Counting points on projective hypersurfaces

David Harvey

New York University

19th October 2010
Workshop on Elliptic Curve Computation
Microsoft Research, Redmond, Washington, USA

Contents

- previous results
- main result
- computational examples
- sketch of algorithm

Notation

$q = p^a$

$X =$ variety over $\mathbf{F}_q$

$$Z_X(T) = \exp\left(\sum_{r \geq 1} \frac{\#X(\mathbf{F}_{q^r})}{r} T^r\right)$$

Weil Conjectures: $Z_X(T) \in \mathbf{Q}(T)$

Goal: compute $Z_X(T)$ efficiently when p is "large"

Previous results

Schoof (1985) and descendants, ℓ-adic/CRT method:

Curve of genus g in time

$$(a \log p)^{g^{O(1)}}.$$

Asymptotically best known approach for fixed g and large p.

Practical for $g \leq 2$, not aware of implementations for $g \geq 3$.

Not yet available for higher-dimensional varieties (except abelian varieties).

Previous results

Lauder (2004), p-adic deformation method:

Degree d smooth hypersurface $X \subset \mathbf{P}^n$ in time

$$p^{2+\epsilon} \operatorname{poly}(d^n a).$$

Dense input size is $d^n a \log p$ bits.

Note $p^{2+\epsilon}$ contribution is *independent of dimension*.

Previous results

Kedlaya (2000), using Monsky–Washnitzer cohomology:

Genus g hyperelliptic curve in time

$$p^{1+\epsilon}\,\mathrm{poly}(ga).$$

H. (2007), modification of Kedlaya's algorithm:

$$p^{0.5+\epsilon}\,\mathrm{poly}(ga).$$

e.g. $g = 3$, $p \approx 3 \times 10^{16}$ is feasible (30 hours on single CPU)

Minzlaff (2008): superelliptic curve in time

$$p^{0.5+\epsilon}\,\mathrm{poly}(ga).$$

Main result

Setup for today:

$X \subset \mathbf{P}^n_{\mathbf{F}_q}$ smooth variety defined by $f \in \mathbf{F}_q[x_0, \ldots, x_n]$

$\deg f = d$

Then

$$Z_X(T) = P(T)^{(-1)^n} \prod_{i=0}^{n-1} \frac{1}{1 - q^i T},$$

where $P(T) \in \mathbf{Z}[T]$; our goal is to compute $P(T)$.

Main result

Theorem (tentative)

Assume $X \subset \mathbf{P}^n_{\mathbf{F}_q}$ *satisfies a certain smoothness condition (see following slides).*

Then $P(T)$ *can be computed to* N *significant p-adic digits in time*

$$p^{0.5+\epsilon}\,\mathrm{poly}(d^n a)N^n.$$

In particular taking $N = O(d^n a)$*, we can compute* $Z_X(T)$ *in time*

$$p^{0.5+\epsilon} d^{n^2+O(n)} a^{n+O(1)}.$$

(Significant digits means: modulo $p^{N+\text{“Hodge polygon”}}$)

Main result

What is the smoothness condition?

For $S \subseteq \{0, \ldots, n\}$, let

$$J_S = \langle \partial_i f \rangle_{i \in S} + \langle x_i \, \partial_i f \rangle_{i \notin S}.$$

(Here $\partial_i = \partial / \partial x_i$.)

(Note: if $p \mid d$ we use instead

$$J_S = \langle f \rangle + \langle \partial_i f \rangle_{i \in S} + \langle x_i \, \partial_i f \rangle_{i \notin S};$$

algorithm still works (?), complexity may increase.)

Main result

Smoothness condition:

> *There exists S with $|S| \leq d$ such that J_S defines the empty scheme, i.e.* $\operatorname{rad} J_S = (x_0, \ldots, x_n)$.

Geometric interpretation: for all subsets $T \supseteq S$, the intersection of X with the coordinate hyperplanes defined by $\{x_i\}_{i \notin T}$ is smooth.

If $d > n$, can take $S = \{0, \ldots, n\}$, equivalent to X itself being smooth.

Computational examples

February 2010: toy implementation in Sage.

September 2010: second toy implementation in Sage.

Today we discuss several examples of the second toy implementation, running on a 16-core Opteron server with 96 GB RAM (thanks to Harvard Mathematics Department).

Currently under development: C++/NTL implementation

Computational examples

Random degree 4 in $\mathbf{P}^3$ (K3 surfaces) over a prime field.

$\deg P(T) = 21$

Used $N = 2$ (ok provided that p is not too 'small').

p	cores	wall time
1009	12	3.4h
10007	12	7.7h
100003	12	18.4h
1000003	6	121h

Computational examples

Random degree 4 in $\mathbf{P}^3$ over $\mathbf{F}_{23^2}$ (non-prime field).

Used $N = 3$. (Oops, actually insufficient. Should have used $N = 4$. For $p \geq 43$ it would be ok to use $N = 3$.)

Wall time was 11.0h, running on 12 cores.

Computational examples

Random degree 5 in $\mathbf{P}^3$ over a prime field.

$\deg P(T) = 52$

For $p = 1009$, $N = 2$, wall time was 66 hours running on 12 cores.

Note that $N = 2$ is enough to determine the first coefficient of $P(T)$, i.e. to determine $\#X(\mathbf{F}_p)$.

However, for the whole zeta function we would need $N = 5$, estimated running time 47 days on 12 cores!

Hopefully this can be made more feasible with a tighter implementation, and possibly using the 'interpolation trick'.

Computational examples

Random degree 3 in $\mathbf{P}^4$ over a prime field.

$\deg P(T) = 10$

For $p = 401$, $N = 3$ (sufficient for whole zeta function), wall time was 59 hours running on 12 cores.

(Also tried degree 4 in $\mathbf{P}^4$... didn't terminate in time for this talk.)

Sketch of algorithm

Algorithm based on AKR = Abbott–Kedlaya–Roe ("Bounding Picard numbers of surfaces using p-adic cohomology", 2005).

Basic idea:

$$P(T) = \det(1 - q^{-1}\sigma_q T | H^n_{\text{rig}}(U))$$

where

$$U = \mathbf{P}^n \backslash X$$

$$\sigma_q = q\text{-th power Frobenius.}$$

Sketch of algorithm

$H^n_{\text{rig}}(U)$ is essentially Monsky–Washnitzer cohomology (a p-adic analytic de Rham cohomology); we get

$$H^n_{\text{rig}}(U) \cong H^n_{\text{dR}}(\widetilde{U}/\mathbf{Q}_q),$$

where

$\widetilde{f} = p$-adic lift of f to $\mathbf{Z}_q[x_0, \ldots, x_n]$

$\widetilde{U} =$ lift of U defined by $\widetilde{f}$, i.e. with coordinate ring

$$\widetilde{A} = \text{degree-0 piece of } \mathbf{Z}_q[x_0, \ldots, x_n, \widetilde{f}^{-1}]$$

Sketch of algorithm

Explicit description of $H^n_{\mathrm{dR}}(\widetilde{U}/\mathbf{Q}_q)$ (Griffiths): let

$$\Omega = \sum_{i=0}^{n} (-1)^i x_i \, dx_0 \wedge \cdots (\text{omit } dx_i) \cdots \wedge dx_n.$$

Then $H^n_{\mathrm{dR}}(\widetilde{U}/\mathbf{Q}_q)$ is the quotient of degree-0 piece of

$$\left\langle \frac{G}{\widetilde{f}^m} \Omega : G \in \mathbf{Q}_q[x_0, \ldots, x_n] \right\rangle$$

by

$$\left\langle \left(\frac{\partial_i G}{\widetilde{f}^m} - m \frac{G \partial_i \widetilde{f}}{\widetilde{f}^{m+1}} \right) \Omega \right\rangle,$$

i.e. relations declare that exact forms are zero in cohomology.

Sketch of algorithm

$H^n_{\mathrm{dR}}(\widetilde{U}/\mathbf{Q}_q)$ is finite dimensional.

Using the cohomology relations + linear algebra, we can easily compute a basis consisting of forms

$$\frac{x^w}{\widetilde{f}^m}\Omega$$

where m is 'small' (here $x^w = x_0^{w_0} \cdots x_n^{w_n}$).

Also, there is a *reduction algorithm* that, given any differential $G\Omega/\widetilde{f}^m$, repeatedly subtracts off relations to find the unique linear combination of basis elements cohomologous to the given differential.

This is called the *reduction* of $G\Omega/\widetilde{f}^m$.

Sketch of algorithm

There is a Frobenius action on $H^n_{\mathrm{dR}}(\widetilde{U}/\mathbf{Q}_q)$, induced by $x_i \mapsto x_i^p$. The image of $\widetilde{f}^{-1}$ is given by a p-adically convergent power series.

The image under Frobenius of any cohomology basis element $x^w\Omega/\widetilde{f}^m$ can be p-adically approximated by a linear combination of terms of the form

$$\frac{x_0^{pu_0-1}\cdots x_n^{pu_n-1}}{\widetilde{f}^{pk}}\Omega,$$

where $\sum u_i = kd$.

(AKR used a different series expansion with at least p^n terms.)

Sketch of algorithm

Overall strategy:

1. Compute a basis for H^n_{dR}
2. Compute series approximations for images of cohomology basis elements under absolute Frobenius (need about N^n terms in each expansion to get precision N in final result)
3. Apply reduction algorithm to reduce to basis elements; yields matrix of absolute Frobenius acting on H^n_{dR}
4. Multiply by conjugates to obtain matrix of q-power Frobenius
5. Characteristic polynomial is $P(T)$ (up to some normalisations)

Sketch of algorithm

The main novelty of our algorithm (relative to AKR) is a procedure called *controlled reduction*. Consider a differential

$$\frac{x^u G}{\widetilde{f}^m}\Omega,$$

where $\deg G = \beta := dn - n$.

Choose a monomial x^v of degree d. (We assume $S = \emptyset$ and $u_i \gg 0$; otherwise the choice of x^v may be restricted.)

Then there exists G' of degree β such that the above differential is cohomologous to

$$\frac{x^{u-v} G'}{\widetilde{f}^{m-1}}\Omega.$$

In other words, we have reduced the pole order of the differential without increasing the number of terms used to represent it.

Sketch of algorithm

General case of controlled reduction is technical/complicated; we illustrate the idea with a special case.

Consider the diagonal hypersurface in $\mathbf{P}^3$ given by

$$f = x_0^4 + x_1^4 + x_2^4 + x_3^4 = 0.$$

Suppose we want to reduce $x^u G\Omega/\widetilde{f}^m$ in the direction $x^v = x_0x_1x_2x_3$. Let

$$x^w = x_0^{w_0} x_1^{w_1} x_2^{w_2} x_3^{w_3}$$

be a typical monomial in G, so $\sum w_i = \deg G = \beta = 9$.

At least one of the w_i, say w_j, must be ≥ 3.

Sketch of algorithm

Since $\partial_j f = 4x_j^3$ we may use the reduction relations to obtain

$$\begin{aligned}
\frac{x^u x^w}{\widetilde{f}^m}\Omega &= \frac{(x^u x^w x_j^{-3})x_j^3}{\widetilde{f}^m}\Omega \\
&\sim \frac{1}{4(m-1)}\frac{\partial_j(x^u x^w x_j^{-3})}{\widetilde{f}^m}\Omega \\
&= \frac{(u_j + w_j - 3)}{4(m-1)}\frac{(x^u x_j^{-1})(x^w x_j^{-3})}{\widetilde{f}^m}\Omega \\
&= \frac{(u_j + w_j - 3)}{4(m-1)}\frac{x^{u-v}G'}{\widetilde{f}^m}\Omega
\end{aligned}$$

for some G' as desired.

Key point is that one should use different reduction relations for the various terms in G.

Sketch of algorithm

So far this gives an algorithm with running time

$$p^{1+\epsilon}\,\mathrm{poly}(d^n a)N^n.$$

But how to get from $p^{1+\epsilon}$ down to $p^{0.5+\epsilon}$?

The reduction

$$\frac{x^{u-\ell v}G}{\widetilde{f}^{m-\ell}}\Omega \Longrightarrow \frac{x^{u-(\ell+1)v}G'}{\widetilde{f}^{m-\ell-1}}\Omega$$

induces a map $R_{u,v}(\ell)$ on the space of polynomials V_β of degree β, i.e. sending G to G'.

This map is linear. Moreover, choosing a basis for V_β, the entries of the matrix of $R_{u,v}(\ell)$ are *degree* 1 *rational functions of* ℓ.

Sketch of algorithm

Computing p reduction steps is equivalent to computing a matrix product of the form

$$R_{u,v}(p-1)\cdots R_{u,v}(1)R_{u,v}(0).$$

There is an algorithm of Chudnovsky–Chudnovsky (improved recently by Bostan–Gaudry–Schost), depending on asymptotically fast polynomial arithmetic, that can compute such a matrix product in time $O(p^{0.5+\epsilon})$ rather than the naive $O(p)$ (ignoring the dependence on the size of the matrix).

(One data point: for K3 surfaces, these are 220×220 matrices.)

Open questions

1. Any traction from the 'interpolation trick'? (i.e. $\prod_{\ell=0}^{p-1} R_{u,v}(\ell)$ varies p-adically analytically with coordinates of u.)
2. Can we combine with deformation techniques? Can we deform in $p^{0.5}$ time, or even p^1 time? Would this reduce time from $(d^n a)^{n+O(1)}$ to $(d^n a)^{O(1)}$?
3. Any systematic improvements for surfaces with sparse equations?
4. Can we drop the annoying smoothness condition?
5. What about smooth *affine* varieties?
6. Complete intersections?
7. Weighted projective space?
8. Can we drop smoothness condition? (Lauder & Wan?)

Thank you!