C be the zero locus in $\mathbf{P}^2$ of

$$F = X^4 + 2Y^4 - Z^4 + XY^3 - 2YZ^3 + 2ZX^3.$$

By naive point enumeration, we find that

$$\begin{aligned} \#C(\mathbf{F}_p) &= 3, & \#C(\mathbf{F}_{p^4}) &= 2587, \\ \#C(\mathbf{F}_{p^2}) &= 55, & \#C(\mathbf{F}_{p^5}) &= 17183, \\ \#C(\mathbf{F}_{p^3}) &= 345, & \#C(\mathbf{F}_{p^6}) &= 117091, \end{aligned}$$

and thus

$$Z(T) = 1 + 3T + 32T^2 + 202T^3 + 1497T^4 + 10317T^5 + 72400T^6 + \cdots$$

Zeta functions of curves

The zeta function always has the form

$$Z(T) = \frac{L(T)}{(1-T)(1-pT)}$$

where $L(T) \in \mathbf{Z}[T]$ has degree $2g$.

Much of my research is about algorithms for computing $L(T)$.

In our example,

$$L(T) = 1 - 5T + 15T^2 - 33T^3 + 15 \cdot 7T^4 - 5 \cdot 7^2 T^5 + 7^3 T^6.$$

Smooth plane quartics

In this talk we focus on *smooth plane quartics*.

So C is the zero locus in $\mathbf{P}^2$ of a homogeneous $F(X, Y, Z)$ of degree 4.

"Smooth" means that $\frac{\partial F}{\partial X}$, $\frac{\partial F}{\partial Y}$, $\frac{\partial F}{\partial Z}$ have no common zeroes in $\mathbf{P}^2$.

Equivalently, these are the *non-hyperelliptic curves of genus* 3 over $\mathbf{F}_p$.

Smooth plane quartics

For any such curve, $L(T)$ has the form

$$L(T) = 1 + a_1 T + a_2 T^2 + a_3 T^3 + p a_2 T^4 + p^2 a_1 T^5 + p^3 T^6$$

where a_1, a_2, a_3 are integers.

Given F as input, our mission is to compute a_1, a_2, a_3.

Smooth plane quartics

Equivalently, we are computing the *Frobenius eigenvalues* for C.

In our example, the complex roots of $L(1/T)$ are

$$\{-1.315 \pm 2.296i, 1.559 \pm 2.138i, 2.256 \pm 1.382i\}.$$

They all have absolute value $\sqrt{p}$.

The Sato–Tate conjecture

A major application of such algorithms is to study the Sato–Tate conjecture.

Assume C is a smooth plane quartic over the *rational numbers*, defined by $F \in \mathbf{Z}[X, Y, Z]$.

For all but finitely many primes, F modulo p defines a smooth plane quartic over $\mathbf{F}_p$.

So we can define $L_p(T)$ for these primes.

The Sato–Tate conjecture predicts how the Frobenius eigenvalues are distributed as p varies.

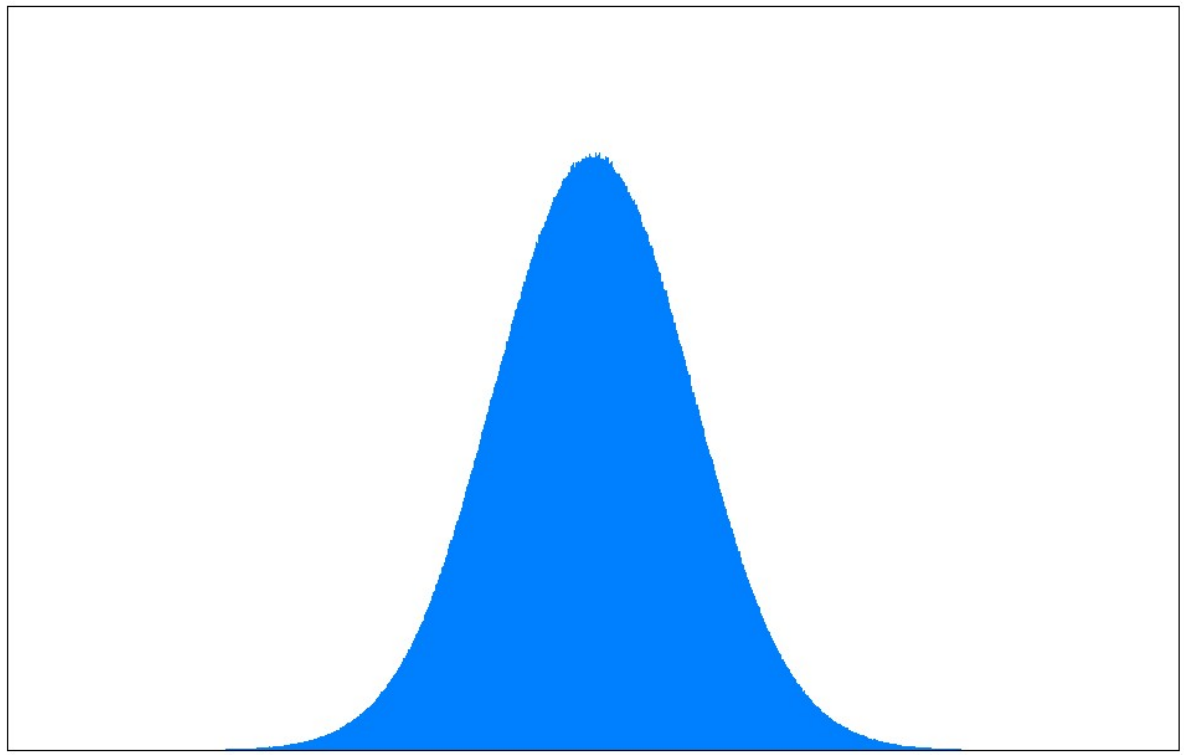
The Sato–Tate conjecture

Sato–Tate recently proved for genus 1 (elliptic curves).

Still wide open for $g \geq 2$.

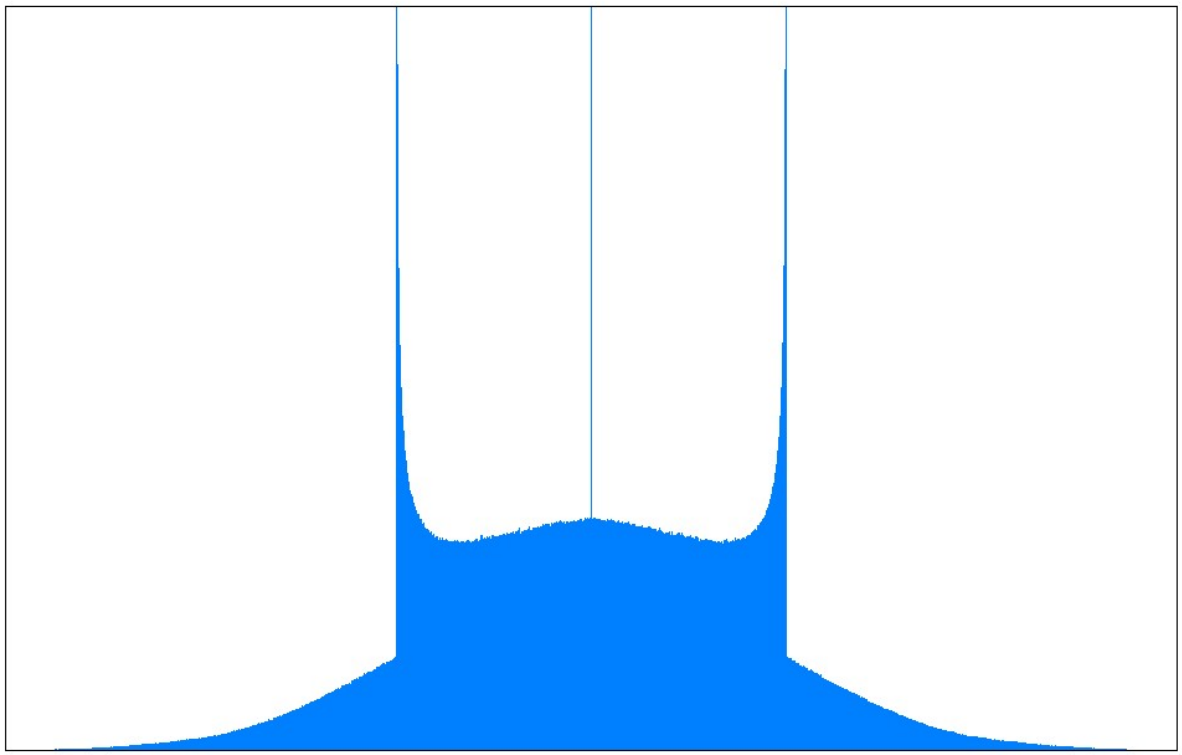
Over the last few years, Kedlaya and Sutherland (and others) have investigated it numerically for *hyperelliptic* curves of genus 2 and 3.

Distribution of the normalized trace (sum of eigenvalues) for the hyperelliptic genus 3 curve $y^2 = x^7 - x + 1$, for $p \leq 2^{30}$:



Took about 4 CPU days to generate the data. (Prior to my "average polynomial time" algorithm, would have needed about 4 CPU years.)

Same graph for $y^2 = x^7 + 3x^6 + 2x^5 + 6x^4 + 4x^3 + 12x^2 + 8x$, which has interesting automorphisms.



Computing $L(T)$ — naive algorithm

To compute $L(T)$, it's enough to count points over $\mathbf{F}_p$, $\mathbf{F}_{p^2}$ and $\mathbf{F}_{p^3}$.

Let's ignore the points at infinity (which are easy to count separately).

To count the number of points over $\mathbf{F}_{p^i}$, the naive algorithm is to plug in every possible value of $x_0 \in \mathbf{F}_{p^i}$ into $F(x_0, y, 1)$, and count the number of roots of the resulting quartic polynomial in $\mathbf{F}_{p^i}[y]$.

Thus we can compute $L(T)$ in time $O(p^{3+\varepsilon})$.

For $p \sim 1000$, an efficient implementation would take a few hours (?).

Computing $L(T)$ — existing methods

I will briefly discuss existing implementations of:

- Deformation method.
- Monsky–Washnitzer cohomology.
- Abbott–Kedlaya–Roe (AKR).

DISCLAIMER:

Performance depends on hardware and choice of language.

This is NOT intended to be a rigorous, fair comparison.

Computing $L(T)$ — existing methods

Deformation method (Lauder, $\sim$2004).

State of the art is C/FLINT implementation of Jan Tuitman and Sebastian Pancratz (2013).

Last week I asked Jan to run some examples for me.

For a random plane quartic over $\mathbf{F}_p$ with $p = 1009$, can compute $L(T)$ in about 3 minutes using 650 MB RAM.

Time and space are both $O(p^{1+\varepsilon})$, both in theory and in practice.

Computing $L(T)$ — existing methods

Monsky–Washnitzer cohomology (Kedlaya, 2001).

Originally only for hyperelliptic curves.

Generalised to nondegenerate curves by Denef–Castryck–Vercauteren (2006). Never been implemented in full.

Jan Tuitman has worked out an even more general variant (2014).

For $p = 1009$, his current Magma implementation gets $L(T)$ (modulo p) in about 40 seconds, using about 20 MB RAM.

Again time and space are $O(p^{1+\varepsilon})$.

Computing $L(T)$ — existing methods

[ASIDE:

To compute $L(T)$, it is enough to compute $L(T)$ modulo p.

Given $L(T)$ modulo p, can lift to $L(T) \in \mathbf{Z}[T]$ by applying a "baby-step giant-step" algorithm to $J(\mathbf{F}_p)$, the Jacobian of the curve.

This has complexity $O(p^{1/4+\varepsilon})$.

Not yet implemented, but anticipated to be *lightning fast*.]

Computing $L(T)$ — existing methods

Abbott–Kedlaya–Roe (2005).

Works for any smooth projective hypersurface, of any degree and dimension.

Original complexity was probably $O(p^{3+\varepsilon})$.

Around 2010, I improved this to $O(p^{1+\varepsilon})$ (unpublished).

The space complexity drops to $O(\log p)$, i.e., essentially *constant*.

Edgar Costa's current C++/NTL implementation can handle $p = 1009$ in about 0.01 seconds.

Computing $L(T)$ — existing methods

Briefly, a few related algorithms:

- Schoof–Pila (1990): theoretically $O((\log p)^C)$ for some C (probably $C \gg 12$?), but not practical for interesting range of p.
- Ritzenthaler (2004): AGM method, but only over $\mathbf{F}_{2^n}$.
- Bauer–Teske–Weng (2005): only Picard curves, i.e.,

$$Y^3 Z = c_0 X^4 + \cdots + c_4 Z^4.$$

New algorithm

Today I will give an overview of a new algorithm.

It is closely related to the one implemented by Costa, but conceptually much simpler.

My current C code can do $p = 1009$ in about 0.001 seconds.

As before, time complexity is $O(p^{1+\varepsilon})$ and space complexity is $O(\log p)$.

New algorithm

Let C be the smooth plane quartic, defined by $F \in \mathbf{F}_p[X, Y, Z]$ (degree 4).

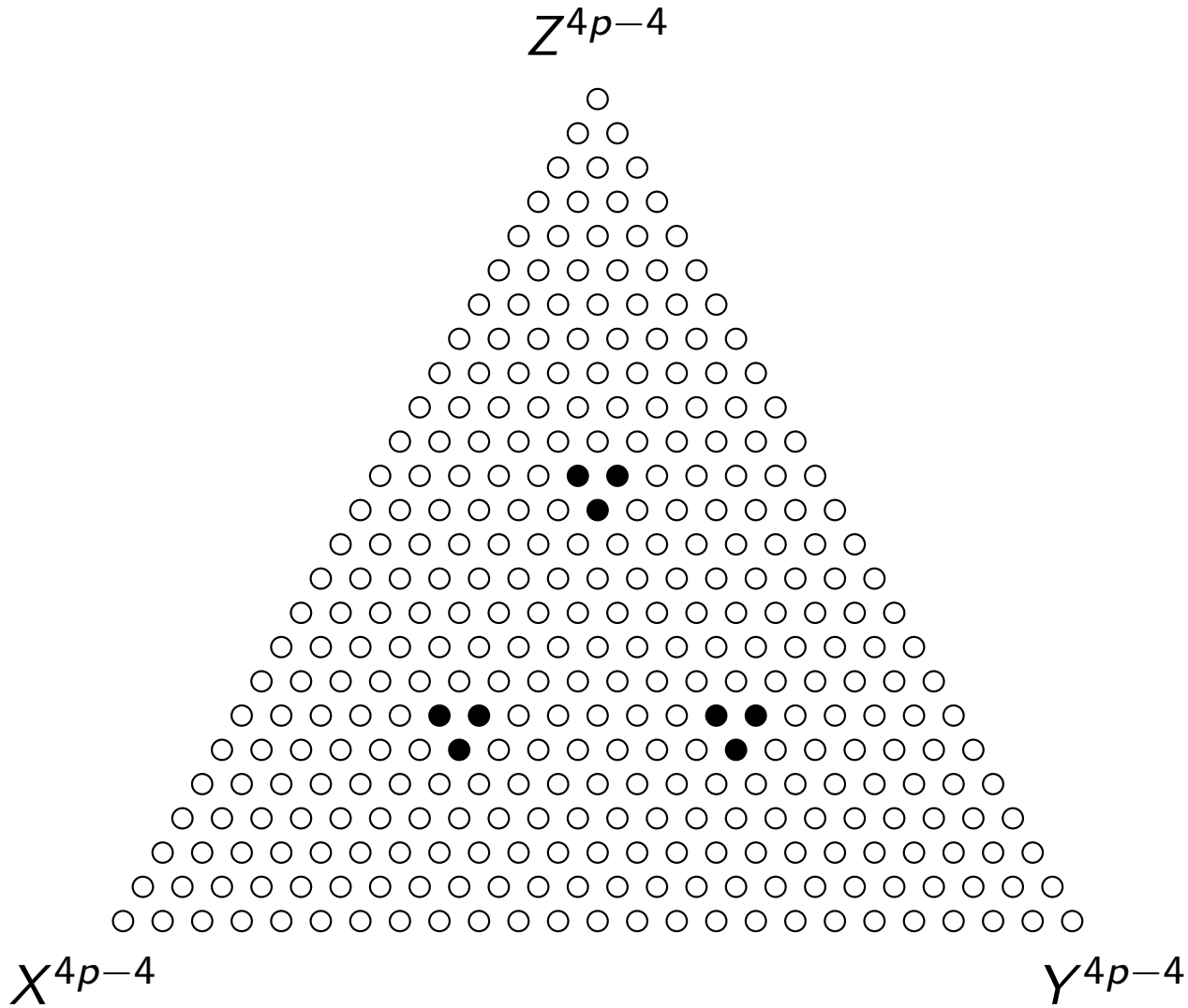
Let $(F^n)_{i,j,k}$ be the coefficient of $X^i Y^j Z^k$ in F^n.

The *Hasse–Witt matrix* of C is the 3×3 matrix over $\mathbf{F}_p$ defined by

$$W := \begin{bmatrix} (F^{p-1})_{p-1,p-1,2p-2} & (F^{p-1})_{2p-1,p-1,p-2} & (F^{p-1})_{p-1,2p-1,p-2} \\ (F^{p-1})_{p-2,p-1,2p-1} & (F^{p-1})_{2p-2,p-1,p-1} & (F^{p-1})_{p-2,2p-1,p-1} \\ (F^{p-1})_{p-1,p-2,2p-1} & (F^{p-1})_{2p-1,p-2,p-1} & (F^{p-1})_{p-1,2p-2,p-1} \end{bmatrix}.$$

(This the matrix of the Cartier–Manin operator with respect to a certain basis for the space of regular differentials on the curve.)

Target coefficients of F^{p-1} for $p = 7$:



New algorithm

It turns out that

$$L(T) = \det(I - WT) \pmod{p},$$

i.e., $L(T)$ is the characteristic polynomial of W modulo p (modulo some change of variables).

So all we need to do is efficiently compute the nine coefficients $(F^{p-1})_{i,j,k}$ appearing in the formula for W.

New algorithm

Key idea: there are *relations* between neighbouring coefficients of F^{p-2}.

Let us sketch how to derive these relations.

Start with

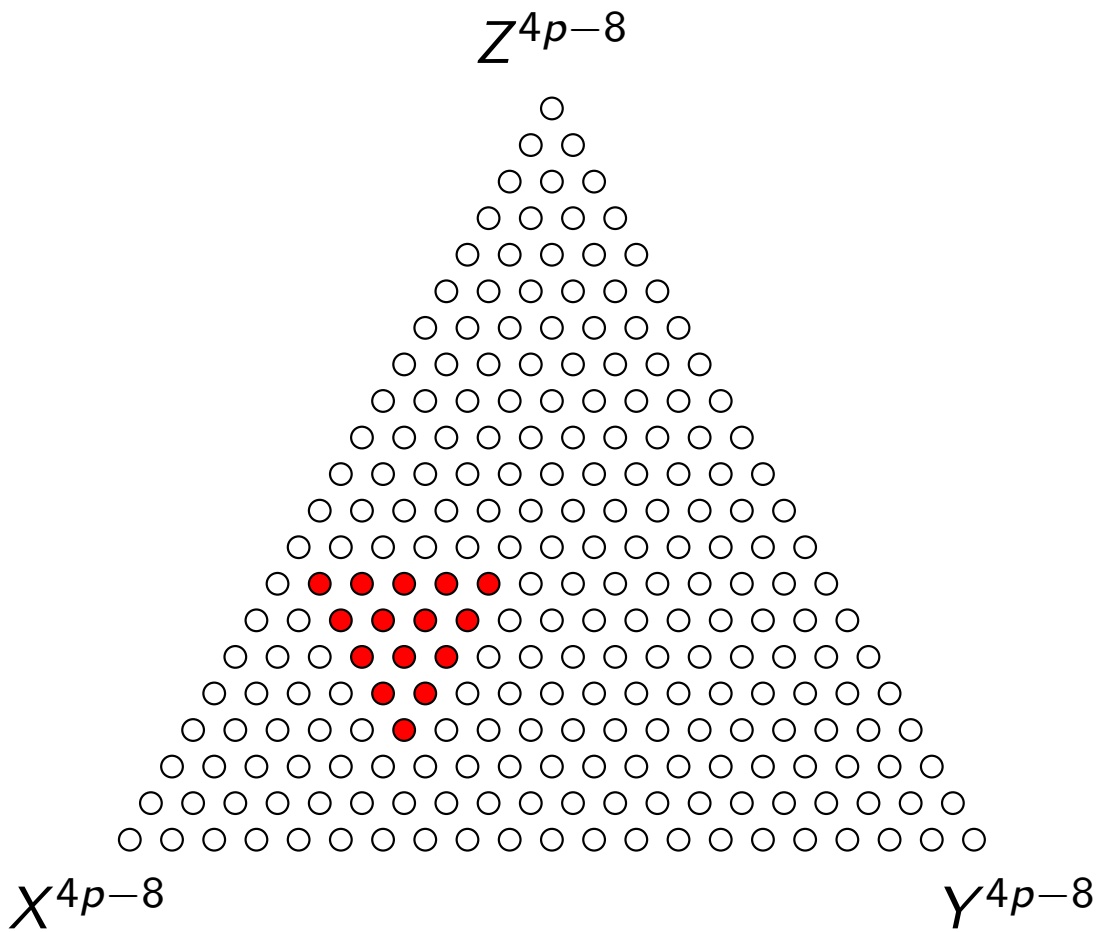
$$F^{p-1} = F \cdot F^{p-2}.$$

Equating coefficients of $X^i Y^j Z^k$ yields

$$(F^{p-1})_{i,j,k} = \sum_{i_0+j_0+k_0=4} F_{i_0,j_0,k_0} (F^{p-2})_{i-i_0,j-j_0,k-k_0}.$$

This expresses a given coefficient of F^{p-1} as a linear combination of a bunch of nearby coefficients of F^{p-2}.

Example: the coefficient of $(F^{p-1})_{12,5,7}$ is a linear combination of the following coefficients of F^{p-2}:



New algorithm

But there is another way to get such a relation.

Define $\partial_X := X\frac{\partial}{\partial X}$.

By the chain rule we have

$$\partial_X(F^{p-1}) = -\partial_X F \cdot F^{p-2}.$$

Equating coefficients gives

$$i(F^{p-1})_{i,j,k} = -\sum_{i_0+j_0+k_0=4} i_0 F_{i_0,j_0,k_0}(F^{p-2})_{i-i_0,j-j_0,k-k_0}.$$

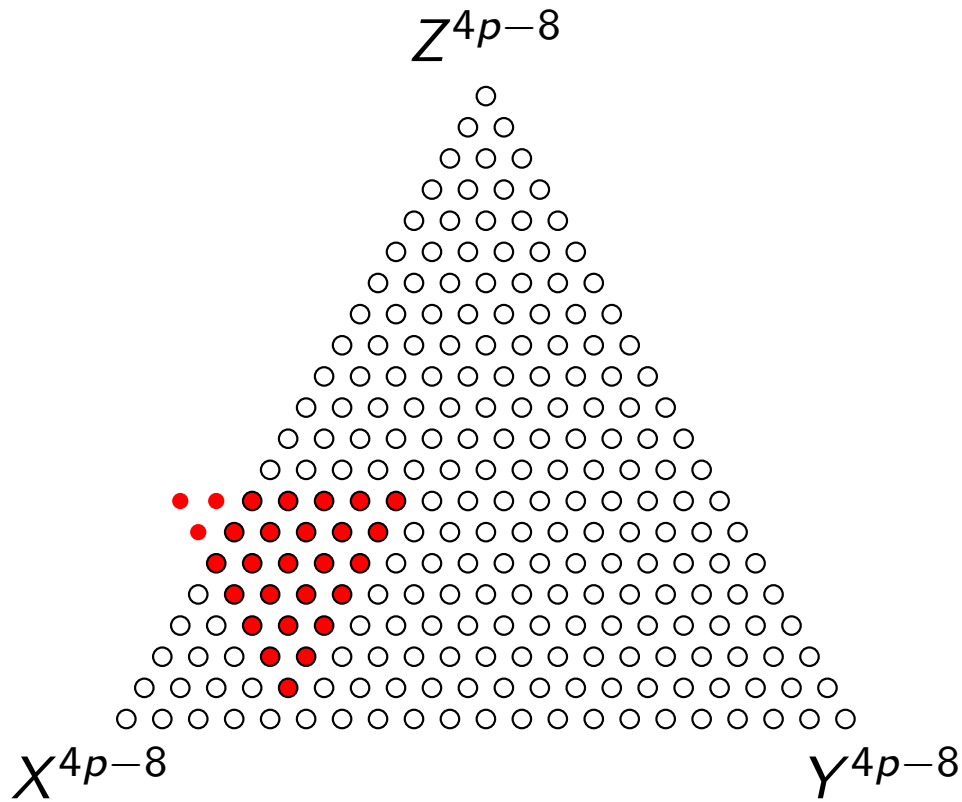
New algorithm

Eliminating the coefficient of F^{p-1}, we get

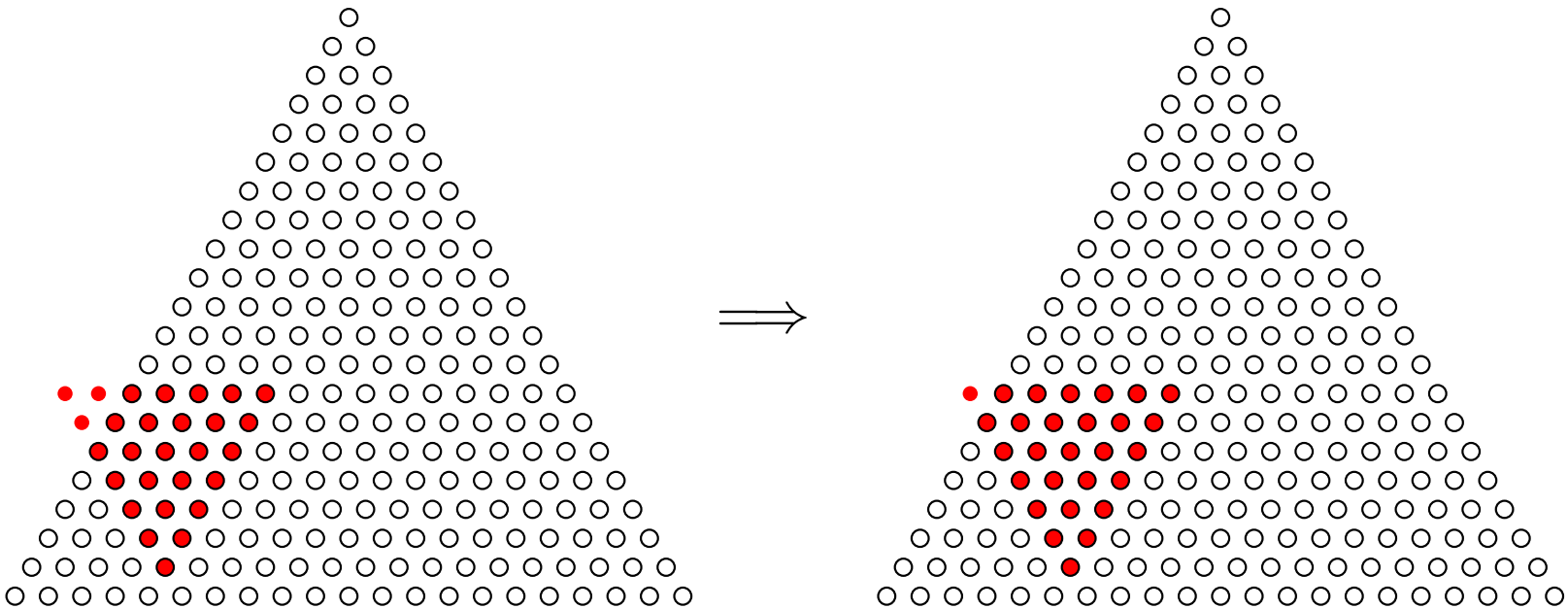
$$\sum_{i_0+j_0+k_0=4} (i+i_0) F_{i_0,j_0,k_0} (F^{p-2})_{i-i_0,j-j_0,k-k_0} = 0.$$

This is a nontrivial *relation* among nearby coefficients of F^{p-2}.

Now consider a larger triangle, of side length 6:



Algebraic magic: the relation derived above, plus an analogous relation for ∂_Y, are enough to *move the triangle around*:



New algorithm

I am being slightly dishonest.

We need one additional condition: that F is *nondegenerate*.

This means:

1. $F(1,0,0)$, $F(0,1,0)$ and $F(0,0,1)$ are nonzero, and
2. $F(0,Y,Z)$, $F(X,0,Z)$ and $F(X,Y,0)$ have no repeated factors.

New algorithm

Condition (1) means that the curve does not pass through the points $(1,0,0)$, $(0,1,0)$, $(0,0,1)$.

Condition (2) means that the curve intersects the coordinates axes ($X=0$, $Y=0$ and $Z=0$) transversally.

Nondegeneracy is a very mild condition. Almost every smooth plane quartic satisfies it.

If we are given an equation that is *not* nondegenerate, a random coordinate change will likely produce a nondegenerate one (provided p is not too small), and this does not change the zeta function.

New algorithm

So now the algorithm is very simple to explain:

Start with a triangle at one of the vertices, e.g. near X^{4p-8}. The coefficients of F^{p-2} here are trivial to compute.

Move it around until we arrive at the target coefficients.

Deduce the relevant coefficients of F^{p-1} by using the relation $F^{p-1} = F \cdot F^{p-2}$.

Assemble the Hasse–Witt matrix W.

We're done!

New algorithm

Notice we have not used the smoothness of C anywhere.

Taking smoothness into account, it is possible to work with 16×16 matrices instead of 28×28.

This yields a speedup by a factor of about 3.

New algorithm

Let's be more precise about the running time.

Let $\mathrm{M}(n)$ be the bit complexity of multiplying n-bit integers.

The new algorithm runs in time $O(p\,\mathrm{M}(\log p))$, i.e., it performs $O(p)$ additions/multiplications of integers with $O(\log p)$ bits.

(The version with Costa is the same.)

Previous algorithms, including Kedlaya, Lauder, etc, all run in time $O(\mathrm{M}(p \log p))$, because they multiply polynomials of degree p with coefficients of bit size $O(\log p)$.

In theory, we save a factor of about $\log p$.

The improved space complexity also makes a huge difference in practice.

Variants

Two possible variants:

- "square root time" version
- "average polynomial time" version

Square root time complexity

Using the Strassen–Pollard approach, can improve time complexity to $O(p^{1/2+\varepsilon})$, at the expense of increasing space complexity to $O(p^{1/2+\varepsilon})$.

To my knowledge this has only been implemented for hyperelliptic curves (by Bostan–Gaudry–Schost and myself) and superelliptic curves (Minzlaff 2010).

Should be competitive for moderate size p; will try it soon.

Average polynomial time complexity

Now suppose C is a plane quartic defined over $\mathbf{Q}$.

Using the same ideas from my paper on the hyperelliptic case (2014), we can construct an algorithm that computes $L_p(T)$ for all primes $p \leq N$ in time $O(N \log^{3+\varepsilon} N)$.

In other words, the average time per prime is $O((\log p)^{4+\varepsilon})$.

Sutherland and I are working on an implementation of this right now.

Based on our experience with hyperelliptic curves, we expect it to be extremely fast.

More details soon!