SQUARE ROOT TIME COLEMAN INTEGRATION ON SUPERELLIPTIC CURVES

ALEX J. BEST

Abstract. Since Kedlaya first introduced a p-adic algorithm for computing zeta functions of hyperelliptic curves, many related algorithms for computing both zeta functions and Coleman integrals on various classes of algebraic curves have been studied. These algorithms compute in the Monsky-Washnitzer cohomology or the rigid cohomology of the curve to determine the action of Frobenius on this cohomology.

We give a new algorithm for explicitly computing Coleman integrals on superelliptic curves over unramified extensions of p-adic fields. The runtime is softly linear with respect to the square root of the size of the residue field, bringing the runtime in line with that of the corresponding zeta function algorithms. We also describe the implementation of this algorithm in Nemo, a new package for the Julia programming language, which adds functionality for computational number theory. We compare Nemo with other systems in use in this area.

1. Introduction

In [18], Kedlaya introduced an algorithm using p-adic cohomology to compute the zeta function of a hyperelliptic curve defined over a finite field. Since then, there have been advances and generalisations of this method in (at least) three different directions. Firstly [13], [14] and [22, 23] have introduced variants that work on more general curves, respectively; superelliptic curves, cyclic covers of $\mathbf{P}^1$ and general curves with a map to $\mathbf{P}^1$. Secondly, Harvey [16] introduced a variant that runs in time quasilinear in $\sqrt{p}$. Finally, Balakrishnan-Bradshaw-Kedlaya [1] gave an algorithm to compute p-adic Coleman integrals on curves over the p-adics. A large part of the runtime of this algorithm is taken up by computing explicit relations in Monsky-Washnitzer cohomology, the same core procedure used in Kedlaya's algorithm.

In [20], Minzlaff showed that the first and second of these could be combined; that is, there exists an algorithm that computes the zeta function of superelliptic curves over $\mathbf{F}_{p^n}$ in $\widetilde{O}(p^{1/2})$ time. Additionally the paper [7] gives an algorithm based on the work of Balakrishnan-Bradshaw-Kedlaya and of Harvey that computes Coleman integrals on hyperelliptic curves over $\mathbf{Q}_p$ in time quasilinear in $\sqrt{p}$.

In this paper, we incorporate the work of Minzlaff into that of [7]. Let a, b be coprime integers with $a > 1, b > 2$, let p be a prime and $q = p^n$, take $h \in \mathbf{Z}_q[x]$ a squarefree polynomial of degree b, and consider the curve

$$C/\mathbf{Z}_q \colon y^a = h(x).$$

Let $g = (a-1)(b-1)/2$ be the genus of C. Let M be the matrix of q-power Frobenius acting on $H^1_{\mathrm{dR}}(C \times_{\mathbf{Z}_q} \mathbf{Q}_q)$ (via comparison with $H^1_{\mathrm{cris}}(C \times_{\mathbf{Z}_q} \mathbf{F}_q, \mathbf{Q}_q)$), in terms of the basis $B = \{\omega_{i,j} = x^i \,\mathrm{d}x/y^j\}_{i=0,\ldots,b-2, j=1,\ldots,a}$ and $N \in \mathbf{N}_{\geq 1}$ be such that both C and points $P, Q \in C(\mathbf{Q}_{p^n})$ are known to precision p^N, and assume $p > (aN-1)b$. Then, if multiplying two $m \times m$ matrices requires $\widetilde{O}(m^\omega)$ ring operations we have:

Theorem 1.1 *The vector of Coleman integrals* $(\int_P^Q \omega)_{\omega \in B}$ *can be computed in time*

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

to absolute p*-adic precision* $N - v_p(\det(M - I))$.

The work of Balakrishnan-Tuitman [4] has shown that there exists an algorithm to compute Coleman integrals on general curves defined over p-adic fields. This article shows that fast Coleman integration algorithms are not limited just to the hyperelliptic case.

We also take the opportunity to work out the practical details of the theory without restricting to the case of monic $h(x)$ or curves only defined over $\mathbf{Q}_p$; instead we work over arbitrary unramified extensions of $\mathbf{Q}_p$ throughout.

The algorithms described in this paper have been implemented using the Nemo computer algebra system [12], which is a specialised package for number theory and related mathematics for the programming language Julia [8]. This is a relatively new system for computing in commutative algebra, number theory and group theory that is based on several low-level libraries such as MPIR, Flint, Arb and Antic. A secondary goal of this paper is therefore to compare the performance of the various computer algebra systems available to mathematicians for running this type of computation. The implementation also allows for h to be non-monic and the curve C to be defined over an unramified extension of $\mathbf{Q}_p$.

1.0.1. *Acknowledgements.* This work was supported by Simons Foundation grant #550023 and a Hariri Institute Graduate Fellowship.

I would like to thank Jennifer Balakrishnan for many valuable pieces of advice regarding this project and her constant encouragement. Tommy Hofman has patiently answered many of my questions about Nemo and Hecke and has also improved the Julia implementation, making it both more idiomatic and faster; I am very grateful for all of his help. I would also like to thank the anonymous referees for their helpful feedback.

2. Set-up and notation

The major outline of the algorithm presented in this paper follows that of [7], adapted to the case of a **(tamely) superelliptic curve**: these will be smooth projective curves C of a specific form, which will be defined over the ring of integers of an unramified extension of $\mathbf{Q}_p$ (such extensions will be denoted as $\mathbf{Q}_q$, where $q = p^n$ is the cardinality of the residue field). The present work can be read in conjunction with [7].

We will denote by σ the Witt vector Frobenius on $\mathbf{Z}_q$ and $\mathbf{Q}_q$. The restriction of being superelliptic for us means that the curve should be given by an affine equation of the form

$$y^a = h(x),\ \deg(h) = b,\ h \text{ squarefree}$$

where $\gcd(a, b) = 1$ and $a > 1, b > 2$. Tameness means that, in addition, we assume $p \nmid a$, though this will be implied by our later assumption (2.1). We will write $h = \lambda x^b + \tilde{h}$ with $\deg \tilde{h} < b$. We can view the projective curve C as living inside of the weighted projective space $\mathbf{P}(a, b, 1)$, where there is a unique point at infinity on the curve, denoted ∞. As h might not be monic, this will be of the form $(\lambda^t : \lambda^s : 0)$ for some $s, t \in \mathbf{Z}$ such that $as - bt = 1$. A superelliptic curve of this form will have genus

$$g = \frac{(a-1)(b-1)}{2}.$$

Each set of points $D \subseteq C(\overline{\mathbf{Q}}_q)$ that is the full preimage of some $x \in C_{\overline{\mathbf{F}}_q}(\overline{\mathbf{F}}_q)$ under the reduction map is known as a **residue disk**.

Such a superelliptic curve comes equipped with a natural map to $\mathbf{P}^1$ given by

$$(x : y : z) \mapsto (x : z^a).$$

We will often be interested in the points where this map is branched (i.e. those with $y = 0$), and call them the **branch points** of this curve. We also call any residue disk containing one of these points a **branch disk**.

These play the analogous role to that of the Weierstrass points in the hyperelliptic case, but we prefer to avoid calling them Weierstrass points as the notion of a Weierstrass point in algebraic geometry would include non-branch points in general, see [21, p. 3372].

We will often work with the affine open subset of the curve obtained by removing all branch points (including ∞), writing

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

This affine space is then

$$\operatorname{Spec} A = U \subseteq C.$$

We denote by $A^\dagger$ the weak completion of A; this is the ring of formal power series

$$A^\dagger = \left\{ \sum_{i=-\infty}^{\infty} \frac{R_i(x)}{y^i} : R_i \in \mathbf{Z}_q[x]_{\deg \leq b-1},\ \liminf_{|i| \to \infty} \frac{v_p(R_i)}{|i|} > 0 \right\}.$$

We will consider the module of 1-forms

$$\Omega^1_{A^\dagger} = A^\dagger \, \mathrm{d}x \oplus A^\dagger \, \mathrm{d}y / (a y^{a-1} \, \mathrm{d}y - h'(x) \, \mathrm{d}x)),$$

and the exterior derivative map is denoted

$$\mathrm{d} \colon A^\dagger \to \Omega^1_{A^\dagger}.$$

We also use A_{loc} to denote the $\mathbf{Q}_q$-algebra of $\overline{\mathbf{Q}}_q$-valued functions on $C(\overline{\mathbf{Q}}_q)$ that are given by a convergent power series on each residue disk and are $\mathrm{Gal}(\overline{\mathbf{Q}_q}/\mathbf{Q}_q)$-equivariant. Note that $A^\dagger \hookrightarrow A_{\mathrm{loc}}$. The Monsky-Washnitzer cohomology of A is then defined to be $H^1_{\mathrm{MW}}(A) = \Omega^1_{A^\dagger}/\mathrm{d}(A^\dagger)$.

From now on, we make the assumption that

$$p > (aN - 1)b; \tag{2.1}$$

this simplifies the analysis of denominators appearing later in the algorithm. This assumption is likely completely removable without affecting the asymptotic complexity, or at least smaller p may be used by doing a more involved precision analysis to ensure the right amount of extra precision is used.

3. Coleman integration

Coleman integration is an integration theory that is in particular defined for 1-forms on curves, viewed as p-adic analytic spaces. This theory has been applied to various questions in the arithmetic of curves over number fields and p-adic fields. These applications include provably determining the set of rational points on a curve using Chabauty-Coleman(-Kim) [19, 3] and determining torsion points on curves. For background on Coleman integration, especially the details of the action of Frobenius when $n > 1$, we refer to [5].

Coleman integration for curves is computable in many cases [6, 1, 4]; that is, given a base point and a 1-form to some finite p-adic precision, the corresponding integral can be computed to some (smaller) precision. This can be

done in many instances of interest to arithmetic geometers. Moreover these algorithms perform well in practice, returning an answer in examples of interest to mathematicians in an acceptable real-world runtime.

Coleman integration requires the choice of a *lift of Frobenius*, that is a map

$$\phi\colon A^\dagger \to A^\dagger$$

such that the reduction of this map to $\mathbf{F}_q$

$$\bar{\phi}\colon A^\dagger_{\mathbf{F}_q} \to A^\dagger_{\mathbf{F}_q}$$

is the p-power Frobenius map and $\phi(k) = \sigma(k)$ for all $k \in \mathbf{Z}_q$. There are many possibilities for such a lift in general; in this paper, we will make an explicit choice of a lift for superelliptic curves that allows us to analyse its action in detail. It is therefore important to note that the theory is known to be independent of the choice of ϕ.

We define the action of ϕ on $C(\overline{\mathbf{Q}}_q)$ via

$$\phi(x_0, y_0) = (\sigma^{-1}(\phi(x)(x_0, y_0)), \sigma^{-1}(\phi(y)(x_0, y_0))). \tag{3.1}$$

On functions $f\colon C(\overline{\mathbf{Q}}_q) \to \overline{\mathbf{Q}}_q$ the action of ϕ is then

$$\phi(f)(P) = \sigma f(\phi(P)). \tag{3.2}$$

The key theorem proved by Coleman describing this integration theory can be stated in our setting as follows.

Theorem 3.1 (Coleman). *There is a unique (up to a global constant of integration) $\mathbf{Q}_q$-linear integration map $\int\colon \Omega^1_{A^\dagger} \otimes \mathbf{Q}_q \to A_{\mathrm{loc}}$ satisfying the following:*

(1) Frobenius equivariance: $\int(\phi\omega) = \phi\left(\int \omega\right)$

(2) the fundamental theorem of calculus: $\mathrm{d} \circ \int$ is the canonical inclusion

$$\Omega^1_{A^\dagger} \otimes \mathbf{Q}_q \to \Omega^1_{\mathrm{loc}}$$

(3) and $\int \circ \mathrm{d}$ is the natural map $A^\dagger \to A_{\mathrm{loc}}/(\text{constant functions})$.

Given points $P, Q \in C(\overline{\mathbf{Q}}_q)$ the definite integral $\int_P^Q \omega$ is then defined as $\left(\int \omega\right)(Q) - \left(\int \omega\right)(P)$, which is a well-defined function of P, Q.

Balakrishnan-Bradshaw-Kedlaya [1] describe how, using these properties, the computation of Coleman integrals of 1-forms on a curve can be broken up into two parts:

(1) The computation of *tiny integrals* between points in the same residue disk, using local coordinates on the curve.
(2) The evaluation of exact forms appearing when reducing Frobenius pullbacks of differentials in Monsky-Washnitzer cohomology.

In the remainder of this section, we give a description of local coordinates that can be used to compute tiny integrals and describe how this method works in our setting in more detail, and in the next section we describe a procedure for evaluating the relevant exact forms.

3.0.1. *Local coordinates on superelliptic curves.* In order to compute Coleman integrals locally on a p-adic disk, we need an explicit local parametrisation of such a disk. To do this we apply Newton's method/Hensel's lemma; to find a power series solution $x(t)$ to a polynomial $F(x(t))$, we start with some $x_0(t)$, and iterate $x_i(t) = x_{i-1}(t) - F(x_{i-1}(t))/F'(x_{i-1}(t))$. The limiting power series is the desired solution; applying this directly with $F(x(t), y(t)) = y(t)^a - h(x(t))$ in the x or y variable leads to the next results, giving local coordinates around a finite branch point or non-branch point.

Proposition 3.2 Local coordinates around a point not in a branch disk. *Let $P \in C(\mathbf{Q}_q)$ be a point not in a branch disk, represented as $P = (X, Y)$ on $y^a = h(x)$. Then local coordinates $(x(t), y(t))$ around P can be given by*

$$\begin{aligned} x(t) &= X + t, \\ y_0(t) &= Y, \\ y_i(t) &= \frac{1}{a}\left((a-1)y_{i-1}(t) + \frac{h(x(t))}{y_{i-1}(t)^{a-1}}\right), \text{ and} \\ y(t) &= \lim_{i\to\infty} y_i(t). \end{aligned}$$

Proof. We may simplify

$$\begin{aligned} y_i(t) &= y_{i-1}(t) - \frac{y_{i-1}(t)^a - h(x(t))}{a y_{i-1}(t)^{a-1}} = y_{i-1}(t) - \frac{1}{a}y_{i-1}(t) + \frac{h(x(t))}{a y_{i-1}(t)^{a-1}} \\ &= \frac{1}{a}\left((a-1)y_{i-1}(t) + \frac{h(x(t))}{y_{i-1}(t)^{a-1}}\right). \end{aligned}$$

■

Proposition 3.3 Local coordinates around a point in a finite branch disk. *Let $P \in C(\mathbf{Q}_q)$ be a point in a non-infinite branch disk, represented as $P = (X, Y)$ on $y^a = h(x)$. Then local coordinates $(x(t), y(t))$ around P can be given by*

$$\begin{aligned} y(t) &= Y + t, \\ x_0(t) &= X, \\ x_i(t) &= x_{i-1}(t) + \frac{y(t)^a - h(x_{i-1}(t))}{h'(x_{i-1}(t))}, \text{ and} \\ x(t) &= \lim_{i\to\infty} x_i(t). \end{aligned}$$

Proof. Direct application of Newton's method. ■

For the infinite disk the choice of uniformiser is less obvious and several different choices can be made. For instance, one could take a local coordinate t at ∞ to be such that $x = t^{-a}$. Then given a point (x_0, y_0) in the same residue disk as ∞, to find the value of the parameter t that gives (x_0, y_0), we can take each of the possible roots $\sqrt[a]{x_0}$ and check which of these give y_0.

Instead, we take t to be an appropriate monomial in x, y. This has the advantage that, to find the value of the local coordinate for a point (X_0, Y_0), we may compute the corresponding coordinate t_0 using only multiplication of the values X_0, Y_0. Precisely, we do the following:

Proposition 3.4 Local coordinates around ∞. *Let ∞ be the point at infinity on C. Then local coordinates $(x(t), y(t))$ around ∞ can be given as follows. Let ℓ, k be such that $b\ell - ak = 1$. Set*

$$\begin{aligned} x_0(t) &= \lambda^{-\ell} t^{-a}, \\ y_0(t) &= \lambda^{-k} t^{-b}, \\ J(x, y) &= \begin{pmatrix} -h'(x) & a y^{a-1} \\ -k x^{k-1} & \ell t y^{\ell-1} \end{pmatrix}, \\ \begin{pmatrix} x_i(t) \\ y_i(t) \end{pmatrix} &= \begin{pmatrix} x_{i-1}(t) \\ y_{i-1}(t) \end{pmatrix} - J(x_{i-1}(t), y_{i-1}(t))^{-1} \begin{pmatrix} t y_{i-1}(t)^\ell - x_{i-1}(t)^k \\ y_{i-1}(t)^a - h(x_{i-1}(t)) \end{pmatrix}. \end{aligned}$$

Then

$$x(t) = \lim_{i\to\infty} x_i(t),$$

$$y(t) = \lim_{i\to\infty} y_i(t).$$

Proof. Using the assumption that $\gcd(a,b) = 1$, we may find ℓ, k such that $b\ell - ak = 1$. We now find $x(t), y(t)$ such that $t = x(t)^k/y(t)^\ell$. To do this, we solve $ty(t)^\ell = x(t)^k$ and $y(t)^a = h(x(t))$ simultaneously using multivariate Newton's method; this gives the recurrence stated. ∎

When working with hyperelliptic curves, one can use the hyperelliptic involution to avoid computing integrals from infinity to points in branch disks, where the exact forms that would need to be evaluated fail to converge. We can also avoid this issue when working with superelliptic curves as follows:

Proposition 3.5 *Fix μ to be the automorphism of the cover*

$$C \xrightarrow{x} \mathbf{P}^1$$

given by

$$\mu\colon C \to C$$
$$(x,y) \mapsto (x, \zeta_a y)$$

for some fixed primitive a-th root of unity $\zeta_a \in \overline{\mathbf{Q}}_q$. If P, Q are branch points and ω is a differential for which $\mu^*\omega = \zeta_a^\beta \omega$ for some $\beta \in \{1, \dots, a-1\}$, then

$$\int_Q^P \omega = 0.$$

Proof. We have

$$\int_P^Q \omega = \int_{\mu P}^{\mu Q} \omega = \int_Q^P \mu^*\omega = \zeta_a^\beta \int_P^Q \omega.$$

∎

Corollary 3.6 *Let P be a point in a branch disk. We have for any branch point Q*

$$\int_P^\infty \omega = \int_P^Q \omega.$$

Choosing Q to be the branch point in the same disk as P we can therefore compute

$$\int_P^\infty \omega$$

by expanding only in local coordinates around Q.

3.0.2. *The action of q-power Frobenius.* We now assume that we know the action of ϕ on $\Omega^1_{A^\dagger}$ in the sense that we have the following:

(1) A fixed basis $(\omega_i)_i$ of $H^1_{MW}(C)$, thought of as a column vector
(2) a matrix of Frobenius M,
(3) and a vector of primitives $(f_i)_i$ such that,

$$(\phi^*\omega_i)_i = M(\omega_i)_i + (\mathrm{d}f_i)_i \in (\Omega^1_{A^\dagger})^{2g}.$$

This data is what will be returned by (our generalisation of) Kedlaya's algorithm.

The next lemma calculates the action of ϕ^{*n} on $\Omega^1_{A^\dagger}$ from this data; this is consistent with [2, Rmk. 1] but appears to be different from [1, Rmk. 12].

Lemma 3.7 *The action of a power of Frobenius on the basis differentials is given by*

$$\phi^{*n}(\omega_i)_i = \sum_{t=n-1,\dots,0} \left(\prod_{s=n-1,\dots,t+1} \phi^s(M) \right) \phi^{*t}(\mathrm{d}f_i)_i + \prod_{s=n-1,\dots,0} \phi^s(M)(\omega_i)_i.$$

Proof. By induction on n, we will apply the relation

$$\phi^*(\omega_i)_i = (\mathrm{d}f_i)_i + M(\omega_i)_i.$$

The base case

$$\phi^{*0}(\omega_i)_i = 0 + 1 \cdot (\omega_i)_i$$

holds trivially (as does the $n = 1$ case which is simply the fundamental relation above), and we have

$$\begin{aligned}
\phi^{*n+1}(\omega_i)_i &= \phi^*\left(\sum_{t=n-1,\dots,0}\left(\prod_{s=n-1,\dots,t+1}\phi^s(M)\right)\phi^{*t}(\mathrm{d}f_i)_i\right) + \phi^*\left(\prod_{s=n-1,\dots,0}\phi^s(M)(\omega_i)_i\right) \\
&= \sum_{t=n-1,\dots,0}\left(\prod_{s=n-1,\dots,t+1}\phi^{s+1}(M)\right)\phi^{*t+1}(\mathrm{d}f_i)_i + \prod_{s=n-1,\dots,0}\phi^{s+1}(M)\phi^*((\omega_i)_i) \\
&= \sum_{t=n-1,\dots,0}\left(\prod_{s=n-1,\dots,t+1}\phi^{s+1}(M)\right)\phi^{*t+1}(\mathrm{d}f_i)_i \\
&\quad + \prod_{s=n-1,\dots,0}\phi^{s+1}(M)(\mathrm{d}f_i)_i + \prod_{s=n-1,\dots,0}\phi^{s+1}(M)M(\omega_i)_i \\
&= \sum_{t=n-1,\dots,0}\left(\prod_{s=n-1,\dots,t+1}\phi^{s+1}(M)\right)\phi^{*t+1}(\mathrm{d}f_i)_i \\
&\quad + \prod_{s=n-1,\dots,0}\phi^{s+1}(M)(\mathrm{d}f_i)_i + \prod_{s=n,\dots,0}\phi^{s}(M)(\omega_i)_i \\
&= \sum_{t=n-1,\dots,0}\left(\prod_{s=n,\dots,t+2}\phi^{s}(M)\right)\phi^{*t+1}(\mathrm{d}f_i)_i + \prod_{s=n-1,\dots,0}\phi^{s+1}(M)(\mathrm{d}f_i)_i + \prod_{s=n,\dots,0}\phi^{s}(M)(\omega_i)_i \\
&= \sum_{t=n,\dots,0}\left(\prod_{s=n,\dots,t+1}\phi^{s}(M)\right)\phi^{*t}(\mathrm{d}f_i)_i + \prod_{s=n,\dots,0}\phi^{s}(M)(\omega_i)_i.
\end{aligned}$$

■

We therefore have

$$\begin{aligned}
\left(\int_P^Q \omega_i\right)_i &= \left(\int_P^{\phi^n P} \omega_i + \int_{\phi^n P}^{\phi^n Q} \omega_i + \int_{\phi^n Q}^{Q} \omega_i\right)_i \\
&= \left(\int_P^{\phi^n P} \omega_i + \int_{\phi^n Q}^{Q} \omega_i\right)_i + \sum_{t=n-1,\dots,0}\left(\prod_{s=n-1,\dots,t+1}\phi^s(M)\right)\left(\int_P^Q \phi^{*t}\,\mathrm{d}f_i\right)_i \\
&\qquad + \prod_{s=n-1,\dots,0}\phi^s(M)\left(\int_P^Q \omega_i\right)_i \\
&= \left(\int_P^{\phi^n P} \omega_i + \int_{\phi^n Q}^{Q} \omega_i\right)_i + \sum_{t=n-1,\dots,0}\left(\prod_{s=n-1,\dots,t+1}\phi^s(M)\right)\left(f_i(\phi^t Q) - f_i(\phi^t P)\right)_i \\
&\qquad + \prod_{s=n-1,\dots,0}\phi^s(M)\left(\int_P^Q \omega_i\right)_i,
\end{aligned}$$

hence

$$\left(1 - \prod_{s=n-1,\dots,0}\phi^s(M)\right)\left(\int_P^Q \omega_i\right)_i = \tag{3.3}$$

(3.4)
$$\left(\int_P^{\phi^n P} \omega_i\right)_i + \left(\int_{\phi^n Q}^{Q} \omega_i\right)_i + \sum_{t=n-1,\ldots,0} \left(\prod_{s=n-1,\ldots,t+1} \phi^s(M)\right) \left(f_i(\phi^t Q) - f_i(\phi^t P)\right)_i .$$

So computing $\left(\int_P^Q \omega_i\right)_i$ reduces to computing (3.4) and inverting the matrix $1 - \prod_{s=n-1,\ldots,0} \phi^s(M)$.

Thus in order to compute the Coleman integrals between two points P, Q we need to know M and evaluations of $\phi^t f_i$ at the points P, Q for $t = 0, \ldots, n-1$. From (3.1), (3.2) we see that to compute each $(\phi^t f_i)(P)$ we need to compute $f_i(\phi^t P)$ and apply σ^t.

Remark 3.8 This makes the Teichmüller point variant of Coleman integration algorithms more appealing when working over extensions. In that approach, the Teichmüller point is used in each non-branch disk as a base point and all integrals are computed in two parts: an integral to the Teichmüller point and then tiny integrals for reaching the rest of the disk. The advantage is that all of the points $\phi^t P$ will be equal for a Teichmüller point P, hence also the values $f_i(\phi^t P)$, so fewer evaluations of f_i need to be carried out. We do not take this approach in the current implementation, however cf. Remark 6.1.

4. Reductions in cohomology

We now describe the reduction process. The foundation for this is the work of Minzlaff in [20], which gives explicit maps, reducing elements of $\Omega^1_{A^\dagger}$ to cohomologous ones with smaller x- or y-degree. To compute Coleman primitives, we must, in addition, record the exact forms subtracted to obtain these cohomologous elements and determine a recurrence that computes the evaluation of the sum of these forms at the end of the reduction process.

We consider the following spaces of differentials for $s, t \in \mathbf{Z}$ and $s \geq -1$:

$$W_{s,t} = \{x^i y^{-j} \cdot x^s y^{-at}\, \mathrm{d}x : 0 \leq i \leq b-2,\ 1 \leq j \leq a-1\},$$

and for $s = -1$ we take only the subspace with $i \geq 1$.

These differentials are such that $W_{-1,0}$ is spanned by the basis chosen in Theorem 1.1.

We may decompose $W_{s,t}$ into eigenspaces under the action of the superelliptic automorphism μ to obtain

$$W_{s,t} = W^1_{s,t} \oplus \cdots \oplus W^{a-1}_{s,t},$$

where

$$W^\beta_{s,t} = \{x^i y^{-\beta} \cdot x^s y^{-at}\, \mathrm{d}x : 0 \leq i \leq b-2\}.$$

As the subspaces indexed by β are all preserved in everything that follows, we consider each $1 \leq \beta \leq a-1$ independently from now.

We may choose a lift of Frobenius on $\widetilde{C}$ by letting $\sigma : \mathbf{Q}_q \to \mathbf{Q}_q$ be the Witt vector Frobenius and setting

$$\sigma(x) = x^p$$

as shown in [20, Sec 4.] this forces us to take

$$\sigma(y) = y^p \sum_{k=0}^{\infty} \binom{\frac{1}{a}}{k} \frac{(\sigma(h) - h^p)^k}{y^{apk}}.$$

Minzlaff also determined an approximation of the action of Frobenius on the basis differentials. Specifically we have

Lemma 4.1 ([20, Prop. 4.1]). *If we let $\lambda_r^{(k)}$ be the coefficient of x^r in h^k and*

$$\mu_{k,r,j} = p\sigma(\lambda_r^{(k)}) \sum_{\ell=k}^{N-1} (-1)^{\ell+k} \binom{-\frac{j}{a}}{\ell} \binom{\ell}{k} \in \mathbf{Z}_q,$$

then the reduction in cohomology of the 1-form

$$\sum_{k=0}^{N-1} \sum_{r=0}^{bk} \mu_{k,r,j} x^{p(i+r+1)-1} y^{-p(ak+j)} \, \mathrm{d}x$$

is congruent to that of $\sigma(x^i \, \mathrm{d}x/y^j)$ modulo p^N. Moreover the exact forms that reduce these respective cohomology classes to the cohomologous linear combination of our chosen basis elements are also congruent modulo p^N.

Proof. We denote by D_η the divisor $C \setminus U$. Then Minzlaff [20, Lem. 3.4] shows that for any $\omega \in \Omega_{A/\mathbf{Z}_q}$ with pole divisor

$$(\omega)_\infty \leq mD_\eta,$$

we have

$$p^{\left\lfloor \log_p(m-1) \right\rfloor} \pi(\omega) \in \Omega_{A/\mathbf{Z}_q}.$$

However the proof of this statement given there shows that in fact if

$$\omega = \pi(\omega) + \mathrm{d}F$$

then we also have

$$p^{\left\lfloor \log_p(m-1) \right\rfloor} F \in A.$$

As in the proof of [20, Prop. 4.1], we apply this to the series

$$\sigma(x^i \, \mathrm{d}x/y^j) = \sum_{k=0}^{\infty} p \binom{-\frac{j}{a}}{k} (\sigma(h) - h^p)^k x^{p(i+1)-1} y^{-p(ak+j)} \, \mathrm{d}x,$$

and writing S_k for the kth summand of the infinite sum on the right, we need to show that if $k \geq N$ and

$$S_k = \pi(S_k) + \mathrm{d}F_k,$$

then $\pi(S_k) \equiv 0 \pmod{p^N}$ and $F_k \equiv 0 \pmod{p^k}$. Fixing $k \geq N$, we can write

$$S_k = p^{k+1} \sum_{\ell=0}^{(k+1)p-1} f_\ell(x) y^{m_\ell} \, \mathrm{d}x, \text{ where } m_\ell = -p(ak+j) + a\ell$$

with $f_\ell \in \mathbf{Z}_q[x]_{\deg \leq b-1}$.

So we apply the above integrality statement to each

$$f_\ell(x) y^{m_\ell} \, \mathrm{d}x.$$

The same power of p used by Minzlaff is therefore sufficient. ∎

The reduction proceeds in two stages, first horizontal reduction that reduces the index s and hence the power of x appearing in the 1-forms and leaves us with forms in $W_{-1,t}$ to consider. Then vertical reduction is performed that reduces the power of y appearing and reduces all forms to $W_{-1,0}$.

We present the vertical reduction step first however, as although it takes place after the horizontal steps when the algorithm is executed, knowing the form of the vertical reduction ahead of time allows us to make some simplifications in the description of the horizontal reduction.

4.0.1. *Vertical reduction.* Because h is squarefree and λ is a unit, we can find for each $i = 0, \ldots, b-2$, a pair of polynomials $R_i, S_i \in \mathbf{Z}_q[x]$ with $\deg R_i \leq b-2$

and $\deg S_i \leq b-1$, such that

$$x^i = R_i h + S_i h'.$$

We fix a choice of such polynomials now.

The vertical reduction proceeds by reducing a 1-form in $W^{\beta}_{-1,t}$ to a cohomologous one in $W^{\beta}_{-1,t-1}$ using the following lemma:

Lemma 4.2 Vertical reduction. *We have*

$$\begin{aligned} &x^i y^{-at-\beta}\,\mathrm{d}x - \frac{-a}{at+\beta-a}\,\mathrm{d}(S_i(x)y^{-at-\beta+a}) \\ &= \frac{(at+\beta-a)R_i(x) + aS_i'(x)}{at+\beta-a} y^{-a(t-1)-\beta}\,\mathrm{d}x \in W^{\beta}_{-1,t-1}. \end{aligned}$$

Proof. Via our choice of R_i, S_i we have that

$$x^i y^{-at-\beta}\,\mathrm{d}x = (R_i(x)h(x) + S_i(x)h'(x))\, y^{-at-\beta}\,\mathrm{d}x,$$

and also that

$$\begin{aligned} \mathrm{d}(S_i(x)y^{-at-\beta+a}) &= S_i'(x)y^{-at-\beta+a}\,\mathrm{d}x + (-at-\beta+a)S_i(x)y^{-at-\beta+a-1}\,\mathrm{d}y \\ &= S_i'(x)y^{-at-\beta+a}\,\mathrm{d}x + (-at-\beta+a)\frac{1}{a}S_i(x)y^{-at-\beta}h'(x)\,\mathrm{d}x \end{aligned}$$

so

$$\frac{-a}{at+\beta-a}\,\mathrm{d}(S_i(x)y^{-at-\beta+a}) = \frac{-a}{at+\beta-a}S_i'(x)y^{-at-\beta+a}\,\mathrm{d}x + S_i(x)y^{-at-\beta}h'(x)\,\mathrm{d}x,$$

and the above equality holds. ∎

This reduction is a $\mathbf{Q}_q$-linear map

$$W^{\beta}_{-1,t} \to W^{\beta}_{-1,t-1}.$$

To express it in matrix form, we denote the coefficients of R_i by $r_{i,j}$ and likewise $s'_{i,j}$ for those of S_i', each R_i and S_i' is of degree at most $b-2$.

To compute evaluations of the primitives for Coleman integration, we augment these linear maps to also include a vector of length L, that holds the data of evaluations of the exact form subtracted so far at several points $P_i \in C(\mathbf{Q}_q)$, $i = 1, \ldots, L$. We assume that all of these points do not lie in a branch disk (including the infinite disk). This implies that $x(P_i)$ and $y(P_i)$ are both integral for all i and that $y(P_i)$ is a unit in $\mathbf{Z}_q$. So we have linear maps

$$W^{\beta}_{s+1,t} \times \mathbf{Q}_q^L \to W^{\beta}_{s,t} \times \mathbf{Q}_q^L.$$

Doing this as described gives a matrix whose entries are non-linear functions of the index t, due to the presence of the term $y^{-at+a-\beta}$ in the exact form

$$\frac{-a}{at+\beta-a}S_i(x)y^{-at-\beta+a}.$$

To remedy this, we use the approach of [7] and modify the reduction process to obtain matrices with entries linear functions of t, by factoring out the powers of y in the exact form. As we reduce monomials from $W^{\beta}_{s,t} \to W^{\beta}_{-1,t}$, we can let each later reduction step multiply the exact form by x. This results in a total power of x^s when the reduction finishes in $W^{\beta}_{-1,t}$, which is the same power of x as in the exact form obtained from $W^{\beta}_{s+1,t} \to W^{\beta}_{s,t}$.

Let $D^{\beta}_V(t)$ be the scalar $at+\beta-a \in \mathbf{Z}_q[t]$ and define a $(b-1+L)\times(b-1+L)$ matrix

$$R^{\beta}_V(t) = M^{\beta}_V(t) D^{\beta}_V(t)^{-1},$$

where $M_V^\beta(t)$ is the matrix

(4.1)
$$\left(\begin{array}{ccc|ccc} D_V^\beta(t)r_{0,0}+as'_{0,0} & \cdots & D_V^\beta(t)r_{b-2,0}+as'_{b-2,0} & & & \\ \vdots & \ddots & \vdots & & & \\ D_V^\beta(t)r_{0,b-2}+as'_{0,b-2} & \cdots & D_V^\beta(t)r_{b-2,b-2}+as'_{b-2,b-2} & & & \\ \hline -aS_0(x(P_1))y(P_1)^{-\beta+a} & \cdots & -aS_{b-2}(x(P_1))y(P_1)^{-\beta+a} & y(P_1)^{-a}D_V^\beta(t) & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ -aS_0(x(P_L))y(P_L)^{-\beta+a} & \cdots & -aS_{b-2}(x(P_L))y(P_L)^{-\beta+a} & 0 & \cdots & y(P_L)^{-a}D_V^\beta(t) \end{array}\right).$$

4.0.2. *Horizontal reduction.* We now describe the horizontal reduction process.

The horizontal reduction proceeds by reducing a 1-form in $W_{s+1,t}^\beta$ to a cohomologous one in $W_{s,t}^\beta$ using the following lemma:

Lemma 4.3 Horizontal reduction. *We have*

$$\begin{aligned} &x^{b+s-1}y^{-at-\beta}\,\mathrm{d}x - \mathrm{d}\left(\frac{a}{as\lambda-(at-a+\beta)\lambda b}x^s y^{-at+a-\beta}\right) \\ &\quad = -\frac{a(s\tilde{h}-\frac{1}{a}(at-a+\beta)x\tilde{h}')}{as\lambda-(at-a+\beta)\lambda\beta}x^{s-1}y^{-at-\beta}\,\mathrm{d}x \in W_{s,t}^\beta. \end{aligned}$$

Proof. We directly compute

$$\begin{aligned} \mathrm{d}(x^s y^{-at+a-\beta}) =& sx^{s-1}y^{-at+a-\beta}\,\mathrm{d}x - (at-a+\beta)x^s y^{-at+a-\beta-1}\,\mathrm{d}y \\ =& \left(sx^{s-1}h - \frac{1}{a}(at+\beta-a)x^s h'(x)\right)y^{-at-\beta}\,\mathrm{d}x \\ =& \left(sx^{s-1}(\lambda x^b+\tilde{h}) - \frac{1}{a}(at+\beta-a)x^s(\lambda b x^{b-a}+\tilde{h}')\right)y^{-at-\beta}\,\mathrm{d}x \\ =& \left(s(\lambda x^b+\tilde{h}) - \frac{1}{a}(at+\beta-a)x(\lambda b x^{b-a}+\tilde{h}')\right)x^{s-1}y^{-at-\beta}\,\mathrm{d}x. \end{aligned}$$

Therefore, by subtracting $\frac{a}{as\lambda-(at-a+\beta)b\lambda}\,\mathrm{d}(x^s y^{-at+a-\beta})$ from $x^b x^s y^{-at-\beta}\,\mathrm{d}x$, the remaining terms are all as stated, and of lower degree so that they lie in $W_{s,t}^\beta$. ∎

As above we augment our linear maps

$$W_{s+1,t}^\beta \to W_{s,t}^\beta$$

to also include a vector of length L that holds the data of evaluations of the exact form subtracted so far at $P_i \in C(\mathbf{Q}_q)$, $i = 1, \ldots, L$ not in the branch disks:

$$W_{s+1,t}^\beta \times \mathbf{Q}_q^L \to W_{s,t}^\beta \times \mathbf{Q}_q^L.$$

Doing this gives a matrix whose entries are non-linear functions of the index, due to the presence of the term x^s in the exact form

$$\frac{-a}{\lambda((at-a+\beta)b-as)}x^s y^{-at+a-\beta}.$$

Instead we multiply the reduction evaluation computed so far by $x(P_i)$ at each reduction step which results in a total power of x^s when the reduction finishes in $W_{-1,t}^\beta$. This is precisely the power of x in the exact form obtained from $W_{s+1,t}^\beta \to W_{s,t}^\beta$. We also do not multiply by $y^{-at+a-\beta}$, as this is the same power of y that is multiplied by in the vertical reduction steps that take place after the horizontal ones.

Rephrasing this in terms of a matrix, the reduction process is given by multiplying by a matrix of the form

$$R_H^{t,\beta}(s) = M_H^{t,\beta}(s) D_H^{t,\beta}(s)^{-1},$$

where $D_H^{t,\beta}(s) = \lambda((at-a+\beta)b - as)$, and $p_i^{t,\beta}$ is the linear function of s obtained as the coefficient of x^i in $as\tilde{h}(x) - (at-a+\beta)x\tilde{h}'(x)$ and where $M_H^{t,\beta}(s)$ is the matrix

$$\left(\begin{array}{cccc|ccc} 0 & \cdots & 0 & p_0^{t,\beta} & & & \\ D_H^{t,\beta}(s) & \cdots & 0 & p_1^{t,\beta} & & & \\ \vdots & \ddots & \vdots & \vdots & & & \\ 0 & \cdots & D_H^{t,\beta}(s) & p_{b-1}^{t,\beta} & & & \\ \hline 0 & \cdots & 0 & -a & x(P_1)D_H^{t,\beta}(s) & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & -a & 0 & \cdots & x(P_L)D_H^{t,\beta}(s) \end{array}\right). \tag{4.2}$$

This is the linear recurrence to which we may now apply the modified Bostan-Gaudry-Schost algorithm [7, Thm. 5.2], to compute the result more efficiently than the step-by-step approach alluded to above.

Remark 4.4 This is a slight modification of the approach used in [7, Sec. 4]. Here we make use of the commonality between the factors appearing in the vertical and horizontal stages to give a cleaner recurrence without the "correction factors" appearing in [7, Thm. 4.4].

Remark 4.5 Regularity. For some applications (such as Chabauty-Coleman) one is only interested in invariant differentials on the Jacobian, or, equivalently, regular 1-forms on the curve, as these are the forms whose integrals define abelian integrals and provide logarithm-type maps to $\mathbf{Q}_q$. In [21, Prop. 2] it is shown that of the $2g$ basis differentials $x^i\,\mathrm{d}x/y^j$, as above, the g regular 1-forms are those for which

$$i < \left\lceil \frac{b}{a} \right\rceil j - 1 - \left\lfloor j\left(\left\lceil \frac{b}{a} \right\rceil - \frac{b}{a}\right)\right\rfloor.$$

It is interesting to note that the algorithm developed here computes the integral of all $2g$ basis differentials simultaneously and there seems to be no way of computing integrals of only a subset of them via this approach.

5. Runtime and precision

We now analyse the runtime of the above algorithm, proving Theorem 1.1. We only analyse the steps which differ from that occurring in [20, Sec. 6]. All preparatory steps remain the same. It is only the main reduction steps where we now have larger matrices to compute the evaluations of exact forms.

We may apply [7, Thm. 5.2] to this as the reduction matrices above are $(m+n)\times(m+n)$ matrices with $m\times n$ block which is zero and a $n\times n$ block which is diagonal. This uses

$$O\left((\mathrm{MM}(m)+\mathrm{MM}(m,n))\sqrt{K} + \left(m^2+mn\right)\mathrm{M}(\sqrt{K})\right)$$

ring operations. We apply this for each row of horizontal reductions (with $K = O(pN)$ in the worst case) and for the vertical reductions also (where $K = O(pN)$ again).

The precision loss and gain throughout the algorithm is as in [7, Sec. 7] and [20, Sec. 5.3]. We can therefore compute (3.4) for $Q = \infty$ and L distinct Teichmüller points P via the above reduction procedure to precision N in time

$$\widetilde{O}\left((g^\omega + Lg^{\omega-1})\sqrt{p}nN^{5/2} + N^4g^3(g+L)n^2\log p\right)$$

where $\widetilde{O}(t) := O(tf(\log t))$ for some polynomial f. The precision loss from inverting $1 - \prod_{s=n-1,\dots,0}\phi^s(M)$ is precisely

$$v_p\left(\det\left(1 - \prod_{s=n-1,\dots,0}\phi^s(M)\right)\right) = \#\operatorname{Jac} C_{\mathbf{F}_{p^n}}(\mathbf{F}_{p^n}).$$

Specifying the above to $L = 2$ gives Theorem 1.1.

6. Implementation

We have implemented the algorithm outlined above in Julia using the functionality provided by the Nemo package [12] and its extension Hecke. This implementation is available online at `https://github.com/alexjbest/Coleman.jl`.

In this section we describe the implementation and discuss some surrounding issues. To check correctness of the algorithm and implementation, we begin with some examples:

6.1. **Examples.**

6.1.1. *A Picard curve.* The curve

$$C\colon y^3 = h(x) = x^4 + 7x^3 + 3x^2 - x$$

was suggested to me by Hashimoto-Morrison [17], it has an algebraic 9-torsion point

$$P = (1, \sqrt[3]{10}),$$

so we take $p = 41$, which splits inside the ring of integers of $\mathbf{Q}(\sqrt[3]{10})$ as a product of a prime of norm 41 and one of norm 41^2, using the latter embedding we can take $P \in C(\mathbf{Q}_{41^2})$. Explicitly if $\mathbf{Q}_{41^2} = \mathbf{Q}_{41}[\alpha]/(\alpha^2 + 38\alpha + 6)$ we have

$$\begin{aligned}P = (1 + O(41^6), &(14 + 30\cdot 41^1 + 19\cdot 41^2 + 24\cdot 41^3 + 35\cdot 41^5)\cdot\alpha\\ &+ 11 + 20\cdot 41^1 + 33\cdot 41^2 + 23\cdot 41^3 + 32\cdot 41^4 + 34\cdot 41^5 + O(41^6)).\end{aligned}$$

As the x-coordinate of this point is 1, it is actually a Teichmüller point (fixed under ϕ), thus we do not have to do any tiny integrals within a disk to compute the Coleman integrals

$$\int_P^\infty \omega$$

for all ω in the basis. Using our package and running the command

```
julia> ColemanIntegrals(3, h, 3, 41, 2, P, :inf)
```

(where respectively the arguments are, degree of the cover, x-polynomial, precision requested, base prime, extension degree, and endpoints) this returns:

$$\begin{pmatrix} 0 \\ (3 + 21\cdot 41^1 + 32\cdot 41^2 + O(41^3))\cdot a + (15 + 37\cdot 41^1 + 37\cdot 41^2 + O(41^3)) \\ (32 + 18\cdot 41^1 + 25\cdot 41^2 + O(41^3))\cdot a + (37 + 10\cdot 41^1 + 9\cdot 41^2 + O(41^3)) \\ 0 \\ 0 \\ (27 + 10\cdot 41^1 + 21\cdot 41^2 + O(41^3))\cdot a + (30 + 20\cdot 41^1 + 7\cdot 41^2 + O(41^3)) \end{pmatrix}.$$

This exactly reflects the fact that $[P - \infty]$ is torsion in the Jacobian and that only the invariant differentials provide group homomorphisms from the

Jacobian to $\mathbf{Q}_{41^2}$ (forcing the images of torsion points to be trivial). Note that the basis of differentials is ordered by j and then i here and that the regular differentials are as described in Remark 4.5.

We can also check Galois equivariance for this example, by computing that

$$\left(\int_{P^\sigma}^{\infty} \omega_i\right)_i = \left(\left(\int_P^{\infty} \omega_i\right)^\sigma\right)_i$$

as both evaluate to

$$\begin{pmatrix} 0 \\ (38 + 19 \cdot 41^1 + 8 \cdot 41^2 + O(41^3)) \cdot a + (24 + 15 \cdot 41^1 + 32 \cdot 41^2 + O(41^3)) \\ (9 + 22 \cdot 41^1 + 15 \cdot 41^2 + O(41^3)) \cdot a + (10 + 35 \cdot 41^1 + 25 \cdot 41^2 + O(41^3)) \\ 0 \\ 0 \\ (14 + 30 \cdot 41^1 + 19 \cdot 41^2 + O(41^3)) \cdot a + (29 + 25 \cdot 41^1 + 19 \cdot 41^2 + O(41^3)) \end{pmatrix}.$$

6.1.2. *An elliptic curve over a quartic field.* The elliptic curve with LMFDB label 4.4.725.1-16.1-a1 is defined over the quartic number field $K = \mathbf{Q}(\alpha)$ where α is a root of $x^4 - x^3 - 3x^2 + x + 1$. This curve can be given by the model

$$y^2 = x^3 + (2\alpha^3 + 6\alpha^2 - 9\alpha - 4)x^2 + (32\alpha^2 - 8)x + (16\alpha^3 + 48\alpha^2 - 16\alpha - 16).$$

On this model there is a K-rational 17-torsion point with coordinates

$$P = (0, 4\alpha^2).$$

We work with $p = 43$, which remains inert in K, and fix an embedding

$$K \hookrightarrow \mathbf{Q}_{43^4}.$$

We can then compute the pair of Coleman integrals

$$\int_P^{\infty} \frac{\mathrm{d}x}{y}, \int_P^{\infty} x\frac{\mathrm{d}x}{y}$$

as before, obtaining the values 0 and

$$\begin{aligned} &(14 + 31 \cdot 43^1 + 29 \cdot 43^2 + 32 \cdot 43^3 + O(43^4)) \cdot a^3 \\ &+ (41 + 5 \cdot 43^1 + 15 \cdot 43^2 + 12 \cdot 43^3 + O(43^4)) \cdot a^2 \\ &+ (11 + 17 \cdot 43^1 + 30 \cdot 43^2 + 11 \cdot 43^3 + O(43^4)) \cdot a \\ &+ (26 + 12 \cdot 43^1 + 28 \cdot 43^2 + 28 \cdot 43^3 + O(43^4)), \end{aligned}$$

respectively.

6.2. **Implementation details.** Minzlaff has made available an open-source implementation of the algorithm in [20]. This includes an implementation of the algorithm of Bostan-Gaudry-Schost and Harvey in the Magma [10] programming language, and computes zeta functions of superelliptic curves over $\mathbf{F}_q$ in $\sqrt{q}$ time. This implementation is included in recent versions of Magma (as `ZetaFunction`) and is separately available online at https://github.com/mminzlaff/superelliptic. Our implementation is built on top of a direct translation of Minzlaff's code into Nemo/Julia.

Remark 6.1 There are two (related) ways to set up a Coleman integration algorithm: integrating to Teichmüller points, or adding the tiny integrals to Frobenius. When using the former, checking that additivity in endpoints holds does not test the implementation for bugs in any serious way as when broken down, the "paths" integrated along form a tree. However with the

latter approach, checking additivity in endpoints of the form

$$\int_P^\infty \omega + \int_{P'}^P \omega = \int_{P'}^\infty \omega$$

for P, P' in the same residue disk, checks that the local and global integrals are consistent, i.e. there do exist "non-degenerate triangles" here.

6.2.1. *A comparison of computer algebra systems.* In this section, we take the opportunity to compare and contrast programming languages available for working with computational number theory and arithmetic geometry. We discuss some similarities and differences between these computer algebra systems and advantages and disadvantages in the author's experience between the way they behave.

Magma and Julia are similar in several syntactic aspects which made translation from the former to the latter easier. Both operate in a functional way, i.e. `factor(M)` rather than `M.factor()`. This syntax is more similar to the syntax used in mathematics. However it presents difficulties for introspection, that is, it is not always easy to find a function performing a particular task, given only a description of that task or its common name. In the Python-based SageMath, given an object `M`, the user can interactively type `M.` and press <TAB> to see a list of all methods associated to the object. It is a common paradigm for a user to search this list to find functions. In languages such as Julia and Magma where the dispatch system makes such functionality difficult, it is important that good searchable documentation exists to help users find the functions they need. For example, a function with a common name like `Normalise` when called as `Normalise;` will show a list of several possibilities of types of objects a Normalise function exists for. The user must then search this list to find the description and signature for the one they wish to call. The Sage user would instead take their object X and type `X.norm`<TAB> to see if a normalise function exists.

Nemo is a new package in comparison to SageMath and Magma, and so some common general-purpose functionality that would be expected from a general computer algebra system is yet to be implemented. At present this can slow down development of mathematical code using Nemo. However Nemo is built on top of the `C` libraries Flint, Arb, and Antic, which have been under development for far longer and contain significant functionality. Missing features often simply need to be wrapped from these libraries. Wrapping involves adding only 3-5 lines of code on average as Nemo is tightly coupled to these underlying libraries. It appears that interfacing with external `C` libraries is simpler when using Julia than in SageMath and Magma.

Magma and SageMath are a lot more stable at present due to the maturity of these systems. This makes continuous integration and testing of code based on Nemo more vital, to prevent code from diverging when the Nemo core is modified in an incompatible way.

The open-source development model and Julia's built-in package manager also mitigates the immaturity problem somewhat, as it is easy to create a public fork of Nemo or to create a package based on it. For instance, at the time of writing there are at least 30 distinct Julia-based repositories on GitHub that use Nemo in some way, and many hundreds of independent Julia packages, some of which also containing functionality relevant to arithmetic geometers. This iterative and modular way of sharing research code is ingrained in the design of Julia and has also been proposed as a more lightweight development model for additional packages on top of the SageMath core https://wiki.sagemath.org/CodeSharingWorkflow. The advantages over more centralised development are a lower barrier for research code to be made available. It is trivial for

anyone to host code in a version-controlled repository and create the requisite Julia package files, so that the code may then be installed using the built-in package system. With a more modular system, the notions of ownership and responsibility for individual packages are more clear, users can report bugs and feature requests directly to the developers of smaller modules, rather than all traffic going to a centralised bug tracker, or worse, a closed email list with no public tracking of bugs at all.

For a working mathematician, writing high-level code that is as close as possible to mathematical language is preferable, for speed of development, sharing of research code and identification of bugs. In general, the overhead of using a high-level language can render many potentially practical algorithms too slow for serious use on large scale. For instance, algorithms such as the accumulating remainder tree of [11, 15] can lose their practicality when a large call overhead (such as that in Python) is introduced at each recursive step. This necessitates the use of low-level languages such as `C` to implement key core functionality, like fundamental arithmetic operations in different rings, operations with polynomials, and linear algebra. When converting from high-level code to low-level, in general more code must be added, to explicitly construct and destruct objects, manage memory, iterate over lists, and handle files. This can often obscure the mathematical process underlying a given algorithm and obstruct understanding and later generalisation.

In SageMath, Cython is used to mitigate such overhead problems somewhat, and to wrap low-level *C* libraries. Its Python-like syntax makes conversion easier and makes comprehension of code easier, and it does reduce the overhead of Python/Sage code.

Using domain-specific Julia packages to write high-level code, which is compiled at runtime to a fast lower-level implementation can provide a useful balance for a mathematical user. While there are applications where low-level control is essential: for instance in [9, Sec. 3.2] working at the machine level is necessary to obtain results on such a large scale in as small amount of time (and also therefore cost) as possible. Using a high-level language that is compiled (at runtime or before) into a lower-level language can strike a good balance between speed of the computation and time taken to write the code.

The distinction between compiling at runtime vs. before the software is run is most apparent when experimenting with new code. Anything that can be done at runtime can of course be done before, but the user has more flexibility when they do not have to exit the CAS and recompile to take advantage of compiled code.

6.3. **Timings.** We provide here some timings of both the Magma implementation of Minzlaff and Julia/Nemo implementation of the underlying zeta function algorithm, which is the only part common to both implementations, and this uses the same algorithm due to Minzlaff. Hence the comparison below is really a test of how the underlying systems handle the operations used by this algorithm (of course further optimizations to both implementations may well be possible). We do not time the Coleman integration code here as the linear recurrence method that underlies the zeta function algorithm is the main component of the runtime for computing Coleman integrals also.

The dash indicates a parameter range where neither implementation applies due to the standing assumption on p in (2.1). All times are measured in seconds.

Table 6.2 Timings for the Magma implementation

$q \setminus (a,b)$	$(2,5)$	$(2,7)$	$(3,7)$	$(4,7)$
257	0.180	0.647	4.357	10.140
521	0.253	0.963	4.373	10.260
1031	0.290	1.017	6.860	16.543
2053	0.413	1.673	7.080	16.680
4099	0.487	0.960	7.473	21.530
8209	0.750	1.440	8.407	21.767
257^2	0.700	2.743	21.643	–
521^2	1.167	4.060	30.500	78.850

Table 6.3 Timings for the Nemo implementation

$q \setminus (a,b)$	$(2,5)$	$(2,7)$	$(3,7)$	$(4,7)$
257	0.197	0.651	4.695	15.267
521	0.366	1.27	4.779	17.892
1031	0.415	1.348	13.287	43.691
2053	0.973	3.727	14.135	44.516
4099	1.133	2.332	30.306	114.239
8209	3.261	6.486	36.414	114.61
257^2	0.339	1.487	23.569	–
521^2	0.803	2.992	44.461	111.966

References

[1] Balakrishnan, Jennifer S., Robert W. Bradshaw, and Kiran S. Kedlaya. *Explicit Coleman Integration for Hyperelliptic Curves*. In ANTS-IX 2010, LNCS 6197, pp. 16-31, 2010.

[2] Balakrishnan, Jennifer S. *Coleman integration for even-degree models of hyperelliptic curves*. LMS Journal of Computation and Mathematics 18.1 (2015): 258-265.

[3] Balakrishnan, Jennifer S., Netan Dogra, J. Steffen Müller, Jan Tuitman, and Jan Vonk. *Explicit Chabauty—Kim for the Split Cartan Modular Curve of Level 13*. Annals of Mathematics 189, no. 3 (2019): 885–944. https://www.jstor.org/stable/10.4007/annals.2019.189.3.6.

[4] Balakrishnan Jennifer S., and Jan Tuitman. *Explicit Coleman integration for curves*. arXiv preprint arXiv:1710.01673. 2020 Jan 13.

[5] Besser, Amnon. *Heidelberg lectures on Coleman integration*. In *The Arithmetic of Fundamental Groups*, pp. 3-52. Springer, Berlin, Heidelberg, 2012.

[6] Besser, Amnon, and Rob De Jeu. *$Li^{(p)}$-Service? An Algorithm for Computing p-Adic Polylogarithms*. Mathematics of Computation 77, no. 262 (2008): 1105–34.

[7] Best, Alex J. *Explicit Coleman integration in larger characteristic*. Proceedings of the Thirteenth Algorithmic Number Theory Symposium, The Open Book Series, 2(1), 85-102, 2019.

[8] Bezanson, Jeff, Alan Edelman, Stefan Karpinski, and Viral B. Shah. *Julia: A Fresh Approach to Numerical Computing*. (2017) SIAM Review, 59: 65-98. doi: 10.1137/141000671. https://julialang.org/research/julia-fresh-approach-BEKS.pdf.

[9] Booker, Andrew R., Jeroen Sijsling, Andrew V. Sutherland, John Voight, and Dan Yasaki. *A Database of Genus 2 Curves over the Rational Numbers*. LMS Journal of Computation and Mathematics 19, no. A (2016): 235–54. https://doi.org/10.1112/S146115701600019X.

[10] Bosma, Wieb, John Cannon, and Catherine Playoust. *The Magma algebra system. I. The user language*, J. Symbolic Comput., 24 (1997), 235–265.

[11] Costa, Edgar, Robert Gerbicz, and David Harvey. *A Search for Wilson Primes*. Mathematics of Computation 83, no. 290 (2014): 3071–91. https://doi.org/10.1090/S0025-5718-2014-02800-7.

[12] Fieker, Claus, William Hart, Tommy Hofmann, and Fredrik Johansson. *Nemo/Hecke: Computer Algebra and Number Theory Packages for the Julia Programming Language*. In: *Proceedings of ISSAC '17*, pages 157-164, New York, NY, USA, 2017. ACM, http://doi.acm.org/10.1145/3087604.3087611.

[13] Gaudry, Pierrick, and Nicolas Gürel. *An extension of Kedlaya's point-counting algorithm to superelliptic curves*. Advances in Cryptology - ASIACRYPT 2001, Springer, Berlin, Heidelberg, 2001.

[14] Gonçalves, Cécile. *A point counting algorithm for cyclic covers of the projective line*. Contemporary mathematics 637 (2015): 145.
[15] Harvey, David. *Counting points on hyperelliptic curves in average polynomial time*. Annals of Mathematics 179, no. 2 (2014): 783-803.
[16] Harvey, David. *Kedlaya's Algorithm in Larger Characteristic*. IMRN: International Mathematics Research Notices 2007 (2007).
[17] Hashimoto, Sachi, and Travis Morrison. *Chabauty-Coleman Computations on Rank 1 Picard Curves*, preprint.
[18] Kedlaya, Kiran S. *Counting points on hyperelliptic curves using Monsky-Washnitzer cohomology*, J. Ramanujan Math. Soc. 16 (2001), no. 4, 323-338; errata, ibid. 18 (2003), 417--418.
[19] McCallum, William, and Bjorn Poonen. *The Method of Chabauty and Coleman*. Explicit Methods in Number Theory 36 (2012): 99–117.
[20] Minzlaff, Moritz. *Computing zeta functions of superelliptic curves in larger characteristic*. Mathematics in Computer Science 3.2 (2010): 209-224.
[21] Towse, Christopher. *Weierstrass Points on Cyclic Covers of the Projective Line*. Transactions of the American Mathematical Society 348, no. 8 (1996): 3355–3378.
[22] Tuitman, Jan. *Counting points on curves using a map to* $\mathbf{P}^1$. Mathematics of Computation 85.298 (2016): 961-981.
[23] Tuitman, Jan. *Counting points on curves using a map to* $\mathbf{P}^1$*, II*. Finite Fields and Their Applications 45 (2017): 301-322.

A. J. Best, Department of Mathematics & Statistics, Boston University, 111 Cummington Mall, Boston, MA 02215, USA
Email address: alex.j.best@gmail.com