Algorithms for p-adic cohomology and p-adic heights

A dissertation presented

by

David Michael Harvey

to

The Department of Mathematics

in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of
Mathematics

Harvard University
Cambridge, Massachusetts

April 2008

Dissertation Advisor: Professor Barry Mazur David Michael Harvey

Algorithms for p-adic cohomology and p-adic heights

Abstract

In Part I, we present a new algorithm for computing the zeta function of a hyperelliptic curve over a finite field, based on Kedlaya's approach via p-adic cohomology. It is the first known algorithm for this task whose time complexity is polynomial in the genus of the curve and quasilinear in the square root of the characteristic of the base field. In Part II, we study and improve the Mazur–Stein–Tate algorithm for computing the p-adic height of a rational point on an elliptic curve $E/\mathbf{Q}$, where $p \geq 5$ is a prime of good ordinary reduction for E.

Contents

Acknowledgments

First I must thank my advisor, Barry Mazur, for giving me the confidence to tackle the problems in this thesis, for helping me to glimpse the bigger picture surrounding my research, and for elucidating not only a wide swathe of mathematics, but also what the path from graduate student to research mathematician could look like.

Kiran Kedlaya has, without a doubt, unofficially played the role of second advisor. His offhand suggestion that linear recurrences might somehow be usefully applied to p-adic cohomology eventually led to the main results of Part I of this thesis. He was closely involved in the development of these ideas, in particular helping to illuminate for me the subtle issues of p-adic error propagation, and encouraging me to consider the cryptographic applications of my results.

Many thanks to my dissertation committee — Barry Mazur, Kiran Kedlaya, and Samit Dasgupta — for their thoughtful comments on the manuscript and their assistance in polishing the final draft.

All of the results in this thesis were ultimately motivated by the problem of computing p-adic heights on elliptic curves. I owe my introduction to this beautiful problem to William Stein, whom I met at a graduate student workshop that he organised in August 2006. Since then I have learned so much from William about the practical aspects of computational number theory, and have been tremendously inspired by his boundless energy and never-say-die approach to solving problems. William also kindly provided access to the machine informally known as `sage.math`, funded by NSF grant #0555776 and hosted at the University of Washington, on which the example computations in Chapter 5 and Chapter 10

were performed.

My most valuable intellectual resource over the past few years has been the graduate student body in the mathematics department at Harvard. I would like to single out Jonathan (Jay) Pottharst for his friendship, generosity, infinite patience, and at times seemingly infinite knowledge.

Thanks to Christian Wuthrich for several discussions about his Ph.D. thesis that led to the algorithm of Chapter 7, several anonymous referees who made helpful suggestions concerning the published versions of the material in this thesis, and Andrew Sutherland, whose practical interest in my results led me to push the algorithm of Part I and its implementation much further than I would otherwise have dared try.

For the last five years, my parents, Helen and Andrew, and my sister Lisa, have supported me with their love and with regular shipments of Vita-Weat crackers and Tim Tams (delicacies native to the Australian continent) from a distance of some 16,000 kilometres ($\approx$ 10,000 miles).

Finally, my wife Lara, also my closest friend, who means everything to me, and who on reading this sentence characteristically decried its lack of a main verb.

Chapter 1

Introduction

This thesis consists of two parts, largely independent of each other.

1.1 Zeta functions of hyperelliptic curves over finite fields

In Part I we present a new algorithm for computing the zeta function of a hyperelliptic curve over a finite field. It is the first known algorithm for this task whose running time is polynomial in the genus of the curve and quasilinear in the *square root* of the characteristic of the base field. More precisely, we have:

Theorem 1. *Let C be a hyperelliptic curve of genus g over $\mathbf{F}_q$, where $q = p^n$. Let*

$$Z_C(T) = \frac{P_C(T)}{(1-T)(1-qT)}$$

be the zeta function of C.

(a) Let $N \geq 1$, and assume that $p > (2N-1)(2g+1)$. Then $P_C(T)$ may be

computed modulo p^N in time

$$\widetilde{O}(p^{1/2}N^{5/2}g^{\omega}n + N^4g^4n^2\log p).$$

(b) If $p > (gn + 4g + 3)(2g + 1)$, then $P_C(T)$ may be computed (as an element of $\mathbf{Z}[T]$) in time

$$\widetilde{O}(p^{1/2}g^{\omega+5/2}n^{7/2} + g^8n^6\log p).$$

(For the meaning of the symbols $\widetilde{O}$ and ω, see §1.3 below.)

Theorem 1 is proved in Chapter 2, §2.4. For sufficiently large p, the running times are dominated by the terms involving $p^{1/2}$. We have made no attempt to optimise the exponents of N, g and n in the secondary terms; no doubt these could be improved.

To develop the new algorithm, we start with Kedlaya's approach based on Monsky–Washnitzer cohomology [Ked01]. After rearranging the algebra of Kedlaya's algorithm somewhat, and writing out explicit formulae for the cohomological reduction steps, we are able to reformulate the algorithm in terms of certain linear recurrences with polynomial coefficients. We then apply a baby-step/giant-step algorithm of Chudnovsky and Chudnovsky [CC88] to solve these recurrences efficiently. The algorithm itself has already appeared in [Har07], but some of the example computations in this thesis are new.

There are of course many other algorithms available for computing $P_C(T)$, varying greatly in their running time complexities as functions of p, n and g. Chapter 17 of [CFA+06] gives an excellent and detailed survey. The 'holy grail' of this area of research is to find an algorithm whose complexity is simultaneously

polynomial in $\log p$, n and g; in other words, polynomial in the amount of data needed to write down the equation of C. No such algorithm is presently known.

Asymptotically speaking, the running time given in Theorem 1(b) does not beat existing algorithms, with respect to any of the individual parameters. For example, if we fix g and n, the Schoof–Pila algorithm [Pil90] has running time only polylogarithmic in p, so theoretically will outstrip our algorithm for sufficiently large p; in the other direction, for fixed p, Kedlaya's algorithm [Ked01] eventually performs better for sufficiently large n or g (indeed our algorithm will no longer be available, due to the hypothesis imposed on p).

Nevertheless, the square-root dependence on p and polynomial dependence on g mean that in practice the new algorithm is very efficient over a wide range of parameter choices. For example, Kedlaya and Sutherland [KS08] recently conducted a theoretical and empirical analysis of a suite of algorithms for determining zeta functions, for the purpose of computing L-functions of low-genus hyperelliptic curves over $\mathbf{Q}$. To compute the L-function of a single such curve $H/\mathbf{Q}$ to some prescribed accuracy, they must compute the zeta function of the reduction of H modulo p for *all* p up to some prescribed bound. For genus three curves they find that our algorithm is easily the best choice for p above about 2^{18} [KS08, Table 4].

Of course, we know that Schoof–Pila will eventually win for sufficiently large p — but while Schoof–Pila has been successfully used for impressively large point-counting problems in genus two [GS04], the dependence on g is exponential, and to the author's knowledge it has never been implemented in genus three or higher. With our new algorithm, the largest example we have tried in genus three is for p near $2^{53} \approx 10^{16}$ (see Chapter 5 for more details); it is not clear whether Schoof–Pila would be competitive yet in this range. Certainly, as pointed out by Kedlaya

and Sutherland, if one needs to handle all p up to some bound for a single curve over $\mathbf{Q}$, Schoof–Pila is not relevant for the feasible range of computation.

We mention also the algorithm of Bostan, Gaudry and Schost [BGS07], which seems to be quite closely related to ours. The running time of their algorithm is, like ours, quasi-linear in $p^{1/2}$ and polynomial in g, but they are only able to recover the zeta function *modulo* p (they must then use other techniques to fill in the missing information). Their approach is based on the Hasse–Witt matrix rather than on p-adic cohomology. At one stage they introduce $O(g)$ artificial p-adic 'safety digits', partly explaining why the exponent of g in their running time is worse than ours, but their algorithm is not truly p-adic in character. We will use many of their results concerning linear recurrences in Chapter 3.

It is natural to ask to what extent our algorithm may be generalised: can we give an algorithm for computing zeta functions of varieties more general than hyperelliptic curves, based on p-adic cohomology, whose running time is quasilinear in $p^{1/2}$? Indeed, one of the most promising aspects of Kedlaya's approach based on p-adic cohomology is that it should be applicable to a wide class of varieties, including those of dimension greater than one. We expect some varieties to succumb to our new method relatively easily, for example superelliptic curves (see [GG01] for a discussion on extending Kedlaya's original algorithm to this case). However, for more general varieties the picture is not so clear. The main difficulty is that our algorithm seems to depend quite strongly on the form of the equation defining the variety. In the case of a hyperelliptic curve $y^2 = f(x)$, the important feature seems to be that there is only a *single* term y^2 on the left, so that its p-th power $(y^2)^p$ still consists of a single monomial, allowing us to control the number of terms appearing in the power series of Chapter 4, §4.2.

Another possible direction for generalisation is to try to compute the zeta function for hyperelliptic curves in a parameterised family, more efficiently than they could be computed one at a time. Hubrechts recently adapted Kedlaya's algorithm for this purpose [Hub07], using Dwork's deformation theory and some ideas of Lauder. One interesting application he gives is to improve the running time complexity of Kedlaya's algorithm with respect to n (the degree of the field extension). However, his algorithm, like Kedlaya's, is only practical for small p. It would be interesting to study how to merge those ideas with our algorithm, to perhaps obtain an algorithm suitable for larger p.

1.2 Computing p-adic heights on elliptic curves

In Part II we study the problem of computing the canonical p-adic height of a rational point on an elliptic curve, in the good ordinary case, assuming that $p \geq 5$. The p-adic height appears, via the p-adic regulator term, in the p-adic analogues of the Birch and Swinnerton–Dyer conjectures [MTT86]. Fast machine evaluation of the height is an essential tool for numerical investigation of these conjectures.

The first computationally feasible method for evaluating the p-adic height to a reasonable degree of p-adic precision was given by Mazur, Stein and Tate [MST06]. Their algorithm proceeds in two steps. First they compute the value of the p-adic modular form $\mathbf{E}_2$ associated to the elliptic curve. The complexity of this step was not explicitly discussed in [MST06]; we will verify the following.

Theorem 2. *Let $E/\mathbf{Q}$ be an elliptic curve with good ordinary reduction at $p \geq 5$, let ω be an invariant differential for E, and let $N \geq 1$.*

(a) The value of $\mathbf{E}_2(E, \omega)$ may be computed modulo p^N in time $\widetilde{O}(pN^2)$.

(b) Suppose that $p > 6N$. The value of $\mathbf{E}_2(E, \omega)$ may be computed modulo p^N in time $\widetilde{O}(p^{1/2}N^{5/2} + N^4 \log p)$.

To prove this, we will follow the method proposed by [MST06], using an interpretation (due to Katz) of $\mathbf{E}_2$ as the 'slope' of the unit-root eigenspace of a Frobenius action on p-adic cohomology. Mazur, Stein and Tate suggest computing the Frobenius action explicitly via Kedlaya's algorithm; this yields the running time estimate in part (a). The estimate in part (b), which is superior for large p, is obtained by replacing Kedlaya's algorithm by our new algorithm for computing the Frobenius matrix (Theorem 5). This is the only link between Part I and Part II of this thesis.

Using the value of $\mathbf{E}_2$ as input, Mazur, Stein and Tate then compute the series expansion of the canonical p-adic sigma function associated to the curve, and use the sigma function to compute the p-adic height $h_p(P)$ of an arbitrary rational point $P \in E(\mathbf{Q})$. The bulk of Part II of this thesis is devoted to improving the asymptotic complexity of their algorithm. Along the way we also perform a thorough analysis of precision, enabling us to guarantee that the output is correct to a prescribed number of p-adic digits; such an analysis was not carried out in [MST06]. We prove the following in Chapter 9, assuming that E and P are fixed:

Theorem 3. *Given $\mathbf{E}_2(E, \omega)$ modulo $p^{N'-2}$ as input, $h_p(P)$ may be computed modulo p^N in time $\widetilde{O}(N^{3/2} \log^2 p)$.*

Here $N' = N + O(1)$ is an adjusted precision parameter, defined in Chapter 9. The complexity achieved by Theorem 3 improves on that of the algorithm of [MST06] with respect to both p and N; the former is reduced from *quadratic* in p to only polylogarithmic in p, the latter from N^4 to only $N^{3/2}$. Much of

Part II appeared in [Har08]; this thesis incorporates several further asymptotic improvements, in particular the use of the parameter λ in Chapter 9, which reduces the running time by a factor of $N^{1/2}$.

1.3 Notation and complexity assumptions

For running time estimates, we will write $F(X) = \widetilde{O}(X)$ to mean that $F(X) = O(X \log^k X)$ for some $k \geq 0$ as $X \to \infty$, and we say that $F(X)$ is quasilinear in X, or soft-linear in X.

Running times will generally refer to bit-complexity, except in Chapter 3, where we temporarily work in an algebraic model of computation.

For the algebraic model, let R be a commutative ring with identity. We denote by $\mathsf{M}(d)$ the number of ring operations required to multiply polynomials of degree d over R. We assume throughout that fast polynomial arithmetic is being used; that is, that $\mathsf{M}(d) = \widetilde{O}(d)$ (in fact we may take $\mathsf{M}(d) = O(d \log d \log\log d)$) by the Cantor–Kaltofen theorem [CK91]). We denote by $\mathsf{MM}(m)$ the number of ring operations required to multiply $m \times m$ matrices with entries in R. We assume that $\mathsf{MM}(m) = O(m^\omega)$ for some $2 \leq \omega \leq 3$. We have $\omega = 3$ for the naive matrix multiplication algorithm; see [Str69] for the simplest example of a matrix multiplication algorithm achieving $\omega < 3$.

For bit-complexity, we assume that asymptotically fast integer arithmetic is used; in particular, that addition, subtraction, multiplication and division in $\mathbf{Z}/L\mathbf{Z}$ each require time $\widetilde{O}(\log L)$. We assume that auxiliary costs (loop indexing, branching, etc) are all subsumed under the total arithmetic cost.

Part I

Computing zeta functions of hyperelliptic curves over finite fields

Chapter 2

The zeta function and Kedlaya's approach

In this chapter we review the definition of the zeta function, describe the theoretical framework shared by Kedlaya's algorithm [Ked01] and our new algorithm, and sketch how Kedlaya's algorithm works.

2.1 The zeta function

Throughout Part I we fix the following notation. Let $q = p^n$ be a prime power; we assume that $p \geq 3$ unless otherwise noted. The finite fields with p and q elements are denoted by $\mathbf{F}_p$ and $\mathbf{F}_q$. The p-adic field is denoted by $\mathbf{Q}_p$, and its degree n unramified extension by $\mathbf{Q}_q$. Their rings of integers are $\mathbf{Z}_p$ and $\mathbf{Z}_q$ respectively. We denote by $v_p : \mathbf{Q}_p \to \mathbf{Z}$ (or $v_p : \mathbf{Q}_q \to \mathbf{Z}$) the additive p-adic valuation, normalised so that $v_p(p) = 1$.

Fix a hyperelliptic curve $C/\mathbf{F}_q$ of genus $g \geq 1$, with a rational Weierstrass

point. We assume that it is the projective closure of the affine plane curve given by

$$y^2 = \overline{Q}(x),$$

where $\overline{Q} \in \mathbf{F}_q[x]$ is monic and squarefree, of degree $2g + 1$.

The zeta function of C is the power series

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

It is a generating function that encodes the number of points of C over all finite extensions of $\mathbf{F}_q$. According to the Weil Conjectures and Riemann Hypothesis for this curve, it is a rational function of the form

$$Z_C(T) = \frac{P_C(T)}{(1-T)(1-qT)},$$

where $P_C(T) = \sum_{i=0}^{2g} a_i T^i$ is a polynomial of degree $2g$ with integer coefficients, and moreover

$$a_0 = 1, \qquad a_{2g-i} = q^{g-i} a_i, \quad 0 \leq i \leq g.$$

Determining the zeta function of C is equivalent to determining $P_C(T)$, or simply the list of integers $a_1, \ldots, a_g$.

2.2 Construction of the Monsky–Washnitzer cohomology

The Monsky–Washnitzer cohomology assigns to each smooth affine variety X over $\mathbf{F}_q$ a sequence of finite dimensional vector spaces $H^i(X)$ $(i = 0, 1, \ldots)$ over $\mathbf{Q}_q$; the q-th power Frobenius on X induces a $\mathbf{Q}_q$-linear map $F : H^1(X) \to H^1(X)$. The idea of Kedlaya's algorithm is to take a certain affine variety C', obtained from C by deleting several points, and explicitly compute the matrix of F on a particularly simple basis for $H^1(C')$. The zeta function of C' (and hence of C) is then determined from this matrix via a Lefschetz fixed-point theorem. The reason that Monsky–Washnitzer cohomology is so attractive for computational purposes is that it is so *explicit*; approximations to elements of $H^1(X)$ can easily be written down and manipulated (at least by a computer).

In this section we describe explicitly the construction of the first Monsky–Washnitzer cohomology of C', which is essentially a de Rham cohomology obtained from a certain ring of overconvergent power series on a characteristic zero lift of C'. For proofs of various facts concerning Monsky–Washnitzer cohomology that we use in this and the following sections, we refer the reader to the article [vdP86] and also to Kedlaya's paper [Ked01]. Another nice expository paper is [Edi03].

Following Kedlaya, we define C' to be the affine variety consisting of C minus the point at infinity and the points whose x-coordinates are the zeroes of $\overline{Q}(x)$. The coordinate ring of C' is

$$\overline{A} = \mathbf{F}_q[x, y, y^{-1}]/(y^2 - \overline{Q}(x)).$$

We select an arbitrary lift $Q \in \mathbf{Z}_q[x]$ of $\overline{Q}(x)$, also monic and of degree $2g+1$. Let

$$A = \mathbf{Z}_q[x, y, y^{-1}]/(y^2 - Q(x))$$

be the lift of $\overline{A}$ associated to $Q(x)$. Let $A^\dagger$ be the weak completion of A introduced by Monsky and Washnitzer; explicitly, $A^\dagger$ is the ring of power series

$$\sum_{i \geq 0} \sum_{j \in \mathbf{Z}} a_{i,j} x^i y^j, \qquad a_{i,j} \in \mathbf{Z}_q,$$

such that $v_p(a_{i,j}) \to \infty$ at least linearly in $\max(i, |j|)$, modulo the relation $y^2 = Q(x)$.

The module Ω of differentials of $A^\dagger$ over $\mathbf{Q}_q$ consists of expressions of the form

$$\sum_{i \geq 0} \sum_{j \in \mathbf{Z}} a_{i,j} x^i y^j dx/y, \qquad a_{i,j} \in \mathbf{Q}_q,$$

where the $a_{i,j}$ are subject to the same decay conditions as above. Two differentials $\omega, \eta \in \Omega$ are cohomologous, denoted $\omega \sim \eta$, if there exists some $f \in A^\dagger \otimes \mathbf{Q}_q$ such that $\omega - \eta = df$. The quotient of Ω by this relation is by definition $H^1(C')$, the first Monsky–Washnitzer cohomology of C'.

We will work mainly in the submodule Ω^- of differentials on which the hyperelliptic involution acts by -1; explicitly, these have series representations of the form

$$\sum_{s \geq 0} \sum_{t \in \mathbf{Z}} a_{s,t} x^s y^{2t} dx/y, \qquad a_{s,t} \in \mathbf{Q}_q,$$

again with the same decay conditions. The corresponding subspace $H^1(C')^-$ of $H(C')$ has dimension $2g$, with basis $\{x^i dx/y\}_{i=0}^{2g-1}$. In other words, any differential

$\omega \in \Omega^-$ is cohomologous to a unique $\eta = B(x)dx/y$ where $B \in \mathbf{Q}_q[x]$ has degree at most $2g-1$. We call η the *reduction* of ω.

2.3 The action of Frobenius

Let $\sigma : \overline{A} \to \overline{A}$ be the absolute p-th power Frobenius map. This may be lifted to a $\mathbf{Z}_p$-endomorphism of $A^\dagger$ as follows (we denote the lift by σ also).

On $\mathbf{Z}_q$, we take the canonical Witt vector Frobenius. We set $x^\sigma = x^p$, and

$$\begin{aligned} y^\sigma &= (Q(x)^\sigma)^{1/2} \\ &= y^p \left(1 + \frac{Q^\sigma(x^p) - Q(x)^p}{y^{2p}}\right)^{1/2} \\ &= y^p \sum_{k=0}^{\infty} \binom{1/2}{k} \frac{(Q^\sigma(x^p) - Q(x)^p)^k}{y^{2pk}}. \end{aligned}$$

The above series converges in $A^\dagger$ (because $Q^\sigma(x^p) - Q(x)^p$ is divisible by p); the definition is essentially forced by the condition that y^2 should get mapped to $Q(x)^\sigma$. Similarly for y^{-1} we put

$$(y^{-1})^\sigma = y^{-p} \sum_{k=0}^{\infty} \binom{-1/2}{k} \frac{(Q^\sigma(x^p) - Q(x)^p)^k}{y^{2pk}}. \tag{2.1}$$

We further extend σ to Ω^- by $\sigma(f\,dg) = f^\sigma d(g^\sigma)$. This induces a map in cohomology

$$\sigma : H^1(C')^- \to H^1(C')^-.$$

This map is *not* $\mathbf{Q}_q$-linear; it is only semi-linear, that is $\sigma(\alpha\omega) = \alpha^\sigma \sigma(\omega)$ for $\alpha \in \mathbf{Q}_q$.

2.4 Computing the zeta function

We first consider the computation of the matrix of σ. Kedlaya's algorithm, which we sketch in §2.4.1 below, completes this task within the following complexity bound.

Theorem 4. *Kedlaya's algorithm computes the matrix of the p-th power Frobenius on $H^1(C')^-$ to precision p^N in time*

$$\widetilde{O}(pN^2g^2n).$$

For the proof, we refer the reader to Kedlaya's analysis [Ked01, pp. 335–336], and the discussion preceding Theorem 2 in [GG03]. Kedlaya assumes that p is fixed (and small), and consequently does not show the linear dependence on p, whereas Gaudry and Gürel study the dependence on p explicitly. Neither paper separates the contribution of N from that of g and n (they both assume throughout that $N = O(gn)$), but it is straightforward to track the contributions separately in their arguments.

The main result of Part I of this thesis is the following.

Theorem 5. *Let $N \geq 1$, and suppose that*

$$p > (2N - 1)(2g + 1). \tag{2.2}$$

Then the matrix of the p-th power Frobenius on $H^1(C')^-$ may be computed to precision p^N in time

$$\widetilde{O}(p^{1/2}N^{5/2}g^{\omega}n + N^4g^4n\log p).$$

In particular, for fixed N, g and n, the running time is $\widetilde{O}(p^{1/2})$.

Our new algorithm is therefore superior to Kedlaya's algorithm for fixed g and N and large enough p, but inferior for fixed p and large enough N or g.

Theorem 5 is proved in Chapter 4. The purpose of the assumption $p > (2N-1)(2g+1)$ is to simplify the analysis of denominators. It could be weakened somewhat, but the algorithm would become more complicated.

We now show how to deduce Theorem 1 from Theorem 5.

Proof of Theorem 1. Let M be the matrix of σ on $H^1(C')^-$, and let M' be the matrix of the q-th power Frobenius on $H^1(C')^-$. Assume that M has been computed via Theorem 5. Following Kedlaya, we may obtain M' via the product

$$M' = MM^{\sigma}M^{\sigma^2}\cdots M^{\sigma^{n-1}},$$

where σ acts entry-wise on M. Ring operations in $\mathbf{Z}_q/(p^N)$ require time $\widetilde{O}(Nn\log p)$, and applying any power of σ to an element of $\mathbf{Z}_q/(p^N)$ requires time $\widetilde{O}(Nn^2\log p)$, by the algorithm suggested in [Ked01, p. 334]. Using a repeated squaring trick [Ked01, p. 334], the product for M' may be evaluated using $O(\log n)$ matrix multiplications and $O(g^2\log n)$ applications of powers of Frobenius, so the total cost to compute M' from M is $\widetilde{O}(Ng^3n^2\log p)$.

Kedlaya shows that, according to a Lefschetz fixed point theorem for Monsky–Washnitzer cohomology [Ked01, p. 327] and a calculation relating the cohomology of C' to that of C [Ked01, p. 331], the numerator $P_C(T)$ of the zeta function of C is recovered as the characteristic polynomial of M'. Computing the characteristic polynomial requires $O(g^3)$ ring operations, or time $\widetilde{O}(Ng^3n\log p)$.

Therefore the cost of computing $P_C(T)$ modulo p^N from the matrix M is

$\widetilde{O}(Ng^3n^2 \log p)$. Combining this with the estimate from Theorem 5, we obtain part (a) of Theorem 1.

For part (b), using the Riemann Hypothesis for C, Kedlaya shows that if we take any

$$N \geq \frac{gn}{2} + (2g+1)\log_p 2,$$

then knowledge of $P_C(T) \bmod p^N$ determines $P_C(T)$ precisely as an element of $\mathbf{Z}[T]$ [Ked01, p. 332]. Since $\log_p 2 < 1$ for $p \geq 3$, it suffices to take $N = \lceil gn/2 \rceil + 2g + 1$, in which case

$$(2N-1)(2g+1) \leq (gn+4g+3)(2g+1).$$

For p larger than this, we may apply part (a). Substituting $N = O(gn)$ into the running time estimate from (a), we obtain (b). □

2.4.1 A sketch of Kedlaya's method for computing the matrix of σ

Kedlaya begins by computing an approximation to $y^{-\sigma}$ of the form

$$y^{-\sigma} \approx y^{-p} \sum_{k=0}^{Np-1} \frac{A_k(x)}{y^{2k}}, \tag{2.3}$$

where each A_k has degree at most $2g$. It is an approximation in two senses: it is truncated at a certain power of y^{-2}, and the coefficients are represented modulo $p^{N'}$, for some appropriately chosen N' (slightly larger than N, and certainly $O(N)$). Note that the time committed is already necessarily at least linear p, for the number of terms in the above series is about Np.

Given the above approximation for $y^{-\sigma}$, it is easy to write down a series approx-

imation for the images under σ of the basis elements $\{x^i\,dx/y\}_{i=0}^{2g-1}$ for $H^1(C')^-$. We have

$$\sigma(x^i\,dx/y) = x^{pi}d(x^p)y^{-\sigma} = px^{pi+p-1}y^{-\sigma}dx, \tag{2.4}$$

and so obtain

$$\sigma(x^i\,dx/y) \approx \sum_j \frac{F_{i,j}(x)}{y^{2j}}dx/y, \tag{2.5}$$

where each $F_{i,j}$ has degree at most $2g$, and where again the series have about Np terms.

For each i, Kedlaya then applies a reduction algorithm to the terms on the right hand side of (2.5). At each step, he uses the identities $y^2 = Q(x)$ and $2y\,dy = Q'(x)dx$, together with the fact that $d(x^s y^t) = 0$ in cohomology for any s and t, to reduce the term $F(x)y^{-2j}dx/y$ to a lower power of y^{-2} (or in some cases, y^2). The terms are swept up sequentially until reaching $j = 0$. At this point one has computed the reduction of $\sigma(x^i dx/y)$, whose coefficients give the $(i+1)$-th column of the Frobenius matrix. The reduction step is performed once for each j, so again the total time is proportional to at least p. We will examine reductions similar to these in some detail in Chapter 4, §4.1.

It is clear from the above description of Kedlaya's algorithm that to develop an algorithm whose running time is less than linear in p, we must abandon the series approximation (2.3) — it simply has too many terms. We will address this issue in Chapter 4, §4.2.

Chapter 3

Linear recurrences with polynomial coefficients

Let R be a commutative ring with identity, and $M(X)$ an $m \times m$ matrix of polynomials in $R[X]$. Given an initial vector $U_0 \in R^m$, define a sequence of vectors $\{U_i\}_{i \geq 0}$ iteratively by

$$U_i = M(i)U_{i-1}, \qquad i \geq 1. \tag{3.1}$$

This is a *linear recurrence with polynomial coefficients*. For simplicity, in the rest of this chapter we assume that the entries of $M(X)$ are *linear* polynomials, since this is all we need for the applications. It is straightforward to adapt the discussion to the case of higher degree polynomials.

Consider the problem of computing U_K from U_0, for some large K. Solving this problem efficiently is at the core of our fast method for computing zeta functions presented in the next chapter.

The naive algorithm is simply to iterate (3.1); this costs $O(m^2K)$ ring operations in R. Note that if M is a *constant* matrix (i.e. the entries are polynomials of degree zero), we can do much better. We have simply $U_K = M^K U_0$, and the matrix power M^K can be computed in $O(\mathsf{MM}(m) \log K)$ ring operations using repeated squaring.

Unfortunately, for a non-constant matrix, there is no known algorithm for computing U_K whose time complexity approaches this logarithmic dependence on K. In this chapter we discuss several algorithms that achieve *square-root* dependence on K.

3.1 The Chudnovsky–Chudnovsky algorithm

Chudnovsky and Chudnovsky [CC88] gave a baby-step/giant-step algorithm that determines U_K in $O(\mathsf{MM}(m)\mathsf{M}(\sqrt{K}) \log K) = \widetilde{O}(m^\omega K^{1/2})$ ring operations. While this is not nearly as good as the constant-matrix case, it still represents a drastic improvement over the naive algorithm when K is large.

The paper [BGS07] gives an excellent exposition of the Chudnovskys' algorithm. We sketch it in the simplest situation, where $m = 1$, $M(X) = X$, and $U_0 = 1$ (this case actually goes back to Strassen's integer factorisation algorithm [Str77]). In this case, we are simply asking to find

$$U_K = K \cdot (K-1) \cdots 2 \cdot 1 = K!$$

as an element of R. To simplify matters, assume that $K = k^2$ for an integer k.

Then we have

$$K! = F(0) \cdot F(k) \cdot F(2k) \cdots F((k-1)k)$$

where $F(X)$ is the polynomial

$$F(X) = (X+1)(X+2)\cdots(X+k). \tag{3.2}$$

The algorithm consists of three steps:

- Compute the coefficients of $F(X)$ using a product tree; i.e. recursively split (3.2) into halves, using fast polynomial multiplication to multiply each pair of factors. The cost is $O(\mathsf{M}(k)\log k) = \widetilde{O}(k)$. (If we instead had simply multiplied by $(X+1)$ up to $(X+k)$ in succession, we would have obtained running time quadratic in k, or *linear* in K.)

- Evaluate $F(X)$ simultaneously at the k points $X = 0, k, 2k, \ldots, (k-1)k$. Using fast multipoint evaluation techniques, the cost of this step is $\widetilde{O}(k)$. (Again, had we evaluated at each point separately, the cost would have been quadratic in k.)

- Recover $K!$ from (3.2), using $k - 1 = O(k)$ multiplications.

Since $k = \sqrt{K}$, we obtain a total running time of $\widetilde{O}(K^{1/2})$ ring operations. For the general case ($m > 1$), Chudnovsky and Chudnovsky replace the polynomial $F(X)$ by a matrix of polynomials, introducing the $\mathsf{MM}(m)$ term into the running time estimate.

3.2 The Bostan–Gaudry–Schost algorithm

Recently Bostan, Gaudry and Schost [BGS07] gave several improvements to the Chudnovskys' algorithm for solving linear recurrences. The algorithms they describe are more complicated, and we will not explain them in detail here. The running time is still $\widetilde{O}(K^{1/2})$ with respect to K, but they improve on the earlier algorithms in two ways. First, they save a factor of $\log K$, which is not insignificant for the feasible range of problem sizes we will consider in Chapter 5. Second, they separate the contributions of the $\mathsf{MM}(m)$ and $\mathsf{M}(K^{1/2})$ terms to the running time. Instead of a bound of the form $O(\mathsf{MM}(m)\mathsf{M}(\sqrt{K}))$, they obtain bounds of the form

$$O(\mathsf{MM}(m)\sqrt{K} + m^2\mathsf{M}(\sqrt{K})).$$

In the typical usage scenario where K is very large and m quite small, the second term dominates, and the practical effect is to improve the speed by a factor of about m over the Chudnovskys' algorithm. They also consider the problem of simultaneously computing *several* U_{K_i}, and show that this is no more expensive than computing a single U_K, as long as not too many U_{K_i}'s are requested.

The following theorem from [BGS07] is not precisely what we will need, but it is close enough that we will be able to adapt it without difficulty. To state it, we need to introduce some notation from [BGS07]. For any integer $s \geq 0$, they define a certain quantity $\mathsf{D}(1, 2^s, 2^s) \in R$. The definition is straightforward, but lengthy, and we will not give it here. The only fact we need (see [BGS07, p. 1787]) is that if $2, 3, \ldots, 2^s + 1$ are units in R, then $\mathsf{D}(1, 2^s, 2^s)$ is invertible in R, and that its inverse may be used to efficiently recover the inverses of certain other elements of R that are needed in the interpolation steps of their algorithm.

Theorem 6 ([BGS07, Theorem 15])**.** *Let* $0 < K_1 < K_2 < \cdots < K_r = K$ *be integers, and let* $s = \lfloor \log_4 K \rfloor$*. Suppose that* $2, 3, \ldots, 2^s + 1$ *are invertible in* R*, and that the inverse of* $\mathsf{D}(1, 2^s, 2^s)$ *is known. Suppose also that* $r < K^{\frac{1}{2}-\varepsilon}$*, with* $0 < \varepsilon < 1/2$*. Then* $U_{K_1}, \ldots, U_{K_r}$ *can be computed using*

$$O(\mathsf{MM}(m)\sqrt{K} + m^2\mathsf{M}(\sqrt{K}))$$

ring operations in R*.*

The theorem we require is a little stronger. For $k < k'$, let

$$M(k, k') = M(k')M(k' - 1) \cdots M(k + 2)M(k + 1).$$

Instead of just computing the images $U_{K_1}, \ldots, U_{K_r}$ of a single vector U_0, our aim is to compute the matrices $M(K_i, L_i)$ for a sequence of intervals (K_i, L_i). The following slight generalisation of Theorem 6 achieves this.

Theorem 7. *Let*

$$0 \leq K_1 < L_1 \leq K_2 < L_2 \leq \cdots \leq K_r < L_r \leq K$$

be integers, and let $s = \lfloor \log_4 K \rfloor$*. Suppose that* $2, 3, \ldots, 2^s + 1$ *are invertible in* R*, and that the inverse of* $\mathsf{D}(1, 2^s, 2^s)$ *is known. Suppose also that* $r < K^{\frac{1}{2}-\varepsilon}$*, with* $0 < \varepsilon < 1/2$*. Then* $M(K_1, L_1), \ldots, M(K_r, L_r)$ *can be computed using*

$$O(\mathsf{MM}(m)\sqrt{K} + m^2\mathsf{M}(\sqrt{K}))$$

ring operations in R*.*

Remark. When we discuss the application to computing zeta functions in the next chapter, we will pass to soft-O estimates of running times, and consequently ignore the distinction between the $\sqrt{K}$ and $\mathsf{M}(\sqrt{K})$ terms in the above bound.

Proof. The algorithm is almost exactly the same as the one given in the proof of [BGS07, Theorem 15], so we will not spell out all the details. To explain it, we first give a very high-level sketch of their algorithm. In "Step 0", they compute a sequence of matrices

$$M(0, H), M(H, 2H), \ldots, M((B-1)H, BH), \tag{3.3}$$

where both H and B are a small constant factor away from $\sqrt{K}$, and where $BH \geq K$. They apply these matrices successively to U_0 to compute U_{kH} for all $0 \leq k \leq B$. Each target index K_i will fall within one of the intervals $[kH, (k+1)H]$. Then they perform a "refining" step, where they deduce U_{K_i} from U_{kH} by evaluating appropriate products of $M(X)$ over (much smaller) subintervals of $[kH, K_i]$. To stay within the time bounds, they use multipoint evaluation techniques to refine towards all target indices simultaneously.

(The main reason that their algorithm is a logarithmic factor faster than the Chudnovskys' algorithm is that in Step 0, they give up some control over which intervals are computed, in exchange for having available a faster method for computing them. This is why the separate refining step is necessary.)

To adapt this to our needs, we need only perform a little extra work. Given the input indices K_i and L_i, we compute the sequence (3.3), using the same method as [BGS07]. We now perform a refining step using the same algorithm as in [BGS07], but we will need to refine over more intervals. Suppose that K_i lies in

$[k_1 H, (k_1+1)H]$ and that L_i lies in $[k_2 H, (k_2+1)H]$, where $k_2 \geq k_1$. If $k_1 = k_2$ then we refine over $[K_i, L_i]$. If $k_2 > k_1$, we must refine over both $[K_i, (k_1+1)H]$ and $[k_2 H, L_i]$.

After computing the products $M(k, k')$ for each of these intervals, we must perform an additional 'gluing' step. Namely, each of our target intervals (K_i, L_i) is a union of intervals (k, k') for which $M(k, k')$ has been computed (either in Step 0 or in the refining step), and so we simply multiply together the $M(k, k')$ for those intervals, in the appropriate order.

To estimate the total time, we note first that our 'Step 0' is identical to their 'Step 0'. The refining steps take at most twice as long as theirs, since we have at most doubled the number of intervals to be considered, and the lengths of those intervals satisfy the same bounds. One must also check the invertibility conditions in R; these are still satisfied since they depend only on the maximum length of the intervals, which has not changed. Finally, the extra gluing step consists of at most $O(\sqrt{K})$ matrix multiplications, costing time $O(\mathsf{MM}(m)\sqrt{K})$, which fits within the required time bound. □

Chapter 4

The new algorithm

In this chapter we present the algorithm implementing Theorem 5, which is the main result of Part I.

We begin in §4.1 by studying some cohomological reductions similar to those in Kedlaya's algorithm. The point is to give explicit formulae, so that the reductions may be reinterpreted as defining linear recurrences. In §4.2 we give an alternative expression for the action of the p-th power Frobenius on the differentials $x^i dx/y$; this is necessary to avoid expressions like (2.3), whose number of terms is at least linear in p. In §4.3, we give the main algorithm, which is essentially a straightforward combination of the ideas of §4.1, §4.2, and Chapter 3. However, to minimise the amount of p-adic precision that must be maintained throughout the execution of the algorithm (and thereby increase its efficiency), it is necessary to perform a careful analysis of error propagation. This analysis takes up the bulk of the presentation, and appears to be unavoidably technical.

4.1 Explicit reduction formulae

Let $s \geq -1$ and $t \in \mathbf{Z}$. Let $W_{s,t}$ be the $\mathbf{Q}_q$-vector space of differentials of the form

$$F(x)x^s y^{-2t} dx/y,$$

where $F(x) \in \mathbf{Q}_q[x]$ has degree at most $2g$. In the case $s = -1$, we impose the additional condition that the constant term of $F(x)$ must be zero (so that none of the differentials ever involve negative powers of x).

In §4.1.1 and §4.1.2 we will give maps between the various $W_{s,t}$ that send differentials to cohomologous differentials. First we discuss 'vertical' reductions, which map $W_{-1,t}$ to $W_{-1,t-1}$; this is the main type of reduction that appears in [Ked01]. Then we discuss 'horizontal' reductions, which map $W_{s,t}$ to $W_{s-1,t}$. The aim is to eventually reduce everything to $W_{-1,0}$, since this space consists of the differentials of the form $G(x)dx/y$, where G has degree at most $2g-1$.

We will generally identify elements of $W_{s,t}$ with vectors in $\mathbf{Q}_q^{2g+1}$ (or $\mathbf{Q}_q^{2g}$ in the case $s = -1$), with respect to the basis $\{x^{i+s}y^{-2t}dx/y\}_{i=0}^{2g}$ (or with respect to $\{x^i y^{-2t}dx/y\}_{i=0}^{2g-1}$ in the case $s = -1$).

4.1.1 Vertical reduction

Let $0 \leq i < 2g$ and $t \in \mathbf{Z}$. Since $\overline{Q}(x)$ has no repeated roots, we can find polynomials $R_i, S_i \in \mathbf{Z}_q[x]$, where $\deg R_i \leq 2g-1$ and $\deg S_i \leq 2g$, such that

$$x^i = R_i(x)Q(x) + S_i(x)Q'(x). \tag{4.1}$$

(To get the integrality of R_i and S_i, we have used the assumption that $p > 2g+1$, so that the leading coefficient of $Q'(x)$ is a unit.) Using the relation $2y\,dy = Q'(x)dx$, we have

$$x^i y^{-2t} dx/y = R_i(x) y^{-2t+2} dx/y + 2S_i(x) y^{-2t} dy.$$

Since $d(S_i(x)y^{-2t+1})$ is zero in cohomology, after a little algebra we find that

$$x^i y^{-2t} dx/y \sim \frac{(2t-1)R_i(x) + 2S_i'(x)}{2t-1} y^{-2t+2} dx/y. \tag{4.2}$$

(The above calculation is essentially the one in [Ked01, p. 329].)

This last relation may be rephrased in terms of the vector spaces $W_{-1,t}$ as follows.

Proposition 8. *Let*

$$M_V(t) : W_{-1,t} \to W_{-1,t-1}$$

be the linear map given by the $2g \times 2g$ matrix whose $(i+1)$-th column consists of the coefficients of the polynomial $(2t-1)R_i(x) + 2S_i'(x)$. Let

$$D_V(t) = 2t - 1.$$

Then for any $\omega \in W_{-1,t}$, we have

$$\omega \sim D_V(t)^{-1} M_V(t)\, \omega \quad (\in W_{-1,t-1}).$$

In other words, $D_V(t)^{-1}M_V(t)$ is the *reduction matrix* for transporting a differential from $W_{-1,t}$ to a cohomologous differential in $W_{-1,t-1}$. Note that the entries of $M_V(t)$ are linear polynomials in $\mathbf{Z}_q[t]$, as is $D_V(t)$.

We will be interested in iterating this process. For $t_0 < t_1$, let

$$M_V(t_0, t_1) : W_{-1,t_1} \to W_{-1,t_0}$$

be defined by

$$M_V(t_0, t_1) = M_V(t_0 + 1)M_V(t_0 + 2) \cdots M_V(t_1).$$

Similarly let

$$D_V(t_0, t_1) = D_V(t_0 + 1)D_V(t_0 + 2) \cdots D_V(t_1).$$

(Note that we have reversed the direction of the matrix products from the notation introduced in Chapter 3; it is trivial to adapt the algorithms of Chapter 3 to operate in the reversed direction.) With this notation we obtain:

Proposition 9. *For any* $\omega \in W_{-1,t_1}$,

$$\omega \sim D_V(t_0, t_1)^{-1} M_V(t_0, t_1)\, \omega \quad (\in W_{-1,t_0}).$$

Example 10 (An elliptic curve)**.** We compute $M_V(t)$ for the elliptic curve $y^2 = Q(x) = x^3 + ax + b$. First we solve (4.1) for $i = 0, 1$, obtaining

$$\begin{aligned} R_0(x) &= \Delta^{-1}(-18ax + 27b) \\ S_0(x) &= \Delta^{-1}(6ax^2 - 9bx + 4a^2) \\ R_1(x) &= \Delta^{-1}(27bx + 6a^2) \\ S_1(x) &= \Delta^{-1}(-9bx^2 - 2a^2x - 6ab), \end{aligned}$$

where $\Delta = 27b^2 + 4a^3$ is the discriminant of the equation of the curve. Therefore

$$(2t-1)R_0(x) + 2S_0'(x) = \Delta^{-1}(-6ax(6t-7) + 9b(6t-5))$$
$$(2t-1)R_1(x) + 2S_1'(x) = \Delta^{-1}(9bx(6t-7) + 2a^2(6t-5)),$$

and so the matrix $M_V(t)$ is given by

$$M_V(t) = \Delta^{-1} \begin{pmatrix} 9b(6t-5) & 2a^2(6t-5) \\ -6a(6t-7) & 9b(6t-7) \end{pmatrix}.$$

4.1.2 Horizontal reduction

Let $s \geq 0$ and $t \in \mathbf{Z}$. In cohomology,

$$\begin{aligned} 0 &\sim d(x^s y^{-2t+1}) \\ &= sx^{s-1}y^{-2t+1}dx - (2t-1)x^s y^{-2t}dy \\ &= \left(sx^{s-1}Q(x) - \frac{1}{2}(2t-1)x^s Q'(x) \right) y^{-2t}dx/y. \end{aligned}$$

Decompose $Q(x)$ as

$$Q(x) = x^{2g+1} + P(x),$$

where $P \in \mathbf{Z}_q[x]$ has degree at most $2g$. After substituting this into the previous equation and rearranging, we obtain

$$x^{s+2g}y^{-2t}dx/y \sim \frac{2sP(x) - (2t-1)xP'(x)}{(2g+1)(2t-1) - 2s} x^{s-1}y^{-2t}dx/y. \tag{4.3}$$

Proposition 11. *Let*

$$M_H^t(s) : W_{s,t} \to W_{s-1,t}$$

be the linear map given by the matrix

$$M_H^t(s) = \begin{pmatrix} 0 & 0 & \cdots & 0 & C_0(s) \\ D_H^t(s) & 0 & & 0 & C_1(s) \\ 0 & D_H^t(s) & & 0 & C_2(s) \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & \cdots & D_H^t(s) & C_{2g}(s) \end{pmatrix},$$

where

$$D_H^t(s) = (2g+1)(2t-1) - 2s,$$

and where $C_h(s)$ is the coefficient of x^h in the polynomial

$$C(x,s) = 2sP(x) - (2t-1)xP'(x).$$

Then for any $\omega \in W_{s,t}$, we have

$$\omega \sim D_H^t(s)^{-1} M_H^t(s)\,\omega \quad (\in W_{s-1,t}).$$

Proof. The bulk of the statement follows from (4.3). In addition, the constant term of $C(x,0)$ is zero, so $M_H^t(0)$ does indeed map into $W_{-1,t}$. □

Note that, for a fixed choice of t, the entries of $M_H^t(s)$ and $D_H^t(s)$ are linear polynomials in $\mathbf{Z}_q[s]$, and $D_H^t(s)$ does not vanish for any s, since it is always odd.

To iterate this process, we define, for $-1 \le s_0 < s_1$,

$$M_H^t(s_0, s_1) : W_{s_1,t} \to W_{s_0,t}$$

by

$$M_H^t(s_0, s_1) = M_H^t(s_0 + 1)M_H^t(s_0 + 2) \cdots M_H^t(s_1),$$

and

$$D_H^t(s_0, s_1) = D_H^t(s_0 + 1)D_H^t(s_0 + 2) \cdots D_H^t(s_1).$$

We obtain:

Proposition 12. *For any* $\omega \in W_{s_1,t}$,

$$\omega \sim D_H^t(s_0, s_1)^{-1} M_H^t(s_0, s_1)\, \omega \quad (\in W_{s_0,t}).$$

Example 13 (An elliptic curve)**.** We compute $D_H^t(s)$ and $M_H^t(s)$ for the elliptic curve $y^2 = Q(x) = x^3 + ax + b$. We have

$$D_H^t(s) = 6t - 2s - 3,$$

and $P(x) = ax + b$, so

$$2sP(x) - (2t - 1)xP'(x) = ax(2s - 2t + 1) + bs.$$

Then $M_H^t(s)$ is given by

$$M_H^t(s) = \begin{pmatrix} 0 & 0 & 2bs \\ 6t - 2s - 3 & 0 & a(2s - 2t + 1) \\ 0 & 6t - 2s - 3 & 0 \end{pmatrix}.$$

4.2 An alternative expression for the Frobenius action

As noted in Chapter 2, one of the barriers to making Kedlaya's algorithm run in time less than linear in p is that the series approximation for $\sigma(x^i dx/y)$ given by (2.5) has about Np terms. The following proposition gives a different approximation for $\sigma(x^i dx/y)$ that requires only $O(N^2 g)$ terms; in particular, the number of terms does not depend on p.

Proposition 14. *Let $C_{j,r} \in \mathbf{Z}_q$ be the coefficient of x^r in $Q(x)^j$. For $0 \leq j < N$, let*

$$B_{j,r} = pC^{\sigma}_{j,r} \sum_{k=j}^{N-1} (-1)^{k+j} \binom{-1/2}{k} \binom{k}{j} \quad \in \mathbf{Z}_q.$$

For $0 \leq i < 2g$, set

$$T_i = \sum_{j=0}^{N-1} \sum_{r=0}^{(2g+1)j} B_{j,r} x^{p(i+r+1)-1} y^{-p(2j+1)+1} dx/y. \tag{4.4}$$

Then the reduction of T_i agrees modulo p^N with the reduction of $\sigma(x^i dx/y)$.

Proof. From (2.1) and (2.4) we obtain

$$\sigma(x^i dx/y) = \sum_{k=0}^{\infty} p \binom{-1/2}{k} (Q^{\sigma}(x^p) - Q(x)^p)^k x^{pi+p-1} y^{-p(2k+1)+1} dx/y. \tag{4.5}$$

Since $Q^{\sigma}(x^p) - Q(x)^p$ is divisible by p, the k-th term U_k of (4.5) is of the form

$$U_k = p^{k+1} F(x) y^{-p(2k+1)} dx,$$

where $F \in \mathbf{Z}_q[x]$ has degree at most

$$((2g+1)p-1)k + pi + p - 1 < (2g+1)(k+1)p.$$

By repeatedly dividing $F(x)$ by $Q(x) = y^2$, we may rewrite this as

$$U_k = p^{k+1} \sum_{j=0}^{(k+1)p-1} F_j(x) y^{-p(2k+1)+2j} dx,$$

where each $F_j \in \mathbf{Z}_q[x]$ has degree at most $2g$.

We must show that the coefficients of the reduction of U_k are divisible by p^N, for all $k \geq N$. The terms for which $0 \leq j < (k+\frac{1}{2})p$ may be handled by [Ked01, Lemma 2], which shows that the reduction of $F_j(x)y^{-p(2k+1)+2j}$ becomes integral on multiplication by $p^{1+\lfloor \log_p(2k+1) \rfloor}$. Assumption (2.2) implies that $\lfloor \log_p(2k+1) \rfloor \leq k - N$, which covers this case. The remaining terms for which $(k+\frac{1}{2})p \leq j \leq (k+1)p-1$ require [Ked01, Lemma 3]. (Note: Lemma 3 as stated in [Ked01] is incorrect. A corrected version is in the errata to [Ked01], and a proof is given in Lemma 4.3.5 of [Edi03].) For these j we find that the reduction of $F_j(x)y^{-p(2k+1)+2j}dx$ becomes integral on multiplication by p^m where

$$m = \lfloor \log_p((2g+1)(-p(2k+1)+2j+2)-2) \rfloor \leq \lfloor \log_p((2g+1)p) \rfloor \leq 1,$$

the last inequality again depending on (2.2).

Consequently the terms in (4.5) for $k \geq N$ do not contribute modulo p^N to

the reduction of $\sigma(x^i dx/y)$, so we may ignore them. Therefore, let

$$T_i = \sum_{k=0}^{N-1} p \binom{-1/2}{k} (Q^\sigma(x^p) - Q(x)^p)^k x^{pi+p-1} y^{-p(2k+1)+1} dx/y.$$

We now replace $Q(x)^p$ by y^{2p}, use the binomial formula to expand $(Q^\sigma(x^p) - y^{2p})^k$, and write out the coefficients $Q^\sigma(x^p)$ explicitly in terms of the $C_{j,r}$. After rearranging the summations, we obtain the representation for T_i indicated in the statement of the proposition. □

Remark. Ultimately, the linear contribution of p to the running time of Kedlaya's original algorithm arises from explicitly expanding out the $Q(x)^p$ term in a formula of the above type. In the proof of Proposition 14, we avoided this by substituting y^{2p} for $Q(x)^p$, and we will see that our algorithm will accordingly never need to compute the coefficients of $Q(x)^p$. At first glance this may seem odd, since in Kedlaya's original algorithm, the expansion of $Q(x)^p$ — more precisely, the congruence modulo p between $Q(x)^p$ and $Q^\sigma(x^p)$ — is precisely what causes the terms in y^σ with high powers of y^{-2} to have p-adically small coefficients. In our case however, one finds that the reduction of each term $B_{j,r} x^{p(i+r+1)-1} y^{-p(2j+1)+1} dx/y$ of T_i generally contributes to *all* N *digits* of the coefficients of the reduction of T_i, regardless of the value of r or j. In fact, even the *sum* of all terms for a given power of y^{-2} (that is, for a given j) contributes to all N digits. It is almost as if our algorithm ignores the decay conditions defining $A^\dagger$. Of course those decay conditions do play a role, by inducing hidden cancellations among the $B_{j,r}$.

4.3 The main algorithm

In this section we prove Theorem 5, giving the main algorithm for computing the Frobenius matrix. The basic idea is to start with the approximation T_i for $\sigma(x^i dx/y)$ given by Proposition 14, and then to use the reduction maps to push each term towards $W_{-1,0}$. Theorem 7 is used to efficiently compute the reduction maps.

Figure 4.1 illustrates the strategy in the case $g = 1$ and $N = 3$. Each vertex corresponds to a $W_{s,t}$, and the arrows correspond to horizontal and vertical reductions. The black vertices are those which are the starting point for at least one term from some T_i. (There are additional vertices and arrows used in the algorithm that for reasons of clarity are not shown on the diagram.)

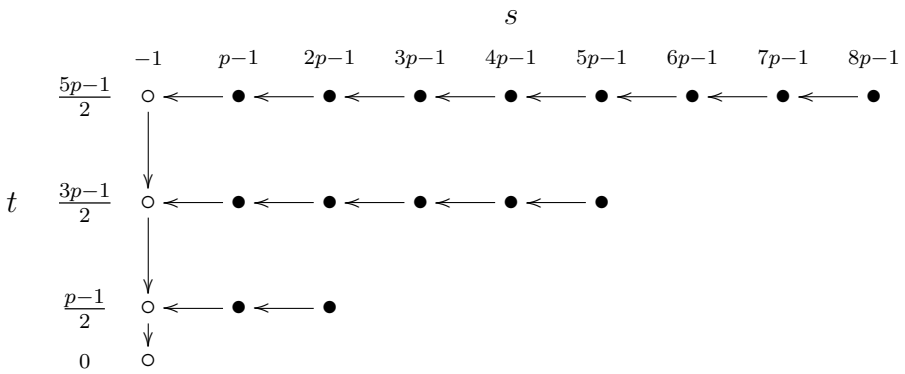

Figure 4.1: Reduction strategy for $g = 1$ and $N = 3$

One of the more magical aspects of Kedlaya's original algorithm is the way that p-adic precision losses propagate through the calculation. Although one needs to perform about N divisions by p, Kedlaya shows that in fact only $O(\log_p N)$ spare digits of precision must be carried.

A similar argument applies to our algorithm, and since we have assumed p to be sufficiently large compared to g and N, it turns out that only one spare digit

is necessary. However, some caution is required. For example, the product of all the $M_H^t(s)$ across a whole 'row' of the horizontal reductions will generally be *zero* modulo p^N, and therefore one must interleave the multiplications by $M_H^t(s)$ and divisions by $D_H^t(s)$ in such a way that the denominators can "catch up with" the build-up of p-divisibility of the numerators. In §4.3.2 we perform a more detailed analysis, showing how to do almost all of the work with *no* spare digits at all. In practical terms, avoiding even this single extra digit yields enormous savings in time and memory when N is small. For the vertical reductions, at least in the case $N > 1$, this kind of analysis seems much more difficult, and consequently we will retain the spare digit.

4.3.1 Preliminaries

The algorithm works in two different rings, $R_0 = \mathbf{Z}_q/(p^N)$ and $R_1 = \mathbf{Z}_q/(p^{N+1})$. At certain stages we will need to compute a/b, where b is not a unit; we may take the result to be any c satisfying $bc = a$. We will see below that such divisions will always be possible in $\mathbf{Z}_q$ when they occur, and that the errors introduced do not contribute to the final result modulo p^N.

As a preliminary step, we compute the coefficients $B_{j,r}$ given in Proposition 14, for $0 \le j < N$ and $0 \le r \le (2g+1)j$, as elements of R_1.

Let us write T_i as

$$T_i = \sum_{j=0}^{N-1} T_{i,j}, \qquad T_{i,j} = \sum_{k=0}^{i+(2g+1)j+1} T_{i,j,k},$$

$$T_{i,j,k} = B_{j,k-i-1} x^{pk-1} y^{-p(2j+1)+1} dx/y,$$

where for convenience we declare that $B_{j,r} = 0$ for $r < 0$. Note that $T_{i,j,k} \in W_{pk-1,t}$,

where $t = \frac{1}{2}((2j+1)p - 1)$.

4.3.2 Horizontal reduction phase

This phase is performed once for each $0 \leq j < N$; throughout this section we regard j as fixed.

Let $t = \frac{1}{2}((2j+1)p - 1)$. The aim is to use the horizontal reduction maps to find differentials $w_{i,j} \in W_{-1,t}$ that are cohomologous to $T_{i,j}$, and whose coefficients are correct modulo p^N, for $0 \leq i < 2g$.

Computing the reduction maps

Let $L = (2g+1)j + 2g$. We must first compute the horizontal reduction matrices

$$\begin{aligned} M(k) &= M_H^t((k-1)p, kp - 2g - 2), \\ D(k) &= D_H^t((k-1)p, kp - 2g - 2), \end{aligned} \tag{4.6}$$

for $1 \leq k \leq L$, with entries in R_0. (Once computed, it may be convenient to lift them to R_1, but it is only necessary to know them modulo p^N.)

This is accomplished in two steps. We will discuss $M(k)$ only; the $D(k)$ are handled entirely analogously.

The first and most time-consuming step is to use Theorem 7 to compute $M(k)$ for $1 \leq k \leq L'$, where $L' = \min(N, L)$. To verify the invertibility hypotheses of Theorem 7, we must check that $\sqrt{K} + 1 < p$, where $K = L'p - 2g - 2$ is the total length of the interval containing all the reduction intervals. From (2.2) we know

that $(2g+1)(2N-1) \leq p-1$, so

$$\begin{aligned} 2K &\leq (2g+1)(2N-1)p + (2g+1)p - 2p - 4g - 4 \\ &\leq (p-1)p + (p-1)p - 2p - 4g - 4 \\ &< 2(p-1)^2, \end{aligned}$$

from which the desired inequality follows.

The second step is to deduce the remaining $M(k)$ for $N < k \leq L$. (This is of course only necessary when $L > N$.) It is possible to simply use Theorem 7 again, but it is much more efficient to take advantage of the known values $M(1), \ldots M(N)$. If $N = 1$ this is trivial, since the $M(k)$ are all equal modulo p. The author thanks Kiran Kedlaya for suggesting the following interpolation method to handle the case $N > 1$.

Consider the matrix

$$F(s) = M_H^t(s-p+1) \cdots M_H^t(s-2g-2),$$

which is a matrix of polynomials in s. Expanding as a Taylor series in s, we obtain

$$M(k) = F(kp) = F(0) + F'(0)kp + \cdots + \frac{1}{(N-1)!} F^{(N-1)}(0)(kp)^{N-1} \pmod{p^N}.$$

Then by simple linear algebra, the values of $F(kp) \pmod{p^N}$ for $1 \leq k \leq N$

determine completely the values of $F^{(i)}(0)p^i/i!$ for $0 \leq i < N$. Namely, we have

$$\begin{pmatrix} F(p) \\ F(2p) \\ \vdots \\ F(Np) \end{pmatrix} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & 2 & & 2^{N-1} \\ \vdots & & & \vdots \\ 1 & N & \cdots & N^{N-1} \end{pmatrix} \begin{pmatrix} F(0) \\ F'(0)p \\ \vdots \\ \frac{1}{(N-1)!}F^{(N-1)}(0)p^{N-1}, \end{pmatrix}$$

and the Vandermonde matrix is invertible modulo p (since $p > N$). After solving for the $F^{(i)}(0)p^i/i!$, the remaining $M(k)$ are computed by substituting the appropriate values of k into the above Taylor series.

Remark. In the case $N = 1$ there is a yet faster method available for computing $D(k)$ (although not $M(k)$). Namely, since $t \equiv -1/2 \pmod{p}$ we have

$$D(k) \equiv D(1) \equiv \prod_{s=1}^{p-2g-2} -2(2g+1) - 2s \pmod{p},$$

which by Wilson's theorem is equal to

$$(-2)^{p-2g-2} \prod_{s=2g+2}^{p-1} s \equiv (2^{2g+1}(2g+1)!)^{-1} \pmod{p}.$$

Performing the reductions

Now we fix $0 \leq i < 2g$, and show how to compute $w_{i,j}$. We will define a sequence of differentials $v_{i+(2g+1)j+1}, \ldots, v_1, v_0$, where $v_m \in W_{mp-1,t}$, with the property that

$$v_m \sim \sum_{k=m}^{i+(2g+1)j+1} T_{i,j,k}. \tag{4.7}$$

In particular we will have $v_0 \sim T_{i,j}$, so this v_0 is the $w_{i,j}$ that we seek. The v_m are computed with entries in R_1. However, not all their p-adic digits will be correct; we will say more about this in a moment.

Naturally, the sequence begins with

$$v_{i+(2g+1)j+1} = T_{i,j,i+(2g+1)j+1}.$$

Then, given v_m, we compute v_{m-1} as follows. We first move from $W_{mp-1,t}$ to $W_{mp-2g-2,t}$, one step at a time, via the following sequence:

$$\begin{aligned}
v_m^{(1)} &= v_m && \in W_{mp-1,t},\\
v_m^{(2)} &= D_H^t(mp-1)^{-1} M_H^t(mp-1) v_m^{(1)} && \in W_{mp-2,t},\\
v_m^{(3)} &= D_H^t(mp-2)^{-1} M_H^t(mp-2) v_m^{(2)} && \in W_{mp-3,t},\\
&\vdots\\
v_m^{(2g+2)} &= D_H^t(mp-2g-1)^{-1} M^t(mp-2g-1) v_m^{(2g+1)} && \in W_{mp-2g-2,t}.
\end{aligned}$$

Using the reduction matrices (4.6) computed above, we set

$$v'_m = D_H^t((m-1)p, mp-2g-2)^{-1} M_H^t((m-1)p, mp-2g-2) v_m^{(2g+2)}, \quad (4.8)$$

and then take one final step to reach

$$v_{m-1} = T_{i,j,m-1} + D_H^t((m-1)p)^{-1} M_H^t((m-1)p) v'_m \in W_{(m-1)p-1,t}.$$

If all of the above computations are performed to infinite precision, then it follows from Propositions 11 and 12 that if v_m satisfies (4.7), then also v_{m-1} also

satisfies (4.7), and then by induction also v_0 satisfies (4.7).

Now we analyse the propagation of errors. To facilitate the analysis, we introduce the following terminology. Suppose that v is a vector of length $2g+1$, with coordinates in R_1. Let $\varepsilon(v)$ denote the error term associated to v. That is, $\varepsilon(v)$ is the difference between the value for v stored by the machine and the value that would have been obtained for v if all computations had been performed to infinite precision. We will say that v is *ℓ-correct* if:

- the ℓ-th coordinate of v is divisible by p;
- the ℓ-th coordinate of $\varepsilon(v)$ is divisible by p^{N+1}; and
- the remaining coordinates of $\varepsilon(v)$ are divisible by p^N.

Note that $v_{i+(2g+1)j+1}$ is 1-correct, since its first coordinate is simply $B_{j,(2g+1)j}$, which has been computed in R_1 and is divisible by p, and the other coordinates are all zero. The following series of claims together show that if v_m is 1-correct, then also v_{m-1} is 1-correct. Consequently v_0 is 1-correct, and in particular $w_{i,j}$ is computed correctly to precision p^N.

Claim 1. *Let $1 \leq \ell \leq 2g$. If $v_m^{(\ell)}$ is ℓ-correct, then $v_m^{(\ell+1)}$ is $(\ell+1)$-correct.*

Proof. We first examine the form of the matrix $M_H^t(mp-\ell)$. Let $P(x)$, $C(s,x)$ and $C_h(s)$ be the polynomials introduced in Proposition 11. We are taking $s = mp-\ell \equiv -\ell \pmod{p}$ and $t \equiv -1/2 \pmod{p}$, so

$$C(x,s) \equiv -2\ell P(x) + 2xP'(x) \pmod{p}.$$

In particular the coefficient $C_\ell(s)$ of x^ℓ is zero modulo p, so the entry in the $(\ell+1)$-th row of the last column of $M_H^t(mp-\ell)$ is zero modulo p. Consequently the

contribution to $v_m^{(\ell+1)}$ from the last entry of $v_m^{(\ell)}$ satisfies the required conditions.

Furthermore, it is clear from Proposition 11 that the only other possibly nonzero entry in the $(\ell+1)$-th row appears in the ℓ-th column. Therefore $v_m^{(\ell+1)}$ also receives a contribution from the ℓ-th entry of $v_m^{(\ell)}$, which by hypothesis already satisfies the required conditions.

Finally, the denominator

$$D_H^t(mp-\ell) = (2g+1)(2t-1) - 2s \equiv -2((2g+1)-\ell) \pmod{p}$$

is a unit, so dividing by it does not disturb ℓ-correctness. □

Claim 2. *If $v_m^{(2g+1)}$ is $(2g+1)$-correct, then $v_m^{(2g+2)}$ is correct modulo p^N.*

Proof. Let $w = M_H^t(mp-2g-1)v_m^{(2g+2)}$. We have

$$D_H^t(mp-2g-1) = (2g+1)(2t-1) - 2(mp-2g-1) \equiv 0 \pmod{p}, \quad (4.9)$$

so by the definition of M_H^t, the first $2g$ columns of $M_H^t(mp-2g-1)$ are zero modulo p. Since the first $2g$ coordinates of $v_m^{(2g+1)}$ are correct modulo p^N, the contribution they make to w is divisible by p and correct modulo p^{N+1}. The contribution from the last coordinate of $v_m^{(2g+1)}$ is by hypothesis already divisible by p and correct modulo p^{N+1}.

We deduce that w is divisible by p and correct modulo p^{N+1}. It suffices now to show that the valuation of $D_H^t(mp-2g-1)$ is precisely 1. Since we know it is odd and divisible by p, we must bound its absolute value below p^2. From (4.9)

and the definition of t we have

$$D_H^t(mp - 2g - 1) = ((2g + 1)(2j + 1) - 2m)p,$$

and then the desired result follows from (2.2), since $0 \le m \le (2g+1)(j+1)-1$. □

Claim 3. *If $v_m^{(2g+2)}$ is correct modulo p^N, then so is v'_m.*

Proof. By (4.8) it suffices to show that $D_H^t((m-1)p, mp-2g-2)$ is a unit. The latter quantity is

$$\prod_{s=(k-1)p+1}^{kp-2g-2} (2g+1)(2t-1) - 2s \equiv \prod_{s=1}^{p-2g-2} -2((2g+1)+s) \pmod{p}$$

since $t \equiv -1/2 \pmod{p}$, so it is a unit. □

Remark. In the above proof, we only needed the values of $M_H^t((m-1)p, mp-2g-2)$ and $D_H^t((m-1)p, mp-2g-2)$ modulo p^N, not modulo p^{N+1}. This is why it is possible to do almost all of the work in the horizontal reductions using only N digits.

Claim 4. *If v'_m is correct modulo p^N, then v_{m-1} is 1-correct.*

Proof. The same argument used in the proof of Claim 1 shows that the first row of $M_H^t((m-1)p)$ is entirely zero modulo p, and that $D_H^t((m-1)p)$ is a unit. Therefore the contribution to v_{m-1} from v'_m is 1-correct. The contribution from $T_{i,j,m-1}$ is also 1-correct. □

4.3.3 Vertical reduction phase

We first prove some lemmas that will be used to analyse the error propagation in the vertical reduction phase.

Lemma 15. *If $t \equiv 1/2 \pmod{p}$, then $M_V(t)$ is invertible modulo p.*

Proof. Under the hypothesis on t, it follows from the definition of $M_V(t)$ that the entries of its $(i+1)$-th column are given by the coefficients of $S_i'(x)$. To show that $M_V(t)$ is invertible modulo p, it suffices to show that the $\overline{S}_i'(x)$ are linearly independent over $\mathbf{F}_q$. If $\sum_{i=0}^{2g-1} \lambda_i \overline{S}_i'(x) = 0$ is some linear relation, then we may integrate (permissible, by (2.2)) to obtain $\sum_{i=0}^{2g-1} \lambda_i \overline{S}_i(x) = \lambda_{2g}$ for some $\lambda_{2g} \in \mathbf{F}_q$. Multiplying this by $\overline{Q}'(x)$, and using (4.1), we obtain

$$\sum_{i=0}^{2g-1} \lambda_i x^i \equiv \lambda_{2g} \overline{Q}'(x) \pmod{\overline{Q}(x)}.$$

But $1, x, \ldots, x^{2g-1}, \overline{Q}'(x)$ are linearly independent in $\mathbf{F}_q[x]/\overline{Q}(x)$, since $Q'(x)$ has degree $2g$ and unit leading term (again due to (2.2)). This forces every $\lambda_i = 0$. □

Remark. It would be interesting to characterise the values of t for which $M_V(t)$ is singular modulo p. For instance, in the case of an elliptic curve, Example 10 shows that $M_V(t)$ is singular precisely when $t \equiv 7/6 \pmod{p}$ or $t \equiv 5/6 \pmod{p}$. By studying the kernels and images of such maps, it may be possible to reduce the working precision in the vertical reduction steps from p^{N+1} to p^N, as was done for the horizontal reductions.

Lemma 16. *If $t_0 \equiv -1/2 \pmod{p}$, then $M_V(t_0, t_0 + p)$ is zero modulo p.*

Proof. Since $M_V(t_0, t_0+p)$ modulo p only depends on t_0 modulo p, we may assume that $t_0 = (p-1)/2$.

Let

$$X = D_V(t_0, t_0 + p + 1)^{-1} M_V(t_0, t_0 + p + 1)$$

be the reduction map from W_{-1,t_0+p+1} to W_{-1,t_0}. First we will show that pX is integral. It is easy to check that p^2X is integral, by inspecting the powers of p dividing $D_V(t_0, t_0 + p + 1)$, but the integrality of pX requires more work. The proof is very similar to the proof of [Ked01, Lemma 2]. Let $\omega \in W_{-1,t_0+p+1}$, say

$$\omega = F(x)y^{-2(t_0+p+1)}dx/y,$$

where $F \in \mathbf{Z}_q[x]$ has degree at most $2g - 1$. Let $\eta = X\omega$, and write

$$\eta = G(x)y^{-2t_0}dx/y$$

where $G \in \mathbf{Q}_q[x]$ has degree at most $2g - 1$. We need to show that $p\eta$ is integral.

Since X is a reduction map, ω and η are cohomologous, and the discussion preceding Proposition 8 shows that $\omega - \eta = dH$ where

$$H = \sum_{t=t_0+1}^{t_0+p+1} H_t(x)y^{-2t+1}$$

for some polynomials $H_t \in \mathbf{Q}_q[x]$ of degree at most $2g$. We may now use the same argument as in the proof of [Ked01, Lemma 2] (namely, comparing the y-expansions of ω, η and dH around each root of $Q(x)$) to deduce that $m\eta$ is integral, provided that $m/(2t-1)$ is integral for $t_0 \leq t \leq t_0 + p + 1$. In particular $p\eta$ is integral, since we have assumed that $t_0 = (p-1)/2$.

Now we may finish the proof of the lemma. We have

$$X = D_V(t_0, t_0 + p + 1)^{-1} M_V(t_0, t_0 + p) M_V(t_0 + p + 1).$$

By Lemma 15 we know that $M_V(t_0 + p + 1)$ is invertible modulo p, so its inverse is integral. Rearranging, we obtain

$$M_V(t_0, t_0 + p) = D_V(t_0, t_0 + p + 1) X M_V(t_0 + p + 1)^{-1}.$$

Note that $D_V(t_0, t_0 + p + 1) = \prod_{t=t_0+1}^{t_0+p+1}(2t - 1)$ is divisible by p^2, since the first and last factors in the product are zero modulo p. The integrality of pX then implies that $M_V(t_0, t_0 + p)$ is zero modulo p. □

Now we may describe the vertical reduction phase. The input consists of the differentials $w_{i,j}$ computed via the horizontal reductions. The output will be a collection of differentials $w_i \in W_{-1,0}$ for $0 \leq i < 2g$ that are cohomologous to T_i, and correct modulo p^N.

The first step is to compute the vertical reduction matrices

$$M_j = \begin{cases} M_V\left(0, \frac{p-1}{2}\right) & j = 0, \\ M_V\left(\frac{(2j-1)p-1}{2}, \frac{(2j+1)p-1}{2}\right) & 1 \leq j < N, \end{cases}$$

and similarly for D_j, using Theorem 7, with entries in R_1. The invertibility hypotheses of Theorem 7 are satisfied, because the total reduction length K satisfies

$$K = \frac{(2(N-1)+1)p - 1}{2} < \frac{(2N-1)p}{2}.$$

The latter is bounded by $p^2/6$ from (2.2), so certainly $\sqrt{K}+1 < p$.

For $j \geq 1$, observe that D_j has valuation precisely 1, because in the product

$$D_j = \prod_{t=\frac{1}{2}((2j-1)p+1)}^{\frac{1}{2}((2j+1)p-1)} (2t-1),$$

the only term divisible by p is the first one, and (2.2) implies that it is less than p^2. Furthermore, M_j is zero modulo p by Lemma 16. Since M_j and D_j have been computed modulo p^{N+1}, we can therefore compute the (integral) matrix

$$X_j = D_j^{-1} M_j$$

correctly modulo p^N. For the $j = 0$ case, the product for D_0 shows that it is a unit, so $X_0 = D_0^{-1} M_0$ may be computed modulo p^N as well. Note that X_j is the vertical reduction map from $W_{-1,\frac{1}{2}((2j+1)p-1)}$ to $W_{-1,\frac{1}{2}((2j-1)p-1)}$ for $j \geq 1$, and to $W_{-1,0}$ for $j = 0$.

Now we fix $0 \leq i < 2g$, and show how to compute w_i. We define a sequence of differentials

$$\begin{aligned}
v_{N-1} &= w_{i,N-1} && \in W_{-1,\frac{1}{2}((2N-1)p-1)},\\
v_{N-2} &= w_{i,N-2} + X_{N-1} v_{N-1} && \in W_{-1,\frac{1}{2}((2N-3)p-1)},\\
&\vdots\\
v_0 &= w_{i,0} + X_1 v_1 && \in W_{-1,\frac{1}{2}(p-1)}.
\end{aligned}$$

Using Proposition 9, one checks by induction that

$$v_m \sim \sum_{j \geq m} T_{i,j}$$

for each $1 \leq m \leq N-1$, and that the coefficients of v_m are correct modulo p^N. Finally one puts $w_i = X_0 v_0 \in W_{-1,0}$, which by Proposition 9 is cohomologous to T_i, and again its coefficients are correct modulo p^N.

Remark. In the case $N = 1$, it is only necessary to compute M_0 modulo p, rather than modulo p^2 as described above, since no divisions by p are involved at all. It is not clear to the author whether a similar optimisation is available when $N > 1$.

4.3.4 Complexity analysis

We first consider the time spent in the applications of Theorem 7, which will be the dominant step when p is large compared to N and g. For both $R_0 = \mathbf{Z}_{p^n}/(p^N)$ and $R_1 = \mathbf{Z}_{p^n}/(p^{N+1})$, basic ring operations (addition, multiplication) have bit-complexity $\widetilde{O}(Nn \log p)$. For the horizontal reductions, for each of N rows, we applied Theorem 7 with $K = O(pN)$ and $m = O(g)$. Therefore each row costs $\widetilde{O}(p^{1/2} N^{3/2} g^\omega n)$. For the vertical reductions, we applied Theorem 7 once, also with $K = O(pN)$ and $m = O(g)$. Therefore the total time is $\widetilde{O}(p^{1/2} N^{5/2} g^\omega n)$.

Now we estimate the time for the remaining steps, which for sufficiently large p will be negligible.

Computing the coefficients $C_{j,r}$ in Proposition 14 requires only $O(N^2 g^2)$ ring operations, even if naive polynomial multiplication is used. In the formulae for the $B_{j,r}$, computing all the necessary binomial coefficients requires $O(N^2)$ ring operations, and then computing all the $B_{j,r}$ requires $O(N^2 g)$ ring operations.

Therefore computing the $B_{j,r}$ requires $O(N^2g^2)$ ring operations altogether.

Solving (4.1) for each i requires $O(g^2)$ ring operations, even by the naive Euclidean extended GCD algorithm, so computing the coefficients of $M_V(t)$ needs at most $O(g^3)$ ring operations. Computing the coefficients of $M_H^t(s)$ for each of the N required values of t requires $O(Ng)$ ring operations.

In the horizontal reduction phase, computing the inverse of the Vandermonde matrix requires $O(N^3)$ ring operations. Then for each of N rows we must perform the following steps. First, compute the values of $F^{(i)}(0)p^i/i!$, costing $O(N^2g^2)$ ring operations. Then use these values to compute $M(k) = F(kp)$ for $O(Ng)$ values of k; for each k this costs $O(Ng^2)$ ring operations, so over all k this costs $O(N^2g^3)$. The total cost over all rows is $O(N^3g^3)$ ring operations.

Finally we must account for the 'single step' reductions during the horizontal reduction phase, as these were performed without the assistance of Theorem 7. Each matrix-vector multiplication requires only $O(g)$ ring operations, due to the sparsity of the matrices. For each of N rows, for each of $O(Ng)$ values of m, and for each of $O(g)$ values of i, there are $O(g)$ such steps, for a total cost of $O(N^2g^4)$ ring operations.

In total the cost is $O(N^3g^4)$ ring operations, with bit-complexity $\widetilde{O}(N^4g^4n\log p)$.

Chapter 5

Example computations

The author implemented the main algorithm in C++, only for the case $n = 1$ (i.e. for curves over $\mathbf{F}_p$). The program, called `hypellfrob`, is freely available from the author's web page, `http://math.harvard.edu/~dmharvey/`, under a GPL license. The functionality is also included in recent versions of the Sage computer algebra system [SJ05], via the `sage.schemes.hyperelliptic_curves.hypellfrob` module.

The underlying polynomial arithmetic uses one of two libraries: either Victor Shoup's NTL library [Sho], or the new `zn_poly` library of the author. The latter is available from the above website, also under a GPL license. The NTL-based code is used whenever the working modulus (p^N or p^{N+1}, depending on which phase of the algorithm is being run) exceeds a single machine word. For smaller moduli, the `zn_poly` code is used.

The matrix multiplication steps use the naive $O(n^3)$ algorithm. The key polynomial shifting steps are performed via the the middle product algorithm [HQZ04] as suggested in [BGS07, p. 1786]. For the NTL-based code, the middle product is

built on top of NTL's low-level number-theoretic transforms, a trivial task thanks to Shoup's wonderfully modular FFT code. The `zn_poly` library has native support for middle products.

The following sample computations were performed on a 1.8 GHz 64-bit AMD Opteron machine with 64 GB RAM, kindly supplied by William Stein. The machine has 16 cores, but only a single core was used. The compiler used was gcc 4.1.2 with optimisation flag `-O3`. Both NTL and `zn_poly` were linked with the GMP library (version 4.2.1, with Pierrick Gaudry's AMD assembly patch) for the underlying integer arithmetic.

5.1 Examples showing dependence on p

Tables 5.1 and 5.2 show the time used to compute the Frobenius matrix over a range of p for a random genus three curve and a random genus six curve, with precision $N = 1$ and $N = 3$ respectively. From Theorem 5, one expects the running time to approximately double for every four-fold increase in p. This is borne out fairly well by the data, except for the jumps in Table 5.2. The jump at $p \approx 2^{16}$ occurs when the increased size of the modulus $p^{N+1} = p^4$ (for the vertical reduction phase) forces the implementation to switch from a single-precision (64 bit) integer representation to a more expensive arbitrary-precision representation. Similarly, the jump at $p \approx 2^{22}$ occurs when the modulus $p^N = p^3$ (for the horizontal reduction phase) reaches the single-precision threshold.

p	time	p	time	p	time
$2^{20}-3$	0.09s	$2^{27}-39$	2.22s	$2^{34}-41$	49.7s
$2^{21}-9$	0.14s	$2^{28}-57$	3.37s	$2^{35}-31$	73.9s
$2^{22}-3$	0.24s	$2^{29}-3$	5.86s	$2^{36}-5$	112s
$2^{23}-15$	0.39s	$2^{30}-35$	8.66s	$2^{37}-25$	161s
$2^{24}-3$	0.62s	$2^{31}-1$	14.4s	$2^{38}-45$	240s
$2^{25}-39$	0.98s	$2^{32}-5$	22.0s	$2^{39}-7$	359s
$2^{26}-5$	1.43s	$2^{33}-9$	32.8s	$2^{40}-87$	515s

Table 5.1: Running times for $g = 3$ and $N = 1$

p	time	p	time	p	time
$2^{14}-3$	0.54s	$2^{18}-5$	8.34s	$2^{22}-3$	143s
$2^{15}-19$	0.75s	$2^{19}-1$	10.1s	$2^{23}-15$	176s
$2^{16}-15$	3.28s	$2^{20}-3$	15.8s	$2^{24}-3$	295s
$2^{17}-1$	5.10s	$2^{21}-9$	22.1s	$2^{25}-39$	382s

Table 5.2: Running times for $g = 6$ and $N = 3$

5.2 Near-cryptographic sizes

For the purpose of constructing secure cryptosystems, it is useful to be able to determine the zeta function of a hyperelliptic curve C of low genus over a large prime field [CFA+06, Ch. 23]. The security of the cryptosystem depends on the difficulty of computing discrete logarithms in the Jacobian of the curve. In particular one hopes to find a curve whose Jacobian order $\#J(C/\mathbf{F}_p) = P_C(1) \approx p^g$ is prime (or is a prime multiplied by a very small integer) and sufficiently large.

For genus three and four, we ran our implementation on a single curve over the largest prime field that seemed feasible with the given hardware. We were able to determine the zeta function for a curve whose Jacobian approaches a cryptographically useful size, although there is still a gap to overcome. Handling a genus *two* curve with a large enough Jacobian is clearly out of reach of this

technique.

Thanks to Kiran Kedlaya for his assistance in using the Magma computer algebra system [BCP97] to perform some of the computations below.

5.2.1 Genus three

We computed the characteristic polynomial of Frobenius modulo p for the curve

$$y^2 = x^7 + 2x^6 + 3x^5 + 4x^4 + 5x^3 + 6x^2 + 7x + 8$$

defined over $\mathbf{F}_p$ where

$$p = 2^{53} - 111 = 9007199254740881.$$

The running time was 21.5 hours, and peak memory usage was 51 GB.

This determines $\#J(C/\mathbf{F}_p)$ modulo p, within an interval of width $O(p^{3/2})$. The search space is only $O(p^{1/2})$, so Magma's baby-step/giant-step implementation is easily (in a few seconds) able to recover the Jacobian order. From this we inferred that the characteristic polynomial of Frobenius is

$$(X^6 + p^3) + a_1(X^5 + p^2X) + a_2(X^4 + pX^2) + a_3X^3,$$

where

$$a_1 = -98254756,$$

$$a_2 = 7863373269694211,$$

$$a_3 = -25865813220240800386 3832.$$

The order of the Jacobian over $\mathbf{F}_p$ is

$$730750810694051686964010072594226615405670581640 \approx 2^{159}.$$

Unfortunately, this is far from prime; it factors as

$$2^3 \cdot 5 \cdot 23 \cdot 113 \cdot 197 \cdot 709 \cdot 1544163711313381 \cdot 32590965878831406770743.$$

The usual benchmark for cryptographic applications is that the Jacobian should have cardinality at least 2^{160}, so in this sense our implementation can already count points on a curve of 'cryptographic size'. However, for genus three hyperelliptic curves, there exist *index calculus attacks* that reduce the difficulty of the discrete logarithm problem [CFA+06, Ch. 21]. To achieve security equivalent to 160 bits, one must produce a curve whose Jacobian order is about 2^{180} [CFA+06, p. 554], implying that p should be increased to about 2^{60}. Since the complexity of our algorithm is essentially $O(p^{1/2})$, we would expect such a counting problem to take about $2^{3.5} \approx 11.3$ times longer than the above example, or about ten days. A more serious obstacle is that the memory requirements would balloon by the same factor, to about 600 GB, rendering the computation completely infeasible on the same hardware. It might be possible to perform such a computation on a contemporary distributed-memory system, but parallelising the algorithm effectively seems quite challenging.

Sutherland very recently gave an ingenious and extremely practical method to find hyperelliptic curves of genus three with Jacobians of prime order, of cryptographic size [Sut08]. However, the curves found by his algorithm are not totally generic: by construction their quadratic twists are guaranteed to have Jacobians

of *highly composite* order. As Sutherland points out [Sut08, p. 3], there are no known attacks that take advantage of this extra structure on the quadratic twist; nevertheless, it would be more satisfying to have available curves without this extra structure for cryptographic use.

5.2.2 Genus four

We computed the characteristic polynomial of Frobenius modulo p^2 for the curve

$$y^2 = x^9 - 23x^8 + 19x^7 - 17x^6 + 13x^5 - 11x^4 + 7x^3 - 5x^2 + 3x - 2$$

defined over $\mathbf{F}_p$ where

$$p = 2^{44} + 7 = 17592186044423.$$

The running time was 45 hours, and peak memory usage was 34 GB.

This does not pin down the zeta function precisely, but it produces a short list of four candidates, which we checked in Magma by testing which proposed Jacobian order m satisfied $mP = 0$ for a number of random points P defined over $\mathbf{F}_p$. We found that the characteristic polynomial of Frobenius is

$$(X^8 + p^4) + a_1(X^7 + p^3X) + a_2(X^6 + p^2X^2) + a_3(X^5 + pX^3) + a_4X^4,$$

where

$$a_1 = 2394254,$$

$$a_2 = 29576915959850,$$

$$a_3 = 88182558522652238508,$$

$$a_4 = 53617874894354547797127991 6.$$

The order of the Jacobian over $\mathbf{F}_p$ is

$$957809843398383438558093102816012304646098000422927 22 \approx 2^{176}.$$

Again, this is highly composite; it factors as

$$2 \cdot 73 \cdot 83 \cdot 1583 \cdot 22145564293481 \cdot 12741505694634797 \cdot 17695381461552209.$$

Part II

Computing p-adic heights on elliptic curves

Chapter 6

The canonical p-adic height

6.1 Notation and definition of the p-adic height

Throughout Part II we fix the following notation. The elliptic curve $E/\mathbf{Q}$ of interest is given by a Weierstrass equation

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6,$$

where the coefficients a_i are integers. We assume that $p \geq 5$, that the curve has good ordinary reduction at p, and that the above equation is minimal at p.

Given any nonzero point $P \in E(\mathbf{Q})$, we may write its affine coordinates uniquely in the form

$$P = (x(P), y(P)) = \left(\frac{\alpha(P)}{d(P)^2}, \frac{\beta(P)}{d(P)^3}\right),$$

where $(\alpha(P), d(P)) = (\beta(P), d(P)) = 1$, and $d(P) \geq 1$. We denote by E_0 the subgroup of $E(\mathbf{Q})$ of points that reduce to zero in $E(\mathbf{F}_p)$, or equivalently, points

for which $p \mid d(P)$ (together with the zero element).

Let $\omega = dx/(2y + a_1x + a_3)$ be the usual invariant differential. The canonical p-adic sigma function associated to (E, ω), defined in [MT91], is a function on the formal group of E. It is given by a power series

$$\sigma_p(t) = t + c_2t^2 + c_3t^3 + \cdots \in \mathbf{Z}_p[\![t]\!] \tag{6.1}$$

where t is the parameter $t = -x/y$. In particular it is defined on all of E_0, regarding $E(\mathbf{Q})$ as a subgroup of $E(\mathbf{Q}_p)$ in the natural way; the series converges there since for such points $v_p(t) = v_p(-d\alpha/\beta) \geq 1$.

Now we explain the definition of the p-adic height function $h_p : E(\mathbf{Q}) \to \mathbf{Q}_p$, following [MST06]. Let $P \in E_0$ be a non-torsion point. If P reduces to a nonsingular point of $E(\mathbf{F}_\ell)$ for all primes ℓ, then $h_p(P)$ is given by the formula

$$h_p(P) = 2\log_p\left(\frac{\sigma_p(P)}{d(P)}\right), \tag{6.2}$$

where $\log_p$ is the Iwasawa p-adic logarithm. This determines h_p on a finite-index subgroup of $E(\mathbf{Q})$. (Note that, following a suggestion of Christian Wuthrich, we normalise the p-adic height differently from that of [MST06]. Our height is equal to the [MST06] height multiplied by $2p$.)

Finally, to define $h_p(Q)$ for an arbitrary nonzero $Q \in E(\mathbf{Q})$, we use the fact that h_p should be a quadratic form: we select an integer k such that $P = kQ$ lies in the finite-index subgroup described above, and then

$$h_p(Q) = \frac{1}{k^2}h_p(P). \tag{6.3}$$

6.2 The Mazur–Stein–Tate algorithm

In this section we sketch the Mazur–Stein–Tate algorithm for computing the p-adic height, and indicate some of the bottlenecks in the time complexity that we will address in this thesis.

The key idea of the Mazur–Stein–Tate algorithm is that there is a characterisation of the p-adic sigma function $\sigma_p(t)$ in terms of the p-adic modular form $\mathbf{E}_2$, and that $\mathbf{E}_2(E,\omega)$ may be computed efficiently using Kedlaya's algorithm. Once $\sigma_p(t)$ is known to some precision, the p-adic height of a point in $E(\mathbf{Q})$ may be computed via (6.2). We now consider these steps in more detail.

6.2.1 The computation of $\mathbf{E}_2(E,\omega)$

Katz observed that $\mathbf{E}_2(E,\omega)$ may be interpreted as the 'direction' of the unit root eigenspace of Frobenius acting on a particular basis for the first p-adic de Rham cohomology of E, leading to the following algorithm to compute $\mathbf{E}_2(E,\omega)$ (see [MST06, p. 590] and [MST06, Algorithm 3.2] for further discussion).

Suppose that we wish to compute $\mathbf{E}_2(E,\omega)$ modulo p^N, where $N \geq 1$. First consider the case where the equation of E is of the form

$$Y^2 = X^3 + A_4X + A_6, \qquad A_4, A_6 \in \mathbf{Z},$$

and is minimal at p. Interpreting $Q(X) = X^3 + A_4X + A_6$ as a polynomial in $\mathbf{Z}_p[X]$, we apply the first stage of Kedlaya's algorithm [Ked01] (see also Chapter 2) to compute the action of the p-th power Frobenius on the basis $\{dX/Y, X\,dX/Y\}$ for the first p-adic de Rham cohomology of E, to precision p^N. This produces

a 2×2 matrix F with entries in $\mathbf{Z}/p^N\mathbf{Z}$. (The entries are p-integral because $p > 2g + 1 = 3$; see the discussion in [Edi03, p. 17].) Then we compute F^N; this has the effect of killing the non-unit root eigenspace modulo p^N. Writing

$$F^N = \begin{pmatrix} a & b \\ c & d \end{pmatrix},$$

it turns out that we recover

$$\mathbf{E}_2(E, \omega) = -12b/d \pmod{p^N}.$$

(Note that d is a p-adic unit, so no precision is lost in the division; see [MST06, p. 591].)

Now consider the general case, where the equation of E is

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

(still minimal at p). Let Δ be the discriminant for this equation, and let Δ' be the discriminant after passing to an isomorphic curve

$$Y^2 = X^3 + A_4X + A_6,$$

permissible since $p \geq 5$. If $\omega' = dX/2Y$ is the invariant differential corresponding to the latter equation, we may compute $\mathbf{E}_2(E, \omega')$ as above, and then we have

$$\mathbf{E}_2(E, \omega) = \left(\frac{\Delta'}{\Delta}\right)^{1/6} \mathbf{E}_2(E, \omega'), \tag{6.4}$$

since Δ is of weight 12 and $\mathbf{E}_2$ of weight 2. Note that $v_p(\Delta) = v_p(\Delta') = 0$, since E was assumed to have good reduction at p and both equations were minimal at p, so no p-adic precision is lost in evaluating (6.4).

We may now prove Theorem 2 from the Introduction.

Proof of Theorem 2. The running time of the algorithm described above is dominated by that of Kedlaya's algorithm, which by Theorem 4 (with $g = n = 1$) is $\widetilde{O}(pN^2)$. Computing F^N from F requires only $O(\log N)$ matrix multiplications (using repeated squaring), so $O(\log N)$ ring operations in $\mathbf{Z}/p^N\mathbf{Z}$, costing time only $\widetilde{O}(\log N \log(p^N)) = \widetilde{O}(N \log p)$. This proves part (a).

For part (b), assume that $p > 6N$. We will replace Kedlaya's algorithm by the algorithm implementing Theorem 5 (with $g = n = 1$). The hypotheses of that theorem are satisfied since

$$(2N-1)(2g+1) = 3(2N-1) < 6N.$$

After computing F in this manner, the rest of the algorithm runs as before. □

6.2.2 The differential equation for the sigma function

The Mazur–Stein–Tate algorithm uses the fact (proved in [MT91]) that $\sigma_p(t)$ is the unique odd function in $\mathbf{Z}_p[\![t]\!]$ of the form $\sigma_p(t) = t + \cdots$ satisfying the differential equation

$$x(t) + c = -\frac{d}{\omega}\left(\frac{1}{\sigma}\frac{d\sigma}{\omega}\right), \tag{6.5}$$

where c is the constant

$$c = \frac{a_1^2 + 4a_2 - \mathbf{E}_2(E,\omega)}{12}, \tag{6.6}$$

and where $x(t) \in \mathbf{Z}_p[\![t]\!]$ is the power series expansion of x at the origin. Since c is known from $\mathbf{E}_2(E, \omega)$, it becomes a matter of computing $x(t)$, and then solving (6.5) for $\sigma_p(t)$.

The method they propose for computing $x(t)$ is based on a certain recursive formula [MST06, p. 590] for the coefficients of the auxiliary power series $w(t) = -1/y(t) = \sum_{k \geq 0} s_k t^k$. The recursive formula for s_k involves a term of the form $\sum_{i_1+i_2+i_3=k} s_{i_1} s_{i_2} s_{i_3}$, which contains $O(k^2)$ products; therefore the cost of computing s_k for all $k \leq n$ is $O(n^3)$ arithmetic operations in the coefficient ring. In Chapter 8 we give a different method for computing the first n terms of $x(t)$ that requires only $\widetilde{O}(n)$ ring operations.

Given $x(t)$, Mazur, Stein and Tate solve (6.5) by using the formal logarithm to change variables from t to the parameter z on the *additive* group. After making this substitution, the differential equation takes the simpler form

$$x(z) + c = -\frac{d}{dz}\left(\frac{1}{\sigma(z)}\frac{d\sigma}{dz}\right),$$

which can be solved by integration and exponentation of power series. A bottleneck arises here also: to perform the change of variables, it is necessary to invoke several power series composition and reversion operations. To the author's knowledge, the Brent–Kung algorithm [BK78], which has complexity $\widetilde{O}(n^{3/2})$ in the number of terms, is the best algorithm available for computing series reversions and compositions. In Chapter 8 we show that it is entirely unnecessary to change variables; the original equation (6.5) can be solved directly, up to t^n, in $\widetilde{O}(n)$ ring operations. Moreover our method avoids the p-adic precision losses caused by the divisions by p in the integration and exponentation steps.

6.2.3 Computing the p-adic height

Let $Q \in E(\mathbf{Q})$. Mazur, Stein and Tate propose using (6.2) together with (6.3) to compute $h_p(Q)$, as follows. Let $n_1 = \#E(\mathbf{F}_p)$, let n_2 be the least common multiple of the Tamagawa numbers of E, and put $n = \mathrm{LCM}(n_1, n_2)$. Then $P = nQ$ is guaranteed to be in the finite-index subgroup on which (6.2) is valid, so $h_q(Q)$ may be computed from (6.3), using the series expansion for $\sigma_p(t)$ obtained earlier.

The above procedure has a serious bottleneck when p is large. The difficulty is that one must actually compute the coordinates of nQ, the numerator and denominator of which will generally have about n^2 digits (assuming that Q is non-torsion). From the Weil bound we have $\#E(\mathbf{F}_p) \approx p$, so n is roughly proportional to p. Therefore the time complexity is at least proportional to p^2, and for sufficiently large p will dwarf even the time required to compute $\mathbf{E}_2(E, \omega)$.

In Chapter 9 (see also Proposition 17) we show how to avoid this problem by computing the "n_1 part" of the point multiple in modular arithmetic instead of over $\mathbf{Q}$, thereby preventing coefficient explosion. In this way we reduce the running time from quadratic in p to only polylogarithmic in p.

We introduce one further optimisation to the Mazur–Stein–Tate method. For a typical $P \in E_0$, with $v_p(t(P)) = 1$ (or not much bigger than 1), the series for $\sigma_p(t)$ converges relatively slowly at $t(P)$. By replacing P by a p-power multiple $p^\lambda P$, we push the point p-adically closer to the origin; $t(p^\lambda P)$ has larger valuation, so the series for $\sigma_p(t)$ converges more quickly there. This allows us to compute $h_p(p^\lambda P)$ more efficiently, and $h_p(P)$ is recovered by using the fact that h_p is a quadratic form. (Alternatively, one could directly use the functional equation $\sigma_p(mP) = \sigma_p(P)^{m^2} f_m(P)$ satisfied by σ_p, where f_m is the m-th division polynomial; see [MT91, p. 670].) This efficiency gain is not for free: one must take into account

the cost of computing $p^\lambda P$ from P. By balancing this point-multiplication cost against the savings from faster evaluation of $\sigma_p(t)$, we find that the optimal choice is $\lambda \approx \sqrt{N}$, where p^N is the desired precision for $h_p(P)$ (see the proof of Theorem 3). This observation saves a factor of about $\sqrt{N}$ from the total running time.

Chapter 7

Point multiplication in modular arithmetic

Let $Q \in E(\mathbf{Q})$ be a non-torsion point, and let $m \geq 2$ be an integer. In the notation introduced earlier, the coordinates of mQ are given by

$$(x(mQ), y(mQ)) = \left(\frac{\alpha(mQ)}{d(mQ)^2}, \frac{\beta(mQ)}{d(mQ)^3} \right).$$

In this chapter we consider the problem of computing $\alpha(mQ)$, $\beta(mQ)$ and $d(mQ)$ *modulo* L, where $L \geq 1$ is an odd integer.

The most straightforward method is to compute the coordinates of mQ in $\mathbf{Q}$, and then reduce $\alpha(mQ)$, $\beta(mQ)$ and $d(mQ)$ modulo L. Unfortunately, this approach suffers from coefficient explosion. Indeed, by using properties of the Néron–Tate (real-valued) canonical height, one finds that the number of digits of (say) $d(mQ)$ grows *quadratically* in m.

The following result improves the complexity to only *logarithmic* in m, pro-

vided that Q lies in a certain finite-index subgroup of $E(\mathbf{Q})$:

Proposition 17. *Suppose that $Q \in E(\mathbf{Q})$ reduces to a non-singular point of $E(\mathbf{F}_\ell)$ for every prime ℓ at which E has bad reduction. Let $L \geq 1$ be odd, and $m \geq 2$. Given the values of $\alpha(Q)$, $\beta(Q)$ and $d(Q)$ modulo L, one may compute $\pm\alpha(mQ)$, $\beta(mQ)$ and $\pm d(mQ)$ modulo L in time $\widetilde{O}(\log L \log m)$.*

(The $\pm$ symbols indicate that $\alpha(mQ)$ and $d(mQ)$ will be correct only up to sign, and that the signs will agree.)

The remainder of this chapter is devoted to the proof of Proposition 17. Our main tools are the division polynomials ψ_m associated to our choice of Weierstrass equation for E, and a non-cancellation result of Wuthrich [Wut04, Prop. IV.2] that in effect controls the amount of cancellation that can occur while computing mQ from Q. For further background on division polynomials, including proofs of the assertions that we use in this section, we refer the reader to [MT91, Appendix I] and [Lan78, Ch. II].

The relevance of division polynomials is that they appear in a simple formula for the coordinates of mQ in terms of the coordinates of Q. For an integer $m \geq 1$, and a non-torsion point Q, we have

$$x(mQ) = \frac{\theta_m(Q)}{\psi_m(Q)^2}, \qquad y(mQ) = \frac{\omega_m(Q)}{\psi_m(Q)^3}, \tag{7.1}$$

where $\theta_m, \omega_m \in \mathbf{Q}(E)$ are certain auxiliary functions defined in terms of the ψ_m.

The quantities $\psi_m(Q), \theta_m(Q), \omega_m(Q) \in \mathbf{Q}$ will generally not be integers, due to the coordinates of Q themselves having denominators. It is convenient to introduce a normalising factor that absorbs these denominators. Accordingly we

set

$$\begin{aligned}\hat{\psi}_m(Q) &= \psi_m(Q)d(Q)^{m^2-1},\\ \hat{\theta}_m(Q) &= \theta_m(Q)d(Q)^{2m^2},\\ \hat{\omega}_m(Q) &= \omega_m(Q)d(Q)^{3m^2}.\end{aligned}$$

Note that $\hat{\psi}_m$, $\hat{\theta}_m$ and $\hat{\omega}_m$ are defined only on $E(\mathbf{Q})$ — they are *not* rational functions on E. One checks, by examining the degrees of ψ_m, θ_m and ω_m, that $\hat{\psi}_m(Q)$, $\hat{\theta}_m(Q)$ and $\hat{\omega}_m(Q)$ are all integers. Therefore we now have *two* representations of $x(mQ)$ and $y(mQ)$ as ratios of integers,

$$\begin{aligned}x(mQ) &= \frac{\alpha(mQ)}{d(mQ)^2} = \frac{\hat{\theta}_m(Q)}{\hat{\psi}_m(Q)^2 d(Q)^2},\\ y(mQ) &= \frac{\beta(mQ)}{d(mQ)^3} = \frac{\hat{\omega}_m(Q)}{\hat{\psi}_m(Q)^3 d(Q)^3}.\end{aligned} \tag{7.2}$$

The point of (7.2) is twofold. First, Proposition IV.2 of [Wut04] guarantees that, under the hypothesis of Proposition 17, the fractions on the right are in fact *reduced* fractions; in other words, that $d(mQ) = \pm\hat{\psi}_m(Q)d(Q)$. Therefore we may conclude from (7.2) that

$$\begin{aligned}d(mQ) &= \pm\hat{\psi}_m(Q)d(Q),\\ \alpha(mQ) &= \hat{\theta}_m(Q),\\ \beta(mQ) &= \pm\hat{\omega}_m(Q),\end{aligned} \tag{7.3}$$

where the choices of signs in the first and third equations agree.

Second, we will show that $\hat{\psi}_m(Q)$, $\hat{\theta}_m(Q)$ and $\hat{\omega}_m(Q)$ can be efficiently computed modulo L using the usual recursion formulae for the division polynomials. For our application, it is not necessary to compute the division polynomials them-

selves — in fact, to do so would completely miss the point, since their degree grows like m^2, which is precisely the rate of growth that we are trying to avoid. Rather, we need only compute their *values at* Q, and only modulo L. Fortunately the standard recursive formulae for division polynomials, with minor modifications, are perfectly suited to this task.

We give a version of the formulae that involve no divisions (apart from one division by 2, which is permitted since we have assumed that 2 is invertible modulo L). This ensures that we never lose p-adic precision in our later applications in the p-adic setting. Also, our formulae are tailored to computing the normalised versions $\hat{\psi}_m(Q)$, $\hat{\theta}_m(Q)$ and $\hat{\omega}_m(Q)$ directly, instead of $\psi_m(Q)$, $\theta_m(Q)$ and $\omega_m(Q)$, so that we can work with integral quantities throughout.

In the formulae below, we abbreviate $\alpha(Q)$ by α, and similarly with the other variables. We start by defining normalised versions of the coefficients a_k of the elliptic curve, setting

$$\hat{a}_k = d^k a_k, \quad k = 1, 2, 3, 4, 6.$$

We next define variables $\hat{b}_k$ and $\hat{B}_k$, which are normalised versions of the b_k and B_k appearing in [MT91]:

$$\begin{aligned}
\hat{b}_2 &= \hat{a}_1^2 + 4\hat{a}_2, \\
\hat{b}_4 &= \hat{a}_1\hat{a}_3 + 2\hat{a}_4, \\
\hat{b}_6 &= \hat{a}_3^2 + 4\hat{a}_6, \\
\hat{b}_8 &= \hat{a}_1^2\hat{a}_6 + 4\hat{a}_2\hat{a}_6 - \hat{a}_1\hat{a}_3\hat{a}_4 + \hat{a}_2\hat{a}_3^2 - \hat{a}_4^2,
\end{aligned}$$

and

$$\begin{aligned}\hat{B}_4 &= 6\alpha^2 + \hat{b}_2\alpha + \hat{b}_4,\\ \hat{B}_6 &= 4\alpha^3 + \hat{b}_2\alpha^2 + 2\hat{b}_4\alpha + \hat{b}_6,\\ \hat{B}_8 &= 3\alpha^4 + \hat{b}_2\alpha^3 + 3\hat{b}_4\alpha^2 + 3\hat{b}_6\alpha + \hat{b}_8.\end{aligned}$$

Similarly, we need normalised versions of the g_m from [MT91], defined by

$$\hat{g}_0 = 0, \quad \hat{g}_1 = 1, \quad \hat{g}_2 = -1, \quad \hat{g}_3 = \hat{B}_8, \quad \hat{g}_4 = \hat{B}_6^2 - \hat{B}_4\hat{B}_8,$$

and then recursively for $m \geq 5$ by

$$\begin{aligned}\hat{g}_{2n+1} &= \begin{cases}\hat{B}_6^2\hat{g}_{n+2}\hat{g}_n^3 - \hat{g}_{n-1}\hat{g}_{n+1}^3, & n \text{ even},\\ \hat{g}_{n+2}\hat{g}_n^3 - \hat{B}_6^2\hat{g}_{n-1}\hat{g}_{n+1}^3, & n \text{ odd},\end{cases}\\ \hat{g}_{2n} &= \hat{g}_n(\hat{g}_{n-2}\hat{g}_{n+1}^2 - \hat{g}_{n+2}\hat{g}_{n-1}^2).\end{aligned} \tag{7.4}$$

Finally, the values of $\hat{\psi}_m$, $\hat{\theta}_m$ and $\hat{\omega}_m$ for $m \geq 2$ are given in terms of the $\hat{g}_m$ by

$$\begin{aligned}\hat{T} &= 2\beta + \hat{a}_1\alpha + \hat{a}_3,\\ \hat{\psi}_m &= \hat{T}^{\sigma(m+1)}\hat{g}_m,\\ \hat{\theta}_m &= \alpha\hat{\psi}_m^2 - \hat{\psi}_{m+1}\hat{\psi}_{m-1},\\ \hat{\omega}_m &= \frac{-1}{2}\left(\hat{T}^{\sigma(m)}(\hat{g}_{m-2}\hat{g}_{m+1}^2 - \hat{g}_{m+2}\hat{g}_{m-1}^2) + \hat{\psi}_m(\hat{a}_1\hat{\theta}_m + \hat{a}_3\hat{\psi}_m^2)\right),\end{aligned} \tag{7.5}$$

where $\sigma(k)$ is 0 or 1 accordingly as k is even or odd.

The algorithm implementing Proposition 17 now runs as follows. All computations are performed modulo L. We are given as input $\alpha(Q)$, $\beta(Q)$ and $d(Q)$, and the constants a_k. From (7.3) it suffices to compute $\hat{\psi}_m$, $\hat{\theta}_m$ and $\hat{\omega}_m$.

We start by computing all of the $\hat{a}_k$, $\hat{b}_k$ and $\hat{B}_k$, and $\hat{g}_0$ through $\hat{g}_4$, using the formulae given above. Then, using (7.4), recursively compute $\hat{g}_{m-2}$ through $\hat{g}_{m+2}$. During this recursive step, it is important to retain the values of $\hat{g}_j$ as they are computed, since many of them will be reused. Finally, the equations (7.5) then determine $\hat{\psi}_m$, $\hat{\theta}_m$ and $\hat{\omega}_m$.

Now we analyse the complexity. Arithmetic operations in $\mathbf{Z}/L\mathbf{Z}$ may be performed in time $\widetilde{O}(\log L)$, so the crux of the matter is to show that the recursive formulae (7.4) are evaluated at most $O(\log m)$ times.

Let $k \geq 4$. To determine $\hat{g}_j$ for all j in the range $k \leq j \leq k+7$, using (7.4) it suffices to know $\hat{g}_j$ for $(k-3)/2 \leq j \leq (k+11)/2$ if k is odd, or $(k-4)/2 \leq j \leq (k+10)/2$ if k is even. In other words, to determine 8 consecutive values of $\hat{g}_j$ near $j = k$, it suffices to know 8 consecutive values of $\hat{g}_j$ near $j = k/2$. Iterating this process, to compute $\hat{g}_k$ one must evaluate (7.4) at most $8 \log_2(k) = O(\log k)$ times. This completes the proof of Proposition 17.

Chapter 8

Computing the canonical p-adic sigma function

The main result of this chapter is Proposition 19, which gives an algorithm for computing the p-adic sigma function $\sigma_p(t)$ modulo a certain ideal $J_{r,\lambda} \subseteq \mathbf{Z}_p[\![t]\!]$. This ideal appears quite naturally in the context of computing p-adic heights (Chapter 9). The running time of the algorithm is quasilinear in the size of the output, so is optimal up to logarithmic factors.

Definition 18. For integers $r \geq 2$ and $\lambda \geq 0$, let $J_{r,\lambda}$ be the ideal of $\mathbf{Z}_p[\![t]\!]$ generated by

$$\{p^{\max(0,r-2+(3-j)(\lambda+1))}t^j\}_{j\geq 0} = \{p^{r+3\lambda+1}, p^{r+2\lambda}t, p^{r+\lambda-1}t^2, p^{r-2}t^3, \ldots\}.$$

The ideal $J_{r,\lambda}$ has a triangular shape: if a power series f is known modulo $J_{r,\lambda}$, then we know its constant term modulo $p^{r+3\lambda+1}$, and the precision of the coefficients drops off linearly with slope $\lambda + 1$. The quotient ring $\mathbf{Z}_p[\![t]\!]/J_{r,\lambda}$ is

finite, and its elements require $O(r^2(\lambda+1)^{-1}\log p)$ bits to store.

Proposition 19. *Assume that* $0 \le \lambda \le r$. *Given* $\mathbf{E}_2(E,\omega)$ *modulo* p^{r-2} *as input,* $\sigma_p(t)$ *may be determined modulo* $J_{r,\lambda}$ *in time* $\widetilde{O}(r^2(\lambda+1)^{-1}\log p)$.

The proof is given in §8.3 below.

8.1 Some auxiliary power series

Let $x(t)$ and $y(t)$ be the power series expansions of x and y around the origin. Let $w(t) = -1/y(t)$, and let

$$s(t) = \frac{x'(t)}{2y(t) + a_1 x(t) + a_3},$$

so that $s(t)dt$ is the series expansion of the invariant differential ω. The first few terms of each series are given by

$$\begin{aligned} x(t) &= t^{-2} - a_1 t^{-1} - a_2 + \cdots, \\ y(t) &= -t^{-3} + a_1 t^{-2} + a_2 t^{-1} + \cdots, \\ w(t) &= t^3 + a_1 t^4 + (a_1^2 + a_2) t^5 + \cdots, \\ s(t) &= 1 + a_1 t + (a_1^2 + a_2) t^2 + \cdots. \end{aligned}$$

(See also [Sil92, Ch. IV], which discusses these expansions in some detail, using slightly different notation.)

Let $k \ge 1$ and $n \ge 1$ be integers, and let $R = \mathbf{Z}/p^k\mathbf{Z}$. Our first task is to compute the above series over R, with n terms each.

Proposition 20. *The series $t^2x(t)$, $t^3y(t)$, $t^{-3}w(t)$ and $s(t)$ may be computed up to $O(t^n)$, with coefficients in R, in time $\widetilde{O}(nk \log p)$.*

Proof. The series $w(t)$ satisfies the algebraic equation

$$w = t^3 + a_1tw + a_2t^2w + a_3w^2 + a_4tw^2 + a_6w^3,$$

and so it may be solved using Newton's method. We start with the initial approximation $w(t) = t^3$, and then repeatedly apply the Newton iteration

$$\begin{aligned} w' &= w - \frac{w - t^3 - a_1tw - a_2t^2w - a_3w^2 - a_4tw^2 - a_6w^3}{1 - a_1t - a_2t^2 - 2a_3w - 2a_4tw - 3a_6w^2} \\ &= \frac{t^3 - a_3w^2 - a_4tw^2 - 2a_6w^3}{1 - a_1t - a_2t^2 - 2a_3w - 2a_4tw - 3a_6w^2}. \end{aligned}$$

The arithmetic is performed using truncated power series in $R[t]$. It is straightforward to check that the number of correct terms doubles with each iteration. Each iteration requires several power series multiplications and one reciprocal. As is typical in applications of Newton's method to power series, the total time to obtain n terms is a constant multiple of the time required for a length n polynomial multiplication. Therefore the total time to compute n terms of $w(t)$ with coefficients in R is $\widetilde{O}(n \log(p^k)) = \widetilde{O}(nk \log p)$.

Given $t^{-3}w(t)$ up to $O(t^n)$, we may then deduce $x(t) = t/w(t)$, $y(t) = -1/w(t)$ and $s(t) = x'(t)/(2y(t) + a_1x(t) + a_3)$, also with coefficients in R, to the desired number of terms, in time $\widetilde{O}(nk \log p)$. □

8.2 A p-adic version of Brent's algorithm

We digress to develop a tool that we will need in §8.3 to solve a differential equation of the form

$$\frac{F'}{F} = f,$$

where f is a power series with p-adic coefficients. Formally the solution is given by

$$F(t) = \exp\left(\int f(t)dt\right),$$

but is this computationally problematic since both the integration and exponentation steps introduce denominators, causing some loss of p-adic precision.

The following result gives an efficient means to find F, without any denominators appearing in intermediate steps, and with good control over precision loss. The algorithm we describe is essentially that of Brent [Bre76], with some additional analysis to track the p-adic error terms (Brent's algorithm was not originally designed for the p-adic setting).

Proposition 21. *Let $k \geq 1$ and $1 \leq n < p^{\lceil k/2 \rceil}$, and let $R = \mathbf{Z}/p^k\mathbf{Z}$. Let $f \in R[t]/(t^{n-1})$, and suppose that there exists $F \in R[t]/(t^n)$, with $F(0) = 1$, such that $F'/F = f$. Then given f as input, F may be determined modulo $\mathcal{J}$ in time $\widetilde{O}(nk \log p)$, where $\mathcal{J}$ is the ideal of $R[t]/(t^n)$ given by*

$$\mathcal{J} = (p^{k-1}t^p, p^{k-2}t^{p^2}, \ldots).$$

Remark. Note that $\mathcal{J}$ is the ideal that captures the types of p-adic error terms that occur in a power series integration. Namely, if $g \in R[t]/(t^{n-1})$, and if the coefficient of t^j in g is divisible by $j+1$ for each j, then g has an integral in

$R[t]/(t^n)$; in general it is not uniquely determined, but it is determined at least modulo $\mathcal{J}$.

Remark. The running time estimate in Proposition 21 is optimal up to logarithmic factors, since the size of the output data is proportional to $nk \log p$ bits.

Proof. We begin with an initial approximation $F_0(t) = 1 \in R[t]/(t^n)$. We will refine it iteratively, obtaining a sequence $F_i \in R[t]/(t^n)$ such that $F_i - F \in \mathcal{J} + (t^{2^i})$ for each $i \geq 0$. After $\lceil \log_2 n \rceil$ steps, we will have $F_i - F \in \mathcal{J}$ as desired. Each step is dominated by a polynomial multiplication and a division of length 2^i, so the total time is a constant multiple of the time required for a single polynomial multiplication of length n, which is $\widetilde{O}(n \log(p^k)) = \widetilde{O}(nk \log p)$.

Now we explain the iterative step. Suppose that $F_i - F \in \mathcal{J} + (t^{2^i})$. Since F is invertible, we have

$$F_i = (1 + \varepsilon)F$$

for some $\varepsilon \in \mathcal{J} + (t^{2^i})$, and so

$$\frac{F_i'}{F_i} - f = \frac{F'}{F} + \frac{\varepsilon'}{1+\varepsilon} - f = \frac{\varepsilon'}{1+\varepsilon}.$$

Morally speaking we would like to integrate $F_i'/F_i - f$ to obtain $\log(1 + \varepsilon)$, but the latter does not make sense in our ring. Instead, we write

$$\frac{\varepsilon'}{1+\varepsilon} = \varepsilon' - \varepsilon'\varepsilon + \frac{\varepsilon'\varepsilon^2}{1+\varepsilon}.$$

The hypothesis $n < p^{\lceil k/2 \rceil}$ implies that $\mathcal{J}^2 = 0$, so that $\varepsilon^2 \in t^{2^i}\mathcal{J} + (t^{2^{i+1}})$. We

also have $t\varepsilon' \in \mathcal{J} + (t^{2^i})$, so $t\varepsilon'\varepsilon^2 \in (t^{2^{i+1}})$, and then $\varepsilon'\varepsilon^2 \in (t^{2^{i+1}-1})$. Consequently

$$\frac{F_i'}{F_i} - f \in \varepsilon' - \varepsilon'\varepsilon + (t^{2^{i+1}-1}).$$

Therefore $F_i'/F_i - f$ may be integrated at least up to $O(t^{2^{i+1}})$; that is, we may compute a $G \in R[t]/(t^n)$ such that

$$G \in \varepsilon - \frac{\varepsilon^2}{2} + \mathcal{J} + (t^{2^{i+1}}).$$

(The extra $\mathcal{J}$ term is introduced by errors in the integration.) Since $\varepsilon^2 \in \mathcal{J} + (t^{2^{i+1}})$ we have in fact

$$G \in \varepsilon + \mathcal{J} + (t^{2^{i+1}}).$$

Now it is straightforward to define F_{i+1}; we simply take

$$F_{i+1} = F_i(1 - G).$$

From the above estimates this satisfies

$$\begin{aligned} F_{i+1} &\in F(1+\varepsilon)(1-\varepsilon) + \mathcal{J} + (t^{2^{i+1}}) \\ &= F - \varepsilon^2 F + \mathcal{J} + (t^{2^{i+1}}) \\ &= F + \mathcal{J} + (t^{2^{i+1}}) \end{aligned}$$

as desired. □

8.3 The differential equation for the sigma function

In this section we prove Proposition 19. We start with the differential equation (6.5), which may be rewritten as

$$x(t) + c = \frac{-1}{s(t)} \left(\frac{\sigma_p'(t)}{\sigma_p(t) s(t)} \right)'.$$

It is convenient to rephrase this in terms of the unit power series

$$\theta(t) = t^{-1}\sigma_p(t) = 1 + \cdots \in \mathbf{Z}_p[\![t]\!].$$

Solving for $\sigma_p(t)$ modulo $J_{r,\lambda}$ is equivalent to solving for $\theta(t)$ modulo $J_{r-\lambda-1,\lambda}$.

We have $\theta'(t)/\theta(t) = \sigma_p'(t)/\sigma_p(t) + t^{-1}$, so $\theta(t)$ satisfies the equation

$$x(t) + c = \frac{-1}{s(t)} \left(\frac{1}{s(t)} \left(\frac{-1}{t} + \frac{\theta'(t)}{\theta(t)} \right) \right)'. \tag{8.1}$$

Manipulating this equation formally, we obtain

$$\frac{\theta'(t)}{\theta(t)} = h(t) \qquad \in \mathbf{Z}_p[\![t]\!], \tag{8.2}$$

where

$$h(t) = -\frac{1}{t} - s(t) \left(\int (x(t) + c) s(t) dt + C \right) \tag{8.3}$$

for some constant of integration C. (The formal integration operator is assumed to output a series with zero constant term.) The constant C is determined by the condition that $\sigma_p(t)$ is an *odd* function; we will see below that this implies that

$C = a_1/2$.

Let

$$n = \left\lceil \frac{r-2}{\lambda+1} \right\rceil + 2.$$

The algorithm begins by computing an order n approximation to $h(t)$, as follows.

First compute $x(t)$ and $s(t)$, up to n terms, with coefficients modulo p^{r-2}, using Proposition 20; that is,

$$x(t) = t^{-2} - a_1 t^{-1} - a_2 + \cdots + O(t^{n-2}),$$
$$s(t) = 1 + a_1 t + (a_1^2 + a_2)t^2 + \cdots + O(t^n).$$

Compute the product

$$(x(t) + c)s(t) = t^{-2} + c + \cdots + O(t^{n-2}).$$

At this stage the coefficients are still correct modulo p^{r-2} (note that c is obtained modulo p^{r-2} from the input $\mathbf{E}_2(E, \omega)$ via (6.6)). This last series is integrable in $\mathbf{Z}_p[\![t]\!]$, since we know from (8.3) that it is the derivative of $-(h(t) + t^{-1})/s(t)$, which lies in $\mathbf{Z}_p[\![t]\!]$. (This fact is the basis of the 'integrality algorithm' of [MST06] for computing $\mathbf{E}_2(E, \omega)$.) After integrating to obtain

$$\int (x(t) + c)s(t)\, dt + C = -t^{-1} + C + ct + \cdots + O(t^{n-1}),$$

the terms are no longer all correct modulo p^{r-2}; indeed, the coefficient of t^j is correct only modulo $p^{r-2-v_p(j)}$.

Multiplying by $s(t)$ and subtracting from $-1/t$ yields

$$h(t) = (a_1 - C) + (a_1^2 - a_1 C + a_2 - c)t + \cdots + O(t^{n-1}).$$

The multiplication by $s(t)$ caused the incorrect digits to wash through to higher order terms, so now the coefficient of t^j is correct only modulo $p^{r-2-\lfloor \log_p j \rfloor}$. Denote by $\hat{h}(t)$ the approximation to $h(t)$ that we computed by the above procedure.

From the above expression for $h(t)$ we may deduce the value of C. Namely, we have

$$\sigma_p(t) = t\theta(t) = \exp\left(\int h(t)dt\right) = t + (a_1 - C)t^2 + \cdots . \tag{8.4}$$

To say that $\sigma_p(t)$ is odd means that $\sigma_p(i(t)) = -\sigma_p(t)$, where $i(t) = -t - a_1 t^2 + \cdots$ is the formal inverse law [Sil92, IV §1]. Substituting $i(t)$ into the above expression for $\sigma_p(t)$ and equating coefficients of t^2 yields $C = a_1/2$.

We now wish to apply Proposition 21 (Brent's algorithm) to solve (8.2) for $\theta(t)$. We cannot quite do this, because we only have the approximation $\hat{h}(t)$, not $h(t)$ itself. This may be sidestepped in the following way. Treating $\hat{h}(t)$ as an element of $\mathbf{Z}_p[\![t]\!]$ (any lift will do), let

$$\hat{\theta}(t) = \exp\left(\int \hat{h}(t)\, dt\right) \quad \in \mathbf{Q}_p[\![t]\!].$$

As argued above, for $0 \leq j < n-1$ the coefficient of t^j in $\hat{h} - h$ has valuation at least $r - 2 - \lfloor \log_p j \rfloor$, so for $1 \leq j < n$ the coefficient of t^j in $\int(\hat{h} - h)dt$ has valuation at least

$$r - 2 - \lfloor \log_p(j-1) \rfloor - \lfloor \log_p j \rfloor.$$

Therefore for $1 \leq j < n$ the coefficient of t^j in

$$\begin{aligned}\frac{\hat{\theta}(t)}{\theta(t)} &= \exp\left(\int (\hat{h}(t) - h(t))\, dt\right) \\ &= 1 + \sum_{k \geq 1} \frac{1}{k!} \left(\int (\hat{h}(t) - h(t))\, dt\right)^k\end{aligned}$$

has valuation at least

$$r - 2 - \lfloor \log_p(j-1) \rfloor - \lfloor \log_p j \rfloor - \left\lfloor \frac{j}{p-1} \right\rfloor,$$

because $v_p(j!) \leq j/(p-1)$. Since $p \geq 5$ we have

$$2 + \lfloor \log_p(j-1) \rfloor + \lfloor \log_p j \rfloor + \left\lfloor \frac{j}{p-1} \right\rfloor \leq j$$

for all $j \geq 2$. (It suffices to estimate the left hand side for $p = 5$. For large enough j, the $j/4$ term dominates; one must also check a few small values of j directly.)

This shows that the coefficients of t^j of $\hat{\theta}(t)$ and $\theta(t)$ agree modulo p^{r-j} for $2 \leq j < n$. For $j = 0, 1$, the coefficients agree modulo p^{r-2} (since no precision was lost in $h(t)$ for those terms at all). In particular, the coefficients of $\hat{\theta}(t)$ are *integral* for $j < n$, since $\theta(t) \in \mathbf{Z}_p[\![t]\!]$ and $n \leq r$. Therefore we may apply Proposition 21 with $F = \hat{\theta}$, $f = \hat{h}$ and $k = r - 2$ to solve the equation

$$\frac{\hat{\theta}'(t)}{\hat{\theta}(t)} = \hat{h}(t)$$

for $\hat{\theta}(t)$. (The hypothesis $n < p^{\lceil k/2 \rceil}$ of Proposition 21 is satisfied as long as $r \geq 3$, since $n \leq r$ and $p \geq 5$. In the special case $r = 2$ we do not need Proposition 21 at all, instead reading off $\theta(t)$ directly from (8.4).)

Let $\overline{\theta}(t)$ be the approximation for $\hat{\theta}(t)$ produced by Proposition 21. We have $\overline{\theta} - \hat{\theta} \in \mathcal{J}$; that is, the coefficients of t^j in $\overline{\theta}(t)$ and $\hat{\theta}(t)$ agree modulo $p^{r-2-\lfloor \log_p j \rfloor}$. Since $j \geq \lfloor \log_p j \rfloor + 2$ for $j \geq 2$ and $p \geq 5$, the coefficients of t^j in $\overline{\theta}(t)$ and $\theta(t)$ agree modulo p^{r-j} for $2 \leq j < n$.

Since $\lambda \geq 0$ we have

$$(r - \lambda - 1) - 2 + (3 - j)(\lambda + 1) = r - 2 + (2 - j)(\lambda + 1) \leq r - j$$

for $j \geq 2$, showing that $\overline{\theta}(t)$ and $\theta(t)$ agree modulo $J_{r-\lambda-1,\lambda}$, except possibly for the coefficients of t^0 and t^1. These two terms may be easily computed separately modulo $p^{r+2\lambda}$ and $p^{r+\lambda-1}$ respectively, via (8.4).

Finally we must analyse the complexity. The application of Proposition 21 costs $\widetilde{O}(r^2(\lambda+1)^{-1}\log p)$ since $n = O(r/(\lambda+1))$. The remainder of the algorithm consists of the polynomial arithmetic needed to compute $\hat{h}(t)$, also costing $\widetilde{O}(r^2(\lambda+1)^{-1}\log p)$. This completes the proof of Proposition 19.

Chapter 9

Computing the p-adic height

In this chapter we prove Theorem 3. Let $P \in E(\mathbf{Q})$ be a non-torsion point, and let $N \geq 1$ be an integer; we wish to compute $h_p(P)$ modulo p^N. For the purposes of estimating running times, we assume that E and P are fixed; we study the running time only as a function of p and N.

Let $n_1 = \#E(\mathbf{F}_p)$, let n_2 be the least common multiple of the Tamagawa numbers of E, and let $n = \mathrm{LCM}(n_1, n_2)$. Let $\lambda \geq 0$ be an integer, and put $R = p^\lambda nP$. Then $R \in E_0$, and R reduces to a nonsingular point of $E(\mathbf{F}_\ell)$ for all ℓ, so (6.2) and (6.3) imply that

$$h_p(P) = \frac{1}{n^2 p^{2\lambda}} h_p(R) = \frac{2}{n^2 p^{2\lambda}} \log_p \left(\frac{\sigma_p(R)}{d(R)} \right). \tag{9.1}$$

Put

$$N' = N + 2v_p(n).$$

Note that $v_p(n_1) \leq 1$, and that $N' = N + O(1)$ since we are assuming that E is fixed (in fact usually $N' = N$). To compute $h_p(P)$ modulo p^N, from (9.1) it

suffices to compute $\log_p(\sigma_p(R)/d(R))$ modulo $p^{N'+2\lambda}$, and therefore it suffices to compute the unit part of $\sigma_p(R)/d(R)$ modulo $p^{N'+2\lambda}$, by the following lemma:

Lemma 22. *Suppose that $u \in \mathbf{Z}_p^*$ is a unit and that $M \geq 1$ is an integer. To determine $\log_p u$ modulo p^M it suffices to know u modulo p^M.*

Proof. Since $\log_p(u+\varepsilon) = \log_p u + \log_p(1+\varepsilon/u)$, the result amounts to checking that if $v_p(\varepsilon) \geq M$, then $v_p(\log_p(1+\varepsilon)) \geq M$. Using the power series expansion of $\log_p(1+x)$, this follows from the elementary estimate $v_p(\varepsilon^n/n) \geq v_p\varepsilon$, that is, $v_p(n) \leq n-1$ for all $n \geq 1$. □

Next we have:

Lemma 23. *To compute the unit part of $\sigma_p(R)/d(R)$ modulo $p^{N'+2\lambda}$, it suffices to determine $\alpha(R)$, $\beta(R)$ and $d(R)$ modulo $p^{N'+2\lambda}$, and $\sigma_p(t)$ modulo $J_{N',\lambda}$.*

(See Chapter 8 for the definition of the ideal $J_{r,\lambda}$.)

Proof. Observe that

$$\frac{\sigma_p(R)}{d(R)} = \frac{-\alpha(R)}{\beta(R)}\left(1 + \sum_{j\geq 1} c_{j+1} t(R)^j\right), \tag{9.2}$$

where the coefficients c_j are as in (6.1). Since $\beta(R)$ is a p-adic unit, it suffices to compute $\alpha(R)$, $\beta(R)$, and $1 + \sum_{j\geq 1} c_{j+1} t(R)^j$ modulo $p^{N'+2\lambda}$.

Furthermore, since $nP \in E_0$ we have $v_p(t(nP)) \geq 1$, and then $v_p(t(R)) \geq 1+\lambda$, since multiplication by p on the formal group increases the p-adic valuation of the parameter t. Therefore we need to know c_{j+1} modulo $p^{N'+2\lambda-j(\lambda+1)} = p^{N'-2+(2-j)(\lambda+1)}$ for $1 \leq j \leq \lfloor (N'+2\lambda)/(\lambda+1) \rfloor$. This is equivalent to knowledge of $\sigma(t)$ modulo $J_{N',\lambda}$. □

Finally we may prove Theorem 3.

Proof of Theorem 3. Let $0 \leq \lambda \leq N'$; we will select the optimal value of λ below. We first use Proposition 19 with $r = N'$ to compute $\sigma_p(t)$ modulo $J_{N',\lambda}$ in time $\widetilde{O}(N^2(\lambda+1)^{-1}\log p)$.

Next compute $Q = n_2 P$. This is a constant-time operation, since P and E (and hence n_2) are assumed fixed. Note that $R = p^\lambda n' Q$, where $n' = n/n_2$, and that Q reduces to a non-singular point of $E(\mathbf{F}_\ell)$ for all ℓ (by definition of n_2).

Now apply Proposition 17 with $m = p^\lambda n'$ and $L = p^{N'+2\lambda}$, obtaining $\pm\alpha(R)$, $\beta(R)$ and $\pm d(R)$ modulo L. Using the fact that $n' \leq n = O(p)$ and $N' = O(N)$, the time cost is $\widetilde{O}(\log L \log m) = \widetilde{O}(N\lambda \log^2 p)$.

We have collected enough information to apply Lemma 23, thereby obtaining the unit part of $\sigma_p(R)/d(R)$ modulo $p^{N'+2\lambda}$ from (9.2). (Note that the sign ambiguities in $\alpha(R)$ and $d(R)$ are inconsequential. In $t = -d\alpha/\beta$ the signs cancel out, and the p-adic logarithm is insensitive to the sign of its input.) The cost of evaluating (9.2) is $O((N'+2\lambda)/(\lambda+1)) = O(N/(\lambda+1))$ ring operations in $\mathbf{Z}/p^{N'+2\lambda}\mathbf{Z}$, for a bit-complexity of $\widetilde{O}(N^2(1+\lambda)^{-1}\log p)$.

Finally we must evaluate the p-adic logarithm in (9.1) to obtain $h_p(P)$. Using the series expansion of $\log_p(1+x)$, this requires $O(N'+2\lambda)$ ring operations, for a cost of $\widetilde{O}(N^2 \log p)$; with a more sophisticated algorithm one can obtain $\widetilde{O}(N \log p)$ [Ber, §16].

Summing all the estimates, we obtain

$$\widetilde{O}(N^2(\lambda+1)^{-1}\log p + N\lambda \log^2 p)$$

for the total running time. By choosing $\lambda = \sqrt{N}$, we balance the exponents of N in these two terms, obtaining the estimate $\widetilde{O}(N^{3/2}\log^2 p)$. □

Chapter 10

Example computations

The author implemented the above algorithms in the computer algebra system Sage [SJ05], building on an implementation of the algorithm of [MST06] by Robert Bradshaw, Jennifer Balakrishnan, Liang Xiao, William Stein, and the author. The code is freely available under a GPL license, and is distributed as a standard component of Sage (version 2.10.2 and later).

The following sample computations were performed on the same machine as described in Chapter 5.

10.1 Large prime case

We will take the elliptic curve '92b1' from Cremona's database, which has equation $y^2 = x^3 - x + 1$ and conductor $92 = 2^2 \cdot 23$. A generator of the group of rational points is $P = (x, y) = (1, 1)$.

Let $p = 10^{11} + 3$ and $N = 6$. We need to compute $\mathbf{E}_2$ modulo p^4. Using the algorithm implementing Theorem 2(b), Sage finds that the matrix of Frobenius

on the standard basis for Monsky–Washnitzer cohomology is given by

$$\begin{pmatrix} 643045857608761156981756803441983181300130 83 & 425039723809360255611246023101867348704772 20 \\ 971285583853682105405681417894577353352735 47 & 356954142511238843023643196558124818699437 49 \end{pmatrix}$$

modulo p^4. The computation time was 42 minutes. As a consistency check, one may verify that the determinant of this matrix is $p \pmod{p^4}$, and that the trace is

$$1000000000120000000005400000000107999999956832 \equiv -43249 \pmod{p^4},$$

which agrees with other point-counting algorithms.

From the matrix Sage finds immediately that

$$\mathbf{E}_2 = 744701682804855332135084234707411222845601 52 + O(p^4).$$

Finally, Sage uses Theorem 3 to find that $h_p(P)$ is

$$p \cdot 92263242705398789443691249592034738060552930445990726 58 + O(p^6).$$

This last step, including computation of the p-adic sigma function to some precision, is virtually instantaneous. Observe that the original algorithm of [MST06] would have required computing the coordinates of the point nP where $n \approx 10^{11}$, which would occupy around 10^{22} bits of storage — not to mention the computation time. Clearly, Proposition 17 is essential for handling such large p.

10.2 High precision case

Continuing with the same curve, we now consider the case $p = 5$ and $N = 3000$. Therefore we require $\mathbf{E}_2$ modulo 5^{2998}. Since p is small relative to N, Sage uses the algorithm of Theorem 2(a), based on Kedlaya's original algorithm [Ked01], for the Frobenius matrix computation. The first and last few digits of $\mathbf{E}_2$ are

$$\mathbf{E}_2 = 3 + 2 \cdot 5 + 2 \cdot 5^3 + 3 \cdot 5^5 + 2 \cdot 5^7 + \cdots + 3 \cdot 5^{2995} + 3 \cdot 5^{2996} + O(5^{2998}).$$

The computation time to obtain $\mathbf{E}_2$ was 189 seconds.

After selecting $\lambda = 54$, the rest of the algorithm implementing Theorem 3 executes in just 2.5 seconds. The p-adic height turns out to be

$$h_p(P) = 3 \cdot 5 + 3 \cdot 5^2 + 2 \cdot 5^3 + 5^4 + \cdots + 4 \cdot 5^{2998} + 2 \cdot 5^{2999} + O(5^{3000}).$$

For such high precision computations, the original algorithm of [MST06] would have been utterly impractical; the N^4 contribution would multiply the above running time by a factor of perhaps 10^8.

10.3 Asymptotic behaviour

In this section we give more data illustrating the running time asymptotics predicted by Theorem 2 and Theorem 3. All the examples below are for the curve '37a' from Cremona's database, with equation $y^2 + y = x^3 - x$ and conductor 37.

Table 10.1 gives the time to compute $\mathbf{E}_2$ via the algorithm of Theorem 2(a), for $p = 5$, with increasing values of N spaced by a factor of $\sqrt{2}$. As expected, the

running time approximately doubles for each successive row.

N	time	N	time	N	time
100	0.27s	400	2.24s	1600	41.4s
141	0.43s	566	4.87s	2263	113s
200	0.62s	800	9.46s	3200	203s
283	1.23s	1131	22.9s		

Table 10.1: Time to compute $\mathbf{E}_2$ via Theorem 2(a) for $p = 5$ and varying N

Similarly, Table 10.2 shows the running time when we fix $N = 100$ and take increasing values of p. Again, we expect the running time to approximately double for each successive row, which matches the data reasonably well.

p	time	p	time
23	1.5s	197	26.3s
47	3.9s	397	62.6s
97	10.2s		

Table 10.2: Time to compute $\mathbf{E}_2$ via Theorem 2(a) for $N = 100$ and varying p

Now we consider Theorem 2(b). In Chapter 5 we already illustrated the asymptotic behaviour of our implementation when N is fixed and p varies. Now we consider the opposite direction: Table 10.3 shows the running time when we fix $p = 10007$ and vary N. We expect the running times in successive rows to increase by a factor of $2^{5/2} \approx 5.7$, matching the data quite well.

N	time	N	time
10	1.5s	40	42s
20	7.4s	80	273s

Table 10.3: Time to compute $\mathbf{E}_2$ via Theorem 2(b) for $p = 10007$ and varying N

For Theorem 3, we are unable to give any timing data that satisfactorily illustrates the $N^{3/2}$ dependence on N or the polylogarithmic dependence on p. In the case of the variable N, the main difficulty is that in the point-multiplication step, the $N^{3/2}$ behaviour would only become visible when N is large enough that quasilinear time integer arithmetic becomes available for integers of size p^N. For such large values of N, the computation of $\mathbf{E}_2$ is essentially infeasible. For the feasible values of N, we tend to see behaviour closer to quadratic in N. The situation for illustrating the dependence on p is even worse, since the running time for the computation of $\mathbf{E}_2$ is *exponential* in $\log p$.

Therefore we should consider the asymptotic running time asserted by Theorem 3 to be mainly of theoretical interest. Nevertheless, *in practice* we find that the algorithm given in the proof of Theorem 3 is extremely efficient, in the sense that the time needed to compute the p-adic height is miniscule compared to the time expended in computing $\mathbf{E}_2$ (witness the examples in §10.1 and §10.2 above).

Bibliography

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust, *The Magma algebra system. I. The user language.*, J. Symbolic Comput. **24** (1997), no. (3-4), 235–265.

[Ber] D. Bernstein, *Fast multiplication and its applications.*

[BGS07] Alin Bostan, Pierrick Gaudry, and Eric Schost, *Linear recurrences with polynomial coefficients and application to integer factorization and Cartier–Manin operator*, SIAM Journal on Computing **36** (2007), no. 6, 1777–1806.

[BK78] R. P. Brent and H. T. Kung, *Fast algorithms for manipulating formal power series*, J. Assoc. Comput. Mach. **25** (1978), no. 4, 581–595.

[Bre76] R. P. Brent, *Multiple-precision zero-finding methods and the complexity of elementary function evaluation*, Analytic computational complexity (Proc. Sympos., Carnegie-Mellon Univ., Pittsburgh, Pa., 1975), Academic Press, New York, 1976, pp. 151–176.

[CC88] D. V. Chudnovsky and G. V. Chudnovsky, *Approximations and complex multiplication according to Ramanujan*, Ramanujan revisited (Urbana-Champaign, Ill., 1987), Academic Press, Boston, MA, 1988, pp. 375–472.

[CFA+06] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen, and Frederik Vercauteren (eds.), *Handbook of elliptic and hyperelliptic curve cryptography*, Discrete Mathematics and its Applications (Boca Raton), Chapman & Hall/CRC, Boca Raton, FL, 2006.

[CK91] David G. Cantor and Erich Kaltofen, *On fast multiplication of polynomials over arbitrary algebras*, Acta Inform. **28** (1991), no. 7, 693–701.

[Edi03] Bas Edixhoven, *Point counting after Kedlaya*, EIDMA-Stieltjes Graduate course, Leiden (unpublished lecture notes), http://www.math.leidenuniv.nl/~edix/oww/mathofcrypt/carls_edixhoven/kedlaya.pdf (retrieved Oct 25th 2006), 2003.

[GG01] Pierrick Gaudry and Nicolas Gürel, *An extension of Kedlaya's point-counting algorithm to superelliptic curves*, Advances in cryptology—ASIACRYPT 2001 (Gold Coast), Lecture Notes in Comput. Sci., vol. 2248, Springer, Berlin, 2001, pp. 480–494.

[GG03] ———, *Counting points in medium characteristic using Kedlaya's algorithm*, Experiment. Math. **12** (2003), no. 4, 395–402.

[GS04] Pierrick Gaudry and Éric Schost, *Construction of secure random curves of genus 2 over prime fields*, Advances in cryptology—EUROCRYPT 2004, Lecture Notes in Comput. Sci., vol. 3027, Springer, Berlin, 2004, pp. 239–256.

[Har07] David Harvey, *Kedlaya's algorithm in larger characteristic*, Int Math Res Notices **2007** (2007), no. rnm095, rnm095–29.

[Har08] ———, *Efficient computation of p-adic heights*, LMS J. Comput. Math. **11** (2008), 40–59.

[HQZ04] Guillaume Hanrot, Michel Quercia, and Paul Zimmermann, *The middle product algorithm. I*, Appl. Algebra Engrg. Comm. Comput. **14** (2004), no. 6, 415–438.

[Hub07] Hendrik Hubrechts, *Point counting in families of hyperelliptic curves*, http://wis.kuleuven.be/algebra/hubrechts/ (retrieved May 26th 2007), 2007, to appear in Foundations of Computational Mathematics.

[Ked01] K. S. Kedlaya, *Counting points on hyperelliptic curves using Monsky-Washnitzer cohomology*, J. Ramanujan Math. Soc. **16** (2001), no. 4, 323–338.

[KS08] Kiran S. Kedlaya and Andrew Sutherland, *Computing L-series of hyperelliptic curves*, arXiv preprint math.NT/0801.2778v2, 2008.

[Lan78] S. Lang, *Elliptic curves: Diophantine analysis*, Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], vol. 231, Springer-Verlag, Berlin, 1978.

[MST06] B. Mazur, W. Stein, and J. Tate, *Computation of p-adic heights and log convergence*, Documenta Math. (Extra Volume: John H. Coates' Sixtieth Birthday) (2006), 577–614.

[MT91] B. Mazur and J. Tate, *The p-adic sigma function*, Duke Math. J. **62** (1991), no. 3, 663–688.

[MTT86] B. Mazur, J. Tate, and J. Teitelbaum, *On p-adic analogues of the conjectures of Birch and Swinnerton-Dyer*, Invent. Math. **84** (1986), no. 1, 1–48.

[Pil90] J. Pila, *Frobenius maps of abelian varieties and finding roots of unity in finite fields*, Math. Comp. **55** (1990), no. 192, 745–763.

[Sho] Victor Shoup, *NTL: A library for doing number theory*, http://www.shoup.net/ntl/.

[Sil92] J. H. Silverman, *The arithmetic of elliptic curves*, Graduate Texts in Mathematics, vol. 106, Springer-Verlag, New York, 1992, Corrected reprint of the 1986 original.

[SJ05] W. Stein and D. Joyner, *Sage: System for algebra and geometry experimentation*, Communications in Computer Algebra (ACM SIGSAM Bulletin) **39** (2005), no. 2, 61–64.

[Str69] Volker Strassen, *Gaussian elimination is not optimal*, Numer. Math. **13** (1969), 354–356.

[Str77] ———, *Einige Resultate über Berechnungskomplexität*, Jber. Deutsch. Math.-Verein. **78** (1976/77), no. 1, 1–8.

[Sut08] Andrew Sutherland, *A generic approach to searching for Jacobians*, arXiv preprint math.NT/0708.3168v2, 2008.

[vdP86] Marius van der Put, *The cohomology of Monsky and Washnitzer*, Mém. Soc. Math. France (N.S.) (1986), no. 23, 4, 33–59, Introductions aux cohomologies p-adiques (Luminy, 1984).

[Wut04] C. Wuthrich, *The fine selmer group and height pairings*, Ph.D. thesis, Cambridge, 2004.