Explicit computation of the Abel-Jacobi map and its inverse

Hugo Labrande

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Explicit computation of the Abel-Jacobi map and its inverse

THÈSE

présentée et soutenue publiquement le 14 novembre 2016

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention Informatique Fondamentale)

par

Hugo Labrande

Composition du jury

Rapporteurs :	John Boxall	Professeur, Université de Caen
	Guillaume Hanrot	Professeur, ÉNS Lyon
Examinateurs :	Pierrick Gaudry	Directeur de recherche, CNRS, Nancy
	Renate Scheidler	Professeur, University of Calgary
Encadrants :	Emmanuel Thomé	Directeur de recherche, INRIA Nancy
	Michael J. Jacobson, Jr.	Professeur, University of Calgary

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503
University of Calgary

Mis en page avec la classe thesul.

To Becky, with love

Remerciements

À la fin of a three year cotutelle, there are beaucoup de people to thank, en français and in English. J'hope ne pas forget anybody, mais tellement m'ont entouré and supported during these 3 ans de thèse that I'm bound to have oublié certains. Sorry!

Je tiens à remercier en premier lieu John Boxall et Guillaume Hanrot pour avoir accepté de relire ce manuscrit, et pour leurs commentaires, qui m'ont grandement aidé à finaliser ce manuscrit. I would also like to thank Matthew Greenberg and Wayne Eberly for accepting to be a part of the examination committee for my candidacy exam in Calgary, as well as Faramarz Famil Samavati for sitting as the neutral chair. Je souhaite également remercier Sylvain Lazard pour m'avoir suivi en tant que référent interne au cours de ces trois dernières années. Je souhaite enfin remercier Pierrick Gaudry d'avoir accepté d'être examinateur à la soutenance de thèse, ainsi qu'à l'examen de candidature ; mais aussi pour son aide et son expertise sur certaines parties ardues, comme la lecture du deuxième livre de Mumford. I also wish to thank Renate Scheidler, who kindly accepted to be part of the defense jury, as well as the jury for my candidacy exam in Calgary.

I also wish to thank my advisors for their guidance. Many thanks to Mike, for his advice, his help and mentorship, during this project and the previous ones, and helping to make me almost a fully-grown researcher over the last six years. Un grand merci à Emmanuel, pour m'avoir proposé ce sujet de thèse, pour s'être toujours montré disponible et patient, pour ses réponses pointues à mes questions fréquentes, pour son aide sur tous les aspects, aussi bien théoriques que d'implémentation que pratico-pratiques, ainsi que de m'avoir dit quand j'écrivais trop, ou trop de bêtises.

Je voudrais également remercier tous ceux qui se sont montrés intéressés par mes travaux de thèse et m'ont permis de les présenter tout au long de ces trois dernières années. I would like to thank the Number Theory Group in Calgary, and more particularly those of the Number Nosh, who have been very welcoming and allowed me to give practice talks there a few times. Enfin, un grand merci à Christophe Ritzenthaler, et à Andreas Enge et Damien Robert, de m'avoir invité dans leurs départements respectifs pour y présenter mes travaux, ainsi que pour les discussions qui s'en suivirent, toujours très stimulantes et dont les fruits se retrouvent dans ce manuscrit.

A cotutelle is a lot of administrative work, and I cannot forget to thank everyone who was involved in making this cotutelle possible. Merci à Vanessa Binet, ainsi que le reste du bureau des Études Doctorales de l'UL, pour leur aide. Thank you to the Graduate Studies at the Computer Science department of the University of Calgary, and many thanks to Britta Travis for answering so many of my questions. Un grand merci à Sophie Drouot pour son aide, notamment pour la planification de mes (nombreux !) voyages. Je souhaite également remercier Suzanne Collin pour m'avoir permis de faire mes heures d'enseignement à Télécom Nancy, tout en composant avec le fait que j'étais absent la moitié de l'année ; finally, I would like to thank the Computer Science department at the UofC for providing me with the opportunity to be a teaching assistant there.

Un grand merci également à tous mes collègues, des deux côtés de l'Atlantique. Thanks to Sebastian for brightening up the office and studying the theory with me. Mille mercis à Marie-Andrée, l'autre moitié du Bureau des Isogénies, pour sa bonne camaraderie. Merci à Aurélien pour ses conseils et ses meubles ; merci également à Hubert et Éric pour ces formations doctorales mémorables. Merci à Jérémie, ainsi qu'à Rémi, pour les discussions occasionnelles entre lyonnais. Merci à Aurore, et nos longs échanges de mails pendant l'hiver canadien. Et bien entendu, un grand merci à toute l'équipe Caramel/Caramba, source inépuisable de rires et de trolls, pour l'ambiance de travail exceptionnelle, que j'étais toujours très heureux de retrouver. Merci à Paul, Jérémie, Marion, Pierre-Jean, et Pierrick, pour leur encadrement bienveillant, et leur aide et leur disponibilité pour les "jeunes" ; merci à Alexander, Enea et Maike, et merci à Svyat et Simon

pour les trolls et la bonne humeur. Merci à mes co-bureaux successifs – et il y en eut – Laurent, Hamza, Cyril, Nick, Stéphane et Luc, pour avoir entrecoupé les silences productifs de discussions animées sur tout et n'importe quoi. Un merci tout particulier à Laurent, qui a sait décrypter les règlements de l'Université de Lorraine comme personne, et sans qui les formations doctorales auraient paru beaucoup plus longues.

And of course, thanks to all that is not work and that allowed me to blow off steam throughout this time. Merci à Azathoth, Natrium, Yoruk, Stormi et Otto, pour leur passion et leur ténacité. Merci aux improvisateurs nancéens, et tout particulièrement à ceux de l'association Improdisiaque ; merci à Steff pour son énergie, à Jess pour son amitié et son rire, et à Fanny pour son soutien constant et les soirées saucisson épiques. Merci à Armand pour son amitié, nos discussions profondes comme légères, et pour sa pêche. Thank you to the UofC Improv Club for always making me feel so welcome every time I came back, and for opening me to so many new perspectives; thanks to the Kinkonauts Level B class for the fun times. Merci à Anh Thy et Laure pour toutes ces heures passées au téléphone ; merci à Clément, Olivier et Guillermo pour leur amitié profonde depuis si longtemps.

Many thanks to my in-laws, for making sure I had a big, loud family whenever I was in Calgary; for their support in so many ways for the last 5 years; and for always teaching me something new about Canada or the English language.

Un grand merci à mes parents, pour leur présence constante, leur soutien dans les moments difficiles, leur joie dans tous les bons moments, leurs encouragements et leur amour, depuis le début du début. Merci à Manon, toujours aussi proche, toujours aussi complice, toujours aussi présente, même avec la distance, que ce soit pour raconter des bêtises ou parler longuement.

And of course, thank you to my wife, for her unconditional love and support throughout the last three years; who stood by me always; who told me to go for it even if she still wonders what the heck is a theta-constant; who was always patient and understanding; who knows exactly when I need to stop working and go for a hot chocolate or a Blizzard; who gives me so much every day and whom I dedicate this manuscript to. Et merci à la petite Anaïs de m'avoir laissé dormir un peu pour que je finisse d'écrire ce manuscrit. Love you both.

Contents

List of Algorithms

Introduction

Elliptic and hyperelliptic curves are classical objects in algebraic geometry, and their properties have been studied for centuries. They have also been proposed for use in cryptography more than thirty years ago, which along with the advent of computers rekindled the interest in effective algorithms to compute objects and perform calculations related to these curves. In particular, the computation of isogenies, which are morphisms between curves, has cryptographic applications: isogenies transport the discrete logarithm problem on a curve (on which the security of the cryptosystem depends) into an instance of the problem on another, potentially weaker, curve. Hence isogeny computation has been used to outline a decrease of the security of a cryptosystem based on curves; however, these applications only concern very specific cases so far, and elliptic curves are still considered to be secure, and are widely deployed.

Elliptic and hyperelliptic curves over the complex numbers have an "analytic representation" as complex tori, in addition to the usual algebraic representation. Analytic representations are particularly interesting, in the sense that computing isogenies (and other maps) between two such representations is as simple as multiplying two complex numbers. The translation between both representations is done via the *Abel-Jacobi map*, an explicit isomorphism from the algebraic to the analytic representation. In genus 1, this map is fairly well-known, and the links with the arithmetico-geometric mean are explicit; its inverse is linked to the Weierstrass $\wp$ function, which is also well-known. However, algorithms to compute the inverse of the Abel-Jacobi map are currently more costly than the ones which compute the Abel-Jacobi map, which is not satisfactory. Furthermore, the situation with respect to the Abel-Jacobi map in higher genus is not as explicit, and there are no fast algorithms to compute the map and its inverse.

In this thesis, we look at the θ function, a function of complex numbers which can be defined in any genus. The θ function can be linked to complex Riemann varieties, including elliptic and hyperelliptic curves over the complex numbers, and can be used to compute the Abel-Jacobi map. Furthermore, it has links with other complex functions of number theory, including the Weierstrass $\wp$ function.

We took a closer look at the computation of this function with arbitrary precision: the main contribution of this manuscript is to outline, in genus 1 and genus 2, fast algorithms which compute the value of this function with complexity roughly linear in the precision needed. A similar approach could be applied to the general genus g case, but this requires solving a few problems first, which we solved in genus 1 and 2 but leave as future work for higher genus. As a result, our work also gives fast algorithms for the computation of the Abel-Jacobi map and its inverse, which is also of general interest. The algorithms for $\theta, \wp$ and the Abel-Jacobi map that we present in this manuscript have the best known asymptotic complexity. Finally we also study one application of this: a new algorithm to compute isogenies of a given kernel over $\mathbb{C}$, $\mathbb{F}_p$ or a number field.

1 Contributions of this thesis

Main contributions

The main contributions of this thesis lie in the design and study of algorithms to compute theta functions in quasi-linear time.

Generalizing algorithms studied in [Dup06], we first give an algorithm to compute the value of $\theta(z,\tau)$ (Jacobi's theta function) with precision P in $O(\mathcal{M}(P)\log P)$, an improvement over the $O(\mathcal{M}(P)\sqrt{P})$ running time of previous algorithms. Our implementation of the algorithm is publicly available, and shows this algorithm is faster than previous algorithms for precisions greater than 260 000 decimal digits. We present this in Chapter 6.

We then generalized this approach to higher genera, as presented in Chapter 7. We managed to obtain an algorithm to compute genus 2 theta functions in the same $O(\mathcal{M}(P)\log P)$ time, although the invertibility of the Jacobian (which arises when using Newton's method) remains to be proven; this is an improvement over previous algorithms, of complexity $O(\mathcal{M}(P)P)$. Once again, our implementation of the algorithm is publicly available and shows a speedup with respect to the naive algorithm for precisions greater than 3 000 digits. Furthermore, we studied the generalization of the algorithm in genus g; a similar complexity seems achievable, but this requires solving some problems which arise when attempting to use Newton's method, and which were more or less solved in genus 2.

We then applied the algorithms and ideas of these chapters to the computation of the Abel-Jacobi map and its inverse in Chapter 8. The fast algorithms for the computation of θ allowed us to show that both the complex Abel-Jacobi map and its inverse can be computed with precision P in $O(\mathcal{M}(P)\log P)$ in genus 1 and 2, provided the Jacobian of the system is invertible. In genus g, the same result for the Abel-Jacobi map is achievable, but our algorithm for the fast computation of its inverse hinges on the fast computation of θ in genus g, which is not fully solved yet.

Finally, as an application of the fast computation of the genus 1 Abel-Jacobi map and its inverse, we studied an isogeny computation algorithm in Chapter 9. The algorithm allows one to compute an isogeny with given kernel over $\mathbb{C}$, a number field, or a finite field. Its complexity is worse than that of Vélu's formulas, but it seems easier to generalize to genus g; however, we left the study of this generalization to future work.

Other contributions

We wish to highlight a few results we obtained in this manuscript; some of them may be of secondary importance, but we think they may be of independent interest. As these results are scattered throughout this manuscript, we summarize them here. The list includes:

- We give a new algorithm to compute $\wp(z,\tau)$ with precision P in time $O(\mathcal{M}(P)\log P)$. The algorithm uses the link between values of $\wp$ and the Landen transform, which gives a recurrence relation; it is described in Section 4.3. We also compare this algorithm to the one deriving from the well-known link between $\wp$ and θ, which is also of quasi-linear complexity using our algorithms for θ; the comparison in Section 8.1.3 shows that the new algorithm has a smaller constant and is faster in practice.

- We give a fast algorithm to compute $E_{2k}(\tau)$ with precision P in $O(P^{1+\epsilon}k^{1+\epsilon})$ bit operations; this is a better asymptotic complexity than the naive algorithm, which has complexity $O((P+k)^{2+\epsilon}\log k)$. Furthermore, our algorithm actually computes $E_{2k'}(\tau)$ for all $k' \leq k$ in the same asymptotic running time. We refer to Section 8.4 for a more detailed analysis.

- Our algorithm for the Abel-Jacobi map in genus g can be applied to the genus 1 setting (see Section 8.1.4), which gives a new quasi-linear algorithm to compute the elliptic logarithm using the link between $\wp$ and θ. The resulting algorithm is only twice slower than the more direct algorithms based on the Landen transform.
- We show in Chapter 5 that using recurrence relations in the naive algorithm to compute θ can be used in any genus, and optimize the genus 1 and genus 2 naive algorithms to lower the asymptotic constant.
- We improve the uniform algorithm for genus 1 theta-constants given in [Dup06] and include the computation of θ_2 in the same uniform complexity (see Section 6.1.2).
- Finally, we study a global torsion lifting procedure in Section 9.3. In particular, we study a univariate polynomial derived from the generic ℓ-torsion polynomials; we manage to give a bound on the module of its roots, but its irreducibility is left as an open problem.

2 Outline of the manuscript

This manuscript is structured in nine chapters.

Background

Chapter 1 deals with background concerning elliptic curves and isogenies, the primary motivation for our study of the computation of the Abel-Jacobi map; we also discuss this map and its potential applications, and show how these notions generalize to genus 2.

Chapter 2 establishes background on general (i.e. genus g) theta functions, outlining the ideas and the formulas that we use throughout the manuscript. We also mention two explicit reduction algorithms which seem relevant in the context of genus g θ functions, although they are weaker than the reduction to the genus g fundamental domain, for which there are no explicit reduction algorithms. Section 2.5 and Section 2.6 show all the formulas we use in the context of genus 1 and genus 2 theta functions, which is handy when reading the more involved chapters.

Computation of the Abel-Jacobi map

Chapter 3 deals with the arithmetico-geometric mean (AGM), an important object in this manuscript since it can be computed in quasi-linear time. We outline the well-known connection with the theta-constants; the connection with elliptic integrals is discussed in the next chapter. We also discuss a nice generalization of the AGM, the Borchardt mean.

Chapter 4 discusses the Landen isogeny, which shows the connection between the AGM and the elliptic integrals, and allows one to compute the periods of an elliptic curve and the Abel-Jacobi map in quasi-linear time. The full proof for general complex elliptic curves has been obtained rather recently, and we recall the notions involved in this proof. We also discuss in that chapter an original algorithm to compute the Weierstrass $\wp$ function in quasi-linear time using the Landen isogeny; finally we discuss similar ideas for the θ function.

Theta functions

Chapters 5, 6 and 7 deal with algorithms to compute the θ function; they form the core of this manuscript. Chapter 5 looks at the algorithm consisting in evaluating the sum defining θ, with enough terms so that the result is accurate to the desired precision; we show how one can use

recurrence relations to evaluate the terms efficiently, and fully evaluate the complexity of the algorithm in any genus, although we were not able to determine the dependency in τ in the general genus g case.

Chapter 6 outlines an original algorithm which computes the genus 1 theta function in quasi-linear time; we use a similar strategy to the theta-constants: find a function $\mathfrak{F}$ which can be evaluated in quasi-linear time and takes a special value at the theta functions (using a sequence inspired by the AGM), then invert this function using Newton's method. We provide a full analysis of the running time of this algorithm and of the precision loss that is incurred during the computation, and give an algorithm with complexity uniform in z and τ. We implemented this algorithm in low-level GNU MPC and show that this algorithm is faster than the naive algorithm described in Chapter 5 for precisions greater than 100 000 decimal digits. The results in this chapter have been accepted for publication in the journal *Mathematics of Computation* in November 2015 [Lab15].

Chapter 7 shows how the quasi-linear time algorithm can be generalized to theta functions of higher genera. The results in Chapter 7 were obtained with Emmanuel Thomé and written up in a paper [LT16] which was accepted to the 2016 *Algorithmic Number Theory Symposium* (ANTS-XII). We outline explicitly the algorithm in genus 2: we define $\mathfrak{F}$, the function to invert, explictly and we prove that it can be evaluated in quasi-optimal time; we also solve a tricky issue, which is that Newton's method cannot be applied directly to this function. The resulting algorithm was implemented in genus 2 in Magma, and it runs faster than the naive algorithm for precisions larger than 3000 decimal digits, which is much less than in genus 1; note that this algorithm relies on a conjecture, which is that the Jacobian of $\mathfrak{F}$ is invertible. In the general, genus g case, we show the construction of $\mathfrak{F}$, whose evaluation has conjectured quasi-optimal running time. However, a similar problem with Newton's method arises, and we were not able to show that one can solve it in the general case.

Abel-Jacobi map

Chapter 8 deals with the computation of the Abel-Jacobi map, and the links between this map and theta functions. We obtained results of different strength depending on the genus.

In genus 1, known AGM-based methods allow the computation of the Abel-Jacobi map in quasi-linear time; we propose an algorithm using the function $\mathfrak{F}$ of Chapter 6, but it is twice slower than these methods. The inverse of the Abel-Jacobi map is given by Weierstrass's $\wp$ function, which can be expressed as a function of θ to yield a first quasi-linear time algorithm. We compare this algorithm to another original algorithm for $\wp$ based on the Landen isogeny, which we outlined in Chapter 4; implementations show that the latter is two to three times faster than the former.

In genus 2, no methods which compute the Abel-Jacobi map in quasi-linear time in the general complex case seem to exist. Our algorithm using the function $\mathfrak{F}$ can be generalized here, and yields a quasi-linear time algorithm. The computation of the inverse of the Abel-Jacobi map relies once again on the computation of θ, which gives a quasi-linear time algorithm modulo the conjecture on the Jacobian of $\mathfrak{F}$.

In genus g, we can once again use the function $\mathfrak{F}$ of Chapter 7, which can be evaluated in quasi-linear running time; this yields a quasi-linear time algorithm for the Abel-Jacobi map, which is the best known complexity. As for the computation of the inverse of the Abel-Jacobi map, it reduces to the computation of θ in genus g; hence, it depends on being able to find a way to apply Newton's method to $\mathfrak{F}$, which we were not able to achieve.

Isogeny computation

The purpose of the last chapter, Chapter 9, is to outline and study an algorithm which, given a complex elliptic curve and a subgroup of order ℓ, computes an ℓ-isogenous curve such that the subgroup is the kernel of the isogeny. This algorithm uses the Abel-Jacobi map to transpose this problem to the complex tori, where it is easy to solve. The asymptotic complexity of this algorithm is worse than the complexity of existing algorithms (e.g. Vélu formulas); however, it seems like it can be generalized quite nicely to the computation of isogenies between genus 2 curves.

We then show how one can build an algorithm which, given a curve defined over a number field K, finds an isogenous curve with given kernel. This requires embedding the number field into the complex numbers, then recognizing the coefficients of the complex isogeny as elements of the number field; however, we were unable to find an explicit formula giving the precision required to do this. Finally, this strategy can be extended to curves defined over finite fields, which requires lifting the curve to a curve defined over a number field; we propose a conjecture on the precision needed in this case.

3 Computational model

Throughout this manuscript, we will perform computations on complex numbers, and we will always use the same computational model in those cases. We will outline algorithms to compute mathematical quantities with arbitrary precision, i.e. using *multiprecision arithmetic*. Hence, the cost of our arithmetic operations depends on the number of digits, which we always denote P, of the quantities we are working with.

3.1 Precision

We introduce the notion of *precision* as follows:

Definition 0.3.1. Let $\alpha \in \mathbb{R}$; we say that $\hat{\alpha}$ is an approximation of α with *absolute precision* P if

$$|\alpha - \hat{\alpha}| \leq \frac{1}{2^P}.$$

We say that $\hat{\alpha}$ is an approximation of $\alpha \in \mathbb{R}^*$ with *relative precision* P if

$$\left|\frac{\alpha - \hat{\alpha}}{\alpha}\right| \leq \frac{1}{2^P}.$$

Throughout this manuscript, we will consider the problem of computing quantities *with absolute precision* P – that is to say, computing P exact bits after the radix point, or computing the quantity up to 2^{-P}. This choice is largely without practical consequences: most of the quantities we will compute in this thesis have integral part of bounded size, and hence we could have just as easily required a relative precision $P + c$ with c a small constant. The notion of absolute precision for complex numbers can either be transposed as "the norm of the difference is smaller than 2^{-P}" or "the real part and the imaginary part of the approximation are correct up to 2^{-P}"; these notions only differ by a $\sqrt{2}$ factor, which is not very important in most cases.

Because of this choice, we will work using *fixed-point arithmetic* instead of the commonly used floating-point arithmetic. This simply means that the quantities we will work with will be of the form $a \times 2^{-P}$ for precision P, where a is an integer. This choice is consistent with our choice of working with absolute precision, since every number which can be represented in fixed-point arithmetic of absolute precision P is distant from its neighbours by exactly 2^{-P}, whereas the

distance between two consecutive floating-point numbers can be greater than 2^{-P}, which fails to provide an approximation with absolute precision P.

As we mentioned, the quantities we will work on in this manuscript very often have a bounded size. We will frequently make the (sometimes implicit) assumption that we have taken into account the size of their integral part in the complexity; that is to say, that the integer a in the representation of the quantity has $P + c$ bits, or even simply $O(P)$ bits. Should a quantity have integral part of size larger than $O(P)$, we will mention it and adjust the complexity accordingly; we ask the reader to assume that, if nothing is mentioned, the integral part can be coded on $o(P)$ or $O(P)$ bits, which means that the integer a of the fixed-point representation has $O(P)$ bits.

3.2 Loss of precision

We now define the notion of *loss of precision*, which will be discussed extensively throughout this manuscript.

Definition 0.3.2. Three different notions of loss of precision can be defined:

- *mathematical loss of precision*: let $\hat{x}$ be an approximation of $x \in \mathbb{C}$ with absolute precision P, and f a complex function. The mathematical loss of precision is $c \geq 0$ such that $|f(x) - f(\hat{x})| \leq 2^{-P+c}$. This is also called the *forward error* [Hig02].

- *loss of precision induced by the algorithm*: let f be a complex function and $\mathcal{A}$ an algorithm to compute f which works *using exact arithmetic*, i.e. we assume the arithmetic operations in $\mathcal{A}$ always give exact results. The loss of precision induced by the algorithm is $c \geq 0$ such that $|f(x) - \mathcal{A}(x)| \leq 2^{-P+c}$ for all $x \in \mathbb{C}$. This can be rephrased as the quality of the approximation of f provided by $\mathcal{A}$.

- *loss of precision induced by rounding*: let $\hat{x}$ be a P-bit number, $\mathcal{A}$ an algorithm which works *using exact arithmetic* (i.e. using as many digits as needed to represent the result of arithmetic operations), and $\mathcal{A}_P$ the same algorithm but in which arithmetic operations give a result rounded with precision P. The loss of precision induced by rounding is $c \geq 0$ such that $|\mathcal{A}(\hat{x}) - \mathcal{A}_P(\hat{x})| \leq 2^{-P+c}$.

The approach which has been taken throughout this manuscript is as follows: we do not take into account the mathematical loss of precision; all our algorithms induce no loss of precision (i.e. assuming exact arithmetic, they would return a result which is always within 2^{-P} of the value which is sought); the loss of precision induced by rounding is analyzed and compensated for. Hence, our goal is to provide algorithms to compute an approximation of $f(\hat{x})$ which is within 2^{-P} of that value, i.e. the exact value of f evaluated at the argument. Should one require an approximation of $f(x)$ within 2^{-P} of that value – i.e. the correct value with absolute precision P –, they should analyze the mathematical loss of precision (say, c bits) and provide an approximation of x with precision $P + c$.

In the rest of this manuscript, "loss of precision" will be understood as meaning "loss of precision induced by rounding". We will state all our algorithms as taking P-bit numbers as inputs; we analyze the loss of precision induced by rounding throughout the algorithm and bound it in the worst case, to ensure our algorithms always return approximations accurate to 2^{-P} of the results.

Loss of precision induced by rounding

By loss of absolute precision induced by rounding, we mean the error in the final result which is due to the fact that we worked on approximations of the input coded on P bits, and that all the results of intermediate computations in the algorithm were rounded to give P-bit numbers. For instance, dividing a number by 2 then multiplying it by 2 loses one bit of precision, because the intermediate result was rounded; this is inherent to working with P-bit numbers.

We analyze here the loss of precision induced by elementary functions, which we use as a building block of all our algorithms; the following theorem shows how an error on a rounded number is transformed by the computation of a rounded result. The analysis of precision loss in our algorithms is then a matter of combining these results to compute how large the errors can get.

Theorem 0.3.3. *For $j = 1, 2$, let $z_j = x_j + iy_j \in \mathbb{C}$ and $\tilde{z_j} = \tilde{x_j} + i\tilde{y_j}$ its approximation. Suppose that $|z_j - \tilde{z_j}| \leq k_j 2^{-P}$ and that $k_j \leq 2^{P/2}$. Suppose furthermore that $k_j 2^{-P} \leq |z_j| \leq 2^{P/2-3}$. Then*

1. $|\mathrm{Re}(z_1 + z_2) - \mathrm{Re}(\tilde{z_1} + \tilde{z_2})| \leq (k_1 + k_2)2^{-P}$
2. $|\mathrm{Re}(z_1 z_2) - \mathrm{Re}(\tilde{z_1}\tilde{z_2})| \leq (2 + 2k_1|z_2| + 2k_2|z_1|)2^{-P}$
3. $|\mathrm{Re}(z_1^2) - \mathrm{Re}(\tilde{z_1}^2)| \leq (2 + 4k_1|z_1|)2^{-P}$

with the same bounds applying to imaginary parts, and

4. $|e^{z_1} - e^{\tilde{z_1}}| \leq |e^{z_1}|\frac{7k_1+8.5}{2}2^{-P}$.

Furthermore if $|z_j| \geq 2k_j 2^{-P}$,

5. $|\mathrm{Re}\left(\frac{z_1}{z_2}\right) - \mathrm{Re}\left(\frac{\tilde{z_1}}{\tilde{z_2}}\right)| \leq \left(\frac{6(2+2k_1|z_2|+2k_2|z_1|)}{|z_2|^2} + \frac{2(4+8k_2|z_2|)(2|z_1||z_2|+1)+2}{|z_2|^4}\right)2^{-P}$

and the same bound applies to the imaginary part, and

6. $|\sqrt{z_1} - \sqrt{\tilde{z_1}}| \leq \frac{k_1}{\sqrt{|z_1|}}2^{-P}$.

Proof. See Appendix A for a proof, inspired from the techniques in [ETZ]. □

We will make use of this theorem to compute the amount of precision we lost during the calculations.

Remark 0.3.4. Note that not every operation creates a loss of absolute precision. For instance, the multiplication by a number much smaller than 1 can create a result which is accurate to the full precision from numbers that were not; the same goes for the computation of the square root of a large number.

Guard bits

In order to compensate for precision loss, we use the notion of *guard bits*:

Definition 0.3.5 (guard bits)**.** Let $\mathcal{A}$ be an algorithm on complex numbers with exact arithmetic, and denote by $\mathcal{A}_P$ the same algorithm which operates on numbers with P bits of absolute precision (i.e. in which inputs are of precision P and all the intermediate quantities are rounded off to precision P). Let $\hat{x}$ be a complex number with P bits after the radix point. We say that *a computation is performed with C guard bits* to denote the following process:

- Put $\hat{\hat{x}}$ the number $\hat{x}$ (with P bits after the radix point) followed by $C+1$ zeros, so that $\hat{\hat{x}}$ is a number with $P+C+1$ bits after the radix point and an approximation of $\hat{x}$ with precision $P+C+1$;
- Compute $\alpha_{P+C} = \mathcal{A}_{P+C}(\hat{\hat{x}})$ with precision $P+C+1$;
- Round off the result with precision P to get a number α_P.

If the computation of f loses C bits, then $|\mathcal{A}(\hat{x}) - \alpha_P| \leq 2^{-P}$.

This means that losses of precision throughout the computation can be compensated, and the final result is then an approximation of the result with absolute precision P. However, this means the precision at which one works increases, which can have an impact on the asymptotic cost of the algorithm; analyzing this and finding ways to reduce the precision losses becomes important in some algorithms.

3.3 Cost of computations

Working with fixed-point arithmetic representations of the form $a \times 2^{-P}$, with a an integer, means that the operations on these representations are essentially reduced to operations on integers. For instance, adding two fixed-point numbers of (absolute) precision P can be done in $O(P)$ bit operations.

As we mentioned in the previous subsection, we will assume that we are working with fixed-point numbers of absolute precision P whose integral part can be represented in $O(P)$ bits at the most. The complexity of multiplying such numbers is then $O(\mathcal{M}(P))$, where $\mathcal{M}(P)$ is defined as follows:

Definition 0.3.6. Denote by $\mathcal{M}(P)$ the number of bit operations needed to compute the product of two P-bit integers. We have

- $\mathcal{M}(P) = O(P^2)$ if the naive (schoolbook) algorithm is used;
- $\mathcal{M}(P) = O(P^{\log_2 3}) = O(P^{1.58})$ if Karatsuba's algorithm is used;
- $\mathcal{M}(P) = O(P \log P \log\log P) = O(P^{1+\epsilon})$ if the algorithm of Schönage-Strassen is used;
- $\mathcal{M}(P) = O\left(P \log P 2^{O(\log^* n)}\right)$ if Fürer's algorithm is used.

For more details on these algorithms and their implementations, we refer to [BZ10] and [Für09].

Newton's method

Finally, we mention that Newton's algorithm can be used to compute some quantities with precision P, for a cost which is similar to the cost of computing the function one inverts.

The following theorem is at the basis of the analysis of Newton's method; we present the one-dimensional case here, but the proof (presented in [BCSS97, Section 8.1]) can be immediately generalized to higher-dimensional spaces.

Theorem 0.3.7 ([BCSS97, Chapter 8])**.** *Let $f : \mathbb{C} \to \mathbb{C}$ be an analytic function, and define*

$$\gamma(f, z) = \sup_{k \geq 2} \left| \frac{f^{(k)}(z)}{f'(z) k!} \right|^{\frac{1}{k-1}}.$$

Define

$$N_f(z) = z - \frac{f(z)}{f'(z)}$$

and consider the sequence defined by an initial value z_0 *and the relation* $z_{n+1} = N_f(z_n)$*. Let* ζ *be such that* $f(\zeta) = 0$*; then*

$$|z_0 - \zeta| \leq \frac{3 - \sqrt{7}}{\gamma(f, \zeta)} \quad \Rightarrow \quad |z_n - \zeta| \leq \frac{1}{2^{2^n - 1}} |z_0 - \zeta|.$$

Hence, provided that z_0 is close enough to the zero of f, Newton's method gives an approximation of the zero with an accuracy which roughly doubles at each step.

We note the following corollary, which we will use often:

Corollary 0.3.8. *Let* $f : \mathbb{C} \to \mathbb{C}$ *be an analytic function and* x *such that* $f(x) = 0$*. Let* $\hat{x}$ *be an approximation of* x *with precision* P*. Then computing* $N_f(\hat{x})$ *at precision* $2P$ *gives an approximation of* x *with precision* $2P - \delta$*, where* $\delta > 0$ *depends only on* f *(and precision losses) and* x*.*

Hence, one step of Newton's method "lifts" an approximation with precision P into an approximation with absolute precision roughly $2P$. In practice, computing δ can be done using the following procedure (described in e.g. [ET14a]): if $\hat{x}$ and $N_f(\hat{x})$ agree to k bits, and $N_f(\hat{x})$ and $N_f(N_f(\hat{x}))$ agree to k' bits, we have $\delta = 2k - k'$.

This allows to prove the following result, giving the complexity of applying Newton's method:

Theorem 0.3.9. *Let* $f : \mathbb{C} \to \mathbb{C}$ *be an analytic function and* x *such that* $f(x) = 0$*. Denote* $C(f, P)$ *the cost of evaluating* $\frac{f}{f'}$ *with arguments of precision* P*, and suppose that* $2C(f, P) \leq C(f, 2P)$*. Then one can compute an approximation of* x *with precision* P *in* $O\left(C(f, P)\right)$ *operations.*

Proof. We give a sketch of the proof; a very similar result is proved in detail in [Dup06, Thm. 1.2]. Denote δ the number as in Corollary 0.3.8; note that δ is a constant in P. Let P_0 be sufficiently large so that Theorem 0.3.7 applies; we furthermore impose $P_0 > \delta$. Note again that P_0 is a constant in P. We start by computing an approximation x_{P_0} of x with precision P_0; the cost of computing this approximation (by any method) only depends on P_0, and is hence a constant in P. Computing $N_f(x_{P_0})$ with precision $2P_0$ gives an approximation of x with absolute precision $2P_0 - \delta > P_0$; one can then repeat the process k times in a row. The cost of applying this process is

$$C\left(f, P_0\right) + C\left(f, 2P_0 - \delta\right) + \ldots + C\left(f, 2^k P_0 - (2^k - 1)\delta\right) \leq 2C\left(f, 2^k P_0\right).$$

Taking $k = O(\log P)$ is enough to get a result which is an approximation of x with P bits of precision; the total cost is then $O\left(C(f, P)\right)$. □

This result means that applying Newton's method while doubling the working precision at each step is only as costly as the last full-precision step. Note that applying directly Theorem 0.3.7, i.e. computing each iteration at full precision, gives a $O(C(f, P) \log P)$ cost, which is not as good.

A direct application is that the division of two fixed-point numbers of precision P can be carried out in $O(\mathcal{M}(P))$ bit operations, by inverting the function $f_z(t) = 1 - zt$ via Newton's method. Furthermore, the square root can also be computed in $O(\mathcal{M}(P))$ bit operations, either by applying Newton's method directly (which gives in this case Heron's method), or by computing the inverse square root using Newton's method and multiplying it by the number; as explained

in [BZ10, Section 4.2.3], the latter is more efficient in practice (because there are then no divisions in Newton's method), but both methods have the same asymptotic cost of $O(\mathcal{M}(P))$.

Another application of Newton's method, as we explain in Chapter 3, is the computation of $\exp(z)$ with precision P in the same amount of time as $\log(z)$, which is $O(\mathcal{M}(P)\log P)$ (which we call *quasi-linear time* or *quasi-optimal time*).

Finally, an interesting application of Newton's method is to prove that any algebraic function (i.e. any function which can be defined as the root of a polynomial equation) over $\mathbb{Q}[X]$ can be computed with absolute precision P in $O(\mathcal{M}(P))$ bit operations [BB87, Theorem 6.4]. We will see in this manuscript that a large number of transcendental functions – $\pi, \log, \exp, \theta$ – can be computed in $O(\mathcal{M}(P)\log P)$ bit operations; proving that they cannot be computed in $O(\mathcal{M}(P))$ bit operations would be another proof of their transcendance, but we are nowhere close to obtaining such facts.

Chapter 1

Background on elliptic and hyperelliptic curves

In this introductory chapter, we take a look at elliptic and hyperelliptic curves, which are abelian varieties that have been studied extensively over the years, in particular because of their use in cryptography. One of the ultimate goals of this manuscript is to describe a new algorithm to compute *isogenies*, which are morphisms between elliptic curves with interesting applications; it led us to investigate related problems, among which the computation of the theta function and of the Abel-Jacobi map.

We start this chapter by giving a brief overview of elliptic curves defined over any field; then, we discuss isogenies and their applications, and survey the state of the art in computing them in several settings. In another section, we discuss complex elliptic curves, for which another representation is available, with interesting computational consequences. We then end the chapter by a discussion of hyperelliptic curves.

1.1 Elliptic curves and isogenies

We define elliptic curves over any field, then discuss isogenies; we also survey the state of the art for their computation in several settings. We refer the reader to [CFA+10, Sil86] for more details.

1.1.1 Elliptic curves over a field

Definition 1.1.1 ([CFA+10, Section 13.1.1])**.** Let K be a field. An *elliptic curve E over K*, is the set of points $[X : Y : Z] \in \mathbb{P}^2(\overline{K})$ satisfying the equation

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3$$

with $a_i \in K$. We call $\mathcal{O} = [0 : 1 : 0]$ the point at infinity. We denote $E(K)$ the set of K-rational points, the points of the elliptic curves which are defined over K, i.e. for which there is $\lambda \in \overline{K}$ such that $\lambda X, \lambda Y, \lambda Z \in K$.

For ease of notation, we will often write the equation using non-homogeneous coordinates ($x = X/Z, y = Y/Z$):

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

which works for all the points except the point at infinity.

Remark 1.1.2. In the remainder of this manuscript, we will assume that $\mathrm{char}(K) \neq 2, 3$. Hence, after a change of variables, every elliptic curve can be written in a *short Weierstrass form*:

$$y^2 = x^3 + ax + b.$$

There exists a group law making the elliptic curve an abelian group; this fundamental property is the basis of the cryptographic applications of elliptic curves.

Proposition 1.1.3. *Let E be an elliptic curve over K. There exists a map $E \times E \to E$, the* chord-and-tangent process*, which defines a commutative group law on E. Furthermore the group law preserves K-rationality, i.e. it gives a group law $E(K) \times E(K) \to E(K)$.*

The chord-and-tangent process is described in many references, such as [Sil86, Section III.2], [Gal12, Section 7.9] or [CFA+10, Section 13.1.1]. We do not describe it explicitly in this manuscript, as we do not use it; its existence is sufficient for our purposes.

The elliptic curve discrete logarithm problem (ECDLP)

Elliptic curves over finite fields have found an application in the last decades in the field of cryptography; indeed, cryptosystems based on elliptic curves are among the most widely spread, and are supported in many different standards and applications. As all public-key cryptography schemes, its security relies on a hardness assumption, i.e. a problem that is believed to be hard, and such that breaking the cryptographic scheme seems to be as hard as solving this problem. In the case of elliptic curve cryptography, the problem is

Definition 1.1.4. Let $E(\mathbb{F}_p) : y^2 = x^3 + ax + b$ with $a, b \in \mathbb{F}_p$. The *elliptic curve discrete logarithm problem* is the following problem:

$$\text{Given } P \in E(\mathbb{F}_p) \text{ and } Q = [n]P, \text{ compute } n.$$

We sometimes put $n = \log_P Q$, the discrete logarithm of Q with respect to the base point P. We outline a few facts on the state of the art on solving the ECDLP; for a recent and more in-depth review, we refer to [GG16].

In some clearly-identified cases, the elliptic curve discrete logarithm problem (ECDLP) is easy to solve; this is the case for instance for curves such that $\sharp E(\mathbb{F}_q) \in \{q, q+1\}$, and such cases should be avoided for cryptography. Furthermore, the Silver-Pollig-Hellman algorithm [CFA+10, Section 19.3] allows one to solve the discrete logarithm problem in $O(\sqrt{n})$, where n is the largest prime factor of the order of the group; hence, one important guideline to follow is to work in a subgroup of points of the elliptic curve that has a large prime order. A frequent case is to pick a curve whose number of points is "almost a prime", that is to say $\sharp E(\mathbb{F}_q) = cp$ where c is a small number (for instance $c = 16$) and p is a prime.

Pollard's rho algorithm is currently one of the best general-purpose algorithms to solve the ECDLP in a subgroup of prime order r; it has an average-case complexity of $(\sqrt{2} + o(1))\sqrt{r}$ operations [GG16]. An interesting improvement over this approach is the "two grumpy giants and a baby" algorithm of [BL13], an improvement over the "baby-steps giant-steps" approach. The analysis of [GWZ15] shows one can expect an average running time of $(1.26 + o(1))\sqrt{r}$ steps, an improvement over Pollard's rho. The article [GWZ15] exhibits further improvements, bringing the average complexity down to $(0.38 + o(1))\sqrt{r}$ for Pollard's rho and $(0.36 + o(1))\sqrt{r}$ for the "two grumpy giants and a baby" approach.

Hence, for a curve which has a prime order subgroup of order $\simeq 2^n$, breaking the ECDLP costs around $2^{n/2}$ operations. By contrast, algorithms to factor n-bit numbers (and thus who

break the RSA cryptosystem) require roughly $e^{1.92(\log n)^{1/3}(\log\log n)^{2/3}}$ operations, which is a much faster, subexponential complexity. Comparing those complexities give the result that the key size for a cryptosystem whose security relies on the ECDLP grows as slowly as the cube root of the key size for the RSA cryptosystem; this allows cryptosystems based on elliptic curves to have much smaller key sizes than other cryptosystems, which compensates for a costlier group law. In practice, a few curves have been standardized by NIST, defined over $\mathbb{F}_p$ for p a prime number of size 192, 224, 256, 384, or 521 bits. For reference, the largest instance of ECDLP for a curve over $\mathbb{F}_p$ was for a subgroup with a 112 bit prime order [BK10], while the largest RSA instance ever broken was for a key of size 768 bits [KAF+10].

1.1.2 Isogenies

This section deals with *isogenies*, which are essentially morphisms transporting the group law of a curve onto another one. Those maps are very useful in many domains related to elliptic curve cryptography; we detail in Section 1.2 some applications of isogenies. We introduce isogenies and state the most important mathematical theorems related to them. We do not present proofs, as those are presented in a number of textbooks; our presentation follows the one of [Sil86].

Definition 1.1.5 ([Sil86, III.4])**.** Let E_1, E_2 be two elliptic curves. An *isogeny* from E_1 to E_2 is a morphism $\phi : E_1 \to E_2$ such that $\phi(\mathcal{O}_{E_1}) = \mathcal{O}_{E_2}$ and $\phi(E_1) \neq \{O_{E_2}\}$. We then say that E_1 and E_2 are *isogenous* if there is an isogeny from E_1 to E_2.

This property implies that isogenies transport the group law from one elliptic curve to another one, as follows:

Theorem 1.1.6 ([Sil86, Theorem 4.8])**.** *Let $\phi : E_1 \to E_2$ be an isogeny. Then for any $P, Q \in E_1$, $\phi(P + Q) = \phi(P) + \phi(Q)$.*

We then define the notion of *degree* of isogenies:

Definition 1.1.7 ([Sil86, III.4])**.** An isogeny induces an injection of function fields $\phi^* : \overline{K}(E_2) \to \overline{K}(E_1)$. We then define the *degree of ϕ* as

$$\deg \phi = [\overline{K}(E_1) : \phi^*\overline{K}(E_2)].$$

In all that follows, we may write "ϕ an n-isogeny" for "ϕ an isogeny of degree n".

Remark 1.1.8. An important example of an isogeny is the *multiplication-by-m* map

$$[m]P = \underbrace{P + ... + P}_{m \text{ times}}$$

which is an isogeny from a curve to itself. Note that $\deg([m]) = m^2$.

Remark 1.1.9. We call isogenies $\phi : E \to E$ *endomorphisms*, and consider the *endomorphism ring of the curve* $\mathrm{End}(E)$. Usually, the only endomorphisms are the multiplication-by-m maps, and $\mathrm{End}(E) \simeq \mathbb{Z}$; however, for some curves, the endomorphism ring is larger, and we then say that the curve has *complex multiplication*. This is for instance the case of curves over finite fields, who have at least the Frobenius endomorphism $\phi : (x, y) \mapsto (x^p, y^p)$. We refer to [Sil86, Section III.9 and Section V.3] for more on endomorphism rings in finite fields, and to [Koh96, Sut13] for explicit algorithms.

Theorem 1.1.10 ([Sil86, Theorem III.4.12])**.** *Let E be an elliptic curve and let Φ be a finite subgroup of E. There there exists a unique elliptic curve E' and a (separable) isogeny $\phi : E \to E'$ such that* $\mathrm{Ker}\,\phi = \Phi$.

An explicit construction of the curve and the isogeny is given in Section 1.1.3.

Theorem 1.1.11 ([Sil86, Theorem III.6.1]). *Let $\phi : E_1 \to E_2$ be an isogeny of degree m. Then there is a unique isogeny $\hat{\phi} : E_2 \to E_1$ such that $\hat{\phi} \circ \phi = [m]$. We call $\hat{\phi}$ the* dual isogeny *of ϕ.*

This result has many applications; for instance, we note that it has been used in [DIK06] to propose curves where doubling and tripling are sped up by computing [2] and [3] using specific isogenies and their dual instead of adding points using the classical chord-and-tangent process.

Remark 1.1.12 ([Sut13, p. 6]). Note that for any prime ℓ that does not divide the characteristic of the field, there are $\ell^2 - 1$ non-zero ℓ-torsion points, and hence $\ell + 1$ cyclic subgroups of order ℓ. Each of those subgroups is the kernel of an isogeny over $\overline{K}$. Furthermore, every ℓ-isogeny ϕ from E arises this way, since $P \in \operatorname{Ker}\phi \Rightarrow [n]P = \hat{\phi} \circ \phi(P) = 0$. The isogeny is defined over K if the subgroup is Galois-invariant; [Sut13, Lemma 2] shows there are 0, 1, 2 or $\ell + 1$ isogenies defined over K. This property is useful in the context of SEA algorithm (see Section 1.2.2).

The following theorem gives a criterion to recognize whether two curves over a finite field are isogenous:

Theorem 1.1.13 (Tate). *Let E, E' be two elliptic curves over $\mathbb{F}_q$. Then E and E' are isogenous if and only if $\sharp E(\mathbb{F}_q) = \sharp E'(\mathbb{F}_q)$.*

Hence, counting points of both elliptic curves is enough to determine whether they are isogenous or not. This can be accomplished in polynomial time using the *Schoof-Elkies-Atkin (SEA) algorithm*, which has complexity $O(\log^{4+\epsilon} p)$ [BSS99], an improvement over the $O(\log^{5+\epsilon})$ complexity of the original Schoof algorithm [Sch95]. We discuss this algorithm in Section 1.2.2, as the algorithm itself has a connection with the computation of isogenies.

Finally:

Theorem 1.1.14 ([CFA+10, Corollary 4.76]). *Any isogeny can be decomposed into a sequence of isogenies of prime degree.*

This allows us to assume in some applications that the isogeny has degree a prime number. Also, the decomposition of an isogeny in isogenies of prime degrees can be translated in terms of paths in isogeny graphs; see Section 1.1.5 for more details.

Remark 1.1.15. An isogeny between elliptic curves in their short Weierstrass form can be written as

$$\phi(x, y) = \left(\frac{g(x)}{h(x)^2}, y \frac{k(x)}{h(x)^3} \right),$$

with $h(x)$ a polynomial defined in the next section; furthermore, we have $\deg h = \frac{\ell-1}{2}$, and $\deg g = \ell$, $\deg k = 3\frac{\ell-1}{2}$. Another way to write this formula is given in the next section.

The problem of isogeny computation can be split up in three different settings, depending on what is known and what is sought. We summarize these settings in three different problems, from easiest to hardest: "finding an ℓ-isogenous curve and the isogeny" (Problem 1.1.16), "computing an ℓ-isogeny between two given curves" (Problem 1.1.19) and "computing an isogeny between two given curves" (Problem 1.1.20). We take a look at these problems in the following sections.

1.1.3 Finding a ℓ-isogenous curve: Vélu's formulas

We first consider the following problem:

Problem 1.1.16. *Let ℓ be an odd prime. Given:*

- *an elliptic curve E defined over an algebraically closed field K,*
- *a subgroup $S \subset E(K)$ of cardinality ℓ (or, alternatively, an ℓ-torsion point P, in which case $S = \{[i]P\}$),*

compute:

- *A curve E' that is ℓ-isogenous to E;*
- *An ℓ-isogeny $\phi : E \to E'$ such that* $\operatorname{Ker}\phi = S$.

This problem can be solved explicitly using *Vélu's formulas*, first published in [Vél71]. Recall we suppose here that ℓ is an odd prime and that the curve is in short Weierstrass form. A more general presentation of those formulas can be found in [Vél71, Ler97], and proofs are presented in e.g. [Was08].

The isogeny ϕ is:

$$\begin{aligned} \phi : E(K) &\to E'(K) \\ \mathcal{O} &\mapsto \mathcal{O} \\ P = (X_P, Y_P) &\mapsto \left(X_P + \sum_{Q \in S\setminus\{\mathcal{O}\}} X_{P+Q} - X_Q, Y_P + \sum_{Q \in S\setminus\{\mathcal{O}\}} Y_{P+Q} - Y_Q, \right) \end{aligned}$$

The coefficients defining E' and those defining ϕ can be recovered using the following theorem:

Theorem 1.1.17 ([Ler97, Theorem 39])**.** *Let S be a subgroup of $E(K)$ of cardinality ℓ. Define R as the set satisfying $S\setminus\{\mathcal{O}\} = R \cup (-R)$ with $R \cap (-R) = \emptyset$. For any $Q = (X_Q, Y_Q) \in S\setminus\{\mathcal{O}\}$, define:*

$$t_Q = 6X_Q^2 + 2a \qquad u_Q = 4(X_Q^3 + aX_Q + b)$$

and $t = \sum_{Q\in R} t_Q$ and $w = \sum_{Q\in R} u_Q + X_Q t_Q$. Define

$$E' : y^2 = x^3 + (a - 5t)x + (b - 7w),$$

then the isogeny is given by

$$\begin{aligned} \phi(P)_X &= X_P + \sum_{Q\in R} \frac{t_Q}{X_P - X_Q} + \frac{u_Q}{(X_P - X_Q)^2} \\ \phi(P)_Y &= Y_P \left(1 + \sum_{Q\in R} \frac{t_Q}{(X_P - X_Q)^2} + \frac{2u_Q}{(X_P - X_Q)^3} \right) \end{aligned}$$

Since $\sharp S = \ell$ is odd prime, any point in it generates the group and is an ℓ-torsion point; hence, knowing just one point of S is enough to compute the whole subgroup. This allows one to compute the polynomial

$$h(x) = \prod_{Q\in R} (x - X_Q)$$

whose square is the denominator of the x-coordinate of the isogeny. One can then recover t, w and the coefficients of the rational function defining ϕ from the coefficients of h; we refer to [Ler97, Chapter 4] for details.

Applying Vélu's formulas requires $O(\ell)$ multiplications in $\mathbb{C}$. We then estimate the cost of writing it in the form given in Note 1.1.15; one could think of interpolating the rational

function, or simply computing it from the shape given by the formulas (which requires computing $\prod_{Q \neq Q_i}(X - X_Q)$). In both cases, one uses methods related to remaindering trees (see [VZGG13, Chapter 10] or Section 9.2.3) to get the best complexity, which is $O(\mathcal{M}(\ell)\log \ell)$ field operations.

Remark 1.1.18. A careful inspection of Vélu's formulas reveals that the isogeny is actually of the shape

$$\phi(x, y) = \left(\frac{g(x)}{h(x)^2}, y\left(\frac{g(x)}{h(x)^2}\right)'\right).$$

with $\deg g = \ell$ and $\deg h = \frac{\ell-1}{2}$. This simplifies the computations, as this means one can only compute the x-coordinate of an isogeny. We make use of this in Chapter 9.

1.1.4 Computing an ℓ-isogeny

We now consider a slightly harder problem:

Problem 1.1.19. *Given:*

- *two elliptic curves E and E', defined over K,*
- *a prime integer ℓ such that E and E' are ℓ-isogenous,*

compute:

- *an ℓ-isogeny $\phi : E \to E'$.*

This problem first appeared in the context of speeding up Schoof's algorithm to compute the number of points of an elliptic curve; the polynomial $h(x)$ is used to reduce the cost of polynomial arithmetic in the SEA algorithm (see Section 1.2.2 for details). Several algorithms exist to solve this problem; however, their complexity is more or less advantageous depending on the characteristic of K.

The first case is the large characteristic case, in which $\mathrm{char}(K)$ is either much larger than ℓ or 0. The complexities here are given in terms of number of field operations. Several methods to solve this problem have been proposed over the years; we refer to [BMSS08] for a more in-depth review of each algorithm, including pseudocode. We note that the first algorithm of Elkies, which consists in differentiating the differential equation satisfied by $\wp$ (see Section 1.3.4), and Atkin's algorithm, which consists in computing the exponential of a power series, are two methods which were proposed in the context of the SEA algorithm; we mention why these algorithms were needed in this context in Section 1.2.2. Both of these algorithms require $O(\ell^2)$ field operations; this complexity is improved by the algorithm of [BMSS08], which uses fast algorithms for the computation of power series to achieve a $O(\mathcal{M}(\ell)\log \ell)$ running time – that is to say, a quasi-linear number of field operations with respect to the degree of the isogeny.

The other case is the small characteristic case, for instance $K = \mathbb{F}_q$ with $q = p^n$ and $p \leq \ell$. Two approaches by Couveignes have been seminal in this case; these algorithms, originally published in [Cou94] and [Cou96], are also discussed in [Ler97]. The second algorithm relies on the computation of the isogeny from its image on the p^k-torsion points, which are defined on an extension of $\mathbb{F}_q$; these groups are cyclic groups, and the algorithm roughly attempts to map a generator of the p^k-torsion group of E to a similar generator in E', then interpolate the isogeny and check if it is the right one. The computations on points of p^k-torsion is computationally costly, and one should use computations in Artin-Schreier extensions to attempt to mitigate memory requirements, as described (along with several other improvements) in [DF11]; the complexity of this algorithm is $O(\ell^2 \log q)$ operations in $\mathbb{F}_p$. These algorithms assume p is fixed,

as the dependency in $\log p$ is exponential; however, we note that a recent paper [DFHPS16] outlines a new algorithm with similar running time but without an exponential complexity in $\log p$.

1.1.5 Computing an isogeny

The final problem we consider here is the hardest one:

Problem 1.1.20. *Given:*

- *two isogenous curves E and E', defined over K,*

compute

- *an isogeny $\phi : E \to E'$.*

We consider here the case where $K = \mathbb{F}_q$. This problem is then much harder than Problems 1.1.16 and 1.1.19. In fact, the best known algorithm to solve it has exponential complexity, and [Gal99, Section 8] gives arguments showing that this problem may not have polynomial-time solutions in the general case. Probably due to this, a few cryptosystems who take this problem as a basis for their hardness assumption have been proposed [Sto12, DFJP14]; thus any improvement on this problem has security implications for those cryptosystems.

Since an isogeny can be decomposed as a sequence of isogenies of prime degree by Theorem 1.1.14, the strategy that is generally employed is one of considering *ℓ-isogeny graphs* for ℓ prime, and attempt to navigate them.

Definition 1.1.21 ([Sut13, Def.1])**.** An ℓ-isogeny graph is a graph (V, E) with

$$\begin{aligned} \mathrm{V} &= \{ \text{ elliptic curves over } K \text{ up to } \overline{K}\text{-isomorphism } \} \simeq \mathrm{K} \text{ (via the } j\text{-invariant)} \\ \mathrm{E} &= \{ \text{ pairs } (j_1, j_2) \text{ of } j\text{-invariants of } \ell\text{-isogenous curves } \}. \end{aligned}$$

The isogeny graph is then the union of all ℓ-isogeny graphs for all primes ℓ.

The problem of constructing an isogeny between two curves is then rephrased as the problem of finding a path connecting the two curves in the isogeny graph.

Note that curves that are isogenous to a supersingular curve are themselves supersingular; hence, connected components are either ordinary or supersingular. The supersingular components have different properties from the ordinary components, which is why we distinguish both cases. We mention algorithms on both classical and quantum computers, since the hardness of the problem has led some to investigate its complexity on quantum computers, in an attempt to determine whether cryptosystems based on this hard problem would be quantum-resistant.

Ordinary case

In the ordinary case, ℓ-isogeny graphs have a "volcano" structure. We refer to [Sut13] for more precise statements of the properties in this section.

Definition 1.1.22 (ℓ-volcano; [Sut13, Def.1])**.** An ℓ-volcano is a connected undirected graph partitioned in d levels $V_0, ..., V_d$ such that

- V_0, the *surface*, is a cycle (or more generally a regular graph of degree at most 2);
- each vertex in V_{i+1} has exactly one neighbour in V_i, and this accounts for every edge not on the surface;

- each vertex except those in V_d have degree $\ell+1$.

As outlined in [Koh96, Chapter 4], any[1] ordinary component of the ℓ-isogeny graph is an ℓ-volcano; furthermore, all vertices at a level share the same endomorphism ring. An ℓ-isogeny $\phi : E \to E'$ is said to be "ascending" if $\mathrm{End}(E')$ is larger than $\mathrm{End}(E)$, descending if it is smaller, and horizontal if it is the same, which only happens at the surface; moreover, one cannot ascend further than the surface of the volcano.

Given this structure, the general strategy for solving Problem 1.1.20, i.e. find ϕ between a given E_1 and E_2, is as follows:

1. Compute an ascending chain of ℓ-isogenies from E_1 (resp. E_2) to E_1' (resp. E_2') such that $\mathrm{End}(E_1') = \mathcal{O}_K$ (resp. $\mathrm{End}(E_2') = \mathcal{O}_K$). This is Kohel's algorithm [Koh96].

2. We wish to reach vertices E_i' such that the endomorphism ring is maximal, i.e. $\mathrm{End}(E_i') = \mathcal{O}_K$. The previous steps ensures only that $\ell \nmid [\mathcal{O}_K : \mathrm{End}(E_i')]$; hence, one need to repeat the previous steps and ascend different ℓ-volcanoes. This part is the costliest asymptotically, but is actually very fast in practice.

3. Find a horizontal isogeny between E_1' and E_2'.

The last step is the part which has been improved in different algorithms, which we mention here. The first algorithm has been the one of [Gal99], which uses a meet-in-the-middle strategy; it constructs isogeny trees starting at E_1' and E_2', using the following procedure: pick a prime number ℓ at random (in a carefully chosen set of primes) and construct all ℓ-isogenies starting from a node of each tree. This procedure needs to be iterated $O(\log p)$ times on average, but the size of the trees can be as big as $O(p^{1/4+\epsilon})$. Once a match is found, each ℓ-isogeny in the path linking E_1' and E_2' is reconstructed using algorithms to solve Problem 1.1.19; this step is analyzed as costing $O(p^{3/2} \log p)$ operations in general, but this complexity can actually be made negligible if one assumes smoothness properties, which allows to bound the maximal size of the primes. The algorithm is polynomial time if the class number of the endomorphism ring is small, which is the case for instance for elliptic curves generated using the CM method.

One improvement over this algorithm is given in [GHS02a]: instead of using isogeny trees, one can use random walks over the isogeny graph starting at E_1' and E_2'. This makes storage polynomial (instead of exponential), but the number of expected steps before finding a collision in the walks is $O(p^{1/4+\epsilon})$. This idea is combined with a representation of the isogeny as an ideal, which undergoes a smoothing step before the step which reconstructs the final isogeny; this step costs as much as the random walk, i.e. $O(p^{1/4+\epsilon})$, but allows a faster reconstruction (using ideas resembling the ones from the SEA algorithm of Section 1.2.2). A variant over this algorithm, which saves a constant factor, is described in [GS13].

Finally, we note that there is also a quantum algorithm to solve this problem, i.e. an algorithm running on a theoretical quantum computer instead of a classical one. The application of quantum computers to the resolution of these problems is certainly motivated by the need to find cryptosystems which rely on a hardness assumption that resists attacks using a quantum computer. For example, factoring integers and the ECDLP are both solvable in polynomial time on a quantum computer, while the Shortest Vector Problem in a lattice still requires exponential time. One algorithm to solve the problem we are considering here on a quantum computer has been proposed in [CJS14]; its running time is subexponential, i.e. $O\left(e^{\sqrt{3/2(\ln p \log\log p)^{1/2}}}\right)$. This is accomplished by reducing the problem to the *hidden shift problem*, which is a problem that can be solved by a quantum computer in subexponential time. We note that they also

[1] except those who contain curves of j-invariant 0 or 1728.

propose a classical (non-quantum) algorithm to speed up the ideal reduction step in [GHS02a], using more up-to-date techniques to reduce the ideal and compute the isogeny; this step requires subexponential complexity, instead of the $O(q^{1/4+\epsilon})$ complexity in [GHS02a].

Supersingular case

We now look at the supersingular components of isogeny graphs; these components exhibit a different structure from the volcanoes of the ordinary case, and hence the techniques above do not directly apply. This problem has been used as a hardness assumption for some cryptosystems, most notably the one of [DFJP14]; we discuss the implication of the algorithms below in Section 1.2.3. We refer once again to [Sut13] for details.

Supersingular isogeny graphs have a very regular structure:

Proposition 1.1.23 ([Koh96])**.** *Every vertex in a supersingular component of the ℓ-isogeny graph when considered over $\mathbb{F}_{p^2}$ has out-degree $\ell+1$. If the vertex is not 0 or 1728, nor adjacent to those two vertices, it also has in-degree $\ell+1$. Finally, the supersingular ℓ-isogeny graph is connected for every prime ℓ.*

The connectedness of the isogeny graph means one can simply work with, say, 2-isogenies to construct a path between any two points. Furthermore, those graphs are expander graphs, with a small diameter ($O(\log p)$ [DG16]) and hence a short path between two vertices.

A rather straightforward algorithm consists in, as in [Gal99], performing a "meet-in-the-middle" search in the full supersingular graph over $\mathbb{F}_{p^2}$ to find a path between both elliptic curves. This method will find the shortest path; however, the sheer size of the tree computations makes it a $O(p^{1/2+\epsilon})$ algorithm (both time and storage).

A more complex algorithm is the one in [DG16], which considers a variant of the isogeny graph above with only curves defined over $\mathbb{F}_p$. The graph then looks like a volcano of depth 2; hence, solving Problem 1.1.20 in the case where both E_1 and E_2 are defined over $\mathbb{F}_p$ can be done adapting techniques from the previous section, in average running time $O(p^{1/4+\epsilon})$. The running time of the general algorithm is still $O(p^{1/2+\epsilon})$, since one has to find isogenies from the original curves to curves defined over $\mathbb{F}_p$ (which is done in the article using self-avoiding random walks, and can also be done using Pollard-style stateless random walks) – unless of course the original curves are defined over $\mathbb{F}_p$, which makes the complexity only $O(p^{1/4+\epsilon})$.

Finally, we also mention a quantum algorithm to compute isogenies between supersingular curves [BJS14]. The algorithm improves the first step of [DG16] – that is to say, finding an isogeny to a curve defined over $\mathbb{F}_p$ – using Grover's algorithm, which in this case means that the cost of this step on a quantum computer is $O(p^{1/4+\epsilon})$. It also extends [DG16] in a similar way that [CJS14] extends [GHS02a] – that is to say, in the easier case where the curves are defined over $\mathbb{F}_p$, the structure can be exploited to yield a reduction to the hidden shift problem, thus making the quantum cost for this step $O\left(e^{\sqrt{3/2(\log p \log\log p)^{1/2}}}\right)$. This algorithm has direct implications for the security of a scheme; see Section 1.2.3 for details.

1.2 Applications of isogenies

1.2.1 Isogenies and the ECDLP

The morphism property of isogenies means that they "transport" the problem of the discrete logarithm from one curve to another. More precisely:

$$\begin{array}{rcl} E & \xrightarrow{\phi} & E' \\ Q = [n]P & \Leftrightarrow & \phi(Q) = [n]\phi(P) \\ \log_P Q & \Leftrightarrow & \log_{\phi(P)} \phi(Q) \end{array}$$

Hence, solving the ECDLP on E is only as hard as solving the ECDLP on E' once the isogeny ϕ has been computed. In theory, this could lead to a faster attack on the ECDLP of E, provided one can find a weaker curve E' that is isogenous to E and compute the corresponding isogeny.

Such an attack has been made explicit e.g. in the case of genus 3 hyperelliptic curves [Smi09]. In general, solving the discrete logarithm problem on these curves is hard, requiring $O(q^{4/3+\epsilon})$ operations. However, for a large proportion (around 28%) of hyperelliptic curves of genus 3, there is an isogeny between the curve and a non-hyperelliptic curve; this yields a $O(q^{1+\epsilon})$ attack using the algorithm of [DT08].

Generically, for elliptic curves, this amounts to solving Problem 1.1.20, which current algorithms cannot do in less than exponential time. Note also that identifying a weaker isogenous curve is not a problem one knows how to solve either; in fact, [JMV05] shows that isogenous curves all have similar security with respect to the ECDLP. Hence, this does not seem to constitute a generic threat to the security of the ECDLP.

However, the strategy of using isogenies to fall back on weaker curves can be used in the context of the Gaudry-Hess-Smart attack, presented in [GHS02b]. This attack uses Weil descent to reduce the ECDLP to a discrete logarithm problem on a hyperelliptic curve of high genus, where an index calculus attack (such as the one in [EG02, EGT11]) applies; such attacks are asymptotically faster than attacks such as Pollard's rho. As it happens, the fact that an elliptic curve is vulnerable to the GHS attack can be easily checked, but there is no way to check that a curve on which the GHS attack is unsuccessful is not isogenous to one which is vulnerable. This fact was noted in [GHS02a], where a faster algorithm to compute isogenies is used to extend the probability to find an isogenous curve that is vulnerable to the GHS attack. We refer to [MTW04] for practical implications of this attack.

Finally, the fact that finding an isogenous curve and computing the isogeny is a hard problem can be seen as worrying, as we do not have any guarantee that a curve is not isogenous to a weaker curve. Vulnerability to the GHS attack has once again been used in [Tes06], in which one finds a method to construct a curve E that is vulnerable to the GHS attack, and an isogenous curve E' for which the best attack is Pollard's rho, along with the corresponding isogeny. The curve E' appears to be secure, and finding E is a hard problem; however, if someone knows E and the isogeny, the ECDLP on E' can be easily solved. This opens the possibility of trapdoors in elliptic curves, which is a worrying possibility; [Tes06] recommends choosing the coefficients provably at random to avoid mistrust.

1.2.2 The SEA point counting algorithm

The topic of isogeny computation, and more precisely Problem 1.1.19, is one that was first considered in the context of finding asymptotic improvements to Schoof's algorithm to compute the number of points of an elliptic curve over a finite field. We provide a brief overview of Schoof's

original algorithm, then show the improvements to the algorithm and their link with isogenous curves; we follow the presentation of [BSS99].

In 1986 Schoof proposed in [Sch95] an algorithm to compute the number of points on an elliptic curve E defined over $\mathbb{F}_q$ in polynomial time; this was the first polynomial-time algorithm, down from the $O(q^{1/4+\epsilon})$ complexity of a baby-step / giant-step algorithm. The idea is as follows: since Hasse's theorem indicates that $\sharp E(\mathbb{F}_q) = q+1-t$ with $|t| \leq 2\sqrt{q}$, one computes $t \pmod{p_i}$ for enough primes p_i such that we can reconstruct t using the Chinese Remainder Theorem (i.e. so that the final modulus in the CRT is larger than $4\sqrt{q}$). To compute $t \pmod{p_i}$, the characteristic polynomial of the Frobenius endomorphism shows that for any $P = (x, y) \in E$:

$$\left(x^{q^2}, y^{q^2}\right) + [q](x, y) = [t]\left(x^q, y^q\right)$$

In particular for $P \in E[p_i]$, i.e. p_i-torsion points, we have $\left(x^{q^2}, y^{q^2}\right) + [q \pmod{p_i}](x, y) = [t \pmod{p_i}]\left(x^q, y^q\right)$. The computation of the left-hand side is then performed symbolically, i.e. using polynomial arithmetic. The trick is that this computation can be performed modulo ϕ_{p_i}, the p_i-torsion polynomial, i.e. the polynomial such that $(x, y) \in E[p_i]$ if and only if $\phi_{p_i}(x) = 0$; this polynomial is of degree $O(p_i^2)$, which limits the degree of the polynomials and hence the cost of the computations. Then, one computes iteratively the $[k](x^q, y^q)$ until a solution to the equation is found, which gives the correct value for $t \pmod{p_i}$. We refer to [BSS99] for the complexity of this algorithm.

The Schoof-Elkies-Atkin (SEA) algorithm lowers the complexity to $O(\log^{4+\epsilon} q)$. The prime numbers are split in two different sets: Atkin primes, i.e. primes p_i for which $t^2 - 4q$ is not a square modulo p_i, and Elkies primes, for which it is. Dealing with Atkin primes requires exponential time; hence, the improved running time is achieved using only Elkies primes, although in practice dealing with a small number of Atkin primes provides numerous advantages.

Improvements to this algorithm are given by the SEA algorithm, and a closer study of Elkies primes, i.e. primes p_i for which $t^2 - 4q$ is a square modulo p_i, and hence primes for which the polynomial $x^2 - tx + q$ has two roots in $\mathbb{F}_q$. We are looking to compute one of these roots, λ, by finding a solution to $(x^q, y^q) = [\lambda](x, q)$. The asymptotically significant savings come from the fact that one can work modulo a factor $\mathcal{F}_{p_i}$ of degree $O(p_i)$ of ϕ_{p_i}. As we mentioned in Note 1.1.12, subgroups of the ℓ-torsion define the kernel of an ℓ-isogeny; hence, the factor is simply the polynomial corresponding to the kernel to a p_i-isogeny $\mathcal{I}$ from the curve E to another curve E_1, i.e.

$$\mathcal{F}_{p_i}(X) = \prod_{\pm P_i \in \operatorname{Ker} \mathcal{I} \setminus \mathcal{O}} (X - x(P_i))$$

Computing the isogenous curve is done by first computing its j-invariant, which is a root of the modular polynomial $\Phi_{p_i}(j(E), X)$; we assume we get this polynomial from precomputed tables. Since any isogeny will do, we can afford to only look at isogenies to curves defined over $\mathbb{F}_q$, whose number is given by Note 1.1.12. Hence, we only need to compute a root of $\gcd(X^q - X, \Phi_{p_i}(j(E), X))$, which is typically of degree 2. We then compute the coefficients of the new curve E_1 from the two j-invariant and the coefficients of E (see [BSS99] for details). Once we have this, we essentially need to compute the p_i-isogeny between E and E_1, which corresponds to Problem 1.1.19. We refer to Section 1.1.4 for solutions to this problem, which were originally devised for this very setting.

The cost of the algorithms to solve Problem 1.1.19 does not matter greatly asymptotically here: the cost of computing the eigenvalue (using polynomial arithmetic modulo $\mathcal{F}_{p_i}$) is $O(\log^{3+\epsilon} q)$, which dominates the cost of these algorithms in any case. This is an improvement over the $O(\log^{5+\epsilon} q)$ complexity of the original Schoof algorithm.

1.2.3 Isogeny-based cryptography

A relatively novel idea that has been investigated in recent years is to use isogenies, and more precisely the hardness of Problem 1.1.20, as a basis for strong cryptosystems. Even more enticing is the fact that this problem seems to resist fairly well to quantum computers, unlike other problems like factoring integers or the ECDLP, which can both be solved in polynomial time using a quantum computer.

Cryptosystems based on isogenies between elliptic curves were proposed most notably in Stolbunov's Ph.D. thesis [Sto12], although the idea appears in a previous article by Couveignes [Cou06]. The hardness assumption in these cryptosystems reduces to Problem 1.1.20 on ordinary elliptic curves, for which there are no known algorithms on a classical computer requiring less than an exponential number of operations. However, it does not achieve resistance to quantum computers, as the problem can be solved using only a subexponential number of operations on a quantum computer.

Another, more efficient cryptosystem was proposed in [DFJP14], based on Problem 1.1.20 for supersingular elliptic curves; the hardness assumption is often called the Supersingular Isogeny Diffie-Hellman (SIDH) problem. Solving this problem requires $O(p^{1/2+\epsilon})$ operations on a classical computer and $O(p^{1/4+\epsilon})$ on a quantum one. However, if the elliptic curves are defined over $\mathbb{F}_p$, there is a small risk that the problem could be solved faster using the work in [DG16, BJS14] (respectively $O(p^{1/4+\epsilon})$ on a classical computer and subexponential time on a quantum computer); see [BJS14] for the full discussion, and their recommendation that the cryptosystem should avoid curves defined over $\mathbb{F}_p$. Efficient implementations are discussed in [CLN16].

1.3 The Abel-Jacobi map

In this section, we take a closer look at elliptic curves defined over $\mathbb{C}$, which will be an important part of this manuscript. Complex elliptic curves have another representation, as complex tori, which gives nice computational properties. The map which allows the translation between the Weierstrass form and the corresponding complex torus is the *Abel-Jacobi map*; this is one of the main objects of this manuscript, and we describe fast algorithms to compute this map in Chapter 8.

1.3.1 Definition of the map

Definition 1.3.1. Let $\omega_1, \omega_2 \in \mathbb{C}$ such that $\mathbb{R}\omega_1 + \mathbb{R}\omega_2 = \mathbb{C}$, and define the lattice $\Lambda = \mathbb{Z}\omega_1 + \mathbb{Z}\omega_2 \subset \mathbb{C}$. The associated *complex torus* is $\mathbb{C}/\Lambda$. We call ω_1, ω_2 the *periods* of the lattice, and Λ is the *period lattice*. The group law on $\mathbb{C}/\Lambda$ is just addition modulo the periods. The *fundamental parallelogram* is the set $\{x\omega_1 + y\omega_2, x, y \in [0, 1]\}$.

The "torus" part comes from the fact that one obtains a torus when gluing the left and the right sides of the fundamental parallelogram and the top and bottom sides.

Establishing the link between elliptic curves defined over $\mathbb{C}$ and complex tori can be done using techniques which are classical within the context of complex Riemann surfaces. More precisely, one can take a look at integrals of the invariant differential $\omega = \frac{dx}{y}$ associated to the elliptic curve E:

Proposition 1.3.2 ([Sil86, Section VI.I],[CFA+10, Section 5.1.2])**.** *Let $E(\mathbb{C}) : y^2 = x^3 + ax + b$ be an elliptic curve, and $\omega = \frac{dx}{y} = \frac{dx}{\sqrt{x^3+ax+b}}$ its invariant differential. Then, in general, the integral $\int_{\mathcal{O}}^{P} w$ for $P \in E(\mathbb{C})$ is not path-independent. More precisely, if one takes α, β two paths that generate the first homology group of the corresponding torus (for instance, paths around the*

branch cuts of $\sqrt{x^3+ax+b}$, as in [Sil86, Section VI.I]), then the value of $\int_{\mathcal{O}}^{P} w$ is only defined up to $m\omega_1 + n\omega_2$, where $\omega_1 = \int_\alpha \omega, \omega_2 = \int_\beta$ are the periods *of the elliptic curve.*

Integrals of the form $\int \frac{dx}{\sqrt{P(x)}}$ where $\deg P = 3$ are called *elliptic integrals*, as they appear in the expression of the length of an ellipse; this is where the name of elliptic curves comes from.

We note the following important property:

Proposition 1.3.3 ([CFA+10, Corollary 5.18])**.** *Let $\Lambda = \mathbb{Z}\omega_1 + \mathbb{Z}\omega_2$ be the lattice generated by the periods ω_1, ω_2 of a complex elliptic curve. Then $\Lambda \simeq \mathbb{Z} + \tau\mathbb{Z}$ with $\mathrm{Im}(\tau) > 0$.*

The Abel-Jacobi map is then the map studied in Proposition 1.3.2:

Theorem 1.3.4 ([CFA+10, Definition 5.12])**.** *Let E be an elliptic curve defined over $\mathbb{C}$. Take α, β two paths e.g. along the branch cuts; define $\omega_1 = \int_\alpha \omega, \omega_2 = \int_\beta \omega$. Define $\Lambda = \mathbb{Z}\omega_1 + \mathbb{Z}\omega_2$, the complex torus associated to the elliptic curve. Then the* Abel-Jacobi map

$$\begin{aligned} E(\mathbb{C}) &\to \mathbb{C}/\Lambda \\ P &\mapsto \int_{\mathcal{O}}^{P} \omega \pmod{\Lambda} \end{aligned}$$

is a well-defined isomorphism from the elliptic curve to the complex torus.

In the context of elliptic curves, this map is sometimes called the *elliptic logarithm map*, as its inverse can be considered to be an exponential morphism [Coh93, p. 398]

Remark 1.3.5. In the remainder of this manuscript, we will sometimes use the following shorthands:

- the *algebraic representation of a complex elliptic curve* refers to an elliptic curve E described by its short Weierstrass equation, i.e. its coefficients a and b;
- the *analytic representation of a complex elliptic curve* refers to an elliptic curve described as a torus $\mathbb{C}/\Lambda$, i.e. its periods.

The Abel-Jacobi map then performs the "translation" between these two representations, by converting a point P whose coordinates satisfy the short Weierstrass equation into a point z who lies on the complex torus, and vice-versa with its inverse.

1.3.2 Maps between complex tori

As outlined in [Sil86, Section VI.4], complex analytic maps between complex tori all have the same, very simple form:

Theorem 1.3.6 ([Sil86, Theorem VI.4.1])**.** *Let Λ_1, Λ_2 two lattices in $\mathbb{C}$. The association*

$$\begin{aligned} \{\alpha \in \mathbb{C} \mid \alpha\Lambda_1 \subset \Lambda_2\} &\to \{\phi : \mathbb{C}/\Lambda_1 \to \mathbb{C}/\Lambda_2 \text{ holomorphic and with } \phi(0) = 0\} \\ \alpha &\to \phi_\alpha : z \mapsto \alpha z \end{aligned}$$

is a bijection.

Note that $\phi(0) = 0 \Rightarrow \phi(P+Q) = \phi(P) + \phi(Q)$. This is an interesting result from the computational perspective, as it means one can compute the image of points very easily. We outline a few applications of this theorem.

Isomorphisms

A corollary of Theorem 1.3.6 is the following:

Theorem 1.3.7 ([Sil86, Corollary VI.4.1.1]). *Let E_1, E_2 two complex elliptic curves whose analytic representations are $\mathbb{C}/\Lambda_1$ and $\mathbb{C}/\Lambda_2$. Then E_1 and E_2 are isomorphic over $\mathbb{C}$ iff $\exists \alpha \in \mathbb{C}^*$ such that $\alpha\Lambda_1 = \Lambda_2$.*

Remark 1.3.8. The following result is well-known, and can be obtained with simple calculations (see e.g. [Sil86, Table 3.1]):

Theorem 1.3.9. *Let $E_1 : y^2 = x^3 + a_1x + b_1$ and $E_2 : y^2 = x^3 + a_2x + b_2$ two isomorphic elliptic curves over K. Then there exists $u \in K^*$ such that $a_2 = u^4a_1$ and $b_2 = u^6a_2$, and the isomorphism is given by the map $(x, y) \to (u^2x, u^3y)$.*

Note that for $K = \mathbb{C}$, this result is a direct consequence of Theorem 1.3.7 and the inverse of the Abel-Jacobi map (Section 1.3.4).

Isogenies

The link with isogenies of elliptic curves is as follows:

Theorem 1.3.10 ([Sil86, Theorem VI.4.1]). *Let E_1, E_2 be two elliptic curves, with corresponding analytic representations $\mathbb{C}/\Lambda_1, \mathbb{C}/\Lambda_2$. Then the association*

$$\{ \text{ isogenies between } E_1 \text{ and } E_2 \ \} \quad \to \quad \{\phi : \mathbb{C}/\Lambda_1 \to \mathbb{C}/\Lambda_2 \text{ holomorphic and with } \phi(0) = 0\}$$

is a bijection.

Remark 1.3.11. As we noted in Note 1.1.12, the existence of a dual isogeny means that every point P in the kernel of an ℓ-isogeny ϕ is an ℓ-torsion point, and $\operatorname{Ker}\phi$ is a subgroup of the ℓ-torsion group. Hence, the complex number α describing the isogeny between complex tori necessarily sends some points of the shape $\frac{m\omega_1+n\omega_2}{\ell}$ (for m, n integers) to 0 (mod Λ_2), which gives a relation between periods of isogenous curves. We use this remark in Section 9.1.1.

Remark 1.3.12. This result can be used to compute isogenies between analytic representations of complex elliptic curves, i.e. solving Problem 1.1.20. Let E_1, E_2 be two isogenous elliptic curves with analytic representations $\mathbb{C}/\Lambda_1, \mathbb{C}/\Lambda_2$. Write $P_1 = (\omega_1, \omega_2)$ and $P_2 = (\omega_1', \omega_2')$, the periods of the lattices. Then there is $\alpha \in \mathbb{C}$ such that $\alpha\Lambda_1 \subset \Lambda_2$. The images of periods have to be points of Λ_2 because $\phi(0) = 0$, so we have

$$\alpha P_1 = P_2 M$$

with $M \in M_2(\mathbb{Z})$. We can then compute α and M, for instance using LLL, as in [VW00], who described this method in genus 2 and used it to compute explicit isogenies between a set of curves, thus proving that the curves were indeed isogenous.

CM maps

Another application is determining whether a curve over $\mathbb{C}$ has *complex multiplication*, in the sense outlined in Note 1.1.9, using analytic representations; the method is described in [VW99]. We have that any endomorphism of E corresponds to a map $z \mapsto \alpha z$ for some α such that $\alpha\Lambda \subset \Lambda$. Hence, we have the following relation:

$$\alpha P = PM$$

with M a 2×2 integer matrix; one can then compute α and M using the LLL algorithm. The method was originally described in the case of genus 2 hyperelliptic curves in [VW99].

1.3.3 Elliptic functions and the $\wp$ function

In order to make explicit the inverse of the Abel-Jacobi map, we need a bit of background on elliptic functions. We refer to [Cha85] or [Sil86, Chapter VI] for proofs and a more thorough background.

Note that, as a complex map, the inverse of the Abel-Jacobi map is invariant by translation by ω_1 or ω_2. It is thus rather natural to study elliptic functions:

Definition 1.3.13 ([Cha85, Chapter I])**.** An *elliptic function* is a meromorphic function with two periods, ω_1, ω_2, such that $\mathrm{Im}\left(\frac{\omega_2}{\omega_1}\right) > 0$.

Using integration on paths slightly off the fundamental parallelogram, one can prove

Theorem 1.3.14 ([Cha85, Chapter II])**.** *No non-constant elliptic function is entire. A non-constant elliptic function has at least one pole in any parallelogram. The sum of residues of an elliptic function in a parallelogram is 0; the number of zeros is equal to the number of poles, counted with multiplicity, and we say that this number is the order of the elliptic function.*

The first construction of an elliptic function with any periods was given by Weierstrass:

Definition 1.3.15. Given ω_1, ω_2, define the function $\wp : z \mapsto \wp(z, [\omega_1, \omega_2])$ by

$$\wp(z, [\omega_1, \omega_2]) = \frac{1}{z^2} + \sum_{m,n \in \mathbb{Z}^2} \frac{1}{(z - m\omega_1 - n\omega_2)^2} - \frac{1}{(m\omega_1 + n\omega_2)^2}$$

Then $\wp$ is elliptic of periods ω_1, ω_2, and it is of order 2. It is an even function.

Remark 1.3.16. On top of the ω_1-periodicity and the ω_2-periodicity of $\wp$, we also have the following property:

$$\wp(z, [\omega_1, \omega_2 + \omega_1]) = \wp(z, [\omega_2, -\omega_1]) = \wp(z, [\omega_1, \omega_2])$$

This is easily proved when looking at the definition of $\wp$. Hence, when trying to compute $\wp(z, [\omega_1, \omega_2])$ in Section 4.3 and Section 8.1.2, we will be able to assume that

$$\mathrm{Im}\left(\frac{\omega_2}{\omega_1}\right) \geq \frac{\sqrt{3}}{2}, \qquad |\mathrm{Re}\left(\frac{\omega_2}{\omega_1}\right)| \leq \frac{1}{2} \qquad \text{(cf Chapter 2)},$$
$$0 \leq \mathrm{Im}(z) < \mathrm{Im}\left(\frac{\omega_2}{2\omega_1}\right), \qquad |\mathrm{Re}(z)| \leq \frac{\mathrm{Re}(\omega_1)}{2}.$$

Finally, we mention another proposition:

Proposition 1.3.17 ([Sil86, Theorem VI.3.2])**.** *Every elliptic function of periods ω_1, ω_2 is a rational function of $z \mapsto \wp(z, [\omega_1, \omega_1])$ and $z \mapsto \wp'(z, [\omega_1, \omega_1])$.*

This proposition shows that, once one has an algorithm to compute $\wp$ and $\wp'$ (as in Chapter 8, for instance), it is not very much harder to compute any elliptic function.

1.3.4 Inverse of the Abel-Jacobi map

The connection between the Weierstrass $\wp$ function and Weierstrass equations of elliptic curves is shown in the following proposition:

Proposition 1.3.18. *The function $\wp$ satisfies the following differential equation:*

$$\wp(z,[\omega_1,\omega_2])^2 = 4\wp(z,[\omega_1,\omega_2])^3 - g_2\wp(z,[\omega_1,\omega_2]) - g_3$$

where

$$g_2 = 60 \sum_{m,n\in\mathbb{Z}^2} \frac{1}{(m\omega_1+n\omega_2)^4} \qquad g_3 = 140 \sum_{m,n\in\mathbb{Z}^2} \frac{1}{(m\omega_1+n\omega_2)^6}$$

are related to Eisenstein series of weight 4 and 6.

This proposition is proven using the Laurent expansion, for instance.
Hence, the map

$$\begin{cases} \mathbb{C}/\Lambda & \to E(\mathbb{C}) \\ 0 \pmod{\Lambda} & \mapsto \mathcal{O} \\ z \pmod{\Lambda} & \mapsto [\wp(z,\Lambda) : \wp'(z,\Lambda)) : 1] \end{cases}$$

is an isomorphism; in fact, it is the inverse of the Abel-Jacobi map.

A most important problem we consider in this manuscript is the problem of computing quickly the Abel-Jacobi map and its inverse. Quasi-optimal time algorithms – i.e. algorithms allowing the computation of this map with precision P in $O(\mathcal{M}(P)\log P)$ – for the computation of the periods and of the elliptic logarithm have been outlined, thus giving a quasi-optimal time algorithm for the Abel-Jacobi map; we discuss these algorithms in Chapter 4. We show in Chapter 8 that the coefficients of the Weierstrass equation can be computed from the periods using the θ function (defined in Chapter 2), and that $\wp$ is also related to the θ function. We show in Chapter 6 that the θ function can be computed with quasi-optimal time, which gives a quasi-optimal time algorithm to compute the inverse of the Abel-Jacobi map. We also give in Chapter 4 another algorithm for $\wp$, with conjectural quasi-optimal running time.

Thomae formulas

We mention a last result, which links the coefficients of the curve equation (in short Weierstrass form) and the period lattice, via theta-constants, which are values of the θ function (see Section 2.1). These are the *Thomae formulas*:

Theorem 1.3.19 (Thomae formulas)**.** *Let E be a complex elliptic curve defined by $y^2 = P(x)$ with $P = 4X^3 - g_2X - g_3 = 4(X-e_1)(X-e_2)(X-e_3)$. Let ω_1,ω_2 be the periods of E and $\tau = \frac{\omega_2}{\omega_1}$. Then*

$$e_1 - e_2 = \left(\frac{\pi}{\omega_1}\right)^2 \theta_0^4(0,\tau), \quad e_1 - e_3 = \left(\frac{\pi}{\omega_1}\right)^2 \theta_1^4(0,\tau), \quad e_3 - e_2 = \left(\frac{\pi}{\omega_1}\right)^2 \theta_2^4(0,\tau)$$

Proof. See [BM88] or [Cha85, Cor. 1 of Theorem 5, Chap V]. □

We use this result a few times in this manuscript, for instance to compute the curve coefficients from τ in quasi-optimal time (Section 8.1.1), but also to give a fast algorithm for the computation of the $\wp$ using the Landen transform (Section 4.3). Hence, these formulas are useful in making the inverse of the Abel-Jacobi map computable in quasi-optimal time, as they offer a link with the θ function.

1.4 Hyperelliptic curves and the Abel-Jacobi map

This manuscript deals mostly with the genus 1 case, i.e. elliptic curves, Jacobi's theta function and isogeny computation in genus 1. However, some of the methods we present here generalize to higher genera; in particular, we present in Chapter 8 algorithms to compute the generalization of the Abel-Jacobi map to higher genera. In this context, we give some background here on hyperelliptic curves and the corresponding Abel-Jacobi map. Most of the material presented here is taken from [Gal12, CFA+10].

1.4.1 Hyperelliptic curves over $\mathbb{C}$

We start by defining hyperelliptic curves over any field, then specialize over $\mathbb{C}$.

Definition 1.4.1 ([CFA+10, Section 14.1])**.** Let K be a (perfect) field. A curve given by an equation of the form

$$C : y^2 + h(x)y = f(x)$$

with $f, h \in K[X], \deg f = 2g+1, \deg h \leq g$ and f monic is called a *hyperelliptic curve of genus g over K* if no point (x, y) on the curve over $\overline{K}$, the algebraic closure of K, satisfies both partial derivatives $2y(x) + h(x) = 0$ and $f'(x) - h'(x)y = 0$. Furthermore, we add to this curve a single *point at infinity*, which we denote P_∞.

The curve and the point at infinity can be defined using a projective equation; we refer to e.g. [Gal12, Def. 10.1.10] for details.

Group law on hyperelliptic curves

Much as in genus 1, one can define a group law from hyperelliptic curves of genus g. However, unlike in genus 1, the elements of the group are not simply points of the curve, but rather, some specific sums of points.

Definition 1.4.2 ([Gal12, Section 7.6])**.** Let C be a hyperelliptic curve of genus g over K. A *divisor over K* is defined as formal sum of points with finite support, i.e.

$$D = \sum_{P \in C(K)} n_P P, \qquad n_P \in \mathbb{Z},$$

with $n_P = 0$ for all but finitely many P, and furthermore such that $\sigma(D) = D$ for all σ in the Galois group of K. The set of divisors of a curve is denoted $\mathrm{Div}_K(C)$.

Definition 1.4.3 ([Gal12, Section 7.6])**.** For a divisor $D \in \mathrm{Div}_K(C)$, we define its *degree* as

$$\deg(D) = \sum_{P \in C(k)} n_P \in \mathbb{Z}.$$

We define $\mathrm{Div}^0_K(C)$ as the set of divisors of C of degree 0.

Both $\mathrm{Div}_K(C)$ and $\mathrm{Div}^0_K(C)$ are groups, the group law being simply the addition of finite sums; $\mathrm{Div}^0_K(C)$ is a subgroup of $\mathrm{Div}_K(C)$.

Definition 1.4.4 ([Gal12, Section 7.7])**.** Let $f \in K(C)^*$ a non-zero function defined over the curve (an element of the function field associated to the curve). Then f has finitely many zeroes and poles (see e.g. [Gal12, Thm. 7.7.1]). Define the *divisor associated to f* as

$$\mathrm{div}(f) = \sum_{P \in C(K)} v_P(f) P$$

where $v_P(f)$ is 0 if P is not a pole or a zero of f, n if P is a zero of order n of f, and $-n$ if it is a pole of order n of f.

The divisor of a function is also called a *principal divisor*. We then define

$$\mathrm{Prin}_K(C) = \{\mathrm{div}(f), f \in K(C)^*\},$$

the group of principal divisors. We have:

Proposition 1.4.5 ([Gal12, Thm. 7.7.11])**.** $\mathrm{Prin}_K(C) \subset \mathrm{Div}^0_K(C)$*, i.e. for all* $f \in K(C)$*,* $\deg \mathrm{div}(f) = 0$*.*

Finally, we define the group associated to a hyperelliptic curve:

Definition 1.4.6. We define the *Picard group* or the *divisor class group* of an hyperelliptic curve as

$$\mathrm{Pic}^0_K(C) = \mathrm{Div}^0_K(C)/\,\mathrm{Prin}_K(C).$$

This group is identified with the *Jacobian of* C, denoted $\mathrm{Jac}(C)$, an algebraic variety which is isomorphic to $\mathrm{Pic}^0_K(C)$ on every extension field of K.

Remark 1.4.7. The situation is simpler in the case of genus 1 elliptic curves, as we have $\mathrm{Jac}(C) \simeq C$; we refer to [Gal12, Section 7.9] for a proof.

Representation and addition of divisors

Proposition 1.4.8 ([CFA+10, Thm. 14.5])**.** *Each divisor class in the Jacobian of the curve can be represented using* Mumford coordinates*, i.e. a unique pair of polynomials* $u, v \in K[X]$ *such that*

- u *is monic;*
- $\deg v < \deg u \leq g$*;*
- $u \mid v^2 + vh - f$*.*

For instance, if $D = \sum_{i=1}^r P_i - rP_\infty$*, with* $P_i \neq P_j$*, one can take* $u = \prod_{i=1}^r (X - x_i)$*.*

Mumford coordinates are the usual representation of divisors on hyperelliptic curves.

Remark 1.4.9. Addition of divisors in Mumford representation can be computed using *Cantor's algorithm*, which actually generalizes the chord-and-tangent process to any genus g. We refer to [CFA+10, Algorithm 14.7] or [Gal12, Section 10.3] for an exposition of this algorithm; we actually will not need to use it in this manuscript.

Hyperelliptic curve cryptography

As with elliptic curves, the Jacobian of a hyperelliptic curve and its associated group law can be used in cryptography, via the *hyperelliptic curve discrete logarithm problem* (HECDLP):

$$\text{Given } P \in \mathrm{Jac}(\mathcal{C}(\mathbb{F}_q)) \text{ and } Q = [n]P, \text{ compute } n.$$

Hasse-Weil's theorem for hyperelliptic curves shows that $\sharp\,\mathrm{Jac}(\mathcal{C}(\mathbb{F}_q)) = O(q^g)$. Since the complexity of methods such as Pollard's rho have a complexity depending on the cardinal of the group, this means one can afford to take a smaller q (and hence a smaller finite field) to reach the same level of security as with an elliptic curve. This comes at the cost of a more involved arithmetic,

although some settings (such as the arithmetic on the genus 2 Kummer surface which we define at Section 2.6.4) allow a fast arithmetic, which can even be faster than optimized arithmetic on elliptic curves [BCHL16].

However, hyperelliptic curves of high genus are vulnerable to *index calculus attacks*, as described in Gaudry's seminal paper [Gau00]. If g is fixed, the attack of [GTTD07] shows for example that one can solve the HECDLP in $O(q^{2-2/g})$ operations, which is faster than generic methods (i.e. Pollard's rho) for $g \geq 3$. If, on the contrary, q is fixed (or at least does not grow too fast in size compared to the genus), [EGT11] shows there is actually an algorithm in time subexponential in q^g which solves the DLP. Finally, we also mention that another algorithm shows that the DLP on non-hyperelliptic genus 3 curves can be solved with only $O(q)$ operations [DT08]; since some genus 3 hyperelliptic curves are isogenous to non-hyperelliptic curves (and the isogeny is explicitly computable using [Smi09]), the HECDLP in genus 3 can sometimes be solved more quickly. In practice, genus 2 hyperelliptic curves are the most relevant case to cryptography, as they are not vulnerable to any of these attacks.

Complex hyperelliptic curves

Hyperelliptic curves over $\mathbb{C}$, and most generally complex abelian varieties, can be connected to the following:

Definition 1.4.10 ([CFA+10, Section 5.1.3])**.** Let $g \geq 1$, and let Λ be a lattice in $\mathbb{C}^g$, i.e. a discrete $\mathbb{Z}$-submodule of $\mathbb{C}^g$ of rank $2g$. Then $\mathbb{C}^g/\Lambda$ is a *complex torus of genus g*; it has a (complex Lie) group structure using addition modulo Λ.

We have the following general theorem:

Proposition 1.4.11 ([CFA+10, Corollary 5.17])**.** *Every abelian variety $\mathcal{A}$ of dimension g is isomorphic to a complex torus $\mathbb{C}^g/\Lambda$ with $\Lambda = \mathbb{Z}^g + \Omega\mathbb{Z}^g$, where Ω is symmetric and has a postive definite imaginary part. We call Ω the* period matrix *of $\mathcal{A}$.*

In particular, for a hyperelliptic curve of genus g C, we can write

$$\mathrm{Jac}(C) \simeq \mathbb{C}^g/(\mathbb{Z}^g + \Omega\mathbb{Z}^g).$$

Remark 1.4.12. Much as in genus 1, it is very easy to evaluate maps (and in particular isogenies) using the representation of a hyperelliptic curve as a torus. Indeed, for Λ_1, Λ_2 two lattices of $\mathbb{C}^g$, and for any α a $g \times g$ complex matrix such that $\alpha\Lambda_1 \subset \Lambda_2$, we can define the map

$$\begin{cases} \phi_\alpha : \mathbb{C}^g/\Lambda_1 & \to \mathbb{C}^g/\Lambda_2 \\ z & \mapsto \alpha z \pmod{\Lambda_2} \end{cases}$$

Furthermore one can prove (see e.g. [Cos11, Prop. 2.3.1]) that these maps are the only holomorphic maps preserving 0. This property is used in e.g. [VW99, VW00] to compute maps between genus 2 curves; one can use it to evaluate complex isogenies as with the method we present in Chapter 9.

1.4.2 The Abel-Jacobi map

The propositions from this section are taken from [CFA+10, Section 5.1.2].

Proposition 1.4.13. *Let C be a hyperelliptic curve of genus g; then the space of holomorphic differentials on C is of dimension g.*

Proposition 1.4.14. *Let C be a hyperelliptic curve of genus g and choose a basis $(\omega_1, \ldots, \omega_g)$ of the space of holomorphic differentials on C. Consider the set of paths starting and ending at a fixed point P_0, and identify homologous paths, i.e. paths that can be deformed continuously into one another. The resulting set gives a free abelian group $H_1(C, \mathbb{Z})$ with $2g$ generators, and the map*

$$\begin{aligned} \phi : H_1(C, \mathbb{Z}) &\to \mathbb{C}^g \\ \alpha &\mapsto \left(\int_\alpha \omega_1, \ldots, \int_\alpha \omega_g \right) \end{aligned}$$

satisfies $\operatorname{Im} \phi = \Lambda$*, a full-rank lattice in* $\mathbb{C}^g$*.*

Remark 1.4.15. An explicit construction of the $2g$ generators A_i, B_i of the group $H_1(C, \mathbb{Z})$ is given in [Mum84, Section III.5], complete with pictures; it is the construction which is generally used for hyperelliptic curves. The A_i are constructed as paths circling around the branch cuts chosen for the points a_i, the roots of the polynomial f which defines the hyperelliptic curve; the B_i can be taken as paths going across the branch cuts, so that A_i meets B_j if and only if $i = j$. (We refer to [Mum84, p. 76] or [Cos11, p. 28] for pictures.) The period matrix can then be computed as $\Omega_A^{-1}\Omega_B$ where the columns of Ω_A (resp. Ω_B) are the images of the A_i (resp. B_i) by the map ϕ outlined in the previous proposition; alternatively, one can choose a normalized basis of holomorphic differentials such that $\phi(A_i) = 1$, and define Ω with $\Omega_{i,j} = \int_{B_i} \omega_j$.

We now define the Abel-Jacobi map:

Theorem 1.4.16. *Let C be a hyperelliptic curve of genus g; let $(\omega_1, \ldots, \omega_g)$ be a basis of the space of holomorphic differentials, and let Λ be as in the previous proposition. Define the* Abel-Jacobi map

$$\begin{aligned} J : C(\mathbb{C}) &\to \mathbb{C}^g/\Lambda \\ P &\mapsto \left(\int_{P_\infty}^P \omega_1, \ldots, \int_{P_\infty}^P \omega_g \right) \pmod{\Lambda}. \end{aligned}$$

This map is well-defined, and can be extended to divisors by linear extension. Then for any $D \in \operatorname{Prin}_{\mathbb{C}}(C)$, $J(D) = 0$; hence, J induces a map from $\operatorname{Jac}_{\mathbb{C}}(C)$ to $\mathbb{C}^g/\Lambda$, and this map is a group homomorphism.

We outline in Chapter 8 algorithms to compute this map in quasi-linear time, using techniques which do not require to evaluate the hyperelliptic integrals which define the map. One can also compute the inverse of the Abel-Jacobi map using genus g theta functions; we outline these algorithms in the same chapter.

Chapter 2

Background on theta functions

In this chapter, we introduce the θ function in all generality; this function will be featured in every subsequent chapter of this manuscript. This complex function of two variables, z and τ, has numerous applications in mathematics; what we are most interested in are its links to algebraic geometry and complex Riemann surfaces. We show here all the properties that we will use in this manuscript; the last sections of this chapter outline those properties in the genus 1 and 2 settings, on which our manuscript is focused for the most part.

2.1 Definition

We start by defining the theta function. Most of the background we need is described in [Mum83, Chapter 2].

Definition 2.1.1 ([Mum83, Section II.1])**.** The *Siegel upper-half space* in dimension g, $\mathcal{H}_g$, is the set of symmetric $g \times g$ matrices whose imaginary part is positive definite.

Definition 2.1.2 (θ function)**.** Let $z \in \mathbb{C}^g$ and $\tau \in \mathcal{H}_g$. The θ function is defined as

$$\theta(z,\tau) = \sum_{n\in\mathbb{Z}^g} \exp(i\pi\, {}^t n\tau n)\exp(2i\pi\, {}^t nz).$$

Definition 2.1.3 (θ functions with characteristics)**.** Let $z \in \mathbb{C}^g$, $\tau \in \mathcal{H}_g$ and $a, b \in \frac{1}{2}\mathbb{Z}^g/\mathbb{Z}^g$. The theta function with characteristic $[a;b]$ is defined as

$$\theta_{[a;b]}(z,\tau) = \sum_{n\in\mathbb{Z}^g} \exp(i\pi\, {}^t(n+a)\tau(n+a))\exp(2i\pi\, {}^t(n+a)(z+b)).$$

Definition 2.1.4 (theta-constants)**.** Let $\tau \in \mathcal{H}_g$; the theta-constants are the values at $z = 0$ of the functions $\theta_{[a;b]}$.

Remark 2.1.5. We will often use a certain numbering of the theta functions with characteristics, to lighten notations. The numbering we choose to use is the one used e.g. in [Dup06, Cos11, ET14a]:

$$\theta_{[a;b]} = \theta_i, \qquad i = 2(b_0 + 2b_1 + ... + 2^{g-1}b_{g-1}) + 2^{g+1}(a_0 + 2a_1 + ...2^{g-1}a_{b-1}$$

In other words, the integer we associate with the theta function has the same binary expansion as $(2a||2b)$. Other numbering schemes are possible (see for instance the discussion in [Cos11, p. 37]), with other pros and cons attached to them.

We often deal with the functions $\theta_0, ..., \theta_{2^g-1}$ (i.e. the ones with $a = 0$), sometimes called *fundamental theta functions*.

Remark 2.1.6. Throughout the manuscript, we may use the following shorthands, for notational convenience: $\theta_{i,j}(z,\tau)$ instead of $\theta_i(z,\tau), \theta_j(z,\tau)$; $\theta_{i,j,k}(z,\tau)$ instead of $\theta_i(z,\tau), \theta_j(z,\tau), \theta_k(z,\tau)$; etc.

The θ function is quasi-periodic with respect to the lattice $\mathbb{Z} + \tau\mathbb{Z}$, in the following sense:

Proposition 2.1.7 (([Mum83, p. 120-123])). *For all $m \in \mathbb{Z}^g$, we have*

$$\begin{aligned}\theta_{[a;b]}(z+m,\tau) &= \exp(2i\pi\, {}^t a m)\theta_{[a;b]}(z,\tau) \\ &\qquad \textit{(in particular, } \theta_{[0;b]} \textit{ is invariant by } z \to z+m\textit{)} \\ \theta_{[a;b]}(z+\tau m,\tau) &= \exp(-2i\pi\, {}^t b m)\exp(-i\pi\, {}^t m\tau m)\exp(-2i\pi\, {}^t m z)\theta_{[a;b]}(z,\tau)\end{aligned}$$

One can also study the parity of θ with respect to the first argument:

Proposition 2.1.8. *We have $\theta_{[a;b]}(-z,\tau) = (-1)^{4\,{}^t ab}\theta_{[a;b]}(z,\tau)$. This means in particular that the function $z \mapsto \theta_{[a;b]}(z,\tau)$ is even if and only if $4\,{}^t ab \equiv 0 \pmod 2$, and odd otherwise.*

This means there are $2^{g-1}(2^g+1)$ even theta functions, and $2^{g-1}(2^g-1)$ odd ones; the theta constant associated with an odd theta function is 0. In genus 1, the only odd theta function is θ_3; in genus 2, the odd ones are $\theta_5, \theta_7, \theta_{10}, \theta_{11}, \theta_{13}, \theta_{14}$.

The sequence $\theta(z, 2^n\tau)$ has a rather central place in the algorithms we present in this manuscript (see Chapters 3, 4, 6, 7). We establish its limit in the following proposition:

Proposition 2.1.9. *For any $b \in \frac{1}{2}\mathbb{Z}^g/\mathbb{Z}^g$ we have*

$$\lim_{n\to\infty} \theta_{[0;b]}(z, 2^n\tau) = 1, \qquad \textit{and even} \qquad \lim_{n\to\infty} \theta_{[0;b]}(z, 2^n\tau)^{2^n} = 1.$$

Proof. Let R an orthogonal matrix such that ${}^t R\,\mathrm{Im}(\tau)R = \mathrm{Diag}(\lambda_1, \ldots, \lambda_g)$, where the λ_i denote the eigenvalues of $\mathrm{Im}(\tau)$. Write

$$\begin{aligned}|\theta_{[0;b]}(z,\tau) - 1| &\le \sum_{n\in\mathbb{Z}^g\setminus\{0\}} |e^{i\pi\, {}^t n\tau n + 2i\pi\, {}^t nz}| \\ &\le \sum_{n\in\mathbb{Z}^g\setminus\{0\}} e^{-\pi\, {}^t (Rn)\mathrm{Diag}(\lambda_1,\ldots,\lambda_g)Rn} e^{-2\pi\, {}^t n\,\mathrm{Im}(z)} \\ &\le 2^g \sum_{n\in\mathbb{N}^g\setminus\{0\}} q^{n_1^2+\ldots+n_g^2} w^{-2\sum|n_i|}\end{aligned}$$

with $q = e^{-\pi\lambda}$, where λ is the smallest eigenvalue of $\mathrm{Im}(\tau)$, and $w = e^{-\pi \max_i |\mathrm{Im}(z_i)|}$. Hence $|\theta(z, 2^k\tau) - 1| \le 2^g \sum_{n\in\mathbb{N}^g\setminus\{0\}} q^{2^k(n_1^2+\ldots+n_g^2)} w^{-2\sum n_i}$. Let k_0 be such that $2^{k_0}\lambda \ge 2\max_i|\mathrm{Im}(z_i)|$; then for $k \ge k_0$,

$$\begin{aligned}|\theta_{[0;b]}(z, 2^k\tau) - 1| &\le 2^g \sum_{n\in\mathbb{N}^g\setminus\{0\}} q^{(2^k-2^{k_0})(n_1^2+\ldots+n_g^2)} \\ &\le 4^g \frac{q^{2^k-2^{k_0}}}{1-q^{2^k-2^{k_0}}} \left(1 + \frac{q^{2^k-2^{k_0}}}{1-q^{2^k-2^{k_0}}}\right)^{2^g-1}\end{aligned}$$

which proves the first statement. As for the second one, write $\theta_{[0;b]}(z, 2^k\tau) = 1 + c$, with $c \sim_{k\to\infty} c' q^{2^k}$, which goes to 0 as k grows; then $\theta_{[0;b]}(z, 2^k\tau)^{2^k} - 1 \sim 2^k c' q^{2^k}$, which also goes to 0 as k grows. □

The second statement refines the first one in a way that ensures quasi-optimal time in our algorithms (see Sections 6.2.3 and 7.2.2).

2.2 Addition and duplication formulas

2.2.1 τ-duplication formula

The following formula is of great importance to this manuscript.

Proposition 2.2.1. *For all* $a, b \in \frac{1}{2}\mathbb{Z}^g/\mathbb{Z}^g$,

$$\theta_{[a;b]}(z,\tau)^2 = \frac{1}{2^g} \sum_{\beta \in \frac{1}{2}\mathbb{Z}^g/\mathbb{Z}^g} e^{-4i\pi\, {}^t a\beta} \theta_{[0;b+\beta]}\left(z, \frac{\tau}{2}\right) \theta_{[0;\beta]}\left(0, \frac{\tau}{2}\right). \tag{2.2.1}$$

This formula can be found in [Cos11, formula 3.13], where it is called the change of basis formula from the $\mathcal{F}_2$ basis to the $\mathcal{F}_{(2,2)^2}$ basis; it is derived from [Igu72, Section IV.1, Theorem 2] by taking $m_1 = m_2$ and $z_1 = z_2$.

The consequences of this formula are numerous:

1. Taking $z = 0$ in Equation (2.2.1) shows a relationship between theta-constants at τ and theta-constants at 2τ. A closer look at the equation one gets shows a similarity with the classic *arithmetico-geometric mean* (in genus 1) and the *Borchardt mean* (in genus g). The link between those quadratically convergent means and theta-constants is outlined in Chapter 3; it is the basis of the fast algorithms for theta-constants discussed in Chapter 6 and 7.

2. Equation (2.2.1) is at the basis of our fast algorithms for θ in Chapter 6 and 7.

3. The value at τ of the *fundamental theta functions*, i.e. the ones with $a = 0$, along with the value of the fundamental theta-constants, can be used to recover the value of all the theta functions and theta-constants at 2τ. In particular, should one want to compute the value of all the theta functions at τ, a valid strategy is to compute the value of all the fundamental ones at $\tau/2$; we use this strategy for instance in Chapter 6. We also use this equation repeatedly in the same chapter to get an algorithm whose complexity is uniform in z, τ.

4. Equation (2.2.1) can be combined with Proposition 2.1.9, and the fact that $\sum_\beta (-1)^{{}^t a\beta} = 0$, to prove that, for any $a \in \frac{1}{2}\mathbb{Z}^g/\mathbb{Z}^g \setminus \{0\}$,

$$\lim_{n\to\infty} \theta_{[a;b]}(z, 2^n\tau) = 0.$$

2.2.2 Riemann formulas

The following set of formulas, often called *Riemann formulas*, give relationships between values of θ in any genus:

Theorem 2.2.2 ([Igu72, Theorem IV.1.1])**.** *Let* $(m_1, m_2, m_3, m_4) \in \mathbb{R}^{8g}$ *and* $(z_1, z_2, z_3, z_4) \in \mathbb{C}^{4g}$. *Let*

$$T_g = \frac{1}{2}\begin{pmatrix} I_g & I_g & I_g & I_g \\ I_g & I_g & -I_g & -I_g \\ I_g & -I_g & I_g & -I_g \\ I_g & -I_g & -I_g & I_g \end{pmatrix},$$

and put

$$\begin{aligned}(n_1, n_2, n_3, n_4) &= (m_1, m_2, m_3, m_4)T_{2g},\\ (w_1, w_2, w_3, w_4) &= (z_1, z_2, z_3, z_4)T_g\end{aligned}$$

Then we have (with m_1' the first g coordinates of m_1)

$$\theta_{m_1}(z_1,\tau)\dots\theta_{m_4}(z_4,\tau) = \frac{1}{2^g}\sum_{(\alpha,\beta)} e^{-2^t m_1'\beta}\theta_{n_1+(\alpha,\beta)}(w_1,\tau)\dots\theta_{n_4+(\alpha,\beta)}(w_4,\tau)$$

in which (α,β) runs over a complete set of representatives of $\frac{1}{2}\mathbb{Z}^{2g}/\mathbb{Z}^{2g}$.

Riemann formulas are complementary to the τ-duplication formulas, since they encode a relationship between theta functions with the same τ (but at different z). They can be instantiated in many different ways; we mention in the next section the existence of z-duplication formulas, which are obtained from the Riemann formulas. For other applications, we refer to [Cos11, Chapter 3].

2.3 Reduction of the first argument

This manuscript describes algorithms for the computation of θ (e.g. Chapter 5, Chapter 6 and Chapter 7). We show how one can perform argument reduction in order to restrict ourselves to more favorable cases. We start by describing argument reduction of the first argument of θ, before describing argument reduction for the second argument in Section 2.4.

2.3.1 Quasi-periodicity

The most obvious way to perform argument reduction is to use the quasi-periodicity of $z \mapsto \theta(z,\tau)$ (i.e. Proposition 2.1.7). In all generality, one can expect to achieve the conditions

$$|\mathrm{Re}(z_i)| \le \frac{1}{2}; \qquad |\mathrm{Im}(z_i)| \le \frac{\sum_{j\in[1..g]}|\mathrm{Im}(\tau_{i,j})|}{2}. \tag{2.3.1}$$

using the quasi-periodicity of θ. We say that z is *reduced* if the above conditions are met; this corresponds to $z = x + \tau y$ with $x, y \in \mathbb{R}^g$ and $|x_i|, |y_i| \le \frac{1}{2}$. The value of $\theta(z,\tau)$ can then be obtained from $\theta(z',\tau)$ (with z' reduced) simply by computing an exponential.

However, note that this exponential factor can become quite big. This would not cause problems if one wanted to compute θ to some relative precision P; however, in this manuscript, we wish to compute θ to absolute precision P, i.e. up to 2^{-P}, since $\theta(z,\tau)$ can be close to 0. This means that the exponential factor should be taken into account: if one wants to compute $\theta(z,\tau)$ up to 2^{-P} and use argument reduction, the value $\theta(z',\tau)$ must be computed up to 2^{-P-C}, with C the size of the exponential factor.

Hence, the final complexity of any method relying on this argument reduction will depend on τ and z; however, since the size of the final result depends on it too, we consider this as inevitable. Supposing that z is reduced allows us to essentially (i.e. along with other hypotheses on τ) work on values of θ of bounded size, which is more comfortable, and allows us to get complexities which depend only on P – with the understanding that recovering the original value using argument reduction has a complexity depending on the original values of z and τ.

Remark 2.3.1. In genus 1, writing $z = x + \tau y$ with $|x|, |y| \leq \frac{1}{2}$ is rather easy; we simply subtract $k\tau$ from z with k the integer closest to $\mathrm{Re}\left(\frac{z}{\tau}\right)$), then we subtract the integer which is closest from the real part. In the general case, we can write explicitly $z = x + \tau y$ with $x, y \in \mathbb{R}^{2g}$ as follows. Put $\Lambda = \begin{pmatrix} I_g & \tau \end{pmatrix} \in \mathcal{M}_{g\times 2g}(\mathbb{C})$, so that the lattice associated to τ is $\Lambda\mathbb{Z}^{2g}$. We then have $\begin{pmatrix} \Lambda \\ \overline{\Lambda} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} z \\ \overline{z} \end{pmatrix}$, and we can compute x, y easily. It is then easy to subtract $k + \tau k'$ with $k, k' \in \mathbb{Z}^g$ to z in order to get the reduced z.

Remark 2.3.2. In the case of genus 1, the function $z \mapsto \theta_i(z, \tau)$ is odd for $i = 3$ and even for the other ones. Hence, without loss of generality, one can suppose that $\mathrm{Im}(z) \geq 0$.

2.3.2 z-duplication formulas

One last way to perform argument reduction is to use the so-called z-duplication formulas. These formulas can be derived from the Riemann formulas mentioned in the previous section; they fall in the more general class of *addition formulas*:

Proposition 2.3.3 ([Igu72, Section IV.1])**.** *Let m_1, m_2, m_3, m_4 denote elements of $\mathbb{R}^{2g}$ and $u, v \in \mathbb{C}$. Put*

$$(n_1, n_2, n_3, n_4) = (m_1, m_2, m_3, m_4)T$$

where T is the matrix defined in Theorem 2.2.2. Then, omitting τ from the equation, Riemann's formulas give (with m_1' the first g coordinates of m_1)

$$\theta_{m_1}(u+v)\theta_{m_2}(u-v)\theta_{m_3}(0)\theta_{m_4}(0) = \frac{1}{2^g}\sum_{(\alpha,\beta)} e^{-2^t m_1'\beta}\theta_{n_1+(\alpha,\beta)}(u)\theta_{n_2+(\alpha,\beta)}(u)\theta_{n_3+(\alpha,\beta)}(v)\theta_{n_4+(\alpha,\beta)}(v)$$

in which (α, β) runs over a complete set of representatives of $\frac{1}{2}\mathbb{Z}^{2g}/\mathbb{Z}^{2g}$. Taking $u = v$ gives the z-duplication formulas

$$\theta_{m_1}(2z)\theta_{m_2}(0)\theta_{m_3}(0)\theta_{m_4}(0) = \frac{1}{2^g}\sum_{(\alpha,\beta)} e^{-2^t m_1'\beta}\theta_{n_1+(\alpha,\beta)}(z)\theta_{n_2+(\alpha,\beta)}(z)\theta_{n_3+(\alpha,\beta)}(z)\theta_{n_4+(\alpha,\beta)}(z)$$

We derive one more formula, who is particularly interesting for our purposes: if $\theta_a(0, \tau) \neq 0$, then

$$\theta_a(2z, \tau) = \frac{\sum_{\alpha,\beta} e^{-2^t m_1'\beta}\theta^4_{a+(\alpha,\beta)}(z, \tau)}{2^g\theta_a^3(0, \tau)}. \tag{2.3.2}$$

This gives $\theta(2z, \tau)$ from the knowledge of $\theta(z, \tau)^2$.

Proposition 2.3.4. *The value of $\theta(z, \tau)$ with absolute precision P can be computed from the computation of the $\theta_i\left(\frac{z}{2^k}, \tau\right)$ with absolute precision $P+p_k$ and the application of the z-duplication formulas.*

We use z-duplication formulas, interwoven with τ-duplication formulas, in Chapter 6, in which we also analyse the precision loss incurred. In genus g, we conjecture that $p_k = O(k)$.

2.4 Reduction of τ via the symplectic group

2.4.1 Symplectic group

Definition 2.4.1 ([Kli90, Def. I.1.1])**.** The *symplectic group* of dimension g, denoted $\mathrm{Sp}_{2g}(\mathbb{Z})$ is the set of matrices $\begin{pmatrix} A & B \\ C & D \end{pmatrix}$ with $A, B, C, D \in \mathcal{M}_g(\mathbb{Z})$ such that:

$$ {}^tAC = {}^tCA, \qquad {}^tBD = {}^tDB, \qquad {}^tAD - {}^tCB = I_g. $$

Proposition 2.4.2 ([Kli90, Prop. I.1.1])**.** $\mathrm{Sp}_{2g}(\mathbb{Z})$ *acts on* $\mathcal{H}_g$ *as follows:*

$$ \begin{array}{rcl} \mathrm{Sp}_{2g}(\mathbb{Z}) \times \mathcal{H}_g & \to & \mathcal{H}_g \\ \left(M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}, \tau \right) & \mapsto & M \cdot \tau = (A\tau + B)(C\tau + D)^{-1} \end{array} $$

Furthermore, M *defines an isomorphism of complex tori between* $\Lambda_\tau = \frac{\mathbb{C}^g}{\tau\mathbb{Z}^g + \mathbb{Z}^g}$ *and* $\Lambda_{M\cdot\tau}$ *as follows:*

$$ \begin{array}{rcl} \Lambda_\tau & \to & \Lambda_{M\cdot\tau} \\ z & \mapsto & M \cdot_\tau z = {}^t(C\tau + D)^{-1} z \end{array} $$

where the shorthand $M \cdot z$ *may be used when the context allows.*

Remark 2.4.3. In genus 1, the symplectic group is simply the group $\mathrm{SL}_2(\mathbb{Z})$ of matrices $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ with $a, b, c, d \in \mathbb{Z}$ and $ad - bc = 1$. The action is $\tau \mapsto \frac{a\tau+b}{c\tau+d}$.

2.4.2 Action of the symplectic group on θ

The following theorem shows how θ functions behave under the action of $\mathrm{Sp}_{2g}(\mathbb{Z})$ on $\mathcal{H}_g$.

Theorem 2.4.4. *Let* $M = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \in \mathrm{Sp}_{2g}(\mathbb{Z})$*, and* $(z, \tau) \in \mathbb{C}^g \times \mathcal{H}_g$*. We have the functional equation:*

$$ \theta_i\left(M \cdot z, M \cdot \tau\right) = \zeta_M \sqrt{\det(C\tau + D)} e^{i\pi\, {}^t(M\cdot z)(Cz)} \theta_{\sigma_M(i)}(z, \tau) $$

where σ_M *is a permutation and* ζ_M *is an eighth root of unity.*

This theorem is proven in [Mum83, Section II.5] in a special case, and in [Igu72, Chapter 5, Theorem 2]; an outline of the proof can also be found in [Cos11, Prop. 3.1.24]. This allows to perform argument reduction:

Proposition 2.4.5. *For any* $z \in \mathbb{C}^g$ *and* $\tau \in \mathcal{H}_g$*, the value of* $\theta(z, \tau)$ *with absolute precision* P *can be computed from the value with absolute precision* $P + c$ *of a* $\theta_i(z', \tau')$*, for some* i *and some* $\tau' \in \mathcal{F}_g$*. Computing the final result requires the computation of a square root and an exponential, for a cost of* $O(\mathcal{M}(P + c) \log(P + c))$*, on top of the cost of computing* $\theta_i(z', \tau')$*. The constant* c *depends on* z, τ, z', τ'*.*

The domain $\mathcal{F}_g$ is defined in the next section. The constant c is necessary to account for the size of the exponential factor; it can be computed using e.g. a low-precision approximation of $\frac{\theta(z,\tau)}{\theta_i(z',\tau')}$.

2.4.3 Fundamental domain for τ

Definition 2.4.6 ([Kli90, Def. I.3.1])**.** The set $\mathcal{F}_g \subset \mathcal{H}_g$ is defined as the matrices satisfying the conditions:

- $\mathrm{Im}(\tau)$ is Minkowski-reduced, i.e. ${}^t g\,\mathrm{Im}(\tau) g \geq \mathrm{Im}(\tau_{k,k})$ for all integral g with $(g_k, \ldots, g_n) = 1$ and $\mathrm{Im}(\tau_{k,k+1}) \geq 0$ for all k;
- $|\mathrm{Re}(\tau_{k,l})| \leq \frac{1}{2}$ for all $k, l \in \{1, ..., n\}, k \leq l$;
- $|\det(C\tau + D)| \geq 1$ for all $\begin{pmatrix} A & B \\ C & D \end{pmatrix} \in \mathrm{Sp}_{2g}(\mathbb{Z})$.

Remark 2.4.7. Computing the Minkowski reduction of a $g \times g$ matrix is, in general, a hard problem. Note that a Minkowski-reduced matrix gives the shortest vector of a lattice, i.e. Minkowski reduction solves the SVP. This problem is a hard problem for which there are currently no subexponential algorithms, only exponential ones; it is a standard hardness assumption in lattice-based cryptography. The best known complexity for computing the Minkowski reduction of a $g \times g$ matrix is $O\left(2^{1.3g^3}\right)$ arithmetic operations [Hel85]. However, note that there is an algorithm for Minkowski reduction in dimensions up to 4 whose running time is polynomial in the size of the longest vector of the given basis [NS04].

This is the *fundamental domain* for the action of $\mathrm{Sp}_{2g}(\mathbb{Z})$ on $\mathcal{H}_g$:

Theorem 2.4.8 ([Kli90, Section I.3])**.** *$\mathcal{F}_g$ is the fundamental domain of the action of $\mathrm{Sp}_{2g}(\mathbb{Z})$ on $\mathcal{H}_g$ – that is to say, for any $\tau \in \mathcal{H}_g$ there is $\gamma \in \mathrm{Sp}_{2g}(\mathbb{Z}), \tau' \in \mathcal{F}_g$ such that $\tau = \gamma\tau'$, and such a τ' is unique if it is not on the boundary of $\mathcal{F}_g$.*

Note that the definition of $\mathcal{F}_g$ includes a condition that one must check on an infinite number of matrices; hence, this definition does not allow us to write an algorithm that would find the representative of a $\tau \in \mathcal{H}_g$ in the fundamental domain. However we have

Proposition 2.4.9 ([Kli90, Section I.3, Prop. 3])**.** *For any g, there exists a finite set $V_g \subset \mathrm{Sp}_{2g}(\mathbb{Z})$ such that the third condition defining $\mathcal{F}_g$ can be replaced by*

- $|\det(C\tau + D)| \geq 1$ *for all* $\begin{pmatrix} A & B \\ C & D \end{pmatrix} \in V_g$.

This proposition gives a procedure for reducing a τ into the fundamental domain $\mathcal{F}_g$; we outline it in Algorithm 1.

We use two subroutines here, which we do not make explicit:

- ReduceRealPart(τ') is a function that returns $\tau' - M$, with $M = (m_{ij})$ and $m_{ij} = \lfloor \mathrm{Re}(z_{ij}) \rceil$. This is equivalent to applying matrices of the form $T_{ij} = \mathcal{I}_g + \delta_{i,j}$, where $\delta_{i,j}$ is the Kronecker matrix, to τ. This subroutine does not require many operations to compute, and we ignore its cost.
- MinkowskiReduce is a function which takes a real matrix τ' as an input and outputs a real matrix γ that is Minkowski-reduced and a matrix M with integer coefficients such that $\gamma = M \cdot \tau'$. Computing the Minkowski reduction of a $g \times g$ matrix has an exponential cost, of $O\left(2^{1.3g^3}\right)$ arithmetic operations [Hel85].

Algorithm 1 Reducing τ to $\mathcal{F}_g$.
Input: $\tau \in \mathcal{H}_g$
Output: $\tau' \in \mathcal{F}_g$

1: $\tau' \leftarrow \tau$
2: $\tau' \leftarrow \text{ReduceRealPart}(\tau')$.
3: $v, M \leftarrow \text{MinkowskiReduce}(\text{Im}(\tau'))$; $\tau' \leftarrow M \cdot \tau'$.
4: **for** $M \in V_g$ **do**
5: **if** $|\det(C_M\tau' + D_M)| < 1$ **then**
6: $\tau' \leftarrow M \cdot \tau'$
7: **goto** 2
8: **end if**
9: **end for**
10: **return** τ'

This outline can be turned into an algorithm provided that V_g is known, or at least computable. However, the description in general of this finite set is not known, which does not help in making reduction to the fundamental domain effective: there is no known algorithm which allows one to compute V_g, and hence to reduce to the fundamental domain in genus $g \geq 3$. However, we note that, if we assume V_g is known, the resulting algorithms terminates, as proven in [Sie89, Chapter 6,Section 5].

Remark 2.4.10. In genus 1, the action of $\text{SL}_2(\mathbb{Z})$ on the upper-half plane is well-known and has been extensively studied; see for example [Mum83]. The fundamental domain $\mathcal{F}_1$ (sometimes denoted $\mathcal{F}$ when the context is clear) is defined as

$$\mathcal{F}_1 = \left\{ \tau \in \mathcal{H} \mid |\tau| \geq 1, |\text{Re}(\tau)| \leq \frac{1}{2} \right\}$$

Using the notations of Algorithm 1, $V_1 = \{S\}$ where $S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$. The resulting algorithm which reduces τ into $\mathcal{F}$ is Gauss's algorithm [VV09] to find a reduced basis of a 2-dimensional lattice (in this case, $\mathbb{Z} + \mathbb{Z}\tau$); its cost is asymptotically negligible. The fundamental domain is represented on Figure 2.1.

Remark 2.4.11. In genus 2, 19 inequalities $|\det(C\tau + D)| \geq 1$ defining the fundamental domain have been determined [Got59]. Each is required: for each inequality, there is a matrix outside the fundamental domain that satisfies the 18 other inequalities but not the chosen one. In the notations of Algorithm 1, this gives a set V_2 of cardinality 19; hence there is an algorithm to reduce in the fundamental domain in genus 2. The resulting algorithm is analyzed in [Str14, Section 6], who proves that it terminates in a number of steps only depending on τ.

2.4.4 Loosened requirements for τ for $g \geq 2$

As we saw above, there is currently no known algorithm to compute the reduction of an element $\tau \in \mathcal{H}_g$ into the fundamental domain $\mathcal{F}_g$. One could introduce a larger domain, $\mathcal{F}'_g$, for which a reduction algorithm is known.

Definition 2.4.12. The set $\mathcal{F}'_g \subset \mathcal{H}_g$ is defined as the matrices satisfying the conditions:

- $\text{Im}(\tau)$ is Minkowski-reduced, i.e. ${}^t g\,\text{Im}(\tau) g \geq \text{Im}(\tau_{k,k})$ for all integral g with $(g_k, ..., g_n) = 1$ and $\text{Im}(\tau_{k,k+1}) \geq 0$ for all k;

- $|\mathrm{Re}(\tau_{k,l})| \leq \frac{1}{2}$ for all $k, l \in \{1, ..., n\}, k \leq l$;
- $|\tau_{1,1}| \geq 1$.

This domain was introduced in genus 2 by Streng [Str14], who calls $\mathcal{B}$ what is called here $\mathcal{F}_2'$.

Put $N_0 = \begin{pmatrix} I_g - \delta_{1,1} & -\delta_{1,1} \\ \delta_{1,1} & I_g - \delta_{1,1} \end{pmatrix}$, where $\delta_{1,1}$ is the $g \times g$ Kronecker matrix (i.e. with top left coefficient equal to 1 and 0 everywhere else). This matrix is symplectic, and we have $|\det(C\tau + D)| = |\tau_{1,1}|$. This remark allows us to prove that

Proposition 2.4.13. *We have $\mathcal{F}_g \subset \mathcal{F}_g'$.*

Note that $\mathcal{F}_1 = \mathcal{F}_1'$.

The algorithm for reducing τ into $\mathcal{F}_g'$ is similar to Algorithm 1, but with the condition "$M \in V_g$" replaced by "$M = N_0$" (see [Str14, §6.3]). This is Algorithm 2.

Algorithm 2 Reducing τ to $\mathcal{F}_g'$.
Input: $\tau \in \mathcal{H}_g$
Output: $\tau' \in \mathcal{F}_g'$

1: $\tau' \leftarrow \tau$
2: $\tau' \leftarrow \mathrm{ReduceRealPart}(\tau')$.
3: $v, M \leftarrow \mathrm{MinkowskiReduce}(\mathrm{Im}(\tau'))$; $\tau' \leftarrow M \cdot \tau'$.
4: **if** $|\tau_{1,1}| < 1$ **then**
5: $\quad \tau' \leftarrow N_0 \cdot \tau'$
6: $\quad$ **goto** Step 2
7: **end if**
8: **return** τ'

Termination of Algorithm 2 in genus 2 (i.e. reduction to $\mathcal{F}_2'$) is proven in [Str14]. The termination of Algorithm 2 in the general, genus g case is an open problem. Note that a few lemmas used in the proof generalize to genus g, namely Lemma 6.9, Lemma 6.11 (for the set of matrices such that $y_1 \leq \frac{1}{t'}$ with $t' > 2$) and Lemma 6.12 (for the set of matrices such that $y_1 \geq \frac{1}{t'}$); unfortunately the generalization of Lemma 6.14 is unclear, as it relies on Equation 6.5, which is only valid in genus 2.

Another approach to argument reduction is to use so-called Siegel reduction, as in [DHB+04]. The conditions are even weaker than the ones we impose here; in particular, LLL reduction is used instead of Minkowski reduction. The article claims that this reduction is enough to limit the number of terms needed for the naive algorithm for θ, although no analysis is provided.

We believe that both reductions – reduction to $\mathcal{F}_g'$ and the explicit Siegel reduction of [DHB+04] – are relevant to our purposes, i.e. argument reduction in the context of the genus g θ function. The first reduction may be costlier, as Minkowski reduction has running time exponential in the genus while the LLL reduction runs in polynomial time; furthermore, termination has not been proven, although we believe it to hold. In either case, both reduction algorithms seem to give conditions similar to the ones we use when analyzing the naive algorithm for θ in Sections 5.1 and 5.2, which indicates that both reduction algorithms are relevant.

In the remainder of this manuscript, the reduction of τ to $\mathcal{F}_g'$ is the one we will mention the most often, and in particular in the analyses of Chapter 5; however, it should be understood that similar results could probably be found for the effective Siegel reduction described in [DHB+04].

2.5 Genus 1 instantiations

This short section is aimed at making the equations described so far more explicit in genus 1. We use quite a lot of them throughout this manuscript, but most importantly in Chapter 6, when describing our quasi-linear time algorithm for θ; such formulas are used both in the actual computation of θ and in the argument reduction strategy we set up in order to get a complexity that is independent of z and τ.

The genus 1 theta function, sometimes called *Jacobi theta function*, is defined as

Definition 2.5.1. Jacobi's theta function is defined as

$$\begin{array}{rcl} \mathbb{C}\times\mathcal{H} & \to & \mathbb{C} \\ (z,\tau) & \mapsto & \displaystyle\sum_{n\in\mathbb{Z}} e^{i\pi\tau n^2}e^{2i\pi zn} = 1+\sum_{n\in\mathbb{N}} q^{n^2}(w^{2n}+w^{-2n}) \\ & & \displaystyle = 1+q\left(w^2+\frac{1}{w^2}\right)+q^4\left(w^4+\frac{1}{w^4}\right)+\dots \end{array}$$

with $q=e^{i\pi\tau}$ (the "nome") and $w=e^{i\pi z}$.

There are four theta functions with characteristics, which we denote $\theta_0,\theta_1,\theta_2,\theta_3$ using the notation of Note 2.1.5:

$$\begin{array}{rcl} \theta_0(z,\tau) & = & \theta(z,\tau) = 1+q\left(w^2+\frac{1}{w^2}\right)+q^4\left(w^4+\frac{1}{w^4}\right)+\dots \\ \theta_1(z,\tau) & = & \theta\left(z+\frac{1}{2},\tau\right) = 1-q\left(w^2+\frac{1}{w^2}\right)+q^4\left(w^4+\frac{1}{w^4}\right)-\dots \\ \theta_2(z,\tau) & = & \exp(\pi i\tau/4+\pi iz)\theta\left(z+\frac{\tau}{2},\tau\right) \\ & = & q^{1/4}w\left(1+\frac{1}{w^2}+q^2\left(w^2+\frac{1}{w^4}\right)+q^6\left(w^4+\frac{1}{w^6}\right)+\dots\right) \\ \theta_3(z,\tau) & = & \exp(\pi i\tau/4+\pi i(z+1/2))\theta\left(z+\frac{\tau+1}{2},\tau\right) \\ & = & iq^{1/4}w\left(1-\frac{1}{w^2}+q^2\left(w^2-\frac{1}{w^4}\right)+q^6\left(w^4-\frac{1}{w^6}\right)-\dots\right) \end{array}$$

Furthermore, the expressions of the theta-constants are as follows:

$$\theta_0(0,\tau) = \sum_{n\in\mathbb{Z}} q^{n^2} = 1+2q^2+2q^4+2q^9+\dots \tag{2.5.1}$$

$$\theta_1(0,\tau) = \sum_{n\in\mathbb{Z}} (-1)^n q^{n^2} = 1-2q^2+2q^4-2q^9+\dots \tag{2.5.2}$$

$$\theta_2(0,\tau) = 2\sum_{n\geq 0} q^{(n+1/2)^2} = 2q^{1/4}+2q^{9/4}+2q^{25/4}+\dots \tag{2.5.3}$$

Recall that $\theta_3(0,\tau)=0$ since this is an odd theta function.

2.5.1 Duplication formulas

The τ-duplication formulas are as follows:

$$\begin{aligned}
\theta_0(z,2\tau)^2 &= \frac{\theta_0(z,\tau)\theta_0(0,\tau)+\theta_1(z,\tau)\theta_1(0,\tau)}{2} & \theta_0(0,2\tau)^2 &= \frac{\theta_0(0,\tau)^2+\theta_1(0,\tau)^2}{2} \\
\theta_1(z,2\tau)^2 &= \frac{\theta_0(z,\tau)\theta_1(0,\tau)+\theta_1(z,\tau)\theta_0(0,\tau)}{2} & \theta_1(0,2\tau)^2 &= \theta_0(0,\tau)\theta_1(0,\tau) \\
\theta_2(z,2\tau)^2 &= \frac{\theta_0(z,\tau)\theta_0(0,\tau)-\theta_1(z,\tau)\theta_1(0,\tau)}{2} & \theta_2(0,2\tau)^2 &= \frac{\theta_0(0,\tau)^2-\theta_1(0,\tau)^2}{2} \\
\theta_3(z,2\tau)^2 &= \frac{\theta_0(z,\tau)\theta_1(0,\tau)-\theta_1(z,\tau)\theta_0(0,\tau)}{2}
\end{aligned} \tag{2.5.4}$$

We will use the right column in Chapter 3, and the left column is a crucial component in Chapter 6. Note that a direct proof of these formulas using the definition of θ is also sometimes presented (e.g. [BB87]), which involves some manipulations and term reorganization akin to

$$\sum_{n+m\equiv 0 \pmod 2} q^{n^2+m^2} = \sum_{i,j\in\mathbb{Z}} q^{(i+j)^2+(i-j)^2}.$$

As for the z-duplication formulas, we will use:

$$\begin{aligned}
\theta_0(z,\tau)\theta_0^3(0,\tau) &= \theta_1^4(z/2,\tau)+\theta_2^4(z/2,\tau) \\
\theta_1(z,\tau)\theta_1^3(0,\tau) &= \theta_0^4(z/2,\tau)-\theta_2^4(z/2,\tau) \\
\theta_2(z,\tau)\theta_2^3(0,\tau) &= \theta_0^4(z/2,\tau)-\theta_1^4(z/2,\tau) \\
\theta_3(z,\tau)(\theta_0\theta_1\theta_2)(0,\tau) &= 2(\theta_0\theta_1\theta_2\theta_3)(z/2,\tau)
\end{aligned} \tag{2.5.5}$$

These will be used in Chapter 6.

2.5.2 Other equations

We highlight two more equations:

$$\theta_0(0,\tau)^4 = \theta_1(0,\tau)^4+\theta_2(0,\tau)^4 \tag{2.5.6}$$

$$\theta_0^2(z,\tau)\theta_0^2(0,\tau) = \theta_1^2(z,\tau)\theta_1^2(0,\tau)+\theta_2^2(z,\tau)\theta_2^2(0,\tau) \tag{2.5.7}$$

In [Mum83], the first one is named *Jacobi's quartic formula*, and the second one the *equation of the variety*. Those equations show another way than the τ-duplication formulas to recover θ_2 from the knowledge of the values of the fundamental theta functions.

Furthermore, one can recover $\theta_3(z,\tau)$ from the other values of θ, using:

$$\theta_3^2(z,\tau)\theta_0^2(0,\tau) = \theta_1^2(z,\tau)\theta_2^2(0,\tau)-\theta_2^2(z,\tau)\theta_1^2(0,\tau). \tag{2.5.8}$$

Finally, we mention a special formula, whose generalization to higher genera is not very obvious: *Jacobi's derivative formula*, which is

$$\theta_3'(0,\tau) = -\pi\theta_0(0,\tau)\theta_1(0,\tau)\theta_2(0,\tau)$$

We use this to prove a formula in Chapter 8. Generalizations to higher genera have been considered in [Igu80, Gra88]; we do not use any here.

2.5.3 Argument reduction

One can make Theorem 2.4.4 more explicit in the case of genus 1 and determine the correspondence $\gamma \mapsto \sigma_\gamma$, using [Mum83, p. 36] or by noticing it is independent of z and using the tables found by Gauss for theta-constants [Cox84, Eq. 2.15].

Theorem 2.5.2 (extension of [Mum83, Theorem 7.1]). *Let $\tau \in \mathcal{H}$ and $z \in \mathbb{C}$, and let $\gamma = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in SL_2(\mathbb{Z})$. Suppose $c > 0$, or $c = 0$ and $d > 0$; if not, take $-\gamma$. Then we have:*

$$\theta_i\left(\frac{z}{c\tau+d}, \frac{a\tau+b}{c\tau+d}\right) = \zeta_{i,\gamma,\tau}\sqrt{c\tau+d}\, e^{i\pi cz^2/(c\tau+d)}\theta_{\sigma(i)}(z,\tau)$$

where the square root is taken with positive real part, $\zeta_{i,\gamma,\tau}$ is an eighth root of unity and σ is a permutation of the elements $(00, 01, 10)$, defined by the following table:

a	b	c	d	$\sigma(00,01,10)$
odd	even	even	odd	$(00,01,10)$
odd	odd	even	odd	$(01,00,10)$
odd	even	odd	odd	$(10,01,00)$
even	odd	odd	even	$(00,10,01)$
odd	odd	odd	even	$(10,00,01)$
even	odd	odd	odd	$(01,10,00)$

Thus, in order to recover $\theta_i(z,\tau)$ from $\theta_i\left(\frac{z}{c\tau+d}, \frac{a\tau+b}{c\tau+d}\right)$, one needs to compute $\sqrt{c\tau+d}$ (which is done in $O(\mathcal{M}(P))$ bit operations) and $e^{\pi icz^2/(c\tau+d)}$ ($O(\mathcal{M}(P)\log P)$ bit operations), and perform a division; determining ζ is asymptotically negligible. The cost of this step is then $O(\mathcal{M}(P)\log P)$ bit operations. We note that in general, because of the permutation σ_γ, we need to have computed all three values $\theta_{0,1,2}\left(\frac{z}{c\tau+d}, \frac{a\tau+b}{c\tau+d}\right)$ in order to be able to use the formula to compute, say, $\theta_0(z,\tau)$. We will occasionally talk about computing θ_3, but this will not be the focus of our algorithms, as it can easily be recovered using Equation (2.5.8).

The proof of Theorem 2.5.2 is usally done using some particularly simple relations, which describe the action of $S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$ on the values of θ [Mum83, Table V, p. 36]:

$$\theta_0^2\left(0, \frac{-1}{\tau}\right) = -i\tau e^{i\pi z^2/\tau}\theta_0^2(0,\tau), \quad \theta_2^2\left(0, \frac{-1}{\tau}\right) = -i\tau e^{i\pi z^2/\tau}\theta_1^2(0,\tau), \tag{2.5.9}$$

and their equivalent for theta constants

$$\theta_0^2\left(0, \frac{-1}{\tau}\right) = -i\tau\theta_0^2(0,\tau), \quad \theta_2^2\left(0, \frac{-1}{\tau}\right) = -i\tau\theta_1^2(0,\tau). \tag{2.5.10}$$

These can be proven e.g. using Poisson's summation formula. Theorem 2.5.2 is then proven by determining the action of $T = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ on the values of θ, then using the fact that $\mathrm{SL}_2(\mathbb{Z}) = \langle S, T\rangle$.

Theorem 2.5.2 allows us to suppose that $\tau \in \mathcal{F}$, the fundamental domain for the action of $\mathrm{SL}_2(\mathbb{Z})$ on $\mathcal{H}$. This translates into the conditions

$$|\tau| \geq 1, \qquad |\mathrm{Re}(\tau)| \leq \frac{1}{2}, \qquad \text{and hence } \mathrm{Im}(\tau) \geq \frac{\sqrt{3}}{2}.$$

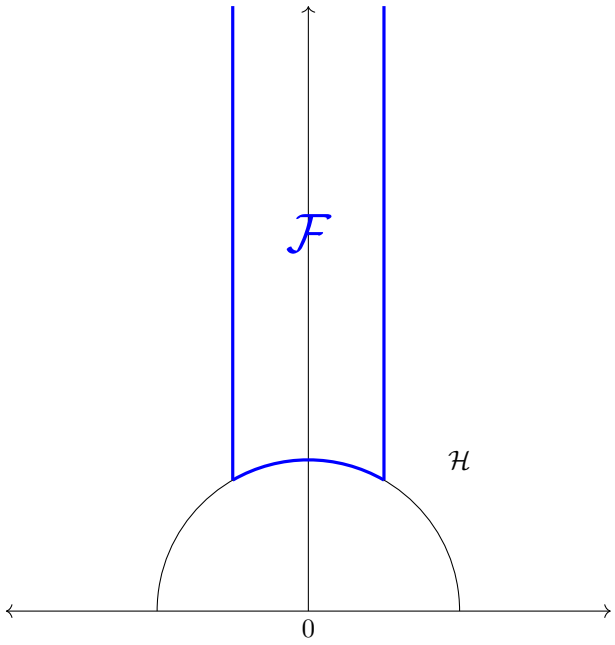

Figure 2.1: Fundamental domain $\mathcal{F}$.

The fundamental domain is depicted on Figure 2.1.

Argument reduction in z is carried out using quasi-periodicity:

$$\theta(z+b+a\tau,\tau) = e^{-\pi i a^2\tau - 2\pi i a z}\theta(z,\tau)$$

Finally, as mentioned in Note 2.3.2, we will suppose that $\mathrm{Im}(z) \geq 0$. Combining all the argument reduction strategies allows us to suppose (e.g. in Chapter 5 and Chapter 6) that:

$$\begin{aligned} &|\tau| \geq 1, \quad |\mathrm{Re}(\tau)| \leq \frac{1}{2}, \quad \mathrm{Im}(\tau) > 0, \\ &|\mathrm{Re}(z)| \leq \frac{1}{2}, \quad 0 \leq \mathrm{Im}(z) \leq \frac{\mathrm{Im}(\tau)}{2}. \end{aligned} \tag{2.5.11}$$

2.6 Genus 2 instantiations

We discuss in Chapter 5 and Chapter 7 the computation of the θ function in genus g. We often take a closer look at the case $g = 2$ in order to show how one can generalize existing algorithms which apply to the (genus 1) Jacobi θ function; hence, we outline explicitly in this section a few of the formulas we will use later, when discussing the case $g = 2$.

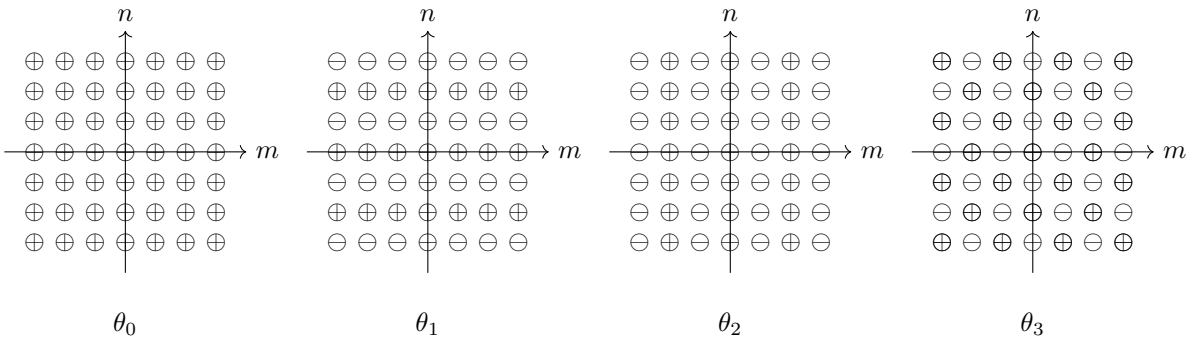

Figure 2.2: Signs of the terms in the sums defining respectively $\theta_0, \theta_1, \theta_2, \theta_3$.

2.6.1 Definition

We usually write $z = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$ and $\tau = \begin{pmatrix} \tau_1 & \tau_3 \\ \tau_3 & \tau_2 \end{pmatrix}$. The definition of the theta function can thus be rewritten as

$$\begin{aligned} \theta(z,\tau) &= \sum_{m,n\in\mathbb{N}} q_1^{m^2} q_2^{n^2} \left(q_3^{2mn}\left(w_1^{2m}w_2^{2n} + w_1^{-2m}w_2^{-2n}\right) + q_3^{-2mn}\left(w_1^{2m}w_2^{-2n} + w_1^{-2m}w_2^{2n}\right)\right) \\ &= 1 + 2q_1\left(w_1^2 + \frac{1}{w_1^2}\right) + 2q_2\left(w_2^2 + \frac{1}{w_2^2}\right) + q_1q_2\left(q_3^2\left(w_1^2w_2^2 + \frac{1}{w_1^2w_2^2}\right) + q_3^{-2}\left(\frac{w_1^2}{w_2^2} + \frac{w_2^2}{w_1^2}\right)\right) + \dots \end{aligned}$$

where $q_j = e^{i\pi\tau_j}$ and $w_j = e^{i\pi z_j}$.

Note that, in genus 2, there are 16 different theta functions, that we number θ_0 to θ_{15}. Using the notation we outlined in Note 2.1.5, the fundamental genus 2 theta functions are thus denoted $\theta_0, \theta_1, \theta_2, \theta_3$. Note that the series defining the fundamental theta functions are made of the same terms but with different signs, which allows their simultaneous computation by a naive algorithm at no extra cost; we summarize the patterns of the signs in Figure 2.2. The even theta functions are θ_i for $i \in \{0, 1, 2, 3, 4, 6, 8, 9, 12, 15\}$, while the odd ones are θ_i for $i \in \{5, 7, 10, 11, 13, 14\}$; this is the notation used in [Cos11, CR15, Str14], but differs from the notation used in e.g. [Gau07].

2.6.2 Reduction

The condition that $\mathrm{Im}(\tau)$ must be Minkowski-reduced can be rewritten as

$$0 \le 2\,\mathrm{Im}(\tau_3) \le \mathrm{Im}(\tau_1) \le \mathrm{Im}(\tau_2).$$

As for the condition given in Equation (2.3.1), it translates into

$$|\mathrm{Re}(z_i)| \le \frac{1}{2}, \qquad |\mathrm{Im}(z_1)| \le \frac{\mathrm{Im}(\tau_1) + \mathrm{Im}(\tau_3)}{2} \le \frac{3}{4}\,\mathrm{Im}(\tau_1), \qquad |\mathrm{Im}(z_2)| \le \frac{\mathrm{Im}(\tau_2) + \mathrm{Im}(\tau_3)}{2} \le \frac{3}{4}\,\mathrm{Im}(\tau_2)$$

2.6.3 Duplication

An important equation we will be using is the one of the τ-duplication formula. This formula can be written as follows for the fundamental theta functions:

$$\begin{aligned}
\theta_0(z,2\tau)^2 &= \frac{\theta_0(z,\tau)\theta_0(0,\tau)+\theta_1(z,\tau)\theta_1(0,\tau)+\theta_2(z,\tau)\theta_2(0,\tau)+\theta_3(z,\tau)\theta_3(0,\tau)}{4}\\
\theta_1(z,2\tau)^2 &= \frac{\theta_0(z,\tau)\theta_1(0,\tau)+\theta_1(z,\tau)\theta_0(0,\tau)+\theta_2(z,\tau)\theta_3(0,\tau)+\theta_3(z,\tau)\theta_2(0,\tau)}{4}\\
\theta_2(z,2\tau)^2 &= \frac{\theta_0(z,\tau)\theta_2(0,\tau)+\theta_1(z,\tau)\theta_3(0,\tau)+\theta_2(z,\tau)\theta_0(0,\tau)+\theta_3(z,\tau)\theta_1(0,\tau)}{4}\\
\theta_3(z,2\tau)^2 &= \frac{\theta_0(z,\tau)\theta_3(0,\tau)+\theta_1(z,\tau)\theta_2(0,\tau)+\theta_2(z,\tau)\theta_1(0,\tau)+\theta_3(z,\tau)\theta_0(0,\tau)}{4}
\end{aligned}$$

We can also write these relations for $z=0$, i.e. for theta-constants; this will emphasize the link with Borchardt mean that we use in Chapter 3 and Chapter 7.

$$\begin{aligned}
\theta_0(0,2\tau)^2 &= \frac{\theta_0(0,\tau)^2+\theta_1(0,\tau)^2+\theta_2(0,\tau)^2+\theta_3(0,\tau)^2}{4}\\
\theta_1(0,2\tau)^2 &= \frac{\theta_0(0,\tau)\theta_1(0,\tau)+\theta_2(0,\tau)\theta_3(0,\tau)}{2}\\
\theta_2(0,2\tau)^2 &= \frac{\theta_0(0,\tau)\theta_2(0,\tau)+\theta_1(0,\tau)\theta_3(0,\tau)}{2}\\
\theta_3(0,2\tau)^2 &= \frac{\theta_0(0,\tau)\theta_3(0,\tau)+\theta_1(0,\tau)\theta_2(0,\tau)}{2}
\end{aligned}$$

2.6.4 The Kummer surface

Finally, we mention a last relation, which in a sense generalizes Equation (2.5.7), in that it gives the equation of the *Kummer variety* defined by the values of theta in genus 2. This equation is useful in Chapter 7, where it appears as a fix which allows the use of Newton's method.

Proposition 2.6.1 ([Gau07]). *Let*

$$\begin{aligned}
&x=\theta_0(z,\tau), \quad y=\theta_1(z,\tau) \quad ,z=\theta_2(z,\tau), \quad t=\theta_3(z,\tau)\\
&a=\theta_0(0,\tau), \quad b=\theta_1(0,\tau), \quad c=\theta_2(0,\tau), \quad d=\theta_3(z,\tau)\\
&A=\theta_0(0,2\tau), \quad B=\theta_1(0,2\tau), \quad C=\theta_2(0,2\tau), \quad D=\theta_3(z,2\tau)
\end{aligned}$$

and define

$$\begin{aligned}
E &= \frac{256abcdA^2B^2C^2D^2}{(a^2b^2-c^2d^2)(a^2c^2-b^2d^2)(a^2d^2-b^2c^2)}\\
F &= \frac{a^4+b^4-c^4-d^4}{a^2b^2-c^2d^2}\\
G &= \frac{a^4-b^4+c^4-d^4}{a^2c^2-b^2d^2}\\
H &= \frac{a^4-b^4-c^4+d^4}{a^2d^2-b^2c^2}.
\end{aligned}$$

Note that the τ-duplication formulas of the previous subsection show that A,B,C,D can be written very simply in terms of a,b,c,d; hence, E can also be written as a function of a,b,c,d. Then

$$(x^4+y^4+z^4+t^4)+2Exyzt=F(x^2y^2-z^2t^2)+G(x^2z^2-y^2t^2)+H(x^2t^2-y^2z^2),$$

which is the equation of the Kummer surface.

The Kummer surface is isomorphic to the Jacobian modulo the 2-torsion points, as the fundamental theta functions are even.

Remark 2.6.2. The Kummer surface has the advantage of having simple (and hence fast) arithmetic, and very regular formulas, which provides natural protection against side-channel attacks. We refer to [Gau07, Cos11] for the description of this surface and the corresponding arithmetic; [Cos11] furthermore shows how to use Kummer surfaces in the context of factorization of integers, in a manner similar to the Elliptic Curve Method (ECM), and with even better implementation results. Finally, we refer to two recent papers, [BCHL16] and [CCS15], which show how Kummer surfaces can be used in cryptographic schemes and offer arithmetic that is even faster than with elliptic curves.

Chapter 3

AGM and Borchardt mean

This chapter is dedicated to outlining the classical theory behind the *arithmetico-geometric mean* and its generalization to higher genus the *Borchardt mean*. Their study has computational applications, namely the computation of theta-constants, in genus 1 for the AGM, and in genus g for the Borchardt mean; we outline the corresponding algorithms in Section 6.1 and Section 7.1.

The main problem in the study of these means is the study of the *choice of signs*: either of those means requires the extraction of a square root, and hence two possible complex values. Choosing one square root over another changes the value to which the sequence converges, and sometimes even its rate of convergence; we will make this explicit. The results presented in this section are taken from [Cox84] and [Dup06, Dup11].

3.1 The real AGM

3.1.1 Rate of convergence

Recall the *arithmetic mean* of two positive numbers $(a,b) \mapsto \frac{a+b}{2}$, and the *geometric mean* $(a,b) \mapsto \sqrt{ab}$. The following proposition gives the definition of the *arithmetico-geometric mean*:

Proposition 3.1.1. *Let $a, b \in \mathbb{R}^+$ such that $a \geq b$. Define the sequences $(a_n)_{n\in\mathbb{N}}, (b_n)_{n\in\mathbb{N}}$ as follows:*

$$\begin{aligned} a_0 &= a, \qquad b_0 = b \\ a_{n+1} &= \frac{a_n + b_n}{2}, \qquad b_{n+1} = \sqrt{a_n b_n} \end{aligned}$$

where $b_{n+1} > 0$. Then the sequences $(a_n)_{n\in\mathbb{N}}, (b_n)_{n\in\mathbb{N}}$ are adjacent, i.e. $(a_n)_{n\in\mathbb{N}}$ is decreasing, $(b_n)_{n\in\mathbb{N}}$ is increasing, and $(a_n - b_n)_{n\in\mathbb{N}}$ goes to zero; thus $(a_n)_{n\in\mathbb{N}}$ and $(b_n)_{n\in\mathbb{N}}$ converge to the same limit. Define the arithmetico-geometric mean (AGM) *of a and b, denoted* $\mathrm{AGM}(a,b)$*, as the limit of either $(a_n)_{n\in\mathbb{N}}$ or $(b_n)_{n\in\mathbb{N}}$.*

Proof. The concavity of $x \mapsto \log x$ can be used to prove that, $\forall x, y \in \mathbb{R}^+$, $\frac{x+y}{2} \geq \sqrt{xy}$. In fact, we have

$$a_n \geq \frac{a_n + b_n}{2} = a_{n+1} \geq b_{n+1} \geq \sqrt{b_n b_n} = b_n$$

which proves that $a \geq a_1 \geq ... \geq a_n \geq b_n \geq ... \geq b_n \geq b$. We even have

$$a_{n+1} - b_{n+1} \leq a_{n+1} - b_n = \frac{a_n - b_n}{2}$$

which proves $(a_n - b_n)_{n\in\mathbb{N}}$ converges to 0: hence $(a_n)_{n\in\mathbb{N}}, (b_n)_{n\in\mathbb{N}}$ are adjacent and converge, and $a_n - b_n \leq \frac{a-b}{2^n}$. □

The notion of quadratic convergence is central in this manuscript:

Definition 3.1.2. Let $(x_n) \in \mathbb{C}^{\mathbb{N}}$ be a sequence; we say that (x_n) *is quadratically convergent to* ℓ, or that (x_n) *converges quadratically to* ℓ if there exists a $C > 0$ and a $N \in \mathbb{N}$ such that for all $n > N$

$$|x_{n+1} - \ell| \leq C|x_n - \ell|^2.$$

Note that this implies that for n large enough, the number of exact digits one gets by taking x_{n+1} as an approximation of ℓ is roughly twice as much as the number of exact digits one gets from x_n. This means that the first n for which $|x_n - \ell| \leq 2^{-P}$ satisfies $n = O(\log P)$: only $O(\log P)$ steps are needed to compute P bits of the limit of the sequence.

AGM sequences can be shown to converge quadratically (see e.g. [BB87, BM88]):

Proposition 3.1.3.

$$a_{n+1} - b_{n+1} \leq \frac{1}{8b}(a_n - b_n)^2.$$

Proof. Write:

$$a_{n+1} - b_{n+1} = \frac{1}{2}(\sqrt{a_n} - \sqrt{b_n})^2 \leq \frac{1}{2}\frac{(a_n - b_n)^2}{(\sqrt{a_n} + \sqrt{b_n})^2} \leq \frac{1}{8b}(a_n - b_n)^2.$$ □

In practice, the AGM sequences converge quite quickly; for instance, one can compute $\mathrm{AGM}(1, \sqrt{2})$ with 14 decimal digits of precision in only 4 steps. This calculation was done by Gauss in 1809, who then noticed that the result corresponded to $\frac{\pi}{\omega}$, where ω is the length of the lemniscate, thus outlining a link between the AGM and elliptic integrals. We refer the reader to [Cox84] for more historical details on the AGM.

Given that square roots can be computed with precision P in $O(\mathcal{M}(P))$ (Section 0.3.3), we have

Proposition 3.1.4. $\mathrm{AGM}(a, b)$ *can be computed with absolute precision* P *in* $O(\mathcal{M}(P)\log P)$ *bit operations.*

This is an example of a *quasi-optimal complexity*, or *quasi-linear complexity*, which we define in this manuscript as a complexity which is essentially linear in the output size, up to logarithmic factors. Finally note that contrary to, say, Newton's method, the AGM is not self-correcting; hence, each iteration has to be carried out at maximal precision.

3.1.2 Brent-Salamin algorithm

An interesting application of the AGM over the positive reals is the Brent-Salamin algorithm for the asymptotically fast computation of digits of π. This algorithm was found independently by Brent and Salamin in 1975; we refer to [BB87] for more details than what is presented here. We use this algorithm in several settings, e.g. Chapter 5.

Proposition 3.1.5 ([BB87, Algorithm 2.2])**.** *Let* $a_0 = 1, b_0 = \frac{1}{\sqrt{2}}$. *Define*

$$\pi_n = \frac{2a_{n+1}^2}{1 - \sum_{k=0}^{n} 2^k c_k^2}, \qquad c_n = \sqrt{a_n^2 - b_n^2} = \frac{c_{n-1}^2}{4a_n}$$

where a_n, b_n are the sequences computed by the AGM iteration. Then (π_n) converges quadratically to π, and

$$\pi - \pi_n \leq \frac{\pi^2 2^{n+4} e^{-\pi 2^{n+1}}}{\mathrm{AGM}(1, 1/\sqrt{2})^2}.$$

This proposition can be proved by using the properties of elliptic integrals, and more precisely the change of variables given by the Landen transform. The connection between the AGM and elliptic integrals via the Landen transform is outlined in Chapter 4. The proposition gives a quasi-optimal time algorithm to compute P bits of π: compute $\frac{1}{\sqrt{2}}$ with P bits of precision in time $O(\mathcal{M}(P))$, then compute (π_n) until it is within 2^{-P} of π, which only requires $O(\log P)$ terms, for a total cost of $O(\mathcal{M}(P) \log P)$ bit operations.

This is currently the best known asymptotic running time for the computation of P digits of π; however, the Brent-Salamin algorithm is not necessarily the fastest algorithm in practice. We refer to [BB87] for many similar algorithms designed to compute approximations of π. Some of these algorithms have been used in the past to compute a record number of digits of π; the main problem is that they require a lot of memory, since one has to work with (and hence, store) numbers of size roughly P (and even $P + \log P$ accounting for guard bits). Other algorithms, such as the one using Chudnovsky's formula [CC89], are used nowadays for records of decimals of π [Bel10, YK11]; their convergence is slower (about 14 decimal digits per term, i.e. per step), but combined with a binary splitting strategy, they are faster in practice than the AGM-based algorithms, most notably because they support parallelization and checkpointing.

3.2 The complex AGM

It is possible to generalize the results above to the complex case, i.e. $a, b \in \mathbb{C}$. However, in this case, there are two possibilities at every step for computing the square root; this gives an uncountable number of *AGM sequences*, and defining unambiguously the AGM requires a bit more work.

3.2.1 Choice of signs and optimal AGM sequences

Definition 3.2.1. Let $a, b \in \mathbb{C}$, and let $(a_n)_{n\in\mathbb{N}}, (b_n)_{n\in\mathbb{N}} \in \mathbb{C}^{\mathbb{N}}$ such that $a_0 = a, b_0 = b$. The sequence (a_n, b_n) is *an AGM sequence* if, for all n:

$$a_{n+1} = \frac{a_n + b_n}{2}, \qquad b_{n+1}^2 = a_n b_n$$

As [Cox84], our discussion here assumes $a_0 \neq 0, b_0 \neq 0, a_n \neq \pm b_n$, as the limit of AGM sequences in these cases is trivial. It is easy to see by induction that these conditions are satisfied for a_n, b_n if and only if they are satisfied for a_{n-1}, b_{n-1}.

Hence, there are uncountably many AGM sequences, since there are two distinct possible choices of sign at each step. The following notion distinguishes choices of signs:

Definition 3.2.2. Let $\alpha \in \mathbb{C}^*$ such that $\alpha^2 = b_n^2$. We say that α is a *good choice of square roots* or a *good choice of signs* if setting $b_n = \alpha$ gives the relations

$$|a_n - b_n| < |a_n + b_n|, \qquad \text{or} \quad |a_n - b_n| = |a_n + b_n| \text{ and } \operatorname{Im}\left(\frac{b_n}{a_n}\right) > 0$$

If that is not the case, α is a bad choice of square roots.

Note that this is equivalent to "$\mathrm{Re}\left(\frac{b_n}{a_n}\right) > 0$, or $\mathrm{Re}\left(\frac{b_n}{a_n}\right) = 0$ and $\mathrm{Im}\left(\frac{b_n}{a_n}\right) > 0$".

Definition 3.2.3. Define the *optimal AGM sequence* [CT13] (also called the *standard AGM sequence*, in e.g. [ET14a]), as the AGM sequence for a, b where *all* the choices of sign are good. This sequence converges quadratically to a non-zero value; this value is defined to be $\mathrm{AGM}(a, b)$ (also sometimes called the *simplest value* [Cox84, Jar08]).

The question of the choice of sign is important in practical applications of the AGM, most notably for theta constants; we give related results in Chapter 6.

3.2.2 Convergence of optimal AGM sequences

Lemma 3.2.4 ([Dup11, Theorem 1])**.** *Let (a_n, b_n) be an AGM sequence in which all the choices of signs are good. Then for all $n \geq 0$ we have*

$$|a_{n+1} - b_{n+1}| \leq \frac{\pi}{8\min(|a_0|, |b_0|)}|a_n - b_n|^2.$$

This can be used to prove quadratic convergence:

Proposition 3.2.5. *Let $a, b \in \mathbb{C}$ such that $ab \neq 0$ and $a \neq -b$. Any AGM sequence for a, b with only a finite number of bad choices converges quadratically to a non-zero limit. Any AGM sequence for a, b with infinitely many bad choices converges (at least linearly) to 0.*

Proof. The properties of the limit come from [Cox84, Prop. 2.1]; the proof of the quadratic convergence uses a lot of the same arguments, and can be found in [Dup06, Section 3.4] or [Dup11, Theorem 1 and Prop. 12]. □

In particular, optimal AGM sequences converge quadratically, which means our definition of the complex AGM enjoys similar properties as the real AGM.

The convergence of the AGM can actually be studied more precisely, with a precise bound on the number of iterations required to compute $\mathrm{AGM}(1, z)$, as well as a bound on the number of guard bits:

Theorem 3.2.6 ([Dup11, Prop. 12 & Cor. 1])**.** *Let z be a complex number in the upper-right quadrant of the complex plane, and denote (a_n, b_n) the optimal AGM sequence for $(1, z)$. Put*

$$n_P = \max\left(\lceil \log_2|\log_2|z|| \rceil, 1\right) + \lceil \log_2(P+3) - 1 \rceil.$$

Then a_{n_P} is an approximation of $\mathrm{AGM}(1, z)$ with relative precision P bits. Furthermore, each iteration loses a constant number of bits of relative precision, which means it is enough to work at precision $P + 2 + 2n_P = O(P + \log_2|\log_2|z||)$.

Note that this means that more iterations are needed the larger $|z|$ is, but also the smaller $|z|$ is.

In the end, this gives the result that the complex AGM can be computed with precision P in $O(\mathcal{M}(P)\log P)$ bit operations. Note that the constant in the O depends on z; however, in our applications, we find a way to get rid of the dependency in z.

3.2.3 Theta-constants and arithmetico-geometric mean

Recall the τ-*duplication formulas*, which were mentioned in Chapter 2:

$$\theta_0(0,2\tau)^2 = \frac{\theta_0(0,\tau)^2 + \theta_1(0,\tau)^2}{2}, \qquad \theta_1(0,\tau)^2 = \theta_0(0,\tau)\theta_1(0,\tau) \tag{3.2.1}$$

A proof for these formulas can also be obtained easily by manipulating the definition of the series; see [BB87]. This means that the sequence $\left(\theta_0(0,2^n\tau)^2, \theta_1(0,2^n\tau)^2\right)_{n\in\mathbb{N}}$ is an AGM sequence for $\theta_0(0,\tau)^2, \theta_1(0,\tau)^2$. Studying the choice of signs in this AGM sequence was done first in [Gep28], while [Cox84] gave a more modern treatment.

Optimal sequences

Definition 3.2.7. Define

$$\mathcal{D}_1 = \left\{\tau \in \mathcal{F} \mid |\mathrm{Re}(\tau)| \leq 1, |\mathrm{Re}\left(\frac{1}{\tau}\right)| \leq 1\right\}$$

and define $\tilde{\mathcal{D}_1}$ as the domain obtained by translating $\mathcal{D}_1$ by $\pm 2, \pm 4$, etc.

Proposition 3.2.8 ([Cox84, Lemma 2.8 and Lemma 2.9])**.** *For all $\tau \in \mathcal{D}_1$, or in $\tilde{\mathcal{D}_1}$, we have* $\mathrm{Re}\left(\frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right) > 0$, *or* $\mathrm{Re}\left(\frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right) = 0$ *and* $\mathrm{Im}\left(\frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right) > 0$.

Now, pick a $\tau \in \mathcal{F}$, and consider the AGM sequence starting with $(\theta_0^2(0,\tau), \theta_1^2(0,\tau))$. Picking the right choice of sign for the first step is then equivalent to the condition

$$\begin{aligned}
& |\theta_0^2(0,2\tau) - \theta_1^2(0,2\tau)| < |\theta_0^2(0,2\tau) + \theta_1^2(0,2\tau)| \\
\text{or} \quad & |\theta_0^2(0,2\tau) - \theta_1^2(0,2\tau)| = |\theta_0^2(0,2\tau) + \theta_1^2(0,2\tau)| \text{ and } \mathrm{Im}\left(\frac{\theta_1^2(0,2\tau)}{\theta_0^2(0,2\tau)} > 0\right) \\
\Leftrightarrow \quad & \mathrm{Re}\left(\frac{\theta_1^2(0,2\tau)}{\theta_0^2(0,2\tau)}\right) > 0 \qquad \text{or } \mathrm{Re}\left(\frac{\theta_1^2(0,2\tau)}{\theta_0^2(0,2\tau)}\right) = 0 \text{ and } \mathrm{Im}\left(\frac{\theta_1^2(0,2\tau)}{\theta_0^2(0,2\tau)}\right) > 0
\end{aligned}$$

Hence:

Proposition 3.2.9 ([Cox84, Lemma 2.9])**.** *Define $\mathcal{D}_2 = \frac{1}{2}(\mathcal{D}_1 \setminus B)$, where B is the border of the half circle on the right. Define also $\tilde{\mathcal{D}_2}$ as $\mathcal{D}_2$ translated by $\pm 1, \pm 2$, etc. Then for $\tau \in \tilde{\mathcal{D}_2}$, the sequence $\left(\theta_0^2(0,2^n\tau), \theta_1^2(0,2^n\tau)\right)_{n\in\mathbb{N}}$ is an optimal AGM sequence, and hence*

$$\mathrm{AGM}(\theta_0(0,\tau)^2, \theta_1(0,\tau)^2) = 1.$$

Hence, AGM sequences starting with the fundamental theta-constants at τ are optimal sequences for τ in the striped domain on Figure 3.1.

Remark 3.2.10. Note that the AGM is homogeneous, i.e.

$$\mathrm{AGM}(\lambda x, \lambda y) = \lambda\,\mathrm{AGM}(x,y)$$

Hence, a direct consequence of Proposition 3.2.9 is that

$$\mathrm{AGM}\left(1, \frac{\theta_1^2(0,\tau)}{\theta_2^2(0,\tau)}\right) = \frac{1}{\theta_0^2(0,\tau)}$$

for any $\tau \in \mathcal{D}_2$.

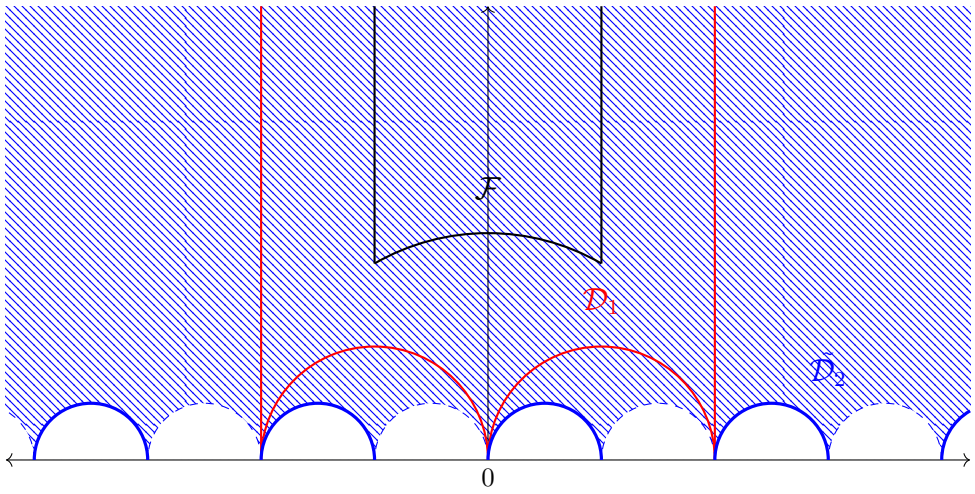

Figure 3.1: The domains $\mathcal{D}_1$ (in red) and $\tilde{\mathcal{D}}_2$ (blue lines), with the fundamental domain $\mathcal{F}$.

Limits of AGM sequences

Proposition 3.2.11 ([Cox84, Lemma 2.5 and Lemma 2.7])**.** *Define the principal congruence subgroup of level 2*

$$\Gamma(2) = \left\{ \gamma \in \mathrm{SL}_2(\mathbb{Z}) \;\middle|\; \gamma \equiv \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \pmod 2 \right\}.$$

Then given $\tau \in \mathcal{F}$, there is $\gamma \in \Gamma(2)$ such that $\gamma \cdot \tau \in \mathcal{D}_1$; in fact, the fundamental domain of $\Gamma(2)$ is $\mathcal{D}_1$ minus the left half-circle.

Similarly, the following proposition determines the set for which $\mathcal{D}_2$ is a fundamental domain:

Proposition 3.2.12 ([Cox84, Lemma 2.7])**.** *Define $\Gamma_2(4)$, a subgroup of Γ_2, as*

$$\Gamma_2(4) = \left\{ \gamma = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathrm{SL}_2(\mathbb{Z}) \;\middle|\; b \equiv 0 \pmod 2, c \equiv 0 \pmod 4 \right\}.$$

Then given $\tau \in \mathcal{F}$, there is $\gamma \in \Gamma_2(4)$ such that $\gamma \cdot \tau \in \mathcal{D}_2$. In fact, the fundamental domain of $\Gamma_2(4)$ is $\mathcal{D}_2$ minus the two left half-circles.

Furthermore:

Proposition 3.2.13 ([Dup06, Prop. 2.14])**.** *The function $\tau \to \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}$ is modular for $\Gamma_2(4)$, and in particular invariant under its action.*

Now, for any $\tau \in \mathcal{F}$ take $\gamma \in \Gamma_2(4)$ such that $\gamma \cdot \tau \in \mathcal{D}_2$, and write

$$\mathrm{AGM}(\theta_0^2(0,\tau), \theta_1^2(0,\tau)) = \frac{\theta_0^2(0,\tau)}{\theta_0^2(0,\gamma\cdot\tau)} \mathrm{AGM}\left(\theta_0^2(0,\gamma\cdot\tau), \theta_1^2(0,\gamma\cdot\tau)\right) = \frac{\theta_0^2(0,\tau)}{\theta_0^2(0,\gamma\cdot\tau)}$$

using the homogeneity of the AGM and Proposition 3.2.13. This proves that:

Theorem 3.2.14 ([Cox84], [Dup06, Theorem 3.1])**.** *The set of limits of AGM sequences starting at $(\theta_0^2(0,\tau), \theta_1^2(0,\tau))$ is*

$$\left\{ \frac{\theta_0^2(0,\tau)}{\theta_0^2(0,\gamma\cdot\tau)}, \gamma \in \Gamma_2(4) \right\} \cup \{0\}.$$

This can be rephrased into a result describing the set of limits of AGM sequences starting at $(a, b) \in \mathbb{C}^2$; this result was known to Gauss [Cox84]. In particular, this gives the rather striking result that the inverse of the limits of the AGM sequences starting at (a, b) form a lattice.

3.3 Applications of the complex AGM

We highlight a few of the computational applications of the complex AGM. The first application is the study of elliptic integrals, which actually led to the discovery of the complex AGM in the first place. A second application is the discovery, in the 1970s, of fast algorithms based on the AGM to compute important mathematical quantities, such as π (see Section 3.1.2) and $\log(z)$, and thus $\exp(z)$.

A most interesting application of the AGM is the computation of theta-constants; we give extensive details on this algorithm in Chapter 6, in which we also generalize the blueprint of the algorithm to theta functions.

3.3.1 Elliptic integrals

As explained in [Cox84, Section 3], the history of the arithmetico-geometric mean is very tightly connected to the study of some integrals, starting with the integral $\int_0^1 \frac{dz}{\sqrt{1-z^4}}$, which is a quarter of the arc length of the lemniscate. The lemniscate, and its sibling the elastic curve, was discovered in the 17th century by Bernoulli, then studied by Stirling and Euler; Lagrange, in 1785, came very close to linking the integral to the arithmetico-geometric mean, but seem to have missed this discovery.

Gauss studied related notions around 1795, where he attempted to study lemniscatic (elastic) functions much in the same way as one does circular geometry. He came back to it in 1798, this time studying quantities that in hindsight are closely related to theta functions; his explorations culminated to a calculation to the 11th decimal place made on May 30th 1799, which led him to conjecture that

$$\int_0^1 \frac{dz}{\sqrt{1-z^4}} = \frac{\pi}{2\,\mathrm{AGM}(\sqrt{2}, 1)}.$$

In his famous words, "the demonstration of this fact will surely open an entirely new field of analysis".

A classical proof of this identity will be given in Chapter 4; in fact, there are many more identities of this type, including one of particular interest for us:

$$\int_0^{2\pi} \frac{dt}{\sqrt{a^2\cos^2 t + b^2 \sin^2 t}} = \frac{2\pi}{\mathrm{AGM}(a, b)}.$$

3.3.2 Computing the complex logarithm

Note that θ can be looked at as a function of $q = e^{i\pi\tau}$. In this context, Equation (2.5.10) is an equation linking θ to the logarithm of q; hence, one could think of computing the logarithm of a complex number using this equation and the properties of the AGM. We first outline the algorithm of Sasaki and Kanada to compute $\log x$ for real x, then show a different approach by Dupont to compute the complex logarithm in quasi-optimal time.

The algorithm of Sasaki and Kanada

Sasaki and Kanada [SK82] proposed in 1982 an algorithm based on the AGM and theta-constants to compute $\log x$ in time $O(\mathcal{M}(P)\sqrt{P})$; we also refer to [BZ10, p.158] for more comments and implementation remarks. They start from the relation:

Proposition 3.3.1. *For $\tau \in \mathcal{H}$ such that $\frac{-1}{\tau} \in \mathcal{D}_2$, we have*

$$\mathrm{AGM}(\theta_0(0,\tau)^2, \theta_2(0,\tau)^2) = \frac{i}{\tau}$$

This proposition can be proven from Equation (2.5.10) and Proposition 3.2.9, and the homogeneity of the AGM. Putting $q = e^{i\pi\tau}$, this can be rephrased as

$$\frac{\log q}{i\pi} = \tau = \frac{i}{\mathrm{AGM}\left(\left(\sum_{n\in\mathbb{Z}} q^{n^2}\right)^2, \left(\sum_{n\in\mathbb{Z}} q^{(n+1/2)^2}\right)^2\right)}$$

This relation is valid for all complex numbers τ such that $\frac{-1}{\tau} \in \mathcal{D}_2$; in particular, it is valid for $\tau \in \mathcal{F}$.

Sasaki and Kanada take a look in particular at the case where q is real and smaller than 1, which corresponds to $\tau \in i\mathbb{R}$. The result above then shows how to compute the logarithm of q:

$$\log q = \frac{-\pi}{\mathrm{AGM}\left(\left(\sum_{n\in\mathbb{Z}} q^{n^2}\right)^2, \left(\sum_{n\in\mathbb{Z}} q^{(n+1/2)^2}\right)^2\right)}$$

Alternatively, one can apply this result to q^4 in order to avoid the $q^{1/4}$ which appears in the definition of θ_2.

Given q, the evaluation of the sums costs $O(\mathcal{M}(P)\sqrt{P})$ bit operations (see for example our analysis in Chapter 5). The evaluation of the AGM then costs $O(\mathcal{M}(P)\log P)$, which is dominated by the cost of the evaluation of the sums; in the end, this gives a $O(\mathcal{M}(P)\sqrt{P})$ algorithm for $\log x$.

Lastly, as noticed in [BB87, Section 7.2], if one works with arithmetic in base b, then for $q = \frac{1}{b}$ the computation of the arguments of the AGM is easy, since they are just sequences of 0s and 1s that can be computed very quickly (in linear time). Hence, the algorithm can compute $\frac{\pi}{\log b}$ in base b in $O(\mathcal{M}(P)\log P)$ time.

We will not use this algorithm in the rest of this manuscript; indeed, it uses theta-constants to compute logarithms, when the quasi-optimal time algorithms we consider (most notably in Chapter 6) require the computation of logarithms to compute theta-constants (and theta functions).

A quasi-optimal time algorithm for $\log z$

This section describes a quasi-optimal time algorithm to compute the logarithm of a complex number with precision P; this algorithm is presented e.g. in [Dup06, Theorem 3.3,p. 90]. Compared to the algorithm in the previous section, this algorithm does not require the computation of theta-constants, and has better running time; this is the algorithm we use in the rest of this manuscript to compute $\log z$.

The following approximation is key to the algorithm:

Theorem 3.3.2. *For any z such that $|z| \leq 2^{-10}$ and $|\mathrm{Arg}(z)| \leq \frac{\pi}{4}$:*

$$|\log \frac{z}{4} + \frac{\pi}{2\,\mathrm{AGM}(1,z)}| \leq 0.26|z|^2\left(1 + |\log \frac{z}{4}|\right)$$

The proof of this theorem uses the fact that the function $\tau \mapsto \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}$ is surjective, and hence proves in particular for z satisfying the hypotheses of the theorem, there exists $\tau_z \in \mathcal{F}$ such that $z = \frac{\theta_2^2(0,\tau_z)}{\theta_0^2(0,\tau_z)}$. All the properties in terms of $\mathrm{AGM}(1,z)$ are then rephrased in terms of theta-constants, and Equation (2.5.10) is used to provide the link between theta-constants and τ_z; a careful bounding of the series defining the theta-constants in that case provides the result.

Computing log(2) Theorem 3.3.2 directly gives an algorithm that computes $\log 2$ with precision P in $O(\mathcal{M}(P)\log P)$ operations. Put $z = \frac{1}{2^n}$, then the theorem above proves that (at least for $n \geq 12$)

$$\left|\frac{\log z + \frac{\pi}{2\,\mathrm{AGM}(1,4z)}}{\log z}\right| \leq \frac{1}{2^{2n-3}}$$

hence

$$\frac{-\pi}{2\,\mathrm{AGM}(1,4z)} \text{ is an approximation of } -n\log 2 \text{ with relative precision } 2n-3 \text{ bits.}$$

Finally, put $n = \lceil \frac{P+4}{2} \rceil$ and compute $\frac{-\pi}{2n\,\mathrm{AGM}\left(1,\frac{1}{2^{n-2}}\right)}$; this gives an approximation of $\log 2$ with relative precision P. Note that since $\log 2 \simeq 0.69$, this also gives an approximation of $\log 2$ up to 2^{-P}. This algorithm requires the computation of $O(P)$ digits of π, which is done using Section 3.1.2; as for the computation of the AGM, one can show (using Theorem 3.2.6) that the number of iterations is $O(\log P)$, and hence the total running time is $O(\mathcal{M}(P)\log P)$.

Computing log(z) This in turn yields an algorithm to compute the complex logarithm with relative precision P. Put $M = \lceil \frac{P+4}{2} \rceil$; one can assume that $2^{-M-1} \leq |z| \leq 2^{-M}$, since one can just use the previous algorithm to add the right multiple of $\log 2$ to the final result. One can also suppose that $|\mathrm{Arg}(z)| \leq \pi/4$, even if it means to add a multiple of $i\pi/2$ to the final result. Then, one applies Theorem 3.3.2 to compute an approximation of $\log z$ with relative precision P bits. This gives Algorithm 3.

Algorithm 3 Compute $\log z$ with absolute precision P.
Input: $z \in \mathbb{C}$ with absolute precision P.
Output: $\log z$ with absolute precision P.

Work with precision $\mathcal{P} = P + 6\log P + 3|\log(2+|z|)| + 20$.
$f \leftarrow 0$
while $|\mathrm{Arg}(z)| \leq \pi/4$ **do**
    $z \leftarrow iz$, $f \leftarrow f+1$
end while
$M \leftarrow \lceil \frac{\mathcal{P}+4}{2} \rceil$
$n \leftarrow 0$
while $|z| \geq 2^{-M}$ **do**
    $z \leftarrow z/2$, $n \leftarrow n+1$
end while
$\pi_{\mathcal{P}} \leftarrow \pi$ with precision $\mathcal{P}$.
$s \leftarrow \mathrm{AGM}\left(1, \frac{1}{2^{M-2}}\right)$ with precision $\mathcal{P}$.
$r \leftarrow \mathrm{AGM}\left(1, \frac{1}{4z}\right)$ with precision $\mathcal{P}$.
return $\pi_{\mathcal{P}}(-\frac{1}{2r} + f\frac{i}{2} - \frac{n}{2Ms})$

Theorem 3.3.3. *For $z \in \mathbb{C}$ with absolute precision P, Algorithm 3 returns an approximation of $\log z$ with absolute precision P in $O(\mathcal{M}(P + |\log|z||)(\log P + \log|\log|z||))$.*

Proof. If $|z| \leq 2^{-M}$, Theorem 3.3.2 proves that $\frac{-\pi}{2\,\mathrm{AGM}(1,4z)}$ is an approximation of $\log z$ with relative precision $\mathcal{P}+1$ bits. According to Theorem 3.2.6, computing $\mathrm{AGM}(1, 4z)$ with relative precision P requires at most $\log P + \log|\log|z||$ iterations, and requires $2 + 2\log P + 2\log|\log|z||$ guard bits. Given our choice of $\mathcal{P}$, Algorithm 3 returns an approximation of $\log z$ with relative precision at least $P + |\log|z||$, which gives an approximation of $\log z$ with absolute precision P.

Now, if $|z| \geq 2^{-M}$, the computation of $\log \frac{z}{2^n}$ requires the computation of $\mathrm{AGM}\,(1, u)$ with $\log|\log|u|| \leq \log M \leq \log P$. Hence, the correct computation of this AGM with relative precision P requires at most $2\log P$ iterations, and the number of guard bits needed to compensate errors is $2 + 4\log P$. Given our choice of $\mathcal{P}$, $\frac{-\pi}{2r}$ is an approximation of $\log \frac{z}{2^n+1}$ with relative precision at least $P + \log P$, which guarantees absolute precision $P + 2$.

The computation of $\log 2$ with absolute or relative precision P also requires roughly $2\log P$ iterations and $2 + 4\log P$ guard bits. Hence, $\frac{\pi}{2Ms}$ is an approximation of $\log 2$ with absolute precision at least $P + \log P + 2|\log(2 + |z|)| + 20$. Numerical experiments show that

$$\begin{aligned}\log_2 n &\leq \log_2 \log_2 |z| + \log_2(\mathcal{P}+5) - 1\\ &\leq 2\log_2 \log_2(2+|z|) + 1.5\log_2 P + 5\end{aligned}$$

for $P \geq 2$. This proves (using Theorem 0.3.3) that $\frac{n\pi}{2Ms}$ is an approximation of $n\log 2$ with absolute precision $P + 2$, which proves the theorem. □

Note that this gives a $O(\mathcal{M}(P)\log P)$ running time if z is assumed to be in a compact set.

3.3.3 Computing the exponential

Classical methods to compute the exponential of a complex number involve computing a partial summation with enough terms. The series converges rather quickly, since $O\left(\frac{P}{\log P}\right)$ terms are sufficient to get a result with absolute precision P [Bre76]. One can also use argument reduction and compute $\exp\left(\frac{z}{2^i}\right)$ for some i, so that the series converges even faster: around $O(\sqrt{P})$ terms are then needed, as well as $O(\sqrt{P})$ squarings. Moreover, the technique of binary splitting, which evaluate parts of the series recursively, can also take advantage of the fact that the denominators of several consecutive terms have a common factor, and so do the numerators. The computation of the exponential at rational points can then be done quickly, in fact in quasi-optimal time. We refer to [Bre76, BZ10] for an overview of these techniques, which achieve overall an $O(\mathcal{M}(P)\log^2 P)$ running time.

An asymptotically faster method is to use Newton's method to compute $\exp z$ from $\log z$. The Newton iteration to compute $\exp a$ is

$$z_{n+1} = z_n - z_n(a - \log z_n).$$

and requires the computation of $\log z_n$ at each step; however, as we explained in the introduction (Section 0.3.3), the fact that Newton's method is self-correcting means that one can afford to simply compute the k-th iteration with precision $2^{k+1}P_0$. Hence, the total complexity of computing $\exp z$ via Newton's method is the same as applying the complex logarithm to numbers close to e^z, i.e.

$$O(\mathcal{M}(P + |z|)(\log P + \log|z|))$$

or $O(\mathcal{M}(P)\log P)$ if z is in a compact set. However, note that in practice, the algorithms based on binary splitting perform better than this algorithm, even for large (e.g. millions of digits) precisions, as the constant in the O is smaller than the one of the AGM-based algorithms.

3.4 Generalization of the AGM to higher genera

This section is based on [Dup06], who generalized some of the results of the previous sections to genus $g > 1$.

3.4.1 Definition

An AGM-like sequence of four positive numbers was considered by Borchardt in [Bor76, Bor78], with the relations

$$a_{n+1} = \frac{a_n + b_n + c_n + d_n}{4} \qquad b_{n+1} = \frac{\sqrt{a_n}\sqrt{b_n} + \sqrt{c_n}\sqrt{d_n}}{2}$$
$$c_{n+1} = \frac{\sqrt{a_n}\sqrt{c_n} + \sqrt{b_n}\sqrt{d_n}}{2} \qquad d_{n+1} = \frac{\sqrt{a_n}\sqrt{d_n} + \sqrt{b_n}\sqrt{c_n}}{2}$$

This provided an interesting generalization of the AGM: the convergence is quadratic (see, e.g. [SvS12], for a direct proof of this), and the limit does not depend on the order of the arguments, just as with the AGM. Borchardt also notes in his original article that one can consider a generalization to 2^g numbers, but the limit of the sequence depends on the order of the arguments. Note that Borchardt only considered sequences of positive real numbers, and hence the square roots in the definition are unambiguously defined. The Borchardt mean of four numbers is linked to genus 2 hyperelliptic integrals, via the *Richelot isogeny*; we discuss this link in Section 8.2.2, and refer to [BM88, SvS12] for more details on the real case.

We outline here the generalization of this sequence to the case where we have 2^g complex numbers.

Definition 3.4.1 (e.g., [Dup06])**.** A *Borchardt sequence of genus g* is a sequence of 2^g-uples $\left(a_0^{(n)}, \ldots, a_{2^g-1}^{(n)}\right)_{n\in\mathbb{N}}$ such that, for all n, there exists $\alpha_0, \ldots, \alpha_{2^g-1}$, square roots of $a_0^{(n)}, \ldots, \alpha_{2^g-1}^{(n)}$ (i.e. $\alpha_i^2 = a_i^{(n)}$), such that

$$\forall i \in \{0, \ldots, 2^g - 1\}, \qquad a_i^{(n+1)} = \sum_{v_1 \oplus v_2 = i} \alpha_{v_1}\alpha_{v_2}$$

where $\oplus$ denotes the bitwise XOR operation. The choice of complex square roots is *good* at the rank n if for any v_1, v_2 we have

$$|\alpha_{v_1} - \alpha_{v_2}| < |\alpha_{v_1} + \alpha_{v_2}|.$$

Note however that, given 2^g complex numbers, the existence of a good choice of signs is not guaranteed, as highlighted in Figure 3.2.

3.4.2 Choice of roots and convergence

As with the AGM, the notion of choice of square roots is crucial, and determines for instance the limit:

Theorem 3.4.2 ([Dup06, Theorem 7.1])**.** *Let* $\left(a_0^{(n)}, \ldots, a_{2^g-1}^{(n)}\right)_{n\in\mathbb{N}}$ *be a Borchardt sequence. Then there is an* $A \in \mathbb{C}$ *such that for any* $i \in \{0, \ldots, 2^g - 1\}$*,*

$$\lim_{n\to\infty} a_i^{(n)} = A.$$

Furthermore, we have $A = 0$ *if and only if the choice of square roots is not good infinitely many times.*

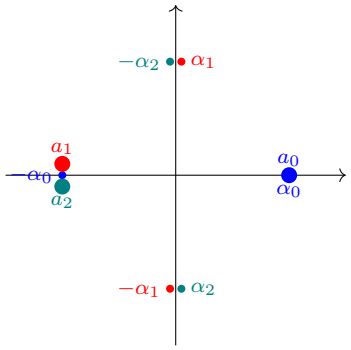

Figure 3.2: Given $a_0 = 1, a_1 = -1 - 0.1i, a_2 = -1 + 0.1i$, any choice of square roots lead to a wrong choice of signs.

Remark 3.4.3. Given the definition of good choices of square roots (in which the inequality must be strict), a choice of square roots is bad as soon as one of the $a_i^{(n)}$ is equal to 0. However, unlike the AGM, this does not necessarily mean the sequence converges to 0, as $a_i^{(n+1)}$ is in general different from 0. On the other hand, if at least half of the $a_i^{(n)}$ are 0, it is easy to check that at least half of the $a_i^{(n+k)}$ will be equal to 0; the limit of the sequence is then 0, and the choice of square roots are necessarily always bad.

Definition 3.4.4. Let $\left(a_0^{(0)}, \dots, a_{2^g-1}^{(0)}\right) \in \mathbb{C}^{2^g}$, and assume that one can define the sequence $\left(a_0^{(n)}, \dots, a_{2^g-1}^{(n)}\right)_{n\in\mathbb{N}}$ with good choices of signs at each step. Then the *Borchardt mean* is the limit of this sequence; we denote it $\mathcal{B}_g\left(a_0^{(0)}, \dots, a_{2^g-1}^{(0)}\right) \neq 0$.

We also have:

Theorem 3.4.5 ([Dup06, Prop. 7.1, p. 163])**.** *Let $\left(a_0^{(n)}, \dots, a_{2^g-1}^{(n)}\right)_{n\in\mathbb{N}}$ be a Borchardt sequence such that $\mathrm{Re}(a_i^{(0)}) > 0$, and such that all the square roots are chosen with positive real part. Let N be such that for all i we have*

$$|a_i^{(N)} - a_0^{(N)}| \leq 0.2247|a_0^{(N)}|. \tag{3.4.1}$$

Then $|A - a_0^{(N+k)}| \leq 1.43 M_N \times 0.7867^{2^k}$, with $M_N = \max_i |a_i^{(N)}|$.

The condition of Equation (3.4.1) is always satisfied for N large enough (see [Dup06, p. 164]). Hence, this theorem establishes the quadratic convergence of any Borchardt sequence for which the elements have positive real parts, and for which the square roots are always chosen with positive real part.

Note that, provided wrong choices of square roots do not happen infinitely often, one can always fall back to this case: the sequence $\left(\frac{a_0^{(n)}}{A}, \dots, \frac{a_{2^g-1}^{(n)}}{A}\right)_{n\in\mathbb{N}}$ (which is a Borchardt sequence) converges to $(1, \dots, 1)$, which means that after a certain number of terms the real part of the sequence is always strictly positive. Furthermore, when the $a_i^{(n)}$ are close to 1 (and to each other), choosing the square root with positive real part corresponds to the good choice of signs; hence, when always choosing the square roots with positive real parts, the choice of signs is good after a while. Hence:

Theorem 3.4.6 ([Dup06, Section 7.4.2]). *Any Borchardt sequence with a finite number of wrong choices of sign converges quadratically; in particular, computing the limit of a Borchardt sequence with precision P can be done in $O(\mathcal{M}(P)\log P)$.*

This means in particular that one can compute $\mathcal{B}_g\left(a_0^{(0)},\dots,a_{2^g-1}^{(0)}\right)$ with precision P in quasi-linear time.

3.4.3 Link with the theta-constants

Recall the genus g τ-duplication formulas (Equation (2.2.1)):

$$\theta_{[a;b]}(z,\tau)^2 = \frac{1}{2^g}\sum_{\beta\in\frac{1}{2}\mathbb{Z}^g/\mathbb{Z}^g} e^{-4i\pi\,{}^t a\beta}\theta_{[0;b+\beta]}\left(z,\frac{\tau}{2}\right)\theta_{[0;\beta]}\left(0,\frac{\tau}{2}\right).$$

The addition in the characteristics is addition modulo $\frac{1}{2}\mathbb{Z}^g/\mathbb{Z}^g$. We have the following group isomorphism:

$$\begin{aligned}\phi:\frac{1}{2}\mathbb{Z}^g/\mathbb{Z}^g &\to \{0,\dots,2^g-1\}\\ (\dots,0,\delta_i,0,\dots) &\mapsto \begin{cases}2^i \text{ if } \delta_i=\frac{1}{2} \pmod 1\\ 0 \text{ if } \delta_i=0 \pmod 1\end{cases}\\ a+b &\mapsto \phi(a)\oplus\phi(b)\end{aligned}$$

where $\oplus$ is the bitwise XOR operation. Hence, using the numbering described in Note 2.1.5, define $a_i(\tau)=\theta_i^2(0,\tau)$; then the τ-duplication formulas for the fundamental thetas can be rewritten as

$$a_i(2\tau)=\sum_{v_1\otimes v_2=i}\sqrt{a_{v_1}(\tau)}\sqrt{a_{v_2}(\tau)}$$

for some choice of square roots: this is exactly the definition of the Borchardt mean. Hence

Proposition 3.4.7. *The sequence $\left(\theta_0^2(0,2^n\tau),\dots,\theta_{2^g-1}^2(0,2^n\tau)\right)_{n\in\mathbb{N}}$ is a Borchardt sequence.*

Furthermore, Proposition 2.1.9 shows that the sequence converges to $(1,1,1,1)$. In fact, this sequence even converges quadratically; this means that the choice of square roots which corresponds to computing $\theta_i(0,2^k\tau)$ from $\theta_i^2(0,2^k\tau)$ (which, when τ is large enough, actually corresponds to picking the square roots with positive real part) is a good choice of square roots (in the sense of Definition 3.4.1) for all but a finite number of steps.

The most interesting case is the one for which good choices of signs always coincide with theta constants at $2^k\tau$:

Definition 3.4.8. Define $\mathcal{U}_g$ as the set of $\tau\in\mathcal{H}_g$ such that $\mathcal{B}_g\left(\theta_0^2(0,\tau),\dots,\theta_{2^g-1}^2(0,\tau)\right)$ is defined and equal to 1; that is to say, good choices of square roots always exist, and always choosing them gives rise to the sequence $\left(\theta_0^2(0,2^n\tau),\dots,\theta_{2^g-1}^2(0,2^n\tau)\right)_{n\in\mathbb{N}}$.

Proposition 3.2.9 proves that $\mathcal{D}_2\subset\mathcal{U}_1$, and in particular $\mathcal{F}_1\subset\mathcal{U}_1$. In genus 2, we have that $\mathcal{F}_2\subset\mathcal{U}_2$ [Dup06, Prop. 9.6, p. 196]. We were also able to prove a slightly better result in genus 2, which we mention in Chapter 7 (Proposition 7.1.2).

Studying this domain is interesting in the context of quasi-linear algorithms to compute theta-constants (see Chapter 7); however, even in genus 2, it is not an easy task. In particular, a result establishing the stability of this domain under the action of some matrices is still a conjecture

(see, e.g. [Dup06, Conjecture 9.1] and [ET14a, Conjecture 9]). We describe how we sidestep this difficulty in practice, in the context of our fast algorithm for theta functions and theta-constants, in Chapter 7.

To finish, recall that, in genus 1, a description of the limits of AGM sequences starting at the squares of theta-constants can be found explicitly (Theorem 3.2.14), which also shows that the inverse of these limits form a lattice. This result generalizes to genus 2, i.e. the set of limits can be written as the set of $\frac{\theta_j^2(0,\tau)}{\theta_j^2(0,\gamma\cdot\tau)}$ for γ in a subgroup of $\mathrm{Sp}_4(\mathbb{Z})$; this is [Dup06, Chapter 8]. However, this result does not appear to generalize to genus 3 and above[2].

[2]The main obstacle is in the generalization of [Dup06, Lemme 8.1], where τ-duplication formulas are used to show that any choice of sign transforms the set of $\theta_i^2(\tau)$ into a set of $\theta_i^2(2\gamma\tau)$ for some γ. Changing the sign of only one of the theta-constants gives formulas with one "minus" sign and $2^{g-1}-1$ "plus" signs, but τ-duplication formulas have an equal number of "plus" and "minus" signs; for $g > 2$, we cannot reconcile the two formulas.

Chapter 4

The Landen isogeny

This section is devoted to the *Landen isogeny*, which is an important 2-isogeny between elliptic curves over the complex numbers (and in fact over the real numbers as well). This isogeny shows up as a change of variables in elliptic integrals. Interestingly, this gives a method to compute periods of an elliptic curve, by repeatedly applying the Landen isogeny; this strategy has been described in [BM88] in a specific case, then generalized (via lattice chains) in [CT13]. A similar strategy allows us to solve the elliptic logarithm; hence, this chapter proves:

Theorem 4.0.1. *The genus 1 Abel-Jacobi map can be computed using quadratically convergent sequences.*

This gives algorithms with complexity of $O(\mathcal{M}(P)\log P)$ to compute the Abel-Jacobi map with absolute precision P, provided one takes into account the loss of precision. In the real case, it seems that a few papers (e.g. [LO94] or [LO98]) have shown that the precision lost is of no asymptotic importance. As for the complex elliptic logarithm, we prove that only $O(\log P)$ guard bits are needed. Hence throughout this chapter, we will assume that precision loss has no impact over the running time of the algorithm, i.e. that they are at most $O(P)$; this allows us to claim a $O(\mathcal{M}(P)\log P)$ algorithm for the complex genus 1 Abel-Jacobi map.

We also show an algorithm to compute the Weierstrass $\wp$ function with estimated complexity $O(\mathcal{M}(P)\log P)$, once again using the Landen isogeny. However, this algorithm suffers from numerical instability: as the periods get close to the edges of the fundamental parallelogram, the accuracy of the result is greatly reduced, even for τ in the fundamental domain. Another approach, not based on the Landen transform, achieves this complexity without this numerical instability; we outline it in Chapter 8.

4.1 The real case (Bost-Mestre)

In this section, E is an elliptic curve defined over $\mathbb{R}$ by the Weierstrass equation $E : y^2 = 4(x-e_1)(x-e_2)(x-e_3)$, with e_1, e_2, e_3 *real* and distinct, and $\sum e_i = 0$. This section establishes the connections between periods of this elliptic curve, the AGM, and a certain chain of 2-isogenies. We follow the presentation of [BM88], and refer to it for the proof of most statements.

4.1.1 Elliptic integrals and period computation

The following proposition was known to Gauss and Lagrange:

Proposition 4.1.1. *For any positive real numbers a, b:*

$$\int_0^{2\pi} \frac{dt}{\sqrt{a^2\cos^2 t + b^2 \sin^2 t}} = \frac{2\pi}{\mathrm{AGM}(a,b)}$$

Proof. If one makes the change of variables

$$\sin t = \frac{2a \sin t'}{(a+b) + (a-b)\sin^2 t'}$$

a careful calculation shows that

$$\int_0^{2\pi} \frac{dt}{\sqrt{a^2\cos t + b^2 \sin t}} = \int_0^{2\pi} \frac{dt'}{\sqrt{\left(\frac{a+b}{2}\right)^2 \cos^2 t' + (ab)\sin^2 t'}}$$

Hence, for all $n \geq 0$

$$\int_0^{2\pi} \frac{dt}{\sqrt{a^2\cos t + b^2 \sin t}} = \int_0^{2\pi} \frac{dt}{\sqrt{a_n^2 \cos^2 t + b_n^2 \sin^2 t}}$$

and, taking the limit when $n \to \infty$,

$$\int_0^{2\pi} \frac{dt}{\sqrt{a^2\cos t + b^2 \sin t}} = \int_0^{2\pi} \frac{dt}{\sqrt{\mathrm{AGM}(a,b)^2 \cos^2 t + \mathrm{AGM}(a,b)^2 \sin^2 t}} = \frac{2\pi}{\mathrm{AGM}(a,b)^2}. \qquad \square$$

Furthermore:

Proposition 4.1.2. *Let $P = 4(X-e_1)(X-e_2)(X-e_3)$ with $e_3 < e_2 < e_1$. Then*

$$\begin{aligned} \int_{e_1}^{+\infty} \frac{dx}{\sqrt{P(x)}} &= \int_{e_3}^{e_2} \frac{dx}{\sqrt{P(x)}} = \frac{\pi}{2\,\mathrm{AGM}(\sqrt{e_1-e_3}, \sqrt{e_1-e_2})} \\ \int_{-\infty}^{e_3} \frac{dx}{\sqrt{-P(x)}} &= \int_{e_2}^{e_1} \frac{dx}{\sqrt{-P(x)}} = \frac{\pi}{2\,\mathrm{AGM}(\sqrt{e_1-e_3}, \sqrt{e_2-e_3})} \end{aligned}$$

Proof. The first half of the first identity can be proven using the change of variables

$$x' = \frac{e_2 x - e_1 e_2 + e_1 e_2 - e_2 e_3}{x - e_2}$$

while the second half can be proven using the change of variables

$$x = e_3 + (e_2 - e_3)\sin^2 t. \qquad \square$$

Recall (Proposition 1.3.2) that periods can be defined as integrals of the invariant differential following paths around the branch cuts, as in [Sil86, Section VI.1]. Hence:

Proposition 4.1.3. *Let $P = 4(X-e_1)(X-e_2)(X-e_3)$ with $e_3 < e_2 < e_1$, and let E be the elliptic curve defined over $\mathbb{C}$ by $E : y^2 = P(x)$. Define*

$$\omega_1 = 2\int_{e_3}^{e_2} \frac{dx}{\sqrt{P(x)}}, \qquad \omega_2 = 2i\int_{e_2}^{e_1} \frac{dx}{\sqrt{-P(x)}}$$

Then ω_1, ω_2 are periods of E. Hence, Proposition 4.1.2 shows that

$$\omega_1 = \frac{\pi}{\mathrm{AGM}(\sqrt{e_1-e_3}, \sqrt{e_1-e_2})}, \qquad \omega_2 = \frac{\pi}{\mathrm{AGM}(\sqrt{e_1-e_3}, \sqrt{e_2-e_3})}$$

We evaluate precision loss in this algorithm; we suppose that the periods are of bounded size, in order to turn results on relative precision into results on absolute precision. The worst case happens when the roots are very close to each other at precision P, for instance $e_1 - e_3 = 2^{-P}$. In that case, the computation of the arguments of the AGM loses up to $P/2$ bits; furthermore, since $\log_2|\log_2|z|| = O(\log P)$, Theorem 3.2.6 shows that $O(\log P)$ guard bits are needed, and the number of iterations is still $O(\log P)$. This proves that, for P large enough, it is enough to work at precision $4P$ to get a result accurate to P bits. Hence, this gives indeed a $O(\mathcal{M}(P)\log P)$ algorithm.

4.1.2 2-isogenies

The changes of variable of the previous subsection can be interpreted in terms of isogenies. Define $E : y^2 = (x-e_1)(x-e_2)(x-e_3)$ with $e_3 < e_2 < e_1$, $\sum e_i = 0$. Put

$$a = \sqrt{e_1 - e_3}, \quad b = \sqrt{e_1 - e_2}, \qquad a_1 = \frac{a+b}{2}, \quad b_1 = \sqrt{ab}$$

and put

$$e_1' = \frac{a_1^2 + b_1^2}{3}, \quad e_2' = \frac{a_1^2 - 2b_1^2}{3}, \quad e_3' = \frac{b_1^2 - 2a_1^2}{3}.$$

Then, $e_3' < e_2' < e_1'$, $e_1' - e_3' = a_1^2$, $e_1' - e_2' = b_1^2$ and $\sum e_i' = 0$. Equation (4.1.1), along with the changes of variables of Proposition 4.1.2, gives:

$$\int_{e_1}^{\infty} \frac{dx}{\sqrt{4(x-e_1)(x-e_2)(x-e_3)}} = \int_{e_1'}^{\infty} \frac{dx}{\sqrt{4(x-e_1')(x-e_2')(x-e_3')}}$$

In fact, the combination of these changes of variables allows us to write this equation as a consequence of the change of variables

$$x = x' + \frac{(e_3' - e_1')(e_3' - e_2')}{x' - e_3'} \tag{4.1.1}$$

This change of variables actually defines an isogeny called the *Landen isogeny* or the *Landen transform*:

Theorem 4.1.4 (Landen transform, e.g. [BM88])**.** *Let $E : y^2 = 4(x-e_1)(x-e_2)(x-e_3)$ be an elliptic curve over $\mathbb{R}$, with $e_1 > e_2 > e_3$, and $E' : y^2 = 4(x-e_1')(x-e_2')(x-e_3')$ with e_1', e_2', e_3' defined from e_1, e_2, e_3 as previously. Define the map*

$$\begin{aligned}
\phi : E' &\to E \\
[0:1:0] &\mapsto [0:1:0] \\
[e_3' : 0 : 1] &\mapsto [0:1:0] \\
[x' : y' : 1] &\mapsto \left[x' + \frac{(e_3' - e_1')(e_3' - e_2')}{x' - e_3'} : y' \left(1 - \frac{(e_3' - e_1')(e_3' - e_2')}{(x - e_3')^2} \right) : 1 \right]
\end{aligned}$$

Then ϕ is a 2-isogeny, i.e. an isogeny of degree 2.

Note that this amounts to considering the 2-isogeny whose kernel is the 2-torsion point $(e_3', 0)$, instead of other ones whose kernels are $(e_1, 0)$ or $(e_2, 0)$. Note that $(e_1', 0), (e_2', 0), (e_3', 0)$ are respectively the images by $\wp$ of the points $\frac{\omega_1'}{2}, \frac{\omega_1'+\omega_2'}{2}, \frac{\omega_2'}{2}$; hence, $\frac{\omega_2'}{2} = 0 \pmod{\Lambda}$. The action of the Landen isogeny on the periods is thus

$$\omega_1' = \omega_1, \qquad \omega_2' = 2\omega_2 \tag{4.1.2}$$

Furthermore, the isogeny between the two tori can be written as $\phi(z) = z \pmod{\Lambda}$.

The successive changes of variables can be written as a "chain of 2-isogenies". Define (a_n, b_n) to be the sequence one obtains when computing $\mathrm{AGM}(a, b)$, and

$$e_1^{(n)} = \frac{a_n^2 + b_n^2}{3}, \qquad e_2^{(n)} = \frac{a_n^2 - 2b_n^2}{3}, \qquad e_3^{(n)} = \frac{b_n^2 - 2a_n^2}{3} \tag{4.1.3}$$

Isogenies can once again be defined, using formulas that are analogous to Theorem 4.1.4:

$$f_n : E_{n+1} : y^2 = 4(x - e_1^{(n+1)})(x - e_2^{(n+1)})(x - e_3^{(n+1)}) \to E_n : y^2 = 4(x - e_1^{(n)})(x - e_2^{(n)})(x - e_3^{(n)})$$

This constructs a chain of 2-isogenies:

$$\ldots \to E_{n+1} \xrightarrow{f_n} E_n \to \ldots \to E' \xrightarrow{f} E$$

Since $\lim a_n = \lim b_n = \mathrm{AGM}(a, b)$, we have

$$\lim e_1^{(n)} = \frac{2}{3}\,\mathrm{AGM}(a, b)^2, \qquad \lim e_2^{(n)} = \lim e_3^{(n)} = -\frac{1}{3}\,\mathrm{AGM}(a, b)^2.$$

This means the equation of the curve "at the limit" is

$$y^2 = P_\infty(x) = 4\left(x + \frac{1}{3}\,\mathrm{AGM}(a, b)^2\right)^2\left(x - \frac{2}{3}\,\mathrm{AGM}(a, b)^2\right)$$

We have

$$\begin{aligned}\int \frac{dt}{\sqrt{P_\infty(t)}} &= \int \frac{du}{\mathrm{AGM}(a, b)^2 + u^2} \qquad \left(\text{putting } u = \sqrt{x - \frac{2}{3}\,\mathrm{AGM}(a, b)^2}\right) \\ &= \frac{1}{\mathrm{AGM}(a, b)}\mathrm{Arctan}\left(\frac{\sqrt{x - \frac{2}{3}\,\mathrm{AGM}(a, b)^2}}{\mathrm{AGM}(a, b)}\right)\end{aligned}$$

and hence we have for instance

$$2\int_{e_1}^{\infty} \frac{dx}{\sqrt{P(x)}} = \omega_1 = 2\int_{\frac{2}{3}\,\mathrm{AGM}(a,b)^2}^{+\infty} \frac{dx}{\sqrt{P_\infty(x)}} = \frac{\pi}{\mathrm{AGM}(a, b)}.$$

4.1.3 Elliptic logarithm

Recall the definition of the elliptic logarithm map (Theorem 1.3.4):

$$P = [x : y : 1] \mapsto \int_x^{\infty} \frac{dx}{\sqrt{P(x)}}$$

In the real case, this incomplete integral can be computed in a similar way as the periods, that is to say by repeatedly applying Landen's transform. This requires keeping track of the bound of the integral, i.e. being able to compute x' such that

$$\int_x^{\infty} \frac{dt}{\sqrt{4(t - e_1^{(0)})(t - e_2^{(0)})(t - e_3^{(0)})}} = \int_{x'}^{\infty} \frac{dt}{\sqrt{4(t - e_1^{(1)})(t - e_2^{(1)})(t - e_3^{(1)})}}$$

Algorithm 4 Compute the elliptic logarithm over the reals.
Input: $u \in \mathbb{R}$ with absolute precision P.
Output: $\int_u^\infty$ with absolute precision P.

1: $a \leftarrow \sqrt{e_1 - e_3}, \quad b \leftarrow \sqrt{e_1 - e_2}$
2: $x \leftarrow u$
3: **while** $a \neq b$ at the given precision **do**
4: $\quad A = \frac{a+b}{2}, \quad B = \sqrt{ab}$
5: $\quad X = \frac{1}{2}\left(x - \frac{a^2+b^2}{6} + \sqrt{\left(x + \frac{a^2+b^2}{6}\right)^2 - \left(\frac{a^2-b^2}{2}\right)^2}\right)$
6: $\quad a = A, b = B, x = X$
7: **end while**
8: return $\frac{2}{a}\left(\frac{\pi}{2} - \arctan\frac{\sqrt{x - \frac{2}{3}a^2}}{a}\right)$

The computation of x' can be done by writing Equation (4.1.1) as a degree 2 polynomial in x' (with coefficients depending on x) and solving for x'. The resulting algorithm, given in [BM88], is Algorithm 4.

As with the computation of periods, the loss of precision can be significant, most of all because of the extraction of square roots for numbers potentially close to 0. We did not manage to evaluate the precision loss in the computation of X; however, [LO94] comments that if u is close e_1 it is possible that up to $P/2$ bits are inaccurate in the final result[3], which does not change the asymptotics. Hence, working with precision $O(P)$ should be enough to compensate for the losses in precision. Note that, in any case, a variation on this algorithm which reduces the loss of relative precision to $O(\log P)$ is studied in [LO98].

As for the running time, we analyze it as follows. Each step in the loop costs $O(\mathcal{M}(P))$ bit operations; since the AGM is quadratically convergent, there are $O(\log P)$ such steps. The last step requires us to compute $\arctan(x)$, which can be done in $O(\mathcal{M}(P)\log P)$ bit operations, as in [BZ10, Section 4.8.5]; we use

$$\arctan(x) = \mathrm{Im}\left(\log(1 + ix)\right)$$

and use the AGM to compute $\log(x)$ in $O(\mathcal{M}(P)\log P)$ operations. In the end, this algorithm computes the elliptic logarithm in $O(\mathcal{M}(P)\log P)$.

4.2 The complex case (Cremona-Thongjunthug)

We now turn to the general case, i.e. an elliptic curve E defined over $\mathbb{C}$ by a Weierstrass equation of the form $E(\mathbb{C}) : y^2 = (x - e_1)(x - e_2)(x - e_3)$. The previous section dealt with the case where the e_i are real, and computed the periods and the elliptic logarithm using a link with the real AGM.

Generalizing this approach to the complex case involves working with the complex AGM, which means one must consider the problem of choosing the correct signs; furthermore, unlike the real case, there is no neat way to pick an ordering of the roots. The general method and theorems in this section are taken from [CT13]; we skip over some details, for instance the case of rectangular lattices, so as to streamline the presentation.

[3]The comment seems to be about relative precision; however, if $a - b$ creates a large loss of relative precision, there is also a large loss of absolute precision since we compute $\sqrt{a - b}$, and the remark is thus still valid.

4.2.1 Lattice chains

The notion of *lattice chains* is introduced in order to study the behavior and the properties of the period lattices of the curves appearing in the chain of 2-isogenies given by the Landen transform.

Definition 4.2.1 ([CT13, Section 3])**.** Let $(\Lambda_n)_{n\in\mathbb{N}}$ be a sequence made of lattices of $\mathbb{C}$; it is a *chain of lattices of index 2* if the following conditions are satisfied:

1. $\Lambda_{n+1} \subset \Lambda_n$ for all $n \geq 0$;
2. $[\Lambda_n : \Lambda_{n+1}] = 2$ for all $n \geq 0$;
3. $\Lambda_{n+1} \neq 2\Lambda_{n-1}$ for any $n \geq 1$.

Thus for each $n \geq 1$ we have

$$\Lambda_{n+1} = \langle w \rangle + 2\Lambda_n$$

for some $w \in \Lambda_n \setminus 2\Lambda_{n-1}$.

Given a lattice Λ_0, there are three possible choices for Λ_1; then, for any $k \geq 2$, there are only two possible choices, the third one being excluded by the last condition. A notion of *right choice* of sublattice can actually be defined:

Definition 4.2.2. For $n \geq 1$ we say that $\Lambda_{n+1} \subset \Lambda_n$ is the right choice of sublattice if $\Lambda_{n+1} = \langle w \rangle + 2\Lambda_n$ where $w \in \Lambda_n \setminus 2\Lambda_{n-1}$ and $|w|$ is minimal in this quotient.

Definition 4.2.3.

- A chain is good if and only if $\Lambda_{n+1} \subset \Lambda_n$ is the right choice for all but finitely many $n \geq 1$.
- A chain is optimal if and only if $\Lambda_{n+1} \subset \Lambda_n$ is the right choice for *all* $n \geq 1$. There is usually one optimal chain for each choice of $\Lambda_1 \subset \Lambda_0$.

Proposition 4.2.4. *A chain is good if and only if Λ_∞ is of rank 1, in which case we note w_∞ a generator of Λ_∞, the* limiting period of the good chain*. For all but finitely many n, w_∞ is the smallest element of Λ_n. If the chain is not good, Λ_∞ is of rank 0.*

The analogy with the complex AGM is very noticeable: at each step of the sequence there are two possible choices, only one of which is defined as the right choice; and a sequence can has all but finitely many right choices, in which case its limit shows a non-zero element, or an infinite number of right choices, in which case we get 0.

The first step in establishing the link between lattice chains and AGM sequences is to link lattices to the initial values (a_0, b_0) of the AGM sequence.

Definition 4.2.5. A *short lattice chain* of order 4 is a chain $\Lambda_2 \subset \Lambda_1 \subset \Lambda_0$ such that Λ_0/Λ_2 is cyclic of order 4.

Proposition 4.2.6. *There a bijection between*

- *Short lattice chains of order 4 $\Lambda_2 \subset \Lambda_1 \subset \Lambda_0$, up to homothety;*
- *Unordered pairs of nonzero complex numbers $a, b \in \mathbb{C}$ such that $a^2 \neq b^2$, identifying (a, b) and $(-a, -b)$;*
- *Triples (E, ω, H) with E an elliptic curve defined over $\mathbb{C}$, ω a holomorphic differential on E, H a cyclic subgroup of $E(\mathbb{C})$ of order 4, up to isomorphisms preserving H.*

Theorem 4.2.7. *Let $\Lambda_2 \subset \Lambda_1 \subset \Lambda_0$ be a short lattice chain corresponding to a, b. Then Λ_2 is the right choice of sublattice for Λ_1 if and only if the pair (a, b) is good in the sense of the AGM.*

4.2.2 2-isogenies

The connection with the real case is made explicit here; however, we do not write down all the formulas, as they are exactly the same as the ones in Section 4.1. We refer to [CT13] for full details.

Let $E = E_0 : y^2 = P(x)$ with P of degree 3, and take Λ_0 such that $E_0(\mathbb{C}) \simeq \mathbb{C}/\Lambda_0$. The change of variables given by the Landen transform (Equation (4.1.1)) gives a 2-isogeny $\phi_1 : E_1 \to E_0$, with $E_1 \simeq \mathbb{C}/\Lambda_1$. However, note that the definition of the isogeny depends on the e_i', which are defined (Equation (4.1.3)) from the quantities $a_0 = \pm\sqrt{e_1 - e_3}, b_0 = \pm\sqrt{e_1 - e_2}$, themselves defined from a labelling of the roots of P. Looking more closely, if e_2 and e_3 are switched, a and b are switched, but this does not affect the e_i' or the rest of the sequence; hence, there are only three possibilities for Λ_1, depending on which root is labeled e_1.

Iterating the change of variables gives a chain of 2-isogenies $\phi_n : E_n \to E_{n-1}$ and the roots $e_i^{(n)}$ of the polynomial P_n such that $E_n : y^2 = P_n(x)$ are defined using Equation (4.1.3):

$$e_1^{(n)} = \frac{a_n^2 + b_n^2}{3}, \qquad e_2^{(n)} = \frac{a_n^2 - 2b_n^2}{3}, \qquad e_3^{(n)} = \frac{b_n^2 - 2a_n^2}{3}$$

Note that $a_n^2 = e_1^{(n)} - e_3^{(n)}, b_n^2 = e_1^{(n)} - e_2^{(n)}$. Each change of variables requires the computation of one term of an AGM sequence starting at (a_0, b_0); hence there are two choices for (a_n, b_n).

Note that Equation (4.1.3) can be rewritten as

$$e_1^{(n+1)} = \frac{e_1^{(n)} + 2a_n b_n}{4}, \quad e_2^{(n+1)} = \frac{e_1^{(n)} - 2a_n b_n}{4}, \quad e_3^{(n+1)} = \frac{-e_1^{(n)}}{2}$$

Hence, taking $(a_n, -b_n)$ instead of (a_n, b_n) switches $e_1^{(n+1)}$ and $e_2^{(n+1)}$; this does not change the equation of E_{n+1}, however it changes the equation of E_{n+2}. Hence, each choice of root for the n-th term of the AGM sequence corresponds to a curve E_{n+2}. Furthermore, the same reasoning can be applied to the choice of signs in a_0, b_0, identifying (a_0, b_0) and $(-a_0, -b_0)$; this gives two possibilities for Λ_2.

In the end, this gives:

Theorem 4.2.8. *There is a bijection between*

- *The AGM sequences starting with (a_0, b_0);*
- *The isogeny chains starting with the short chain $E_2 \to E_1 \to E_0$;*
- *The lattice chains starting with the short chain $\Lambda_2 \subset \Lambda_1 \subset \Lambda_0$.*

Furthermore, we also have

$$\begin{aligned} \Lambda_{n+2} \text{ is the right choice of sublattice for } \Lambda_{n+1} &\iff (a_n, b_n) \text{ is good} \\ \text{The lattice chain is good} &\iff \text{the AGM sequence is good} \\ \text{The lattice chain is optimal} &\iff \text{the AGM sequence is optimal} \end{aligned}$$

4.2.3 Period computation

Note that the results above show that each good lattice chain starting at Λ_0 determines a period (from the definition of a good lattice chain) and a good AGM sequence. The connection between both is given by the proposition:

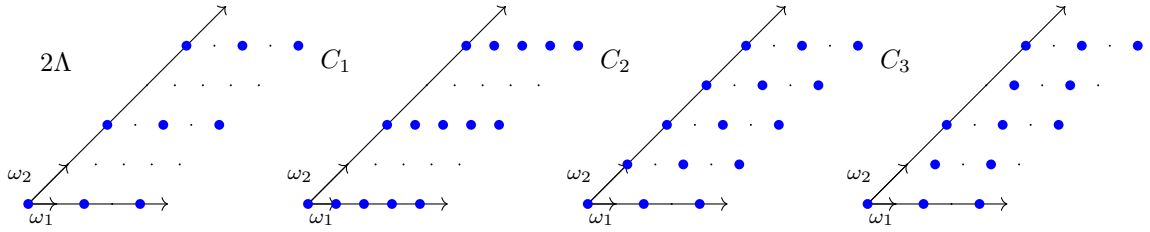

Figure 4.1: Three cosets of 2Λ in $\Lambda = \mathbb{Z}\omega_1 + \mathbb{Z}\omega_2$, for $\omega_1 = 1, \omega_2 = 2\sqrt{2}\,(1+i)$.

Proposition 4.2.9. *Let (Λ_n) be a good lattice chain with limiting period ω, and let the corresponding good AGM sequence be (a_n, b_n) with limit $M \neq 0$. Then $M = \pm\frac{\pi}{\omega}$.*

This means that when ω runs over all the points of Λ_0, $\frac{\pi}{\omega}$ describes all the limits of good AGM sequences starting at (a_0, b_0); this yields another proof of Theorem 3.2.14.

Optimal AGM sequences can actually be used to yield a $\mathbb{Z}$-basis of the lattice. If $\Lambda = \mathbb{Z}\omega_1 + \mathbb{Z}\omega_2$, define the cosets of 2Λ in Λ $C_j = \langle\omega_i\rangle + 2\Lambda$, with $\omega_3 = \omega_1 + \omega_2$. Define a set of *minimal coset representatives* as a triple (c_1, c_2, c_3) such that $|c_i|$ is minimal in C_i. The following proposition confirms an intuition one might have when looking at Figure 4.1:

Proposition 4.2.10. *Let Λ be a non-rectangular lattice, and let w_1, w_2, w_3 be minimal coset representatives of 2Λ in Λ. Then any 2 of them are a $\mathbb{Z}$-basis of the lattice, and $w_3 = \pm(w_1 \pm w_2)$.*

The link with optimal chains is as follows:

Theorem 4.2.11. *A good chain is optimal if and only if w_∞ is one of the minimal coset representatives for Λ_0. This means that in general, any non-rectangular lattice has 3 optimal chains, one per coset C_i.*

Now, setting $\Lambda_1 = C_i$ and picking Λ_2 as the right choice for Λ_1 gives a good pair (a_0, b_0); the optimal AGM sequence can then be computed starting with this good pair, and this gives $\mathrm{AGM}(a_0, b_0) = \frac{\pi}{c_i}$ where c_i is a minimal coset representative. Repeating this with other cosets gives a $\mathbb{Z}$-basis of Λ. Keeping in mind the link with elliptic curves, this gives the following generalization of Proposition 4.1.3:

Theorem 4.2.12. *Let E be an elliptic curve over $\mathbb{C}$ given by $Y^2 = (X - e_1)(X - e_2)(X - e_3)$. Let Λ be the period lattice. Define $a_0 = \sqrt{e_1 - e_3}, b_0 = \sqrt{e_1 - e_2}$ and choose the signs such that (a_0, b_0) is good. Define $\omega_1 = \frac{\pi}{\mathrm{AGM}(a_0, b_0)}$; then ω_1 is a period of E. Labelling the other two roots as e_1 and repeating this process gives two other such periods. Then any two of the three periods form a basis of Λ.*

The process can actually be simplified, as follows. Define $a^2 = e_1 - e_3$, $b^2 = e_1 - e_2$, $c^2 = e_2 - e_3$, so $a^2 = b^2 + c^2$, and pick the signs of a, b, c so that

$$|a - b| \leq |a + b|, \quad |c - ib| \leq |c + ib|, \quad |a - c| \leq |a + c|$$

This is always possible, but we may need to pick another labelling of the roots. Then the pairs $(a, b), (c, ib), (ia, ic)$ are good, and computing $\frac{\pi}{\mathrm{AGM}(a,b)}, \frac{\pi}{\mathrm{AGM}(c,ib)}, \frac{\pi}{\mathrm{AGM}(ia,ic)}$ gives three periods. Note that applying this process to a rectangular lattice gives a procedure very similar (after multiplying the roots by a suitable complex number so they are all real) to [BM88].

4.2.4 Elliptic logarithm

The algorithm for elliptic logarithm in the real case (Algorithm 4) requires the extraction of a square root at each step for the computation of the value of x; hence, in the complex case, the question of which square root to choose needs to be settled. An equivalent way of saying this on elliptic curves is to say that, since ϕ_n is two-to-one as a 2-isogeny, there are uncountably many point sequences (P_n) starting from a point P_0.

The problem is recast in [CT13] in terms of *coherent point sequences*.

Definition 4.2.13. Let (Λ_n) be an optimal lattice chain. A sequence (P_n) of points is called *coherent* if there is $z \in \mathbb{C}$ such that $P_n = (\wp(z, \Lambda_n), \wp'(z, \Lambda_n))$ for all n. If that z exists, it is uniquely determined modulo $\cap \Lambda_n = \langle \omega_1 \rangle$.

The problem of finding a coherent point sequence is solved by writing the change of variables given by the Landen transform as the composition of two maps. Define the curves

$$E'_n : R^2 = \frac{T^2 + a_n^2}{T^2 + b_n^2}$$

These are projective curves in $\mathbb{P}^1 \times \mathbb{P}^1$, with points at infinity $(\infty, \pm 1)$ and $(\pm bi, \infty)$. Then, define the map

$$\alpha_n : E'_{n+1} \to E_n, \qquad \alpha_n(r, t) = (t^2 + e_1^{(n)}, -2rt(t^2 + b_n^2))$$

where $E_n, a_n, b_n, e_i^{(n)}$ are defined as in Section 4.2.2.

The curves E'_n and E_n are also isomorphic, via the isomorphism

$$\begin{aligned}
\theta_n : E'_n &\to E_n \\
\theta_n(t_n, r_n) &= (x_n, y_n) = \left(\frac{t_n^2 + r_n(t_n^2 + a_n^2) + \frac{a_n^2 + b_n^2}{6}}{2}, t_n \left(t_n^2 + r_n(t_n^2 + a_n^2) + \frac{a_n^2 + b_n^2}{2} \right) \right) \\
\theta_n^{-1}(x_n, y_n) &= (r_n, t_n) = \left(\frac{3y_n}{6x_1 + a_n^2 + b_n^2}, \frac{12x_n + 5a_n^2 - b_n^2}{12x_n + 5b_n^2 - a_n^2} \right).
\end{aligned}$$

This is summarized by a commutative diagram:

$$\begin{array}{ccccccccccc}
\ldots & \to & E'_n & \xrightarrow{\phi'_n} & E'_{n-1} & \to & \ldots & \to & E'_1 & & \\
 & & \downarrow \theta_n & & \downarrow \theta_{n-1} & & & & \downarrow \theta_1 & \alpha_0 \searrow & \\
\ldots & \to & E_n & \xrightarrow{\phi_n} & E_{n-1} & \to & \ldots & \to & E_1 & \xrightarrow{\phi_1} & E_0
\end{array}$$

Following a coherent point sequence (r_n, t_n) on the sequence of curves E'_n is easier than on the E_n: this gives the relations

$$r_n^2 = \frac{a_{n-1}(r_{n-1} + 1)}{b_{n-2} r_{n-1} + a_{n-2}}, \qquad t_n = r_n t_{n-1}$$

with $\mathrm{Re}(r_n) \geq 0$ [CT13, Prop. 26], which removes any sign ambiguity. The value of the elliptic logarithm is then given by:

Theorem 4.2.14. *Let (P_n) be a coherent point sequence generated by $z \in \mathbb{C}$. Assume that $2z \notin \Lambda_\infty$. Then for n large enough $P_n \neq \mathcal{O}_{E_n}$. Set $M = \frac{\pi}{\omega_1}$; we have*

$$t_\infty = -\frac{1}{2} \frac{y_\infty}{x_\infty + M^2/3}$$

and $t_\infty \neq 0, \infty$. *Furthermore*

$$z_\infty = \frac{1}{M} \arctan\left(\frac{M}{t_\infty}\right)$$

Note the similarities with the last step of the algorithm in the real case. We also have $\lim_{n\to\infty} r_n = 1$.

The final algorithm, as described by [CT13], is Algorithm 5.

Algorithm 5 Compute the complex elliptic logarithm.
Input: $P_0 = (x_0, y_0) \in E$ with $y_0 \neq 0$, with absolute precision P.
Output: the elliptic logarithm z of P_0 with absolute precision P.

1: $a \leftarrow \sqrt{e_1 - e_3}$, $b \leftarrow \sqrt{e_1 - e_2}$, choosing the numbering of roots and the signs so that $|a_0 - b_0| < |a_0 + b_0|$.
2: $r = \sqrt{\frac{x_0 - e_3}{x_0 - e_2}}$ with $\mathrm{Re}(r) \geq 0$.
3: $t = \frac{-y_0}{2r(x_0 - e_2)}$
4: **while** $a \neq b$ and $r \neq 1$ at the given precision **do**
5: $\quad A = \frac{a+b}{2}$, $B = \sqrt{ab}$ choosing the good sign
6: $\quad r \leftarrow \sqrt{\frac{A(r+1)}{br+a}}$ with $\mathrm{Re}(r) \geq 0$
7: $\quad t \leftarrow rt$
8: $\quad a = A, b = B$
9: **end while**
10: return $\frac{1}{a} \arctan\left(\frac{a}{t}\right)$

The loss of precision incurred in Algorithm 5 is not analyzed in [CT13]; we give a few arguments proving that the number of guard bits is negligible asymptotically in P. Theorem 3.2.6 proves that the number of bits of relative precision lost at each step of the AGM is a constant in P. As for the computation of the r_n, which converge to 1, the inversion and the square root extraction also lose a number of bits which is a constant in P at each step; the same goes for the computation of the t_n. Theorem 3.2.6 thus proves that only $O(\log P)$ bits are needed in the loop. Finally, the arctan computation is done via the complex logarithm (for instance $\arctan(z) = \frac{i}{2}(\log(1 - iz) - \log(1 + iz))$), which only requires $O(\log P)$ guard bits; overall, Algorithm 5 requires $O(\log P)$ guard bits.

4.3 An algorithm for the Weierstrass $\wp$ function

In this section, we outline a new algorithm that computes $\wp(z, \tau)$ with absolute precision P in $O(\mathcal{M}(P) \log P)$ operations. We describe in Chapter 8 (Section 8.1.2) another way, based on Chapter 6, to compute $\wp(z, \tau)$ with a similar complexity; we compare the two algorithms in Section 8.1.3.

This algorithm relies on using a backwards recurrence, which we get from the explicit change of variables given by the Landen transform. The analysis of this algorithm and the analysis of its precision loss relies on conjectures, verified experimentally. The algorithm is similar in its principle to Miller's algorithm for the Bessel function of the first kind [BZ10, p.153]; existing results and analyses on Miller's algorithm could open a way to prove some of the statements we make in this section. However, we did not explore this direction.

As explained in Note 1.3.16, we can assume that the following properties on z, τ are satisfied:

$$\tau \in \mathcal{F}, \qquad 0 \leq \mathrm{Im}(z) < \mathrm{Im}\left(\frac{\omega_2}{2\omega_1}\right), \qquad |\mathrm{Re}(z)| \leq \frac{\mathrm{Re}(\omega_1)}{2}$$

4.3.1 Fast computation of the sequence $\theta_i(0, 2^n\tau)$

Our algorithm relies on an induction formula involving the $\theta^2_{0,1,2}(0, 2^k\tau)$. We first outline an algorithm to compute these quantities in quasi-optimal time.

In order to compute this sequence, we compute its first term, then use the τ-duplication formulas (Equations (2.5.4)) to compute the other terms. Since we suppose that $\tau \in \mathcal{F}$, the choice of signs is always good (cf. Proposition 3.2.9), and we thus know how to extract the square root. We compute the first term using a fast, quasi-optimal algorithm to compute theta-constants, presented in e.g. [Dup11]; we outline this algorithm in Section 6.1 (Algorithm 11). The algorithm computes $\theta_i(0, \tau)$ with precision P in $O(\mathcal{M}(P) \log P)$ operations.

Hence, we can compute the $\theta^2_{0,1,2}(0, 2^k\tau)$ for $k \leq n$ in $O(\mathcal{M}(P)(\log P + n))$. However, note that the sequence we are trying to compute is quadratically convergent to $(1, 1, 0)$; hence, for $n > O(\log P)$, the representation with precision P of the terms of the sequence is stationary. This means the maximal cost of this algorithm is $O(\mathcal{M}(P) \log P)$.

Remark 4.3.1. Obviously, this precomputation can (and should) be cached, as it can be reused when one wants to compute the value of $\wp$ at several different z (keeping the same Λ). This is the case in our main application (cf. Chapter 9).

4.3.2 A backward recurrence for $\wp$

In this section we rewrite the change of variables given by the Landen transform as a recurrence relation between values of $\wp$.

Let $E : y^2 = P(x)$ be a complex elliptic curve of periods $[\omega_1, \omega_2]$. The Landen change of variables describes a 2-isogeny $E_1 \to E$, with $E_1 : y^2 = P_1(x)$ of periods $[\omega_1, 2\omega_2]$. This means that for any u, there is a u' such that

$$\int_u^\infty \frac{1}{\sqrt{P(x)}} = \int_{u'}^\infty \frac{1}{\sqrt{P_1(x)}}.$$

The relationship between u and u' is given by the explicit change of variables (Equation (4.1.1)):

$$u = u' + \frac{(e'_2 - e'_1)(e'_2 - e'_3)}{u' - e'_2}$$

Recall that (for a given polynomial P of degree 3) the function $x \mapsto \int_x^{+\infty} \frac{dt}{\sqrt{P(t)}}$ is the elliptic logarithm function, giving a $z \in \mathbb{C}/\Lambda$. Since $\wp(z, \Lambda) = x$, the function $\wp$ is the inverse of this function; hence, we have

$$u = \wp(z, [\omega_1, \omega_2]), \qquad u' = \wp(z, [\omega_1, 2\omega_2]).$$

We then rewrite Equation (4.1.1) as a function of $\wp$ and of theta constants at $\tau = \frac{\omega_2}{\omega_1}$, using Thomae's formulas (Theorem 1.3.19):

$$\wp(z, [\omega_1, \omega_2]) = \wp(z, [\omega_1, 2\omega_2]) + \frac{\left(\frac{\pi}{\omega_1}\right)^4 \theta_0(2\tau)^4 \theta_2(2\tau)^4}{\wp(z, [\omega_1, 2\omega_2]) + \left(\frac{\pi}{\omega_1}\right)^2 \frac{\theta_0(2\tau)^4 + \theta_2(2\tau)^4}{3}} \tag{4.3.1}$$

This relation allows us to compute $\wp(z,[\omega_1,\omega_2])$ from $\wp(z,[\omega_1,2\omega_2])$; we call this relation a backwards induction formula (as in [BZ10, p. 153]), since we are interested in computing the first term of the sequence $(\wp(z,[\omega_1,2^n\omega_2]))_{n\in\mathbb{N}}$.

The following proposition deals with the behavior of $\wp(z,[\omega_1,2^n\omega_2])_{n\in\mathbb{N}}$:

Theorem 4.3.2 ([CT13], [Kob84, Section 1.6, exercise 7], [Cha85, p. 46])**.**

$$\lim \wp(z,[\omega_1,2^n\omega_2]) = \left(\frac{\pi}{\omega_1}\right)^2 \left(\frac{1}{\sin^2(z\pi/\omega_1)} - \frac{1}{3}\right)$$

Proof. We have

$$\wp(z,[\omega_1,2^n\omega_2]) = \frac{1}{z^2} + \sum_{w\in\mathbb{Z}\omega_1+2^n\mathbb{Z}\omega_2} \frac{1}{(z-w)^2} - \frac{1}{w^2}$$

Write $w = m_1\omega_1 + m_2 2^n\omega_2$ and let n go to infinity: the terms with $m_2 \neq 0$ go to 0, and all that remains is

$$\lim \wp(z,[\omega_1,2^n\omega_2]) = \sum_{m\in\mathbb{Z}} \frac{1}{(z-m\omega_1)^2} - \frac{1}{\omega_1^2} \times 2\frac{\pi^2}{6}$$

But we have

$$\frac{\pi^2}{\sin^2(\pi z)} = \sum_{m\in\mathbb{Z}} \frac{1}{(z-m)^2}$$

which proves the result. □

Recall that sin can be computed with precision P in $O(\mathcal{M}(P)\log P)$, using the AGM-based algorithm for the complex logarithm (Section 3.3.2) and Newton's method (Section 0.3.3) to compute e^{it}. However, computing $\frac{1}{\sin^2}$ with absolute precision P when sin is small (e.g. $z \simeq \omega_1$) causes a large number of bits to be lost.

4.3.3 A quasi-optimal time algorithm

We outline an algorithm for $\wp$ with conjectured quasi-optimal running time. The algorithm directly uses the backwards induction of Equation (4.3.1), combined with Theorem 4.3.2; we compute $\ell = \lim_{n\to\infty} \wp(z,[\omega_1,2^n\omega_2])$, then determine N such that $|\ell - \wp(z,[\omega_1,2^N\omega_2])| \leq 2^{-P}$, and use the backwards induction to compute $\wp(z,[\omega_1,\omega_2])$. This strategy is similar to the one used for the real theta function in e.g. [LO98]; it also resembles other algorithms for special functions, such as Miller's algorithm for the evaluation of the Bessel function [BZ10, section 4.7.1.].

Write

$$\wp(z,[\omega_1,2^k\omega_2]) - \wp(z,[\omega_1,2^{k+1}\omega_2]) = \frac{\left(\frac{\pi}{\omega_1}\right)^4 \theta_0(2^{k+1}\tau)^4\theta_2(2^{k+1}\tau)^4}{\wp(z,[\omega_1,2^{k+1}\omega_2]) + \left(\frac{\pi}{\omega_1}\right)^2 \frac{\theta_0(2^{k+1}\tau)^4+\theta_2(2^{k+1}\tau)^4}{3}} \quad (4.3.2)$$

which gives, using cancellation

$$\wp(z,[\omega_1,\omega_2]) - \wp(z,[\omega_1,2^n\omega_2]) = \sum_{k=0}^{n+1} \frac{\left(\frac{\pi}{\omega_1}\right)^4 \theta_0(2^{k+1}\tau)^4\theta_2(2^{k+1}\tau)^4}{\wp(z,[\omega_1,2^{k+1}\omega_2]) + \left(\frac{\pi}{\omega_1}\right)^2 \frac{\theta_0(2^{k+1}\tau)^4+\theta_2(2^{k+1}\tau)^4}{3}}$$

We have

$$\lim_{n\to\infty} \wp(z,[\omega_1,2^n\omega_2]) + \left(\frac{\pi}{\omega_1}\right)^2 \frac{\theta_0(2^n\tau)^4+\theta_2(2^n\tau)^4}{3} = \left(\frac{\pi}{\omega_1}\right)^2 \frac{1}{\sin^2(z\pi/\omega_1)} \neq 0$$

hence

$$\frac{\left(\frac{\pi}{\omega_1}\right)^4 \theta_0(2^{k+1}\tau)^4\theta_2(2^{k+1}\tau)^4}{\wp(z,[\omega_1,2^{k+1}\omega_2]) + \left(\frac{\pi}{\omega_1}\right)^2 \frac{\theta_0(2^{k+1}\tau)^4+\theta_2(2^{k+1}\tau)^4}{3}} \sim \left(\frac{\pi}{\omega_1}\right)^2 \sin^2(z\pi/\omega_1)\theta_2(2^{k+1}\tau)^4$$

and the series converges because $\theta_2(2^{k+1}\tau)^4$ converges quadratically to 0. We can thus take the limit

$$\wp(z,[\omega_1,\omega_2])-\left(\frac{\pi}{\omega_1}\right)^2\left(\frac{1}{\sin^2(z\pi/\omega_1)}-\frac{1}{3}\right) = \sum_{k=0}^{\infty} \frac{\left(\frac{\pi}{\omega_1}\right)^4 \theta_0(2^{k+1}\tau)^4\theta_2(2^{k+1}\tau)^4}{\wp(z,[\omega_1,2^{k+1}\omega_2]) + \left(\frac{\pi}{\omega_1}\right)^2 \frac{\theta_0(2^{k+1}\tau)^4+\theta_2(2^{k+1}\tau)^4}{3}}$$

In order to turn this into an explicit algorithm, we wish to transform the infinite sum into a finite one. We use the following heuristic: the numerator contains the term $\theta_2^4(0,2^{k+1}\tau)$, which converges quadratically to 0, and thus could make the remainder very small. This also depends on the size of the denominator: when the denominator is close to 0^4, this could create large precision losses. In practice, we take N the first integer such that $\theta_2(0,2^N\tau) \leq 2^{-P}$, then assume that the sum for k greater than N is smaller than 2^{-P}. This heuristic amounts to writing:

$$\wp(z,[\omega_1,\omega_2])-\left(\frac{\pi}{\omega_1}\right)^2\left(\frac{1}{\sin^2(z\pi/\omega_1)}-\frac{1}{3}\right) = \sum_{k=0}^{N-1} \frac{\left(\frac{\pi}{\omega_1}\right)^4 \theta_0(2^{k+1}\tau)^4\theta_2(2^{k+1}\tau)^4}{\wp(z,[\omega_1,2^{k+1}\omega_2]) + \left(\frac{\pi}{\omega_1}\right)^2 \frac{\theta_0(2^{k+1}\tau)^4+\theta_2(2^{k+1}\tau)^4}{3}} + \epsilon$$

with $|\epsilon| \leq 2^{-P}$; we can then evaluate this sum, starting with the approximation of $\wp(z,[\omega_1,2^N\omega_2])$ and using the backwards recurrence, as well as the value of the $\theta_{0,1,2}^2(0,2^k\tau)$. This is Algorithm 6.

Algorithm 6 Compute $\wp$ using the Landen transform.
Input: z,τ with absolute precision P, satisfying conditions (2.5.11).
Output: $\wp(z,[\omega_1,\omega_2])$ with absolute precision P.

1: $N \leftarrow 0$
2: Compute $\theta_{0,1,2}(0,\tau)$ using Algorithm 11.
3: **while** $|\theta_2(0,2^N\tau)| \geq 2^{-P}$ **do**
4: Use the τ-duplication formulas (Equation (2.5.4)) to compute $\theta_{0,1,2}(0,2^{N+1}\tau)$.
5: $N \leftarrow N+1$.
6: **end while**
7: res $\leftarrow \left(\frac{\pi}{\omega_1}\right)^2\left(\frac{1}{\sin^2(z\pi/\omega_1)}-\frac{1}{3}\right)$.
8: **for** i = N downto 1 **do**
9: res $\leftarrow$ res $+ \frac{\left(\frac{\pi}{\omega_1}\right)^4\theta_0(0,2^i\tau)^4\theta_2(0,2^i\tau)^4}{\text{res}+\left(\frac{\pi}{\omega_1}\right)^2\frac{\theta_0(0,2^i\tau)^4+\theta_2(0,2^i\tau)^4}{3}}$.
10: **end for**
11: **return** res

The definition of θ_2 (Equation (2.5.3)) proves that, when N goes to infinity, $\theta_2(0,2^N\tau) \sim 2e^{i\pi 2^N\tau/4}$; hence we have $N = O(\log P)$. As for precision losses, they could stem from two potential problems: if the computation of the limit loses a large amount of absolute precision

[4]For instance when $\wp(z,[\omega_1,2^k\omega_2])$ is close to $e_3^{(k)} = \wp(2^k\omega_2,[\omega_1,2^k\omega_2])$ – note however that this only happens once for every z.

(for instance if z is close to ω_1), or if the heuristic is not correct. Note that the case of small denominators, e.g. $z \simeq \omega_2$, could be easily avoided by checking this condition at the beginning of the algorithm. Provided the heuristic is correct and the precision losses are manageable, we get a quasi-optimal time algorithm to compute $\wp$ with absolute precision P.

We do not provide an analysis of the precision loss incurred, or timings, at this point. Instead, we differ such considerations to Chapter 8, in which we discuss another quasi-linear time algorithm to compute $\wp$; we will compare Algorithm 6 to this other algorithm in terms of timings and precision loss in Section 8.1.3, and show that the precision loss should not affect the asymptotic running time of either algorithm.

4.4 Using the Landen transform to compute θ

The change of variables given by the Landen isogeny shows how one may go from a curve with fundamental parallelogram isomorphic to $[1, 2\tau]$ to a curve with fundamental parallelogram isomorphic to $[1, \tau]$; as such, this change of variables has sometimes been referred to as the *descending* Landen transform. A few algorithms are based on this transformation, although they sometimes only use the vocabulary related to the AGM. Most notably, a couple of algorithms have been proposed to compute the real theta function.

We find in [AS64, Section 16.32] an algorithm (similar to [AS64, Section 16.4] for the elliptic functions) to compute $\theta(u|m)$ from the arithmetico-geometric mean. The algorithm relies on a backwards induction, much like Algorithm 6: one has to compute the terms of an AGM sequence until the first index N such that $\frac{a_N - b_N}{2}$ is negligible at the required precision; then compute a quantity ϕ_N from this, and use recurrence relations to compute $\phi_{N-1}, \ldots, \phi_0$. Since the AGM sequence is quadratically convergent, $N = \log P$; however, each step also requires the computation of $\log \circ \cos(\phi)$, which takes $O(\mathcal{M}(P) \log P)$ if one uses the fast algorithms for log and exp of Chapter 3. Hence, the complexity of this algorithm seems to be $O(\mathcal{M}(P) \log^2 P)$ bit operations in this case, i.e. the real theta function. Note that we were unable to locate a reference which proves this algorithm to be correct and, more importantly, which gives an order of magnitude for the precision loss.

More recently, a better algorithm for the real theta function was proposed and analyzed in [LO98]. The algorithm uses a similar pattern of computing a quadratically convergent sequence until the terms are small enough, then computing a quantity ϕ_N and use a backwards induction to compute the value of θ. In this case, the computations derive directly from the descending Landen transform, i.e. the final result is obtained directly, without having to compute other quantities like $\log \circ \cos$. This gives a $O(\mathcal{M}(P) \log P)$ algorithm; furthermore, the article provides a careful analysis of the precision loss, which shows that the result is not too inaccurate (about $O(\log P)$ guard bits seem to be needed). We were unable to generalize this algorithm (and the corresponding analysis) to the complex case, using instead another method based on Newton's method (reasoning it would be more stable numerically, as the Newton iteration is self-correcting); we refer to Chapter 6 for more details.

Finally, we mention an attempt of ours to get a quasi-optimal time algorithm for θ. We start from the Landen transform formulated in terms of theta functions:

Proposition 4.4.1 ([Bel61, p.54], [WW27, Section 21.52, p.469])**.**

$$\frac{\theta_0(z,\tau)\theta_1(z,\tau)}{\theta_1(2z,2\tau)} = \frac{\theta_0(0,\tau)\theta_1(0,\tau)}{\theta_1(0,2\tau)}$$

We rewrite this formula as

$$\theta_1(2z,2\tau) = \frac{\theta_1(0,2\tau)}{\theta_0(0,\tau)\theta_1(0,\tau)}\frac{\theta_0(z,\tau)}{\theta_1(z,\tau)}\theta_1^2(z,\tau).$$

Putting $a_n = \frac{\log \theta_1(2^n z, 2^n \tau)}{2^n}$, we get the recurrence relation:

$$a_{k+1} = a_k + \frac{1}{2^{k+1}}\left(\log\frac{\theta_0(2^k z, 2^k\tau)}{\theta_1(2^k z, 2^k\tau)} + \log\theta_1(0,2^{k+1}\tau) - \log\theta_0(0,2^k\tau) - \log\theta_1(0,2^k\tau)\right).$$

The advantage of this relation is that we only need the theta-constants and the quotient θ_0/θ_1, which can be computed from other quotients, and for instance from the value of $\wp$. However, the resulting algorithm requires evaluating $\wp(2^k z, 2^k\tau)$ for $1 \leq k \leq N$; the best algorithm to do so uses Equation (4.3.2) and the z-duplication formula for $\wp$:

$$\wp(2z) = \frac{(\wp(z)^2 + \frac{g_2}{4})^2 + 2g_3\wp(z)}{4\wp(z)^3 - g_2\wp(z) - g_3}$$

We then use a recursive algorithm to compute the leaves of the tree which vertices are the values $\wp(2^i z, 2^j\tau)$ and the edges are either the use of Equation (4.3.2) (going from 2τ to τ) or the z-duplication; this is an optimal strategy, as analysed in [DFJP14], and requires $O(\log P \log\log P)$ multiplications. We thus potentially get a $O(\mathcal{M}(P)\log P\log\log P)$ algorithm to compute θ this way, which is not the best potential running time; we did not analyze this algorithm further.

Chapter 5

Naive algorithms for theta functions in any genus

This chapter is dedicated to presenting some algorithms to compute $\theta(z,\tau)$ by partial summation. We show in this chapter that $\theta(z,\tau)$ can be computed with absolute precision P in $O(\mathcal{M}(P)P^{g/2})$ bit operations; in genus 1 and 2, and for τ in the fundamental domain, the complexity is even $O\left(\mathcal{M}(P)\left(\frac{P}{\mathrm{Im}(\tau_{1,1})}\right)^{g/2}\right)$ bit operations. We conjecture that this result holds in genus g.

In order to determine how many terms are needed, an analysis of the size of the tail of the sum is required. We produce such an analysis in genus 1 and 2, in the case where z,τ are reduced; the results we obtain show that the number of terms needed to get a result accurate to 2^{-P} decreases as $\mathrm{Im}(\tau_{1,1})$ increases. This is of importance in the context of fast algorithms with uniform running time, such as in Chapter 6 and Chapter 7. In genus g, we make the result of [DHB+04] more explicit, which generalizes the results in genus 1 and 2 nicely; however, the running time of this method does not seem to be as good as the results we got in genus 1 and 2.

Another aspect is to determine an efficient way to compute the terms of the series. The simplest method is to compute each term independently, using an exponentiation for each term; this does not yield the best asymptotic running time, but the resulting algorithm is rather simple to implement. We discuss another way, based on recurrence relations of degree 2 linking the terms together; this method requires more storage (presumably around $O(2^{2g})$), but the amortized cost for each term is only a few multiplications. This gives the best running time; we give explicit algorithms in genus 1 and 2, which we implemented[5], and just sketch the corresponding relations and the algorithm in genus g.

5.1 Genus 1

Argument reduction for z and τ has been studied in Section 2.5.3 (which instiatiated Section 2.3 and Section 2.4). As a result, the computational problem we study here is the computation of $\theta(z,\tau)$ to absolute precision P with z,τ such that

$$|\tau| \geq 1, \quad |\mathrm{Re}(\tau)| \leq \frac{1}{2}, \quad \mathrm{Im}(\tau) > 0, \qquad |\mathrm{Re}(z)| \leq \frac{1}{2}, \quad 0 \leq \mathrm{Im}(z) \leq \frac{\mathrm{Im}(\tau)}{2}.$$

[5]Timings are provided respectively in Section 6.4 and Section 7.3.

5.1.1 Partial summation of the series defining θ

The analysis we present here is inspired by [Dup11].

Define the following partial summation of the series defining $\theta(z,\tau)$:

$$S_B(z,\tau) = 1 + \sum_{0<n<B} q^{n^2}(e^{2i\pi nz} + e^{-2i\pi nz})$$

where we use the notation $q = e^{i\pi\tau}$. We have

Proposition 5.1.1. *Suppose that* $\mathrm{Im}(\tau) \geq 0.35$ *and that* $\mathrm{Im}(z) \leq \frac{\mathrm{Im}(\tau)}{2}$ *(which is looser than the conditions we specified at the beginning of the section). Then* $|\theta(z,\tau) - S_B(z,\tau)| \leq 3|q|^{B-1}$.

We use the following lemma, which bounds the remainder of a series by a geometric series whose sum is easy to compute:

Lemma 5.1.2. *Let* $q \in \mathbb{C}$ *such that* $|q| < 1$. *Let* $f : \mathbb{N} \to \mathbb{N}$ *be an increasing function such that* $(f(k+1) - f(k))_{k\in\mathbb{N}}$ *is increasing. Then:*

$$\sum_{n\geq n_0} q^{f(n)} \leq \sum_{n\geq n_0} q^{f(n_0)+(n-n_0)(f(n_0+1)-f(n_0))} = \frac{q^{f(n_0)}}{1-q^{f(n_0+1)-f(n_0)}}.$$

We use this lemma a few times throughout this manuscript, among which in the proof of Proposition 5.1.1:

Proof of Proposition 5.1.1. We look at the remainder of the series:

$$\begin{aligned}
|\theta(z,\tau) - S_B(z,\tau)| &\leq \sum_{n\geq B} |q|^{n^2}(|e^{2i\pi nz}| + |e^{-2i\pi nz}|) \\
&\leq \sum_{n\geq B} |q|^{n^2}(1 + |q|^{-n}) \leq 2\sum_{n\geq B} |q|^{n^2-n} \\
&\leq 2\sum_{n\geq B} |q|^{(n-1)^2} \leq 2\sum_{n\geq 0} |q|^{(B-1+n)^2} \\
&\leq 2|q|^{(B-1)^2} \sum_{n\geq 0} |q|^{2n(B-1)+n^2} \\
&\leq 2\frac{|q|^{(B-1)^2}}{1-|q|^{2B-1}} \qquad (5.1.1)
\end{aligned}$$

the last line being a consequence of Lemma 5.1.2. A numerical calculation then proves that for $\mathrm{Im}(\tau) \geq 0.35$, we have $\frac{2}{1-|q|} \leq 3$, which proves the proposition. □

Note that we can prove the same inequality for θ_1, since the series that define it has the same terms, up to sign, as the series for θ.

Unlike the analysis of [Dup11] for naive theta-constant evaluation, we cannot get a bound for the relative precision: since $\theta(\frac{1+\tau}{2},\tau) = 0$, there is no lower bound for $|\theta(z,\tau)|$. If we set

$$B(P,\tau) = \left\lceil \sqrt{\frac{P+2}{\pi\,\mathrm{Im}(\tau)\log_2(e)}} \right\rceil + 1.$$

we have $4|q|^{(B-1)^2} \leq 2^{-P}$, which means the approximation is accurate with absolute precision P. We just showed that:

Theorem 5.1.3. *To compute $\theta(z,\tau)$ with absolute precision P bits, it is enough to sum over all $k \in \mathbb{Z}$ such that*

$$|k| \leq \left\lceil \sqrt{\frac{P+2}{\pi \operatorname{Im}(\tau) \log_2(e)}} \right\rceil + 1$$

Note that this bound is of the same order than the one of [Dup11, p. 5], since it is greater than it by only 2 (i.e. only 4 more terms are needed).

5.1.2 Naive algorithm

We now present a naive algorithm to compute not only the value of $\theta(z,\tau)$, but also the value of $\theta_1(z,\tau), \theta_0(0,\tau), \theta_1(0,\tau)$ for only a marginal amount of extra computation; this is the algorithm we will use for comparison to the fast algorithm we propose in Chapter 6. The algorithm computes with internal precision $\mathcal{P}$, which we determine later so that the result is accurate to the desired precision P.

Define the sequence $(v_n)_{n\in\mathbb{N}}$ as

$$v_n = q^{n^2}(e^{2i\pi nz} + e^{-2i\pi nz})$$

so that $\theta(z,\tau) = 1 + \sum_{n\geq 1} v_n$. This sequence satisfies a recurrence relation for $n > 1$:

$$v_{n+1} = q^{2n} v_1 v_n - q^{4n} v_{n-1}.$$

We use this recursion formula to compute v_n efficiently, which is similar to the trick used by [ET14a, Prop. 3]. This removes the need for divisions and the need to compute and store $e^{-2i\pi nz}$, which can get quite big; indeed, computing it only to multiply it by the very small q^{n^2} is wasteful. The resulting algorithm is Algorithm 7.

Algorithm 7 Naive algorithm to compute $\theta_0(z,\tau), \theta_0(0,\tau), \theta_1(z,\tau), \theta_1(0,\tau)$.
Input: $z \in \mathbb{C}, \tau \in \mathcal{H}$ with absolute precision P, satisfying conditions (2.5.11).
Output: $\theta_0(z,\tau), \theta_0(0,\tau), \theta_1(z,\tau), \theta_1(0,\tau)$ with absolute precision P.

Work with precision $\mathcal{P}$.
$a \leftarrow (1,1), \quad b \leftarrow (1,1)$ ▷ These arrays will hold respectively $\theta_i(z,\tau)$ and $\theta_i(0,\tau)$
$B \leftarrow \left\lceil \sqrt{\frac{P+2}{\pi \operatorname{Im}(\tau) \log_2(e)}} \right\rceil + 1$
$q \leftarrow \text{UniformExp}(i\pi\tau)$
$v_1 \leftarrow \text{UniformExp}(2i\pi(z+\tau/2)) + \text{UniformExp}(-2i\pi(z-\tau/2))$
$q_1 \leftarrow q, \quad q_2 \leftarrow q, \quad v \leftarrow v_1, \quad v' \leftarrow 2$
for $n = 1..B$ **do**
    /* $q_1 = q^n, q_2 = q^{n^2}, v = v_n, v' = v_{n-1}$ */
    $a_0 \leftarrow a_0 + v, \quad a_1 \leftarrow a_1 + (-1)^n \times v$
    $b_0 \leftarrow b_0 + 2q_2, \quad b_1 \leftarrow b_1 + (-1)^n \times 2q_2$
    $q_2 \leftarrow q_2 \times (q_1)^2 \times q$
    $q_1 \leftarrow q_1 \times q$
    $\text{temp} \leftarrow v, \quad v \leftarrow \left(q_1^2 \times v_1\right) \times v - q_1^4 \times v' \quad v' \leftarrow \text{temp}$
end for
return a_0, b_0, a_1, b_1

We use a subroutine in this algorithm, which we call UniformExp(z), and which computes e^z with absolute precision P using the following algorithm: if $\operatorname{Re}(z) \leq -\frac{P}{\log_2 e}$, return 0, if not compute $\exp(z)$ using Section 3.3.3. It is then easily seen that:

Proposition 5.1.4. *UniformExp computes e^z with absolute precision P in $c\mathcal{M}(P)\log P$ bit operations for any z such that $\mathrm{Re}(z) \leq 0$, where c is a constant independent of z.*

Hence, the complexity of UniformExp is *uniform* over all z with $\mathrm{Re}(z) \leq 0$.

5.1.3 Error analysis and complexity

This section is dedicated to proving:

Theorem 5.1.5. *For z, τ with absolute precision P and satisfying conditions (2.5.11), Algorithm 7 with $\mathcal{P} = P + \log B + 7$ computes $\theta_0(z,\tau)$, $\theta_1(z,\tau)$, $\theta_0(0,\tau)$, $\theta_1(0,\tau)$ with absolute precision P bits. This gives an algorithm which has bit complexity $O\left(\mathcal{M}(P)\sqrt{\frac{P}{\mathrm{Im}(\tau)}}\right)$.*

Remark 5.1.6. Note that the running time of this algorithm gets better as $\mathrm{Im}(\tau)$ increases (the remainder is smaller than 2^{-P} much quicker in that case). At the limit, if $P = c\,\mathrm{Im}(\tau)$ where c is a fixed constant, one only needs a constant number of terms to get the final result. The running time of this algorithm is then dominated by the computation of $q = e^{i\pi\tau}$ and v_1 at the beginning of the algorithm, which costs $c\mathcal{M}(P)\log P$ operations with c independent of z, τ by Proposition 5.1.4. Note that, in that case, the running time of the algorithm is independent of z and τ; this remark will be of use in Section 6.1.2.

Analyzing this algorithm requires bounding the error that is incurred during the computation. We then compensate the number of inaccurate bits by increasing the precision. We use Theorem 0.3.3 to estimate the number of bits lost.

Proof of Theorem 5.1.5. We first determine the size of the quantities we are manipulating; this is needed to evaluate the error incurred during the computation, as well as the number of bits needed to store fixed-precision approximations of absolute precision P of the intermediate quantities. Taking $B = 1$ in Proposition 5.1.1 gives $|\theta(z,\tau) - 1| \leq 3$, so $|\theta(z,\tau)| \leq 4$; actually, this also proves $|S_B(z,\tau)| \leq 4$, which means that $|a_0|, |a_1|, |b_0|, |b_1|$ are bounded by 4. We also have $|q| \leq 0.07$, and $|q_2| \leq |q|^{n^2} \leq |q|^n = |q_1| \leq |q| \leq 0.07$. As for the v_i, we have $v_0 = 2$, and for $n \geq 1$

$$|v_n| \leq |q|^{n^2+n} + |q|^{n^2-n} \leq (1 + |q|^{2n})|q|^{n^2-n} \leq 1.0049|q|^{n^2-n} \leq 1.0049$$

Hence, storing all the complex numbers above, including our result, with absolute precision P only requires $P + 2$ bits, since their integral part is coded on only 2 bits. Note that, had we computed $e^{-2i\pi nz}$ before multiplying it by q^{n^2}, we would have needed $O(\mathrm{Im}(\tau))$ more bits, which changes the asymptotic complexity.

Computing the absolute precision lost during this computation is done using Theorem 0.3.3. We start with the bounds $|\tau - \tilde{\tau}| \leq \frac{1}{2}2^{-\mathcal{P}}$ and $|z - \tilde{z}| \leq \frac{1}{2}2^{-\mathcal{P}}$, coming from the hypothesis that the approximations of z and τ are correctly rounded with precision $\mathcal{P}$. We then need to estimate k_{v_1} and k_q, which can be done using the formula giving the absolute error when computing an exponential from Theorem 0.3.3. Given that $\tau \in \mathcal{F}$, we have

$$\begin{aligned} |q - \tilde{q}| &\leq 0.07\frac{7 \times 1/2 + 8.5}{2}2^{-P} \leq 0.42 \times 2^{-P} \\ |v_1 - \tilde{v_1}| &\leq 6(|e^{-\pi(\mathrm{Im}(\tau)+2\,\mathrm{Im}(z))}| + |e^{\pi(2\,\mathrm{Im}(z)-\mathrm{Im}(\tau))}|) \times 2^{-P} \\ &\leq 6(|q|+1)2^{-P} \leq 6.42 \times 2^{-P} \end{aligned}$$

which means that $k_q \leq 0.42$ and $k_{v_1} \leq 6.42$. We then need to evaluate the loss of precision for each variable and at each step of the algorithm, which gives recurrence relations with non-constant coefficients. Solving those is rather tedious, and we use loose upper bounds to simplify

the computation; we do not detail this proof here. The results obtained by this method show that the error on the computation of the theta-constants is bounded by $(0.3B + 105.958)2^{-\mathcal{P}}$, and the one on the computation of the theta function is smaller than $(5.894B + 28.062)2^{-\mathcal{P}}$. This proves that the number of bits lost is bounded by $\log_2 B + c$, where c is a constant smaller than 7; hence we set $\mathcal{P} = P + \log B + 7$.

Finally, evaluating π and $\exp(z)$ with precision $\mathcal{P}$ can be done in $O(\mathcal{M}(\mathcal{P}) \log \mathcal{P})$, using respectively the Brent-Salamin algorithm (Section 3.1.2) and our subroutine UniformExp (Proposition 5.1.4); this is negligible asymptotically. In the end, computing an approximation up to 2^{-P} of $\theta(z,\tau)$ can be done in $O\left(\mathcal{M}\left(P + \log(P/\operatorname{Im}(\tau)) + c\right)\sqrt{\frac{P}{\operatorname{Im}(\tau)}}\right) = O\left(\mathcal{M}(P)\sqrt{\frac{P}{Im(\tau)}}\right)$ bit operations. □

5.1.4 Computing θ_2

We mentioned in Section 2.5.3 the need to compute $\theta_2(z,\tau)$ and $\theta_2(0,\tau)$ as well. One could think of recovering those values using Jacobi's quartic formula and the equation of the variety, which we mentioned in Section 2.5:

$$\begin{aligned} \theta_0(0,\tau)^4 &= \theta_1(0,\tau)^4 + \theta_2(0,\tau)^4 \\ \theta_0^2(z,\tau)\theta_0^2(0,\tau) &= \theta_1^2(z,\tau)\theta_1^2(0,\tau) + \theta_2^2(z,\tau)\theta_2^2(z,\tau) \end{aligned}$$

that is to say, compute

$$\begin{aligned} \theta_2(0,\tau) &= \left(\theta_0(0,\tau)^4 - \theta_1(0,\tau)^4\right)^{1/4} \\ \theta_2(z,\tau) &= \frac{\sqrt{\theta_0^2(z,\tau)\theta_0^2(0,\tau) - \theta_1^2(z,\tau)\theta_1^2(0,\tau)}}{\theta_2(0,\tau)}. \end{aligned}$$

However, this approach induces an asymptotically large loss of absolute precision for both $\theta_2(0,\tau)$ and $\theta_2(z,\tau)$. According to Theorem 0.3.3, both square root extraction and inversion induce a loss of precision proportional to $|z|^{-1}$; since $\theta_2(0,\tau) \sim 4q^{1/2}$, the number of bits lost by applying those formulas is $O(\operatorname{Im}(\tau))$. Also note that those formulas would induce a big loss in relative precision as well, since $\theta_0(0,\tau)$ and $\theta_1(0,\tau)$ are very close when $\operatorname{Im}(\tau)$ goes to infinity, and the subtraction induces a relative precision loss of $O(\operatorname{Im}(\tau))$ bits; for more details, see [Dup11, Section 6.3]. Either of those analyses show that, in order to compensate precision loss, the naive algorithm should actually be run with a precision of $O(P + \log B + \operatorname{Im}(\tau))$, which gives a running time that worsens, insteads of getting better, when $\operatorname{Im}(\tau)$ gets big. We do not recommend this approach.

Instead, one should compute partial summations of the series defining θ_2, much in the same way as we did for $\theta(z,\tau)$. We write $\theta_2(z,\tau) = q^{1/4}w(1+\sum_{n\geq 1} v_n)$ with $v_n = q^{n^2+n}(w^{2n}+w^{-2n})$; since $\operatorname{Im}(z) \geq 0$ we can just look at the problem of evaluating $1 + \sum v_n$. We have

$$v_{n+1} = q^{2n}v_1v_n - q^{4n+2}v_{n-1}$$

The analysis in this case is very similar to the one for θ. We have $|v_n| \leq |q|^{n^2}$, hence $|\theta_2(z,\tau) - S_B| \leq 3|q|^{B^2}$, so that the bound on B is one less than the one for θ. We have that $q^{2n}|v_1|$ is bounded by 2 instead of 1, which in the worst case means $\log B$ more guard bits are needed. This gives Algorithm 8; its asymptotic complexity is, just like Algorithm 7, $O\left(\mathcal{M}(P)\sqrt{\frac{P}{\operatorname{Im}(\tau}}\right)$ bit operations, which gets better as $\operatorname{Im}(\tau)$ increases.

Algorithm 8 Naive algorithm to compute $\theta_2(z,\tau), \theta_2(0,\tau)$.
Input: z, τ with absolute precision P, satisfying conditions (2.5.11).
Output: $\theta_2(z,\tau), \theta_2(0,\tau)$ with absolute precision P.

$a \leftarrow 1, \quad b \leftarrow 1$
$B \leftarrow \left\lceil \sqrt{\frac{P+2}{\pi \operatorname{Im}(\tau) \log_2(e)}} \right\rceil$
Work with precision $P + 2\log B + 7$.
$q \leftarrow e^{i\pi\tau}, \qquad q_1 \leftarrow q, \qquad q_2 \leftarrow q$
$v_1 \leftarrow q^2(w^2 + w^{-2}), \qquad v \leftarrow v_1, \qquad v' \leftarrow 2$
for $n = 1..B$ **do**
    /* $q_1 = q^n, q_2 = q^{n^2+n}, v = v_n, v' = v_{n-1}$ */
    $a \leftarrow a + v, \qquad b \leftarrow b + 2q_2$
    $q_2 \leftarrow q_2 \times (q_1)^2 \times q^2$
    $q_1 \leftarrow q_1 \times q$
    $\text{temp} \leftarrow v, \qquad v \leftarrow \left(q_1^2 \times v_1\right) \times v - (q_1^4 \times q^2) \times v' \qquad v' \leftarrow \text{temp}$
end for
$a \leftarrow a \times q^{1/4} w$, $b \leftarrow b \times q^{1/4}$
return a, b

We note that similar considerations apply to the problem of computing θ_3. One can compute $\theta_3(z,\tau)$ using the formula [Mum83, p.22]

$$\theta_3(z,\tau)^2 = \frac{\theta_1(z,\tau)^2\theta_2(0,\tau)^2 - \theta_2(z,\tau)^2\theta_1(0,\tau)^2}{\theta_0(0,\tau)^2}. \tag{5.1.2}$$

Using this formula loses only a few bits of precision since $\theta_0(0,\tau)$ is bounded; however, one then needs to compute a square root, which potentially loses $O(\operatorname{Im}(\tau))$ bits. Hence, a summation of the series, which directly gives θ_3, is preferable.

5.2 Genus 2

We now study the naive algorithm in the genus 2 case, i.e. $z = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$ and $\tau = \begin{pmatrix} \tau_1 & \tau_3 \\ \tau_3 & \tau_2 \end{pmatrix}$. The number of variables is small enough that we can write the sum explicitly, as in Section 2.6, and perform an analysis of its remainder using the triangle inequality; this allows us to make the dependency in τ explicit. We can also write explicit recurrence relations between the terms to compute the sum faster.

Recall that the series defining the fundamental theta functions $\theta_0, \theta_1, \theta_2, \theta_3$ only differ by the signs in front of the terms; the patterns are summarized in Figure 2.2. This means one can compute all the values $\theta_{[0;b]}(z,\tau)$ at the same time for the same computational cost, i.e. with no extra multiplications. This also justifies that the analysis that we perform in Section 5.2.1 is valid for all the fundamental theta functions, since the differences of signs vanish when using the triangle inequality.

As in genus 1, we assume that z, τ are reduced as explained in Section 2.3 and Section 2.4.4 (e.g. $\tau \in \mathcal{F}'_g$). This translates into the conditions:

$$|\operatorname{Re}(\tau_i)| \leq \tfrac{1}{2}, \qquad 0 \leq 2\operatorname{Im}(\tau_3) \leq \operatorname{Im}(\tau_1) \leq \operatorname{Im}(\tau_2),$$

$$|\tau_1| \geq 1 \ \ (\text{i.e. } \operatorname{Im}(\tau_1) \geq \tfrac{\sqrt{3}}{2}) \ \ ; \qquad |\operatorname{Re}(z_i)| \leq \tfrac{1}{2}$$

$$|\mathrm{Im}(z_1)| \leq \tfrac{\mathrm{Im}(\tau_1)+\mathrm{Im}(\tau_3)}{2} \leq \tfrac{3}{4}\,\mathrm{Im}(\tau_1), \quad |\mathrm{Im}(z_2)| \leq \tfrac{\mathrm{Im}(\tau_2)+\mathrm{Im}(\tau_3)}{2} \leq \tfrac{3}{4}\,\mathrm{Im}(\tau_2).$$

5.2.1 Truncated sums

Proposition 5.2.1. *Let z, τ be reduced as previously. Let S_B denote the sum of the series defining θ with $(m,n) \in [-B,B]^2$. Then taking*

$$B = \sqrt{\frac{2(P+5)}{\pi \log_2 e\, \mathrm{Im}(\tau_1)}} + 2$$

i.e. summing no more than $O\left(\frac{P}{\mathrm{Im}(\tau_1)}\right)$ terms, is enough to get an approximation of $\theta(z,\tau)$ that is accurate to 2^{-P}.

Proof. Let $q_j = e^{i\pi\tau_j}$ and $w_j = e^{i\pi z_j}$. Using the triangle inequality, we can write

$$\begin{aligned}|\theta(z,\tau) - S_B| \leq S_1 &\\ &+ \sum_{n\geq B+1} |q_2|^{n^2}(|w_2|^{2n} + |w_2|^{-2n}) + \sum_{m\geq B+1} |q_1|^{m^2}(|w_1|^{2m} + |w_1|^{-2m})\\ &+ \sum_{|m|>0} \sum_{|n|\geq B+1} |q_1|^{m^2}|q_2|^{n^2}|q_3|^{2mn}|w_1|^{2m}|w_2|^{2n}\end{aligned}$$

where $S_1 = \sum_{|m|\geq B+1}\sum_{0<|n|\leq B}|q_1|^{m^2}|q_2|^{n^2}|q_3|^{2mn}|w_1|^{2m}|w_2|^{2n}$. The second line can be bounded using a calculation very similar to the one in Section 5.1.1. The third line can be bounded as follows: we have

$$|q_1^{m^2}q_2^{n^2}q_3^{2mn}| \leq |q_1^{m^2}q_2^{n^2}q_3^{-m^2-n^2}| \leq |q_1^{m^2/2}q_2^{n^2/2}|$$

and furthermore, given the assumptions on z, we have:

$$|w_1| + |w_1^{-1}| \leq 1 + e^{-\pi(-|\mathrm{Im}(z_1)|)} \leq 2e^{\pi\frac{3}{4}\mathrm{Im}(\tau_1)}, \qquad |w_2| + |w_2^{-1}| \leq 2e^{\pi\frac{3}{4}\mathrm{Im}(\tau_2)}.$$

Hence $(|w_1^{2m}| + |w_1^{-2m}|)(|w_2^{2n}| + |w_2^{-2n}|) \leq 4e^{\pi(\frac{3}{2}m\,\mathrm{Im}(\tau_1)+\frac{3}{2}n\,\mathrm{Im}(\tau_2))}$. Overall, we get the bound

$$|\theta(z,\tau) - S_B| \leq S_1 + \frac{4|q_1|^{(B-1)^2-4}}{1-|q_1|} + \frac{4|q_1|^{\frac{(B-1)^2}{2}-3}}{1-|q_1|}$$

Assuming that $\mathrm{Im}(z_i) \geq 0$, which does not change the proof, we bound S_1 as follows:

$$\begin{aligned}S_1 \leq\ & |q_2| \sum_{m\geq B+1} |q_1|^{m^2}\left(|q_3|^{2m}(|w_1|^{2m}|w_2|^2 + |w_1|^{-2m}|w_2|^{-2}) + |q_3|^{-2m}(|w_1|^{2m}|w_2|^{-2} + |w_1|^{-2m}|w_2|^2)\right)\\ &+ |q_2|^4 \sum_{m\geq B+1} |q_1|^{m^2}\left(|q_3|^{4m}(|w_1|^{2m}|w_2|^4 + |w_1|^{-2m}|w_2|^{-4}) + |q_3|^{-4m}(|w_1|^{2m}|w_2|^{-4} + |w_1|^{-2m}|w_2|^4)\right)\\ &+ 4 \sum_{m\geq B+1}\sum_{2<n\leq B} |q_1|^{\frac{m^2}{2}-\frac{3}{2}m}|q_2|^{\frac{n^2}{2}-\frac{3}{2}n}.\end{aligned}$$

The last line can be bounded by $\frac{4q^{\frac{(B-1)^2-4}{2}}}{(1-|q_1|)(1-|q_2|)}$. We can refine the bound on $|w_2|^{-1}$, writing $|q_2||w_2|^{-2} \leq q_3^{-1}$, which gives

$$\begin{aligned} S_1 \leq & \frac{q_1^{(B+1)^2+1}}{1-|q_1|} + \frac{q_1^{(B-1)^2-4}}{1-|q_1|} + \frac{q_1^{(B-1)^2-\frac{3}{2}}}{1-|q_1|} + \frac{|q_1|^{(B-1)^2-3}}{1-|q_1|} \\ & + \frac{q_1^{(B+1)^2+1}}{1-|q_1|} + \frac{q_1^{(B-1)^2-2}}{1-|q_1|} + \frac{q_1^{B^2+1}}{1-|q_1|} + \frac{q_1^{(B-1)^2-2}}{1-|q_1|} \\ & + \frac{4q^{\frac{(B-1)^2-4}{2}}}{(1-|q_1|)(1-|q_2|)}. \end{aligned}$$

Collecting terms yields the stated result. □

5.2.2 Genus 2 naive algorithm

Following and extending the strategy in Section 5.1.2 or [ET14a, §5.1], we use recurrence relations to compute terms more efficiently. Let

$$\begin{aligned} Q(m,n) &= q_1^{m^2} q_2^{n^2} q_3^{2mn} \\ T(m,n) &= Q(m,n)(w_1^{2m} w_2^{2n} + w_1^{-2m} w_2^{-2n}). \end{aligned}$$

With minor rewriting, we have that

$$\theta_i(z,\tau) = \sum_{m,n\in\mathbb{N}} s(i,m,n)(T(m,n) + T(m,-n))$$

$$\theta_i(0,\tau) = \sum_{m,n\in\mathbb{N}} s(i,m,n)(Q(m,n) + Q(m,-n))$$

where $s(i,m,n)$ is $\frac{1}{4}$ for $m = n = 0$, $\frac{\pm 1}{2}$ along the axes $(m,0)$ and $(0,n)$, and ± 1 elsewhere.

We have the following recurrence relations.

$$(w_1^2 + w_1^{-2})T(m,n) = q_1^{-2m-1} q_3^{-2n} T(m+1,n) + q_1^{2m-1} q_3^{2n} T(m-1,n) \tag{5.2.1}$$

$$(w_2^2 + w_2^{-2})T(m,n) = q_2^{-2n-1} q_3^{-2m} T(m,n+1) + q_2^{2n-1} q_3^{2m} T(m,n-1). \tag{5.2.2}$$

We propose an algorithm, Algorithm 9, that uses those recurrence relations in a way such that the memory needed is only $O(1)$. The algorithm consists in iteratively computing the terms for $m = 0$ and $m = 1$ (using Equations (5.2.2)), and use them as soon as they are computed to initialize the other induction (Equations (5.2.1)); this corresponds to horizontal sweeps (from bottom to top) in the square $[0,B]^2$, with the first two terms of each line iteratively computed from the terms below them, as illustrated in Figure 5.1.

In Algorithm 9, terms of the form q_i^k as well as products thereof must also be computed recursively. We did so in our implementation, but this is deliberately omitted here for brevity. Also, despite the use of notations $T(m,n)$, it shall be understood that only constant storage is used by this algorithm, as can be seen by inspecting where values are actually used. Note that further speed-ups are possible if one tolerates using $O(\sqrt{P})$ extra memory, for instance by caching the q_1^m and the q_2^n.

Taking B as in Section 5.2.1 yields the right number of terms; we did not analyze the number of guard bits required by this algorithm, but as the arguments would be similar to the genus 1

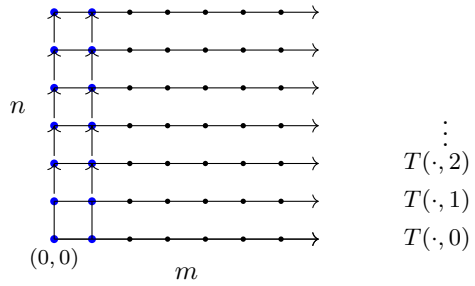

Figure 5.1: Order of summation of the terms in Algorithm 9. Arrows going right correspond to applying Equation (5.2.1), arrows going up correspond to applying Equation (5.2.2). 4 terms need to be stored to initialize the horizontal sweeps, starting with the corners of the bottom left square, then moving that square up when increasing n. Each horizontal sweep requires the storage of 2 terms.

Algorithm 9 Naive algorithm for $\theta(z,\tau)$ in genus 2.
Input: z, τ with absolute precision P, and B a summation bound.
Output: $\theta_i(z,\tau), \theta_i(0,\tau)$ for $i \in \{0..3\}$, with absolute precision P.

$a \leftarrow (1,1,1,1)$; $b \leftarrow (1,1,1,1)$ ▷ These arrays will store $\theta_i(z,\tau)$ and $\theta_i(0,\tau)$.
Compute q_1, q_2, q_3, w_1, w_2 using our UniformExp routine (cf. Proposition 5.1.4).
Compute $R(m,n) = Q(m,n) + Q(-m,n)$ for $m,n \in \{0,1\}$.
Compute $T(m,n)$ for $m \in \{0,1\}$ and $n \in \{-1,0,1\}$.
Add contributions for $(0,1)$ to a and b, with the correct sign.
$u \leftarrow w_1^2 + w_1^{-2}$, $v \leftarrow w_2^2 + w_2^{-2}$
for $n = 1$ to $B-1$ **do**
    $R(0,n+1) \leftarrow q_2^{2n+1} R(0,n)$
    Add contribution for $(0,n+1)$ to b, with the correct sign.
    $T(0,n+1) \leftarrow q_2^{2n+1} v T(0,n) - q_2^{4n} T(0,n-1)$
    Add contributions for $(0,n+1)$ to a, with the correct sign.
end for
for $m = 1$ to B **do**
    $\rho_m \leftarrow q_3^{2m} + q_3^{-2m}$ ▷ can be computed inductively
    Add contributions for $(m,0)$ and $(m,1)$ to a and b, with the correct sign.
    for $n = 1$ to $B-1$ **do** ▷ One may refine the bound depending on m
        $R(m,n+1) \leftarrow q_2^{2n+1} \rho_m R(m,n) - q_2^{4n} R(m,n-1)$
        Add contribution for $(m,n+1)$ to b, with the correct sign.
        $T(m,n+1) \leftarrow q_2^{2n+1} q_3^{2m} v T(m,n) - q_2^{4n} q_3^{4m} T(m,n-1)$
        $T(m,-(n+1)) \leftarrow q_2^{2n+1} q_3^{-2m} v T(m,-n) - q_2^{4n} q_3^{-4m} T(m,-(n-1))$
        Add contributions for $(m,n+1)$ to a, with the correct sign.
    end for
    $R(m+1,0) \leftarrow q_1^{2m+1} R(m,0)$
    $R(m+1,1) \leftarrow q_1^{2m+1} (q_3^2 + q_3^{-2}) R(m,1) - q_1^{4m} R(m-1,1)$
    $T(m+1,0) \leftarrow q_1^{2m+1} u T(m,0) - q_1^{4m} T(m-1,0)$
    $T(m+1,1) \leftarrow q_1^{2m+1} q_3^2 u T(m,1) - q_1^{4m} q_3^4 T(m-1,1)$
    $T(m+1,-1) \leftarrow q_1^{2m+1} q_3^{-2} u T(m,-1) - q_1^{4m} q_3^{-4} T(m-1,-1)$
end for
return a, b.

case, it does not seem like it will change the asymptotic running time. Finally, the remarks made in Note 5.1.6 can be generalized to apply to this algorithm; in particular, for $P \geq c\mathrm{Im}(\tau_{1,1})$, the number of terms is bounded by a constant, and the asymptotic complexity is the one of UniformExp, which is quasi-linear and uniform in z, τ. We implemented Algorithm 9 in Magma, and will discuss timings in Section 7.3.

5.3 Genus g

We outline the analysis of two strategies to evaluate the series defining θ with absolute precision P, i.e. up to 2^{-P}. The first strategy is the one outlined in [DHB+04], while the second one attempts to generalize the naive algorithms we outlined in genus 1 and genus 2; the number of terms that are summed in each strategy is asymptotically the same in P. However, note that the first strategy considers an ellipsoid while the second one considers a cube; thus, it is likely that the second strategy is coarser (and hence perhaps slower in practice), although we did not manage to link the ellipsoid to the cube. We also discuss a way to use recurrence relations to compute the terms, which lowers the overall asymptotic complexity by a $\log P$ factor; this method can be applied to either of these two strategies.

Recall that the argument reduction strategies we discussed in Section 2.3 and Section 2.4.4 (e.g. $\tau \in \mathcal{F}'_g$) allow us to assume that

$$|\mathrm{Re}(\tau_{i,j})| \leq \tfrac{1}{2}, \quad \mathrm{Im}(\tau) \text{ is Minkowski-reduced}, \quad |\tau_{1,1}| \geq 1$$

$$|\mathrm{Re}(z_i)| \leq \tfrac{1}{2}, \qquad \mathrm{Im}(z_i) \leq \tfrac{1}{2} \textstyle\sum_{j \in \{1,\ldots,n\}} \mathrm{Im}(\tau_{i,j}).$$

Note that these conditions are not very well adapted to the analyses we present below. The first analysis singles out an exponential factor which cannot be controlled or dealt with using only these conditions, while the second analysis requires a conjecture (Conjecture 5.3.3) in order to exploit these conditions.

5.3.1 Deckoninck et. al's analysis

We find in [DHB+04] a first method to compute the series defining θ up to 2^{-P}. The authors determine an ellipsoid which contains the indices over which one should sum to get a final result precise up to 2^{-P}. This method does not seem to depend on any conditions on z, τ; however, the authors mention that using argument reduction is beneficial to the process, as it reduces the eccentricity of the ellipsoid.

Their argument reduction strategy is visibly inspired by [Sie89], who determined the fundamental domain $\mathcal{F}_g$ much in the same way as [Kli90] (see Section 2.4); however, the reduction they actually appear to be using is a bit different, as the conditions that are imposed are

$$|\mathrm{Re}(\tau_{i,j})| \leq \frac{1}{2}, \qquad \text{the matrix } T \text{ such that } \mathrm{Im}(\tau) = {}^tTT \text{ is LLL-reduced}, \qquad |\tau_{1,1}| \geq 1.$$

This reduction seems even weaker than the reduction in $\mathcal{F}'_g$ of Section 2.4.4, as the LLL reduction (which runs in polynomial time but does not necessarily find the smallest vector) is used instead of the Minkowski reduction (which runs in exponential time and finds the smallest vector). The termination of the algorithm is claimed to derive from the termination of the algorithm reducing in the fundamental domain, which is proven in [Sie89, Chapter 6, Section 5]; no indications are given on the number of steps that are needed before termination. The effect of this strategy on the number of terms is not quantified, but the article claims that it reduces it.

Note that the analysis presented in [DHB+04] gives results which are valid for a series which is equal to $\theta(z,\tau)e^{-\pi^t \operatorname{Im}(z)\operatorname{Im}(\tau)\operatorname{Im}(z)}$, hence disregarding an "exponential growth" factor [DHB+04, p. 3], and only computing what they call the "oscillatory part" of the theta function. The size of this exponential factor grows to infinity as $\operatorname{Im}(z)$ grows. Note that the conditions given by our argument reduction strategies (Section 2.3 and Section 2.4.4) do not allow us to control the size of this factor: for instance, if (z,τ) are reduced with these strategies, $(2^k z, 2^k \tau)$ is also reduced, yet the exponential factor goes to infinity as k grows. However, this does not mean that the size of θ grows, merely that the oscillatory part has to be computed at an increasingly larger precision in order to compensate for this factor.

The following theorem takes a closer look at the oscillatory part of the theta function.

Theorem 5.3.1 ([DHB+04, Theorem 2])**.** *Denote $[[V]] = V - [V]$, where $[V]$ is the vector with integer coordinates closest to V. Define $\Lambda = \{\sqrt{\pi}T(n+c), n \in \mathbb{Z}^g\}$ with $\tau = {}^tTT$ (Cholesky decomposition) and $c = [[\operatorname{Im}(\tau)^{-1}\operatorname{Im}(z)]]$. For $B > 0$ define the ellipsoid of size B*

$$S_B = \{n \in \mathbb{Z}^g \mid ||\sqrt{\pi}T(n+c)|| < B\},$$

where $||\cdot||$ is the L^2 norm. Then the oscillatory part of the theta function can be approximated to 2^{-P} by summing over the terms whose indices are inside the ellipsoid of size R, where R is the solution to the equation

$$2^{-P} = \frac{g}{2}\frac{2^g}{\rho^g}\Gamma(g/2, (R-\rho/2)^2)$$

where Γ is the incomplete gamma function and ρ is the length of the shortest vector of Λ.

Note that, if τ is reduced as above, $\rho = \sqrt{\operatorname{Im}(\tau_{1,1})} \geq \sqrt{\sqrt{3}/2}$, and hence the number of terms needed can be upper bounded with a bound which is independent from τ.

Neglecting the dependency in τ and z, we get the rather coarse bound of $O(R^g)$ terms needed. We complete the analysis in [DHB+04] by computing an explicit estimate on R:

Proposition 5.3.2. *Treating z, τ (and hence ρ) as constants, we have $R = O(\sqrt{P})$, i.e. summing $O(P^{g/2})$ terms is sufficient to get a result accurate to P bits.*

Proof. Assuming that g is even (which we can do since Γ is growing in the first parameter for R large enough), we use integration by parts $g/2$ times to prove that

$$\begin{aligned}\Gamma(g/2,d) &= (g/2-1)!e^{-d} + \sum_{i=1}^{g/2}(g/2-1)\cdots(g/2-i)d^{g/2-i}e^{-d} \\ &\leq \frac{g}{2}(g/2-1)!d^{g/2-1}e^{-d} \leq e^{-d+g/2(\log d+\log(g/2))}\end{aligned}$$

Hence:

$$\frac{g}{2}\frac{2^g}{\rho^g}\Gamma(g/2,d) \leq 2^{-d\log_2 e + g/2(\log d+\log(g/2))+\log(g/2)+g\log(2/\rho)}$$

Asymptotically, i.e. for R large enough, taking $d = P\log_2 e + g\log P + g\log(2/\rho) + g = O(P)$ is enough for the right hand side to be smaller than 2^{-P}. Hence $R = O(\sqrt{P})$. □

Note that this is not as good as the asymptotics in genus 1 and 2, which showed the number of terms to be $O\left(\left(\frac{P}{\operatorname{Im}(\tau_{1,1})}\right)^{g/2}\right)$. In fact, we have $R = O\left(\operatorname{Im}(\tau_{1,1}) + \sqrt{P - \log\operatorname{Im}(\tau_{1,1})}\right)$ (since $\rho = \sqrt{\operatorname{Im}(\tau_{1,1})}$), which gets worse as $\operatorname{Im}(\tau_{1,1})$ increases. This, combined with the fact that the size of the exponential factor cannot be bounded even for z, τ reduced, means that one cannot use this analysis to build a fast algorithm with uniform complexity, as we do in genus 1 and 2 (see e.g. Algorithm 11 or Algorithm 15 for genus 1, and Note 7.2.7 for genus 2).

5.3.2 Truncated sums

Another method is to attempt to bound the remainder of the series defining θ by a series which can be computed more easily, e.g. a geometric series; this is the method we used in genus 1 and genus 2.

Recall the proof of Proposition 2.1.9: we took R an orthogonal matrix such that ${}^tR\tau R$ is diagonal, and denote λ the smallest eigenvalue of $\mathrm{Im}(\tau)$. Then

$$|\theta_{[0;b]}(z,\tau)-1| \leq 2^g \sum_{n\in\mathbb{N}^g\setminus\{0\}} q^{n_1^2+\ldots+n_g^2} w^{-2\sum n_i}$$

with $q=e^{-\pi\lambda}$ and $w=e^{-2\pi\max \mathrm{Im}(z_i)}$. We can apply similar arguments to $|\theta_{[0;b]}(z,\tau)-S_R(z,\tau)|$, where $S_R(z,\tau)=\sum_{n_i\in[-R,R]} e^{i\pi^t n\tau n}e^{2i\pi^t nz}$. We can then write for R large enough

$$|\theta_{[0;b]}(z,\tau)-S_R(z,\tau)| \leq 2^g \sum_{n_1,\ldots,n_g\geq R} e^{-\frac{\pi}{\lambda}\left((n_1-c)^2+\ldots+(n_g-c)^2\right)}$$

with $c=\frac{\max \mathrm{Im}(z_i)}{\lambda}$. Taking $R=O(\sqrt{P/\lambda})$ is enough to get a sum which is accurate to P bits, which means one needs to sum at most $O\left(\left(\frac{P}{\lambda}\right)^{g/2}\right)$ terms.

This result is not entirely satisfactory with regard to the argument reduction strategies that are deployed in genus g – either the reduction to $\mathcal{F}_g$ (Section 2.4) or the weaker reductions of Section 2.4.4. Indeed, these reductions give conditions on the coefficients of τ (or $\mathrm{Im}(\tau)$) but none on the eigenvalues of $\mathrm{Im}(\tau)$. To the best of our knowledge, there is no result linking the eigenvalues of $\mathrm{Im}(\tau)$ to the coefficients of τ; we note, however, that [Dup06, p. 127] puts forward the following conjecture:

Conjecture 5.3.3 ([Dup06, p. 127])**.** *For any g, there exists a constant c_g such that for any matrix $M\in\mathcal{M}_g(\mathbb{R})$ such that M is symmetric, positive definite and Minkowski-reduced, its smallest eigenvalue λ is such that $\lambda\geq c_g M_{1,1}$.*

The conjecture holds in genus 1 and 2 [Dup06, p. 137]. Should that conjecture be proven for any genera, this would prove that the number of terms needed is in fact $O\left(\left(\frac{P}{\mathrm{Im}(\tau_{1,1})}\right)^{g/2}\right)$, which corresponds exactly to the complexities we found in genus 1 and 2. However, if this conjecture is not true, the number of terms needed in this algorithm for a $\tau\in\mathcal{F}_g$ could be arbitrarily big.

5.3.3 Recurrence relations

One method of computing the sum above is to compute each term explicitly, i.e. with the computation of an exponential for each term. This algorithm is fairly straightforward to implement, as one iterates over all the indices and computes the corresponding exponential term; its cost is $O(\mathcal{M}(P)\log P P^{g/2})$, and it requires only a small amount of memory. Note that this is more expensive than the algorithms we showed in genus 1 (Algorithm 7) and genus 2 (Algorithm 9).

We show a faster method, which uses the recurrence relations of degree two that exist for any choice of index; this allows to compute all the terms using only multiplications once the initial $O(g^2)$ exponentiations $e^{i\pi\tau_{j,k}}, e^{i\pi z_j}$ are computed. This is a generalization of Algorithm 7 and Algorithm 9. However, note that this method has a larger memory requirement; we conjecture that one needs to store at least $O(2^{2g})$ P-bit numbers. Furthermore, the implementation of this method seems tricky in the general case, as one can certainly notice when looking at the

algorithm in genus 2 (Algorithm 9); it may be possible to implement this as a recursive function with some global variables, but we did not undertake this.

We outline the recurrence relations in the g-th (i.e. last) index. For $(\epsilon_1, \dots, \epsilon_g) \in \{-1, 1\}^g$ define

$$\alpha_{n_g}^{(\epsilon_1,\dots,\epsilon_g)} = e^{i\pi^t n \tau n}(e^{2i\pi \sum \epsilon_j n_j z_j} + e^{-2i\pi \sum \epsilon_j n_j z_j})$$

treating $n_i, i < g$ as constants. Note that since $\alpha_{n_g}^{(\epsilon_1,\dots,\epsilon_g)} = \alpha_{n_g}^{(-\epsilon_1,\dots,-\epsilon_g)}$, we only need to consider the case $\epsilon_g = 1$; hence this defines 2^{g-1} quantities. We have

$$\text{approx}_P\left(\theta(z,\tau)\right) = \sum_{n_1,\dots n_{g-1} \in [0,R]^{g-1}} \left(\sum_{(\epsilon_1,\dots,\epsilon_{g-1}) \in \{-1,1\}} \sum_{n_g \in [0,R]} \alpha_{n_g}^{(\epsilon_1,\dots,\epsilon_{g-1},1)} \right) \tag{5.3.1}$$

Taking a closer look to $\alpha_{n_g}^{(1,\epsilon_2,\dots,\epsilon_g)}$ yields a recurrence relation:

$$(e^{2i\pi z_g} + e^{-2i\pi z_g})\alpha_{n_g}^{(\epsilon_1,\dots,\epsilon_{g-1},1)} = \left(\prod_{i=1}^{g-1} q_{i,g}^{-2\epsilon_i |n_i|}\right) q_{g,g}^{-2n_g-1} \alpha_{n_g+1}^{(\epsilon_1,\dots,\epsilon_{g-1},1)} + \left(\prod_{i=1}^{g-1} q_{i,g}^{2\epsilon_i |n_i|}\right) q_{g,g}^{2n_g-1} \alpha_{n_g-1}^{(\epsilon_1,\dots,\epsilon_{g-1},1)}$$

i.e.

$$\alpha_{n_g+1}^{(\epsilon_1,\dots,\epsilon_{g-1},1)} = \left(\prod_{i=1}^{g-1} q_{i,g}^{2\epsilon_i |n_i|}\right) q_{g,g}^{2n_g+1} (e^{2i\pi z_g} + e^{-2i\pi z_g})\alpha_{n_g}^{(\epsilon_1,\dots,\epsilon_{g-1},1)} - \left(\prod_{i=1}^{g-1} q_{i,g}^{2\epsilon_i |n_i|}\right)^2 q_{g,g}^{4n_g} \alpha_{n_g-1}^{(\epsilon_1,\dots,\epsilon_{g-1},1)}$$

Recall that we assume that the n_i are constants; hence, we can suppose that $\left(\prod_{i=1}^{g-1} q_{i,g}^{-2\epsilon_i |n_i|}\right)$ is precomputed. The $q_{g,g}^{-2n_g-1}, q_{g,g}^{2n_g-1}$ can be computed iteratively; hence, this recurrence relation allows one to compute all the $\alpha_{n_g+1}^{(\epsilon_1,\dots,\epsilon_{g-1},1)}$ from the $\alpha_{n_g}^{(\epsilon_1,\dots,\epsilon_{g-1},1)}, \alpha_{n_g-1}^{(\epsilon_1,\dots,\epsilon_{g-1},1)}$ in $3 \times 2^{g-1} + 3$ multiplications. Hence evaluating the inner sum in Equation (5.3.1) for all $b \in \frac{1}{2}\mathbb{Z}^g/\mathbb{Z}^g$ can be done in $O(R2^{g-1})$ multiplications.

Hence, we just showed how, given $n_1, \dots, n_{g-1}$, $\epsilon_1, \dots, \epsilon_{g-1}$ and $\alpha_0^{(\epsilon_1,\dots,\epsilon_{g-1},1)}, \alpha_1^{(\epsilon_1,\dots,\epsilon_{g-1},1)}$, we can compute the following terms of the sum using a recurrence relation. Computing these first two terms can be done using similar recurrence relations on the other variables, with a reasoning which is very similar to the one we just used: similar recurrence relations of degree 2 exist for these, and it only costs a constant number of multiplications to increase one of the indices.

This means that the total cost of this method is $O(2^g R^g)$ multiplications and $O(g^2)$ exponentiations, which gives a cost of $O(\mathcal{M}(P)P^{g/2})$ bit operations. This agrees with the running time of Algorithm 7 and Algorithm 9. Finally, note that once again, one can compute all the fundamental theta functions at once for the same cost, since one simply needs to change the sign of the terms accordingly.

Chapter 6

Fast computation of the theta function in genus 1

Recall that Jacobi's theta function is defined as

$$\begin{array}{rcl} \mathbb{C}\times\mathcal{H} & \to \mathbb{C} & \\ (z,\tau) & \mapsto & \displaystyle\sum_{n\in\mathbb{Z}} e^{i\pi\tau n^2} e^{2i\pi z n} = 1 + \sum_{n\in\mathbb{N}} q^{n^2}(w^{2n}+w^{-2n}) \end{array}$$

with $q = e^{i\pi\tau}$ (the "nome") and $w = e^{i\pi z}$. This chapter provides an asymptotically fast algorithm to compute $\theta(z,\tau)$ with absolute precision P in $O(\mathcal{M}(P)\log P)$ bit operations. We start with the easier case of the fast, quasi-optimal time algorithm to compute theta-constants featured in [Dup11], which we refine; we then use a similar approach for the general case. The results from this chapter are taken from a paper that has been accepted for publication in the *Mathematics of Computation* journal [Lab15].

6.1 Preamble: fast theta-constants

We discuss an algorithm which computes those three values with absolute precision P in time $O(\mathcal{M}(P)\log P)$ bit operations. A fuller description of the algorithm can be found in [Dup11]. This algorithm has been applied to the computation of modular functions such as $j(\tau)$ and Dedekind's η function, using formulas linking these quantities to theta-constants. This is of use for instance in the CM method, where the computation of class polynomials (which zeroes are values of $j(\tau)$) has to be performed; see [Eng09] for more details.

Recall the definition of theta-constants:

$$\theta_0(0,\tau) = \sum_{n\in\mathbb{Z}} q^{n^2}, \qquad \theta_1(0,\tau) = \sum_{n\in\mathbb{Z}} (-1)^n q^{n^2}, \qquad \theta_2(0,\tau) = \sum_{n\in\mathbb{Z}} q^{\left(n+\frac{1}{2}\right)^2}$$

with $q = e^{i\pi\tau}$.

As shown in Section 2.5.3, one only needs to consider the case $\tau \in \mathcal{F}$. Theorem 3.2.8 then shows that the AGM of $\theta_0^2(0,\tau)$ and $\theta_1^2(0,\tau)$ is 1. Finally, recall Note 3.2.10, which uses the homogeneity of the AGM function (i.e. $\mathrm{AGM}(a,b) = a\,\mathrm{AGM}\left(1,\frac{b}{a}\right)$) to get

$$\mathrm{AGM}\left(1, \frac{\theta_1(0,\tau)^2}{\theta_0(0,\tau)^2}\right) = \frac{1}{\theta_0(0,\tau)^2} \tag{6.1.1}$$

This gives a way to recover $\theta_0(0,\tau)^2, \theta_1(0,\tau)^2$ from the knowledge of their quotient.

6.1.1 A quasi-optimal time algorithm to compute theta-constants

We outline the algorithm in [Dup06] to compute theta-constants. Recall Equation (2.5.10):

$$\theta_2(0,\tau)^2 = \frac{i}{\tau}\theta_1\left(0, \frac{-1}{\tau}\right)^2, \qquad \theta_0(0,\tau)^2 = \frac{i}{\tau}\theta_0\left(0, \frac{-1}{\tau}\right)^2.$$

This means that $\frac{\theta_2(0,\tau)^2}{\theta_0(0,\tau)^2} = \frac{\theta_1(0,\tau')^2}{\theta_0(0,\tau')^2}$ with $\tau' = \frac{-1}{\tau}$. Note that for $\tau \in \mathcal{F}$, $\frac{-1}{\tau} \in \mathcal{D}_1 \subset \mathcal{D}_2$ (see e.g. [Dup06, Figure 2.2, p.60]), and hence the choices of signs for $\frac{-1}{\tau}$ are all good (Prop. 3.2.9). Hence, for all $\tau \in \mathcal{F}$,

$$\mathrm{AGM}(\theta_0^2(0,\tau), \theta_2^2(0,\tau)) = \frac{i}{\tau}. \tag{6.1.2}$$

Furthermore, the Jacobi formula (Equation (2.5.6)) is $\theta_2^4(0,\tau) = \theta_0^4(0,\tau) - \theta_1^4(0,\tau)$, which can be rewritten as

$$\frac{\theta_2^2(0,\tau)}{\theta_0^2(0,\tau)} = \sqrt{1 - \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}},$$

the square root being the one with positive real part since, for $\tau \in \mathcal{F}$, $\tau' = \frac{-1}{\tau} \in \mathcal{D}_1$ and $\mathrm{Re}\left(\frac{\theta_2^2(0,\tau)}{\theta_0^2(0,\tau)}\right) = \mathrm{Re}\left(\frac{\theta_1^2(0,\tau')}{\theta_0^2(0,\tau')}\right) \geq 0$.

Putting it all together, and using the homogeneity of the AGM (Note 3.2.10), we have for all $\tau \in \mathcal{F}$

$$\frac{\mathrm{AGM}\left(1, \sqrt{1 - \frac{\theta_1^4(0,\tau)}{\theta_0^4(0,\tau)}}\right)}{\mathrm{AGM}\left(1, \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right)} = \frac{i}{\tau}$$

which means that the function

$$f_\tau : z \mapsto \tau\,\mathrm{AGM}(1, \sqrt{1-z^2}) - i\,\mathrm{AGM}(1, z)$$

is 0 at $\frac{\theta_1(0,\tau)^2}{\theta_0(0,\tau)^2}$.

The algorithm then consists in using Newton's method on f_τ to compute $\frac{\theta_1(0,\tau)^2}{\theta_0(0,\tau)^2}$. The starting point of the method must be close enough to $\frac{\theta_1(0,\tau)^2}{\theta_0(0,\tau)^2}$; [Dup06] finds that computing (using the naive algorithm) an approximation of the quotient with precision $4.16\,\mathrm{Im}(\tau)$ bits is enough to make Newton's method converge. Finally, recall Theorem 0.3.9, which shows that one should apply Newton's method while doubling the working precision at each step. The full algorithm is Algorithm 10.

We refer to [Dup11] for notes on the fast computation of $\frac{f_\tau(t)}{f'_\tau(t)}$; note however that one can simply use finite differences $\frac{f_\tau(t+\epsilon) - f_\tau(t)}{\epsilon}$ as an approximation of this derivative.

Proposition 6.1.1. *For $\tau \in \mathcal{F}$ with absolute precision P, taking $\mathcal{P} = O(P + \mathrm{Im}(\tau))$ in Algorithm 10 allows the computation of $\theta_0^2(0,\tau), \theta_1^2(0,\tau)$ with absolute precision P in time*

$$O\left(\mathcal{M}(P + \mathrm{Im}(\tau)) \times (\log P + \log \mathrm{Im}(\tau))\right)$$

which is quasi-optimal time if τ is assumed to be in a compact set.

Algorithm 10 Compute $\theta_0^2(0,\tau), \theta_1^2(0,\tau)$ with absolute precision P.
Input: τ with absolute precision P.

1: Compute $\theta_0^2(0,\tau), \theta_1^2(0,\tau)$ with absolute precision P_0 using Algorithm 7.
2: $t \leftarrow \frac{\theta_1(0,\tau)^2}{\theta_0(0,\tau)^2}$
3: $p \leftarrow P_0$
4: **while** $p \leq \mathcal{P}$ **do**
5: $\quad t \leftarrow t - \frac{f_\tau(t)}{f'_\tau(t)}$, computed with precision $2p$
6: $\quad p \leftarrow 2p - \delta$ ▷ with δ defined in Corollary 0.3.8
7: $\quad$ Remove the last δ digits of t.
8: **end while**
9: $t_0 \leftarrow \mathrm{AGM}(1, t)$
10: **return** $(t_0, t_0 t)$.

Proof. We refer to [Dup06, Dup11] for a full analysis of this algorithm. Correctness follows from the fact that the choice of square roots are good for τ and $\frac{-1}{\tau}$ since $\tau \in \mathcal{F}$.

First of all, recall that the number of iterations necessary to compute the AGM increases as $\sqrt{1-z^2}$ gets small (Theorem 3.2.6). Hence the number of iterations needed in the computation of $\mathrm{AGM}\left(1, \frac{\theta_2^2(0,\tau)}{\theta_0^2(0,\tau)}\right)$ with precision $\mathcal{P}$ is $O(\log_2 \mathcal{P} + \log_2 \mathrm{Im}(\tau))$.

Secondly, recall that

- Computing $a - b$ can induce a potentially large loss of *relative precision*, of the order of $-\log_2|a-b|$ bits, whereas multiplication and square roots only induce a loss of at most 2 bits of relative precision [Dup06, Section 1.1.2];
- Computing $\sqrt{a}$ with a very close to 0 induces a loss of *absolute precision* of the order of $-\frac{1}{2}\log_2|a|$, whereas adding two complex numbers only induces a loss of at most 1 bit of precision (Theorem 0.3.3).

In either case, i.e. whether we are looking to compute the quantities above with *absolute* or *relative* precision, the computation of $\sqrt{1-z^2}$ for $z \simeq \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}$ will cause a potentially large loss of precision. In fact, since $\theta_2^4(0,\tau) \sim 16q$, the loss of precision is $O(\mathrm{Im}(\tau))$. This means that one should take $\mathcal{P} = O(P + \mathrm{Im}(\tau))$ in the algorithm in order to compensate for the loss of precision.

Putting it all together, the cost of the evaluation of f_τ in the algorithm is

$$O\left(\mathcal{M}(P + \mathrm{Im}(\tau)) \times (\log P + \log \mathrm{Im}(\tau))\right).$$

This is also the final complexity, since applying Newton's method is as costly as the last, full-precision iteration. □

An application of this algorithm can be found in [Eng09], where the computation of theta-constants is used to compute certain modular forms such as Dedekind's η function. These computations are useful to compute class polynomials, which are of interest in the CM method, used to generate cryptographically safe elliptic curves. The use of this algorithm provides the best theoretical complexity for the computation of η. In practice, this approach gives a better running time than a naive algorithm (exploiting the sparseness of the series defining η) for precisions above 250 000 bits, which corresponds to the largest example (a field of class number 100 000) that is computed in [Eng09].

6.1.2 A faster algorithm with uniform complexity

This section gives an algorithm with overall better complexity, which furthermore does not depend on τ at all.

Making the complexity uniform

The τ-duplication formulas (Equation (2.5.4)) allow us to compute easily $\theta_0(0,\tau)^2$ and $\theta_1(0,\tau)^2$ from $\theta_0\left(0,\frac{\tau}{2^n}\right), \theta_1\left(0,\frac{\tau}{2^n}\right)$. Hence, they can be used for argument reduction purposes:

Proposition 6.1.2. *Let $\tau \in \mathcal{F}$, with absolute precision P. Let $s \geq 0$ be such that $1 \leq \frac{|\tau|}{2^s} < 2$. Then running Algorithm 10 on $\frac{\tau}{2^s}$ costs $c\mathcal{M}(P)\log P$ operations with c independent of τ. One can then recover the final result using s τ-duplication formulas, for a cost of $O(\mathcal{M}(P)s) = O(\mathcal{M}(P)\log \mathrm{Im}(\tau))$.*

Note that in the case $P \geq c\,\mathrm{Im}(\tau)$, we have $s = O(\log P)$ in Proposition 6.1.2, and hence quasi-optimal running time uniformly in τ. If this is not the case, we can use the remark made in Section 5.1.3: if $P \leq c\,\mathrm{Im}(\tau)$ for c a constant, the complexity of the algorithm is dominated by the complexity of evaluating π and $\exp(i\pi\tau)$, which is $O(\mathcal{M}(P)\log P)$.

However, note that the complexity of evaluating $e^{i\pi\tau}$ at a precision P depends in τ unless one uses the UniformExp subroutine (see Proposition 5.1.4); this difficulty does not seem to appear in the analysis of [Dup06, Algorithm 8], although it is relevant when computing $e^{i\pi\tau}$ with relative precision P and large $\mathrm{Im}(\tau)$. In the case of theta-constants, it is actually even simpler, since

$$|\theta_0(0,\tau) - 1| \leq 2\frac{|q|}{1-|q|} \leq 4|q|$$

Hence, if $P \leq \pi \log_2 e\,\mathrm{Im}(\tau) - 2$, we have $4|q| \leq 2^{-\pi\log_2 e\,\mathrm{Im}(\tau)+2} \leq 2^{-P}$; hence, 1 is an approximation of $\theta_0(0,\tau)$ (and of $\theta_1(0,\tau)$) with absolute precision P. Furthermore, since θ is close to 1, a similar argument can be used for relative precision.

Computation of θ_2

We now show how to compute $\theta_2(0,\tau)$ in the same uniform quasi-linear time.

Recall from Section 5.1.4 (or from [Dup06, Section 4.2.4]) that $\theta_2(0,\tau)^2$ can be recovered using Jacobi's formula (2.5.6), at the cost of $O(\mathrm{Im}(\tau))$ bits. This requires computing $\theta_{0,1}(0,\tau)^2$ with relative precision $O(P+\mathrm{Im}(\tau))$, which changes the asymptotics of the algorithm in the case where one needs to use the naive method: the complexity becomes worse as $\mathrm{Im}(\tau)$ grows. Hence, we cannot use Jacobi's formula to compute $\theta_2(0,\tau)^2$ if $P \leq c\,\mathrm{Im}(\tau)$.

Instead, one could use Algorithm 8 (Section 5.1.4); however, the same considerations on the computation of $e^{i\pi\tau}$ apply. Note once again that the triangle inequality gives, for $\tau \in \mathcal{F}$:

$$\left|\frac{\theta_2(0,\tau)}{q^{1/4}}\right| \leq 1 + \frac{2|q|^2}{1-|q|^4} \leq 1.009$$

Hence, if $P \leq \frac{\pi\log_2 e}{4}\,\mathrm{Im}(\tau) - 1$, an approximation of $\theta_2(0,\tau)$ with P bits of absolute precision is 0.

In the case where $P \geq \frac{\pi\log_2 e}{4}\,\mathrm{Im}(\tau) - 1$, we use a trick to compute θ_2 from the other theta-constants. Recall (cf Equation (2.5.4)) the τ-duplication formula for the third theta-constant:

$$\theta_2(0,2\tau)^2 = \frac{\theta_0(0,\tau)^2 - \theta_1(0,\tau)^2}{2},$$

Hence, if we use τ-duplication formulas for $\theta_0(0,\tau), \theta_1(0,\tau)$, we can also at the same time recover θ_2; this loses much fewer bits than Jacobi's formula.

We then need to make sure that for every $\tau \in \mathcal{F}$, at least one τ-duplication formula is used. Note that this is not necessarily the case in [Dup06, Algorithm 8], e.g. for $|\tau| < 2$. We have:

Proposition 6.1.3. *Consider the domain*

$$D = \{\tau \in \mathcal{H} \mid |\mathrm{Re}(\tau)| \leq \frac{1}{4}, \frac{1}{2} \leq |\tau| < 1\}$$

Then Algorithm 10 outputs the correct result, i.e. the choices of signs are always good.

Proof. We have $D \subset \mathcal{D}_1$ (see Figure 3.1). Now, for $\tau \in D$, we have

$$\frac{1}{|\tau|} > 1, \qquad |\mathrm{Re}\left(\frac{-1}{\tau}\right)| = \frac{|\mathrm{Re}(\tau)|}{|\tau|^2} \leq 1$$

hence $\frac{-1}{\tau} \in \mathcal{F}$. This means that $f_\tau\left(\frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right) = 0$ for $\tau \in D$, which proves correctness. □

Hence, we simply take the argument reduction strategy of [Dup06, Algorithm 8] one step further: for $\tau \in \mathcal{F}$, we take $s \geq 0$ such that $\frac{\tau}{2^{s+1}} \in \mathcal{D}$, then use at least one τ-duplication formula to recover $\theta(0,\tau)^2$.

We combine the remarks here in Algorithm 11, which is a variant on [Dup06, Algorithm 8].

Algorithm 11 Compute $\theta_{0,1,2}(0,\tau)$ with absolute precision P.
Input: $\tau \in \mathcal{F}$ with absolute precision P.

1: **if** $P \leq 1.13\,\mathrm{Im}(\tau) - 2$ **then**
2: **return** (1,1,0).
3: **else**
4: Let $s \in \mathbb{N}$ such that $1 \leq \frac{|\tau|}{2^s} < 2$.
5: Compute $\theta_0\left(0, \frac{\tau}{2^{s+1}}\right)^2, \theta_1\left(0, \frac{\tau}{2^{s+1}}\right)^2$ using Algorithm 10.
6: **for** $i = 1$ to $s+1$ **do**
7: Compute $\theta_0\left(0, \frac{\tau}{2^{s-i}}\right)^2, \theta_1\left(0, \frac{\tau}{2^{s-i}}\right)^2$ using the τ-duplication formulas.
8: **if** $i = s+1$ **then**
9: Compute $\theta_2(0,\tau)^2 = \frac{\theta_0(0,\tau/2)^2 - \theta_1(0,\tau/2)^2}{2}$.
10: **end if**
11: **end for**
12: **return** $\theta_{0,1,2}(0,\tau)$.
13: **end if**

Proposition 6.1.4. *Algorithm 11, working with internal precision $O(P)$, has complexity $O(\mathcal{M}(P)\log P)$, uniformly in τ.*

Proof. For $P \leq 1.13\,\mathrm{Im}(\tau) - 2$, we have $P \leq \frac{\pi \log_2 e}{4}\,\mathrm{Im}(\tau) - 1$ and $P \leq \pi \log_2 e\,\mathrm{Im}(\tau) - 2$; hence $(1,1,0)$ is an approximation of the theta-constants. We then look at the other case.

Algorithm 10 is called in Step 5 with the argument $\frac{\tau}{2^{s+1}}$, which is of norm smaller than 1. One can thus bound its imaginary part by 1 in Proposition 6.1.1, which shows that the cost of Step 5 is $O(\mathcal{M}(P)\log P)$, independently of τ. Furthermore, note that $s \leq \log_2|\mathrm{Im}(\tau)|$, and hence $s \leq \log_2 P + c$ given our bound on $\mathrm{Im}(\tau)$ in this case. The τ-duplication formulas are thus applied $O(\log P)$ times; the analysis in [Dup06, Section 4.2.3] shows that this loses $O(\log P)$ bits of precision. In the end, we get a $O(\mathcal{M}(P)\log P)$ algorithm with asymptotic complexity independent of τ. □

6.2 A function related to $\theta(z,\tau)$

We generalize the algorithm presented in Section 6.1, which gives an algorithm that computes $\theta_{0,1,2}(z,\tau)$ in $O(\mathcal{M}(P)\log P)$, for z,τ verifying the usual argument reduction conditions (see Chapter 5). We use the same strategy as [Dup11], that is to say, find a quadratically convergent sequence and attempt to invert the function computing its limit using Newton's method. To be more precise, we exhibit a function $\mathfrak{F}$ such that

$$\mathfrak{F}\left(1, \frac{\theta_1(z,\tau)^2}{\theta_0(z,\tau)^2}, 1, \frac{\theta_1(0,\tau)^2}{\theta_0(0,\tau)^2}\right) = (z,\tau)$$

Furthermore, our algorithm has uniform complexity $O(\mathcal{M}(P)\log P)$, i.e. the complexity is independent of z,τ.

6.2.1 The F sequence

Definition of the F sequence

Recall the τ-duplication formulas in genus 1:

$$\begin{aligned}\theta_0(z,2\tau)^2 &= \frac{\theta_0(z,\tau)\theta_0(0,\tau)+\theta_1(z,\tau)\theta_1(0,\tau)}{2}\\ \theta_1(z,2\tau)^2 &= \frac{\theta_0(z,\tau)\theta_1(0,\tau)+\theta_1(z,\tau)\theta_0(0,\tau)}{2}\end{aligned}$$

We then define a function F as:

$$\begin{aligned}F:\mathbb{C}^4 &\to \mathbb{C}^4\\ (x,y,z,t) &\mapsto \left(\frac{\sqrt{x}\sqrt{z}+\sqrt{y}\sqrt{t}}{2}, \frac{\sqrt{x}\sqrt{t}+\sqrt{y}\sqrt{z}}{2}, \frac{z+t}{2}, \sqrt{z}\sqrt{t}\right)\end{aligned}$$

The τ-duplication formulas (Equation (2.5.4)) show that for some appropriate choice of roots we have

$$F\left(\theta_0^2(z,\tau),\theta_1^2(z,\tau),\theta_0^2(0,\tau),\theta_1^2(0,\tau)\right) = \left(\theta_0^2(z,2\tau),\theta_1^2(z,2\tau),\theta_0^2(0,2\tau),\theta_1^2(0,2\tau)\right).$$

Remark. One can also write rewrite F using Karatsuba-like techniques

$$\begin{aligned}F(x,y,z,t) = \quad &\left(\frac{(\sqrt{x}+\sqrt{y})(\sqrt{z}+\sqrt{t})+(\sqrt{x}-\sqrt{y})(\sqrt{z}-\sqrt{t})}{4},\right.\\ &\frac{(\sqrt{x}+\sqrt{y})(\sqrt{z}+\sqrt{t})-(\sqrt{x}-\sqrt{y})(\sqrt{z}-\sqrt{t})}{4},\\ &\left.\frac{(\sqrt{z}+\sqrt{t})^2+(\sqrt{z}-\sqrt{t})^2}{4}, \frac{(\sqrt{z}+\sqrt{t})^2-(\sqrt{z}-\sqrt{t})^2}{4}\right)\end{aligned} \tag{6.2.1}$$

to speed up computations. □

Good choices of sign

Similarly to the complex AGM (Section 3.2), we define a *good choice* for square roots at the rank n as the following conditions being satisfied:

1. $\mathrm{Re}(\sqrt{x_n}) \geq 0, \qquad \mathrm{Re}(\sqrt{z_n}) \geq 0;$
2. $|\sqrt{x_n} - \sqrt{y_n}| < |\sqrt{x_n} + \sqrt{y_n}|, \quad$ or $|\sqrt{x_n} - \sqrt{y_n}| = |\sqrt{x_n} + \sqrt{y_n}|$ and $\mathrm{Im}\left(\frac{\sqrt{y_n}}{\sqrt{x_n}}\right) > 0;$
3. $|\sqrt{z_n} - \sqrt{t_n}| < |\sqrt{z_n} + \sqrt{t_n}|, \quad$ or $|\sqrt{z_n} - \sqrt{t_n}| = |\sqrt{z_n} + \sqrt{t_n}|$ and $\mathrm{Im}\left(\frac{\sqrt{t_n}}{\sqrt{z_n}}\right) > 0.$

Note that the condition $|x-y| < |x+y|$ is equivalent to $\mathrm{Re}\left(\frac{y}{x}\right) > 0$.

Again, similarly to the AGM, we define an *optimal F sequence*:

Definition 6.2.1. Let $x, y, z, t \in \mathbb{C}$, and define the *optimal F sequence* associated to x, y, z, t as the sequence $((x_n, y_n, z_n, t_n))_{n\in\mathbb{N}}$ such that:

$$(x_0, y_0, z_0, t_0) = (x, y, z, t)$$
$$(x_{n+1}, y_{n+1}, z_{n+1}, t_{n+1}) = F\left(x_n, y_n, z_n, t_n\right)$$

where all the choices of sign for the square roots are good. We sometimes denote $F^\infty(x,y,z,t)$ the limit of this optimal F sequence.

The study of this sequence and its convergence is done in Section 6.2.4.

Note that

$$z_{n+1} - t_{n+1} = \frac{z_n + t_n}{2} - \sqrt{z_n t_n} = \frac{(\sqrt{z_n} - \sqrt{t_n})^2}{2}$$
$$z_{n+1} + t_{n+1} = \frac{(\sqrt{z_n} + \sqrt{t_n})^2}{2}$$
$$\text{hence} \qquad |z_{n+1} - t_{n+1}| < |z_{n+1} + t_{n+1}| \quad \Leftrightarrow \quad |\sqrt{z_n} - \sqrt{t_n}| < |\sqrt{z_n} + \sqrt{t_n}|$$

Hence, our third condition and the condition "b_{n+1} is the good choice of square roots" in the AGM are very similar. There are, however, subtle differences, which is why we wrote $\sqrt{z}\sqrt{t}$ instead of $\sqrt{zt}$ in the definition of F. The computation of the AGM involves only one square root computation, and the study of the choice of signs for theta-constants (Section 3.2.3) involves determining the sign of $\frac{\theta_1^2(0,2\tau)}{\theta_0^2(0,2\tau)}$. However, the definition of F requires the computation of both $\sqrt{z}$ and $\sqrt{t}$, which we then multiply (the final result being the same as in the AGM); hence, our third condition leads us to study the sign of $\frac{\theta_1(0,\tau)}{\theta_0(0,\tau)}$ instead. This is accomplished in Proposition 6.2.4, which uses different methods than the ones in [Cox84].

Finally, note that the condition $|\sqrt{z_n} - \sqrt{t_n}| < |\sqrt{z_n} + \sqrt{t_n}|$ can be satisfied in two ways, depending on which sign we pick for $\sqrt{z_n}$; this is of no consequence for the value of z_{n+1}, t_{n+1}, but the sign matters for the computation of x_{n+1}, y_{n+1}. This is why we choose to impose $\mathrm{Re}(\sqrt{z_n}) \geq 0$; this choice is justified in the proof of Theorem 6.2.6.

6.2.2 Link with theta functions

More argument reduction

Recall (from Section 2.5.3) that argument reduction in genus 1 allows us to work under the conditions (2.5.11):

$$\tau \in \mathcal{F}, \quad |\mathrm{Re}(z)| \leq \frac{1}{2}, \quad 0 \leq \mathrm{Im}(z) \leq \frac{\mathrm{Im}(\tau)}{2}.$$

We go slightly further than these conditions in order to justify the forthcoming results.

We define the following domain:

$$\mathcal{A} = \left\{ \tau \in \mathcal{H}, z \in \mathbb{C} \mid 0 \leq \mathrm{Im}(z) \leq \frac{\mathrm{Im}(\tau)}{4} \right\}$$

This condition allows us to avoid $z = \frac{\tau+1}{2}$, which is a zero of $\theta(z,\tau)$, and hence a pole of quotients of the form $\frac{\theta_i}{\theta_0}$, which we consider in our algorithm much in the same way as [Dup11]. We prove Proposition 6.2.4 and Theorem 6.2.6 under the assumption $(z,\tau) \in \mathcal{A}$.

Proposition 6.2.2. *Let $(z,\tau) \in \mathcal{A}$ such that $|\mathrm{Re}(\tau)| \leq \frac{1}{2}$ and $|\mathrm{Re}(z)| \leq \frac{1}{8}$. Then $\left(\frac{z}{\tau}, \frac{-1}{\tau}\right) \in \mathcal{A}$.*

Proof. Write

$$|\mathrm{Im}\left(\frac{z}{\tau}\right)| = \frac{1}{|\tau|^2}|\mathrm{Im}(z)\,\mathrm{Re}(\tau) - \mathrm{Re}(z)\,\mathrm{Im}(\tau)| \leq \frac{\mathrm{Im}(\tau)}{|\tau|^2}\left(\frac{1}{4}\frac{1}{2} + \frac{1}{8}\right) = \frac{1}{4}\,\mathrm{Im}\left(\frac{-1}{\tau}\right). \qquad \square$$

Proposition 6.2.3. *Let z, τ such that conditions (2.5.11) are satisfied. Put $\tau' = \frac{\tau}{2}$ and $z' = \frac{z}{4}$. Then $(z',\tau') \in \mathcal{A}$ and $\left(\frac{z'}{\tau'}, \frac{-1}{\tau'}\right) \in \mathcal{A}$. Furthermore, one can compute $\theta_0(z,\tau)$, $\theta_1(z,\tau)$, $\theta_0(0,\tau)$, $\theta_1(0,\tau)$ from their equivalents at (z',τ') in $O(\mathcal{M}(P))$ operations.*

Proof. The first part of the proposition is simply Proposition 6.2.2. The second part is simply the application of the τ-duplication formulas (which we recalled in Section 6.2.1), as well as the z-duplication formulas, which we already mentioned in Chapter 2:

$$\begin{aligned}
\theta_0(2z,\tau)\theta_0^3(0,\tau) &= \theta_1^4(z,\tau) + \theta_2^4(z,\tau) \\
\theta_1(2z,\tau)\theta_1^3(0,\tau) &= \theta_0^4(z,\tau) - \theta_2^4(z,\tau) \\
\theta_2(2z,\tau)\theta_2^3(0,\tau) &= \theta_0^4(z,\tau) - \theta_1^4(z,\tau)
\end{aligned}$$

Using these formulas requires the knowledge of $\theta_2^2(z,\tau)$ and of $\theta_2^2(0,\tau)$; this could be computed using Jacobi's formula (Equation (2.5.6)) and the equation of the variety (Equation (2.5.7)), but we end up using a different trick in our final algorithm. $\square$

Good choices of sign and thetas

We now prove that, for the arguments we consider, the good choices of sign correspond exactly to values of θ:

Proposition 6.2.4. *For $(z,\tau) \in \mathcal{A}$ such that $\mathrm{Im}(\tau) \geq 0.335$ (in particular, for $\tau \in \mathcal{F}$) we have*

$$|\theta_0(z,\tau) - \theta_1(z,\tau)| < |\theta_0(z,\tau) + \theta_1(z,\tau)|,$$

which also proves that $\mathrm{Re}\left(\frac{\theta_1(z,\tau)}{\theta_0(z,\tau)}\right) > 0, \mathrm{Re}\left(\frac{\theta_1(0,\tau)}{\theta_0(0,\tau)}\right) > 0$.

Proof. We compute the following upper bounds using the triangle inequality and Lemma 5.1.2:

$$\begin{aligned}
|\theta_0(z,\tau)+\theta_1(z,\tau)-2| &\leq 2\sum_{n\geq 2, n \text{ even}} |q^{n^2}(w^{2n}+w^{-2n})| \\
&\leq 2\sum_{n\geq 2, n \text{ even}} |q|^{n^2}(1+|q|^{-n/2}) \\
&\leq 2\sum_{n\geq 1} |q|^{4n^2}(1+|q|^{-n}) \\
&\leq 2|q|^3+2|q|^4+\frac{2|q|^{16}}{1-|q|^{20}}+\frac{2|q|^{14}}{1-|q|^{19}} \\
|\theta_0(z,\tau)-\theta_1(z,\tau)| &\leq 2\sum_{n\geq 1, n \text{ odd}} |q|^{n^2}(1+|q|^{-n/2}) \\
&\leq 2|q|^{1/2}+2|q|+\frac{2|q|^9}{1-|q|^{16}}+\frac{2|q|^{7.5}}{1-|q|^{19}}
\end{aligned}$$

Numerical evaluation shows that

$$2|q|^{1/2}+2|q|+\frac{2|q|^9}{1-|q|^{16}}+\frac{2|q|^{7.5}}{1-|q|^{19}} < 2-\left(2|q|^3+2|q|^4+\frac{2|q|^{16}}{1-|q|^{20}}+\frac{2|q|^{14}}{1-|q|^{19}}\right)$$

for $\mathrm{Im}(\tau)\geq 0.335$, which proves the lemma. □

The result, and our proof, seems coarser and less subtle than in [Cox84]; however, note that Cox's methods cannot be applied here[6].

Note that the same method can be used to prove that

Proposition 6.2.5. *For any τ such that $\mathrm{Im}(\tau) > 0.251$ (in particular for $\tau \in \mathcal{F}$) we have $\mathrm{Re}\left(\frac{\theta_1(0,\tau)}{\theta_0(0,\tau)}\right) > 0$.*

Since $\{\tau \in \mathcal{H} \mid \mathrm{Im}(\tau) > 0.251\} \subset \mathcal{D}_1$, the domain for which we have a good choice of sign for F is strictly smaller than the one for which we have a good choice of sign for the AGM (Proposition 3.2.9). Figure 6.1 displays the different domains.

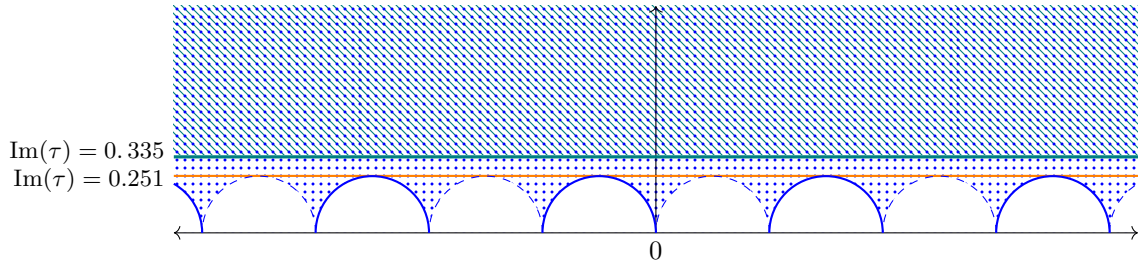

Figure 6.1: The green stripes represent a domain for which the choice of sign in F is good (assuming z is reduced). The blue dots represent Cox's domain for which the choice of sign is good in the AGM (i.e. for quotients of squares of theta-constants at 2τ). Above the orange line is a domain for which the choice of sign is good for quotients of theta-constants at τ; this domain is strictly contained in the one with blue dots.

[6] For instance, Cox uses the invariance of $\frac{\theta_1^2}{\theta_0^2}(0,\tau)$ under the action of the subgroup $\Gamma_2(4)$ (i.e. Proposition 3.2.13), which does not hold for $\frac{\theta_1}{\theta_0}(z,\tau)$.

We are now ready to prove:

Theorem 6.2.6. *Let (x_n, y_n, z_n, t_n) be the optimal F sequence starting with $\theta_0^2(z,\tau)$, $\theta_1^2(z,\tau)$, $\theta_0^2(0,\tau)$, $\theta_1^2(0,\tau)$. If $(z,\tau) \in \mathcal{A}$ and $\mathrm{Im}(\tau) \geq 0.335$, we have*

$$(x_n, y_n, z_n, t_n) = \left(\theta_0^2(z, 2^n\tau), \theta_1^2(z, 2^n\tau), \theta_0^2(0, 2^n\tau), \theta_1^2(0, 2^n\tau)\right)$$

Proof. This is true for $n = 0$; we prove the statement inductively. Suppose it is true for $n = k \geq 0$. For any τ, we have (using Lemma 5.1.2):

$$\theta_0(0,\tau) = 1 + 2q + c, \qquad |c| \leq \frac{2|q|^4}{1-|q|^5}$$

For any τ such that $\mathrm{Im}(\tau) \geq 0.335$, we have $2|q| \leq 0.676$ and $|c| \leq 0.027$; hence $\mathrm{Re}(\theta_0(0, 2^k\tau)) > 0$ for any k, which proves that $\sqrt{z_k} = \theta_0(0, 2^k\tau)$. Proposition 6.2.4 shows that $\mathrm{Re}\left(\frac{\theta_1(0,2^k\tau)}{\theta_0(0,2^k\tau)}\right) > 0$, and we also have $\mathrm{Re}\left(\frac{\sqrt{t_k}}{\sqrt{z_k}}\right) \geq 0$ since the choice of roots is good, hence $\sqrt{t_k} = \theta_1(0, 2^k\tau)$. Equation (2.5.4) then shows that $t_{k+1} = \theta_1^2(0, 2^{k+1}\tau)$ and $z_{k+1} = \theta_0^2(0, 2^{k+1}\tau)$.

Similarly, given that $(z,\tau) \in \mathcal{A}$ and using Lemma 5.1.2, we find:

$$|\theta_0(z,\tau) - 1| \leq |q|^{1/2} + |q| + |q|^3 + |q|^4 + |q|^{7/2} + |q|^9 + \frac{2|q|^{14}}{1-|q|^2} \tag{6.2.2}$$

For $\mathrm{Im}(\tau) \geq 0.335$, this is strictly smaller than 1; hence $\mathrm{Re}(\theta_0(z,\tau)) > 0$, which proves that $\sqrt{x_k} = \theta_0(z, 2^k\tau)$. Again, Proposition 6.2.4 proves that $\mathrm{Re}\left(\frac{\theta_1(z,2^k\tau)}{\theta_0(z,2^k\tau)}\right) > 0$, and since the choice of signs is good, $\mathrm{Re}\left(\frac{\sqrt{y_k}}{\sqrt{x_k}}\right) \geq 0$, necessarily $\sqrt{y_k} = \theta_1(z, 2^k\tau)$. This along with the τ-duplication formulas (Equations (2.5.4)) finishes the induction. □

Note that a consequence of this proposition is:

Proposition 6.2.7. *For $(z,\tau) \in \mathcal{A}$ and $\mathrm{Im}(\tau) \geq 0.335$, the optimal F sequence for $\theta_0^2(z,\tau)$, $\theta_1^2(z,\tau)$, $\theta_0^2(0,\tau)$, $\theta_1^2(0,\tau)$ converges quadratically, and*

$$F^\infty(\theta_0^2(z,\tau), \theta_1^2(z,\tau), \theta_0^2(0,\tau), \theta_1^2(0,\tau)) = (1,1,1,1).$$

6.2.3 A function with quasi-optimal time evaluation

The strategy of [Dup11] is to use an homogenization of the AGM to get a function $f_\tau : \mathbb{C} \to \mathbb{C}$, on which Newton's method can be applied. To generalize this, we homogenize the function F^∞, which gives a function from $\mathbb{C}^2$ to $\mathbb{C}^2$. We call this function $\mathfrak{G}$; this function is a major building block for the function we use to compute our two parameters z, τ using Newton's method.

Proposition 6.2.8. *Let $\lambda, \mu \in \mathbb{C}^*$. Let $((x_n, y_n, z_n, t_n))_{n\in\mathbb{N}}$ be the optimal F sequence for (x,y,z,t), and $((x'_n, y'_n, z'_n, t'_n))_{n\in\mathbb{N}}$ the optimal F sequence for $(\lambda x, \lambda y, \mu z, \mu t)$. Put $\lim_{n\to\infty} z_n = z_\infty$ and $\lim_{n\to\infty} z'_n = z'_\infty$. Then we have*

$$\mu = \frac{z'_\infty}{z_\infty}, \qquad \lambda = \frac{\left(\lim_{n\to\infty}\left(\frac{x'_n}{z'_\infty}\right)^{2^n}\right) \times z'_\infty}{\left(\lim_{n\to\infty}\left(\frac{x_n}{z_\infty}\right)^{2^n}\right) \times z_\infty}$$

Proof. We prove by induction that

$$x'_n = \epsilon_n \lambda^{1/2^n} \mu^{1-1/2^n} x_n, \quad y'_n = \epsilon_n \lambda^{1/2^n} \mu^{1-1/2^n} y_n, \qquad z'_n = \mu z_n, \quad t'_n = \mu t_n,$$

where $\mathrm{Re}(\lambda^{1/2^n}) \geq 0$, $\mathrm{Re}(\mu^{1-1/2^n}) \geq 0$, and ϵ_n is a 2^n-th root of unity. This is enough to prove the proposition above, since then

$$\lim_{n\to\infty} \left(\frac{x'_n}{z'_\infty}\right)^{2^n} = \lim_{n\to\infty} \lambda\mu^{2^n-1} \left(\frac{x_n}{z'_\infty}\right)^{2^n} = \frac{\lambda}{\mu} \lim_{n\to\infty} \left(\frac{x_n}{z_\infty}\right)^{2^n}.$$

Since this is true for $n = 0$, suppose this is true for $n = k$. We have

$$z'_{k+1} = \frac{z'_k + t'_k}{2} = \mu z_{k+1}.$$

As for t_{k+1}, we can write

$$\sqrt{z'_k} = \epsilon_z \sqrt{\mu}\sqrt{z_k}, \qquad \sqrt{t'_k} = \epsilon_t \sqrt{\mu}\sqrt{t_k}$$

where $\epsilon_z = \pm 1$ and $\epsilon_t = \pm 1$, and the square roots are taken with positive real part. This gives $\frac{\sqrt{t'_k}}{\sqrt{z'_k}} = \frac{\epsilon_t}{\epsilon_z}\frac{\sqrt{t_k}}{\sqrt{z_k}}$. Since the sequences we are considering are optimal, we have either $\mathrm{Re}\left(\frac{\sqrt{t'_k}}{\sqrt{z'_k}}\right) > 0$ and $\mathrm{Re}\left(\frac{\sqrt{t'_k}}{\sqrt{z'_k}}\right) > 0$, or $\mathrm{Im}\left(\frac{\sqrt{t'_k}}{\sqrt{z'_k}}\right) \geq 0$ and $\mathrm{Im}\left(\frac{\sqrt{t'_k}}{\sqrt{z'_k}}\right) \geq 0$ if the real parts are zero. In both cases, this proves that $\epsilon_z = \epsilon_t$. Hence

$$t'_{k+1} = \left(\epsilon_z \sqrt{\mu}\sqrt{z_k}\right)\left(\epsilon_z \sqrt{\mu}\sqrt{t_k}\right) = \mu t_{k+1}.$$

As for the other coordinates, we have

$$\sqrt{x'_k} = \epsilon_x \sqrt{\epsilon_k} \lambda^{1/2^{k+1}} \mu^{1/2-1/2^{k+1}} \sqrt{x_k}, \qquad \sqrt{y'_k} = \epsilon_y \sqrt{\epsilon_k} \lambda^{1/2^{k+1}} \mu^{1/2-1/2^{k+1}} \sqrt{y_k}$$

where the roots are taken with positive real part, and $\epsilon_x, \epsilon_y \in \{-1, 1\}$. Once again, since $\frac{\sqrt{y'_k}}{\sqrt{x'_k}} = \frac{\epsilon_y}{\epsilon_x}\frac{\sqrt{y_k}}{\sqrt{x_k}}$ and since the sequences are optimal, we have $\epsilon_x = \epsilon_y$; hence

$$\begin{aligned} x'_{k+1} &= \frac{\sqrt{x'_k}\sqrt{z'_k} + \sqrt{y'_k}\sqrt{t'_k}}{2} = \epsilon_{k+1} \lambda^{1/2^{k+1}} \mu^{1-1/2^{k+1}} \frac{\sqrt{x_k}\sqrt{z_k} + \sqrt{y_k}\sqrt{t_k}}{2} \\ &= \epsilon_{k+1} \lambda^{1/2^{k+1}} \mu^{1-1/2^{k+1}} x_{k+1} \end{aligned}$$

where $\epsilon_{k+1} = \epsilon_x \epsilon_z \sqrt{\epsilon_k}$ is indeed such that $\epsilon_{k+1}^{2^{k+1}} = 1$. This proves the proposition. □

The formulas can be simplified in our case since $\lim_{n\to\infty} \theta(z, 2^n\tau)^{2^n} = 1$ (Proposition 2.1.9). We thus define the function $\mathfrak{G}$ as follows:

Definition 6.2.9. The function $\mathfrak{G} : \mathbb{C}^4 \to \mathbb{C}^2$ is defined as

$$\mathfrak{G}(x, y, z, t) = \left(\left(\lim_{n\to\infty} \left(\frac{x_n}{z_\infty}\right)^{2^n}\right) \times z_\infty, z_\infty\right)$$

where $z_\infty = \lim_{n\to\infty} z_n$.

Proposition 6.2.10. *Let z, τ be as in the hypotheses of Theorem 6.2.6. Then for any $\lambda, \mu \in \mathbb{C}^*$ we have*

$$\mathfrak{G}\left(\lambda\theta_0^2(z,\tau), \lambda\theta_1^2(z,\tau), \mu\theta_0^2(0,\tau), \mu\theta_1^2(0,\tau)\right) = (\lambda, \mu).$$

This is a consequence of Proposition 6.2.8 and Theorem 6.2.6. For instance, we get

$$\mathfrak{G}\left(1, \frac{\theta_1^2(z,\tau)}{\theta_0^2(z,\tau)}, 1, \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right) = \left(\frac{1}{\theta_0^2(z,\tau)}, \frac{1}{\theta_0^2(0,\tau)}\right). \tag{6.2.3}$$

This is similar to Equation (6.1.1), and will play a similar role in the computation of $\theta(z,\tau)$.

6.2.4 Convergence

Let us start by showing that, contrary to the AGM and despite Proposition 6.2.7, an optimal F sequence does not always converge quadratically; for instance, the optimal F sequence for $(2,2,1,1)$ is $((2^{1/2^n}, 2^{1/2^n}, 1, 1))_{n\in\mathbb{N}}$, which does not converge quadratically. This is a big difference from the AGM, and this is why we are reluctant to call optimal F sequences a "generalization of the AGM". However, we now show that the sequence $(\lambda_n) = \left(\left(\frac{x_n}{z_\infty}\right)^{2^n} \times z_\infty\right)_{n\in\mathbb{N}}$ does converge quadratically, whence $\mathfrak{G}$ can be computed in $O(\mathcal{M}(P)\log P)$ bit operations.

Lemma 6.2.11. *Let $(x_0, y_0, z_0, t_0) \in \mathbb{C}^4$. Put $(x_{n+1}, y_{n+1}, z_{n+1}, t_{n+1}) = F(x_n, y_n, z_n, t_n)$ for any integer $n \in \mathbb{N}$, and suppose this is an optimal F sequence. Then there exists positive real constants c, C such $\forall n \geq 1$,*

$$c \leq |x_n|, |y_n|, |z_n|, |t_n| < C.$$

Proof. The upper bound result follows from a trivial induction using the equations defining F.

We now prove the existence of c. Recall that the choice of signs for the square roots are good at all steps, since we assume (x_n, y_n, z_n, t_n) is an optimal F sequence. Thus there exists $\alpha, \beta \in \mathbb{C}^*$ such that

$$\mathrm{Re}(x_1/\alpha) > 0, \quad \mathrm{Re}(y_1/\alpha) > 0, \qquad \mathrm{Re}(z_1/\beta) > 0, \quad \mathrm{Re}(t_1/\beta) > 0.$$

For instance, in most cases one can take $\alpha = x_1$ and $\beta = z_1$. Let us assume without loss of generality that $|\alpha| = |\beta| = 1$, and let $c = \min(\mathrm{Re}(x_1/\alpha), \mathrm{Re}(y_1/\alpha), \mathrm{Re}(z_1/\beta), \mathrm{Re}(t_1/\beta))$. For good choices of square roots, we have (see e.g. [Dup06, Lemme 7.3])

$$\mathrm{Re}\left(\sqrt{\frac{x_1}{\alpha}}\sqrt{\frac{z_1}{\beta}}\right) \geq \min\left(\mathrm{Re}\left(\frac{x_1}{\alpha}\right), \mathrm{Re}\left(\frac{z_1}{\beta}\right)\right) \geq c$$

and the same goes if one replaces x_1 by y_1 or z_1 by t_1. This implies from the definition that $|x_2| \geq \mathrm{Re}(x_2/\sqrt{\alpha\beta}) \geq c$, and the same goes for $|y_2|, |z_2|, |t_2|$. The result follows by induction, with $\sqrt{\alpha\beta}$, of modulus 1, playing the role of α at the next iteration. □

Lemma 6.2.12. *If the choice of square roots is good, we have*

$$|\sqrt{x_n} + \sqrt{y_n}| \geq \sqrt{2c} \qquad |\sqrt{z_n} + \sqrt{t_n}| \geq \sqrt{2c}$$

and hence

$$|\sqrt{x_n} - \sqrt{y_n}| \leq \frac{|x_n - y_n|}{\sqrt{2c}}, \qquad |\sqrt{z_n} - \sqrt{t_n}| \leq \frac{|z_n - t_n|}{\sqrt{2c}}$$

Proof. The parallelogram identity gives

$$\begin{aligned}|\sqrt{x_n}+\sqrt{y_n}|^2 &= 2|\sqrt{x_n}|^2+2|\sqrt{y_n}|^2-|\sqrt{x_n}-\sqrt{y_n}|^2\\ &\geq 2|\sqrt{x_n}|^2+2|\sqrt{y_n}|^2-|\sqrt{x_n}+\sqrt{y_n}|^2 \qquad \text{since choice of signs are good}\end{aligned}$$

and hence $|\sqrt{x_n}+\sqrt{y_n}|^2 \geq 2c$. The proof is the same for $|\sqrt{z_n}+\sqrt{t_n}|$. □

We now prove that $(\lambda_n) = \left(\left(\frac{x_n}{z_\infty}\right)^{2^n} \times z_\infty\right)_{n\in\mathbb{N}}$ converges quadratically, by proving that:

Theorem 6.2.13. *The sequence (λ_n) converges, to a limit λ. Furthermore, for P large enough, there exists a constant $c_1 > 0$, depending on $|z_0|, |t_0|, C, c$ and $|\lambda|$, such that, if k is the first integer such that $|z_k - t_k| \leq 2^{-P-k-c_1}$, then λ_{k+1} is an approximation of λ with absolute precision P bits.*

Proof. The point here is that once z_n and t_n are close enough, x_{n+1} and y_{n+1} are also close and the value of λ_n does not change much after that.

Recall Lemma 3.2.4 (i.e. [Dup11, Theorem 1]), which establishes that

$$|z_{n+k+1} - t_{n+k+1}| \leq A|z_{n+k} - t_{n+k}|^2$$

with $A = \frac{\pi}{8\min(|z_0|,|t_0|)}$. This proves that $(z_n - t_n)$ converges quadratically.

Take a $c_1 \geq 0$; for now, the value of c_1 is unimportant, but throughout the proof we will find sufficient conditions on c_1 for the result to hold. We then consider the first integer n for which $|z_n - t_n| \leq \eta$ with $\eta = 2^{-P-c_1-n}$; the existence of n is guaranteed by the quadratic convergence of $(z_n - t_n)$. Note that $A\eta \leq \frac{\pi}{8} \leq \frac{1}{2}$, since $\min(|z_0|,|t_0|) \geq 2^{-P}$; we will make use of this fact repeatedly throughout the proof.

We then have for all $k \geq 0$:

$$|z_{n+k} - t_{n+k}| \leq A^{2^k-1}\eta^{2^k}.$$

Furthermore, $|z_{n+1} - z_n| = \frac{1}{2}|z_n - t_n|$, so that

$$|z_\infty - z_{n+k}| \leq \frac{1}{2}\sum_{i=k}^{\infty} A^{2^i-1}\eta^{2^i}$$

and we have $|z_\infty - z_{n+k}| \leq \frac{1}{A}(A\eta)^{2^k}$, using the fact that $A\eta \leq \frac{1}{2}$. Finally, using Equation (6.2.1), one can write

$$\begin{aligned}|x_{n+k+1} - y_{n+k+1}| &\leq \frac{|\sqrt{x_n}-\sqrt{y_n}||\sqrt{z_n}-\sqrt{t_n}|}{2}\\ &\leq \sqrt{2C}\sqrt{|z_{n+k+1} - t_{n+k+1}|} \quad \text{since } z_{n+k+1} - t_{n+k+1} = \frac{(\sqrt{z_{n+k}}-\sqrt{t_{n+k}})^2}{2}\\ &\leq \sqrt{2AC}|z_{n+k} - t_{n+k}|.\end{aligned}$$

Now, define $q_n = \frac{(x_n/z_\infty)^2}{x_{n-1}/z_\infty}$, so that $\frac{\lambda_{n+1}}{\lambda_n} = q_n^{2^n}$. Note that if one makes the approximation $x_{n+k+1} \approx y_{n+k+1}$ and $z_{n+k+1} \approx t_{n+k+1} \approx z_\infty$, we have $x_{n+k+2} \approx \sqrt{x_{n+k+1}z_\infty}$ which gives

$q_{n+k+2} \approx 1$. We take a closer look at those approximations:

$$\begin{aligned}
|x_{n+k+2} - \sqrt{x_{n+k+1}}\sqrt{z_{n+k+1}}| &\leq \frac{|\sqrt{y_{n+k+1}} - \sqrt{x_{n+k+1}}||\sqrt{z_{n+k+1}} + \sqrt{t_{n+k+1}}|}{4} \\
&\quad + \frac{|\sqrt{y_{n+k+1}} + \sqrt{x_{n+k+1}}||\sqrt{z_{n+k+1}} - \sqrt{t_{n+k+1}}|}{4} \\
&\leq \frac{\sqrt{C}}{2}(|\sqrt{y_{n+k+1}} - \sqrt{x_{n+k+1}}| + |\sqrt{z_{n+k+1}} - \sqrt{t_{n+k+1}}|) \\
&\leq \frac{\sqrt{C}}{2}\left(\frac{\sqrt{2AC}|z_{n+k} - t_{n+k}|}{|\sqrt{x_{n+k+1}} + \sqrt{y_{n+k+1}}|} + \sqrt{2A}|z_{n+k+1} - t_{n+k+1}|\right) \\
&\leq B(A\eta)^{2^k}
\end{aligned}$$

for some constant B, where we used Lemma 6.2.12 to get a lower bound on the denominators. Put $\xi = x_{n+k+2} - \sqrt{x_{n+k+1}}\sqrt{z_{n+k+1}}$ for notational convenience. Then

$$\begin{aligned}
|q_{n+k+2} - 1| &= \frac{|(x_{n+k+2}/z_\infty)^2 - x_{n+k+1}/z_\infty|}{|x_{n+k+1}/z_\infty|} \\
&\leq \frac{|x_{n+k+2}^2 - x_{n+k+1}z_\infty|}{|x_{n+k+1}z_\infty|} \\
&\leq \frac{|\left(\xi + \sqrt{x_{n+k+1}}\sqrt{z_{n+k+1}}\right)^2 - x_{n+k+1}z_\infty|}{|x_{n+k+1}z_\infty|} \\
&\leq \frac{|\xi^2 + 2\xi\sqrt{x_{n+k+1}}\sqrt{z_{n+k+1}} + x_{n+k+1}(z_{n+k+1} - z_\infty)|}{|x_{n+k+1}z_\infty|} \\
&\leq \frac{|\xi|^2 + 2|\xi\sqrt{x_{n+k+1}}\sqrt{z_{n+k+1}}| + |x_{n+k+1}(z_{n+k+1} - z_\infty)|}{|x_{n+k+1}z_\infty|} \\
&\leq \frac{B^2(A\eta)^{2^{k+1}} + 2BC(A\eta)^{2^k} + \frac{C}{A}(A\eta)^{2^k}}{c^2} \\
&\leq B' \times (A\eta)^{2^k}
\end{aligned}$$

for some constant B'. This proves that (q_n) converges quadratically to 1.

Now, put $X_k = 2^{n+2}\log_2 q_{n+2} + \ldots + 2^{n+k}\log_2 q_{n+k}$. Assume that $B'A\eta \leq 1$, which is true for instance for $c_1 \geq \log_2(\frac{\pi}{8}B')$. We have

$$\begin{aligned}
|X_k| &\leq \sum_{i=0}^{k-2} 2^{n+i+2}|\log 1 + (q_{n+i+2} - 1)| \\
&\leq B' \sum_{i=0}^{k-2} 2^{n+i+2}\frac{(A\eta)^{2^i}}{1 - (A\eta)^{2^i}} \\
&\leq 8B'2^n \sum_{i=0}^{k-2} 2^i (A\eta)^{2^i}
\end{aligned}$$

using $|\log(1+x)| \leq \frac{|x|}{1-|x|}$, which is valid for any $|x| \leq 1$, and $A\eta \leq \frac{1}{2}$. The series converge (using e.g. d'Alembert's ratio test), which means that X_k is absolutely convergent; hence, (λ_n) converges.

Furthermore we can write

$$|X_k| \leq 8B'2^n \sum_{i=0}^{k-2} 2^i (A\eta)^{2^i} \leq 8B' \frac{2^n A\eta}{1-2A\eta} \leq 16AB'2^n\eta$$

assuming $A\eta \leq \frac{1}{4}$, which is true for instance for $c_1 \geq 1$. Assume that $16AB'2^n\eta \leq 1$ (which is true if $c_1 \geq \log_2(2\pi B')$); we then have $|X_k| \leq 1$ and

$$|q_{n+2}^{2^{n+2}} \cdots q_{n+k}^{2^{n+k}} - 1| = |\exp X_k - 1| \leq \frac{|X_k|}{1-|X_k|/2} \leq 32AB'2^n\eta.$$

This proves that

$$\begin{aligned} |\lambda - \lambda_{n+1}| &\leq |\lambda_{n+1}| 32AB'2^n\eta \\ &\leq 32AB'|\lambda_{n+1}|2^{-c_1} \times 2^{-P}. \end{aligned}$$

For $c_1 \geq \log_2(8\pi B')$ we have $|\lambda - \lambda_{n+1}| \leq \frac{|\lambda_{n+1}|}{2}$, which proves that $|\lambda_{n+1}| \leq 2|\lambda|$. Hence if we suppose $c_1 \geq \log_2(64AB'|\lambda|)$, we have that λ_{n+1} is an approximation of λ with P bits of absolute precision.

Collecting the conditions on c_1 throughout the proof, we see that the theorem holds for any c_1 such that

$$c_1 \geq \log_2 \max(64AB'|\lambda|, 8\pi B', 2)$$

which does not depend on P or n. □

Algorithm 12 Compute $\mathfrak{G}$.
Input: x, y, z, t with absolute precision P.
Output: $\mathfrak{G}(x, y, z, t)$ with absolute precision P.

1: Work at precision $\mathcal{P}$.
2: $n \leftarrow 0$
3: **while** $|z - t| \leq 2^{-P-n-c_1}$ **do**
4: $\quad n \leftarrow n + 1$
5: $\quad (x, y, z, t) \leftarrow F(x, y, z, t)$
6: **end while**
7: $(x, y, z, t) \leftarrow F(x, y, z, t)$
8: **return** $\left(\left(\frac{x}{z}\right)^{2^{n+1}} \times z, z\right)$

This gives an algorithm, Algorithm 12, to compute $\mathfrak{G}(x, y, z, t)$.

Proposition 6.2.14. *For any arguments $x, y, z, t \in \mathbb{C}$ with absolute precision P, the time complexity for computing $\mathfrak{G}(x, y, z, t)$ to absolute precision P is $O(\mathcal{M}(P) \log P)$.*

Proof. Theorem 3.2.6 (i.e. [Dup11, Theorem 12]) proves that, if $n = \max(\log|\log|z_0/t_0||, 1) + \log(P + c_1)$, a_n is an approximation of $\mathrm{AGM}\left(1, |\frac{z_0}{t_0}|\right)$ with relative precision P bits. This proves that at the end of the algorithm, $n = O(\log P)$; in fact, we have more precisely $n \leq \log_2 P + C''$ with C'' a constant independent of P. Finally, the next subsection proves that taking $\mathcal{P} = P + O(\log P)$, which proves the result. □

6.2.5 Number of bits lost

We use Theorem 0.3.3 in order to evaluate the precision lost when computing $\mathfrak{G}(x,y,z,t)$. First note that the upper and lower bounds on the terms of the sequence allow us to write

$$\begin{aligned}(\frac{1}{\sqrt{|z_n|}}+\frac{1}{\sqrt{|t_n|}})(\sqrt{|z_n|}+\sqrt{|t_n|}) &\leq b/2\\ (\frac{1}{\sqrt{|z_n|}}+\frac{1}{\sqrt{|t_n|}})(\sqrt{|x_n|}+\sqrt{|y_n|}) &\leq b\\ (\frac{1}{\sqrt{|x_n|}}+\frac{1}{\sqrt{|y_n|}})(\sqrt{|z_n|}+\sqrt{|t_n|}) &\leq b\end{aligned}$$

for some $b>1$; for instance, one can take $b=\max\left(1,4\sqrt{\frac{C}{c}}\right)$. We prove in Section 6.3.3 that, for any values of theta we consider in our final algorithm, the same value for c and C can be taken.

We first evaluate a bound on the error incurred when computing F using Equation (6.2.1). Using those formulas allows us to get error bounds that are identical for F_x and F_y on the one hand, and F_z and F_t on the other hand. For simplicity, we assume that the error on z and t is the same (which we denote k_z), as well as the error on x and y (which we denote k_x). This gives:

$$\begin{aligned}|\mathrm{Re}(F_x)-\mathrm{Re}(\tilde{F_x})| &\leq (1+(\frac{k_z}{\sqrt{|z|}}+\frac{k_z}{\sqrt{|t|}})(\sqrt{|x|}+\sqrt{|y|})+(\frac{k_x}{\sqrt{|x|}}+\frac{k_x}{\sqrt{|y|}})(\sqrt{|z|}+\sqrt{|t|}))2^{-P}\\ |\mathrm{Re}(F_z)-\mathrm{Re}(\tilde{F_z})| &\leq (1+2(\frac{k_z}{\sqrt{|z|}}+\frac{k_z}{\sqrt{|t|}})(\sqrt{|z|}+\sqrt{|t|}))2^{-P}\end{aligned}$$

We thus get the following recurrence relations when looking at what happens when applying F n times in a row:

$$k_x^{(n)}\leq 1+bk_z^{(n-1)}+bk_x^{(n-1)},\qquad k_z^{(n)}\leq 1+bk_z^{(n-1)}$$

The last equation can be rewritten as $k_z^{(n)}+\frac{1}{b-1}\leq b\left(k_z^{(n-1)}+\frac{1}{b-1}\right)$, which gives $k_z^{(n)}\leq b^n\left(k_z+\frac{1}{b-1}\right)$. The induction for x becomes $k_x^{(n)}\leq 1+(k_z+\frac{1}{b-1})b^n+bk_x^{(n-1)}$, which we solve:

$$k_x^{(n)}\leq(1+b+...+b^n)k_x+nb^n\left(k_z+\frac{1}{b-1}\right)\leq b^n\left(nk_z+\frac{b+n}{b-1}\right)$$

For $b>1$, we have for n large enough that $k^{(n)}\leq b^{2n}$, which ultimately means the number of bits lost when applying F n times in a row is bounded by $2n\log b+1$.

Finally we need to find the number of bits lost in the computation of $\left(\frac{x_n}{z_\infty}\right)^{2^n}$. Call E_k the error made after computing k squarings in a row; we have a recurrence relation:

$$E_{k+1}\leq 2+4E_k|x_n/z_\infty|^{2^k}$$

However, since (λ_n) converges, $|\lambda_n|\leq\rho$ for some constant ρ; furthermore, for any $k\leq n$, one has $|x_n/z_\infty|^{2^k}\leq 1+\frac{\rho}{z_\infty}$. Hence the recurrence becomes $E_{k+1}\leq 2+4\left(1+\frac{\rho}{z_\infty}\right)E_k$, which we solve to get

$$E_n\leq 2\frac{C'^{n+1}-1}{C'-1}\leq\frac{2}{C'-1}C'^{n+1}$$

with $C' = 4\left(1 + \frac{\rho}{z_\infty}\right)$. This means the number of bits lost after n successive squarings is at the most $(n+1)\log C' + 1 - \log(C'-1)$.

Overall, if we write that the final value of n in Algorithm 12 is bounded by $\log_2 P + C''$, we have:

Proposition 6.2.15. *The number of bits lost in the computation of $\mathfrak{G}$ is bounded by*

$$(2\log_2 b + \log C')(\log_2 P + C'') + \log C' + 2 - \log C' - 1,$$

where C', C'' and b are constants in P.

As a result, one should take $\mathcal{P} = P + O(\log P)$ in Algorithm 12: this gives a quasi-linear complexity for the evaluation of $\mathcal{G}$ with absolute precision P.

6.3 Fast computation of θ

We use a similar method as [Dup11], that is to say finding a function $\mathfrak{F}$ such that

$$\mathfrak{F}\left(\frac{\theta_1^2(z,\tau)}{\theta_0^2(z,\tau)}, \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right) = (z,\tau),$$

which can then be inverted using Newton's method. One can then compute $\theta(z,\tau)$ by, for instance, using Equation (6.2.3) and extracting a square root, determining the correct choice of sign by computing a low-precision (say, 10 bits) approximation of the value using the naive method; we use a different trick in our final algorithm (Algorithm 15). We build this function $\mathfrak{F}$ using $\mathfrak{G}$ as a building block.

6.3.1 Building $\mathfrak{F}$

Just as with the algorithm for theta-constants, we use formulas derived from the action of $SL_2(\mathbb{Z})$ on the values of θ in order to get multiplicative factors depending on our parameters; this will allow us to build a function which computes z, τ from the values $\theta_i(z,\tau)$.

Definition 6.3.1. We define the function $\mathfrak{F} : \mathbb{C}^2 \to \mathbb{C}^2$ as the result of Algorithm 13.

The forthcoming Proposition 6.3.2 is dedicated to explaining the calculations done by Algorithm 13.

Algorithm 13 Compute $\mathfrak{F}$.
Input: s, t with absolute precision P.
Output: $\mathfrak{F}(s,t)$ with absolute precision P.

1: $b \leftarrow \sqrt{1-t^2}$ ▷ Choose the root with positive real part [Cox84, Prop. 2.9]
2: $a \leftarrow \frac{1-st}{b}$
3: $(x,y) \leftarrow \mathfrak{G}(1,a,1,b)$
4: $(q_1,q_2) \leftarrow \mathfrak{G}(1,s,1,t)$
5: Return $\left(\sqrt{\log\left(\frac{q_2 x}{q_1 y}\right) \times \frac{q_2/y}{-2\pi}}, i\frac{q_2}{y}\right)$, choosing the sign of the square root so that it has positive imaginary part.

Proposition 6.3.2. *Let (z,τ) satisfying the conditions*

$$(z,\tau)\in\mathcal{A},\quad |\mathrm{Re}(\tau)|\leq\frac{1}{2},\quad |\mathrm{Re}(z)|\leq\frac{1}{8},\quad \mathrm{Im}(\tau)\geq 0.335,\quad \mathrm{Im}\left(\frac{-1}{\tau}\right)\geq 0.335\,.$$

Then

$$\mathfrak{F}\left(\frac{\theta_1^2(z,\tau)}{\theta_0^2(z,\tau)},\frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right)=(z,\tau)$$

Remark 6.3.3. Let (z,τ) satisfying conditions 2.5.11 and such that $1\leq|\tau|<2$. Then $\left(\frac{z}{4},\frac{\tau}{2}\right)$ satisfy the hypotheses of this theorem. We use this, along with Proposition 6.2.3, in our final algorithm.

Proof of Proposition 6.3.2. Equation (6.2.3) proves that $(q_1,q_2)=\left(\frac{1}{\theta_0(z,\tau)^2},\frac{1}{\theta_0(0,\tau)^2}\right)$. Furthermore, using Jacobi's formula (2.5.6) and the equation defining the variety (2.5.7), it is easy to see that $b=\frac{\theta_2(0,\tau)^2}{\theta_0(0,\tau)^2}$ and $a=\frac{\theta_2(z,\tau)^2}{\theta_0(z,\tau)^2}$.

The formulas in [Mum83, Table V, p.36] give

$$\left(\theta_0^2(z,\tau),\theta_2^2(z,\tau),\theta_0^2(0,\tau),\theta_2^2(0,\tau)\right)=\left(\lambda\theta_0^2\left(\frac{z}{\tau},\frac{-1}{\tau}\right),\lambda\theta_1^2\left(\frac{z}{\tau},\frac{-1}{\tau}\right),\mu\theta_0^2\left(0,\frac{-1}{\tau}\right),\mu\theta_1^2\left(0,\frac{-1}{\tau}\right)\right)$$

with $\lambda=\frac{e^{-2i\pi z^2/\tau}}{-i\tau}$, $\mu=\frac{1}{-i\tau}$. Proposition 6.2.2 shows we can apply Theorem 6.2.6 to $\frac{z}{\tau},\frac{-1}{\tau}$, which proves that

$$\mathfrak{G}\left(\theta_0^2(z,\tau),\theta_2^2(z,\tau),\theta_0^2(0,\tau),\theta_2^2(0,\tau)\right)=\left(\frac{e^{-2i\pi z^2/\tau}}{-i\tau},\frac{1}{-i\tau}\right).$$

By homogeneity, we get in Step 3 $(x,y)=\left(\frac{e^{-2i\pi z^2/\tau}}{-i\tau\theta_0(z,\tau)^2},\frac{1}{-i\tau\theta_0(0,\tau)^2}\right)$. We thus have

$$\left(\frac{x}{q_1},\frac{y}{q_2}\right)=\left(\frac{e^{-2i\pi z^2/\tau}}{-i\tau},\frac{1}{-i\tau}\right),$$

and Step 5 consists precisely in extracting (z,τ) from these. □

Remark 6.3.4. In genus 1, we can output either z,τ or λ,μ without affecting the rest of the algorithm, although returning z,τ requires one more logarithm computation. We choose to output z,τ for the clarity of the presentation.

This means that, starting from the knowledge of z and τ with precision P and a low-precision approximation of the quotients $\frac{\theta_1(z,\tau)}{\theta_0(z,\tau)}$ and $\frac{\theta_1(0,\tau)}{\theta_0(0,\tau)}$, one can compute those quotients with precision P using Newton's method. This is Algorithm 14. Note that we put $\mathcal{P}'$ a precision that is large enough to ensure that the final result is accurate up to 2^{-P}; we discuss the matter of precision loss later in this subsection.

We make a few remarks:

- Much in the same way as [ET14a], we find it preferable to use finite differences to compute the coefficients a_{11},a_{21},a_{22} of the Jacobian, as it does not require the computation of the derivative of $\mathfrak{F}$, which could be tedious. This requires the computation of $\mathfrak{F}(s+\epsilon,t),\mathfrak{F}(s,t+\epsilon)$ and $\mathfrak{F}(s,t)$.

Algorithm 14 Compute $\theta_0^2(z,\tau), \theta_1^2(z,\tau), \theta_0^2(0,\tau), \theta_1^2(0,\tau)$ with absolute precision P.
Input: (z,τ) with absolute precision P, satisfying the conditions of Proposition 6.3.2.

Compute $\theta_{0,1}^2(z,\tau)$, $\theta_{0,1}^2(0,\tau)$ with absolute precision P_0 using Algorithm 7.
$s \leftarrow \frac{\theta_1(z,\tau)^2}{\theta_0(z,\tau)^2}$, $t \leftarrow \frac{\theta_1(0,\tau)^2}{\theta_0(0,\tau)^2}$
$p \leftarrow P_0$
while $p \leq \mathcal{P}'$ **do**
    $p \leftarrow 2p$
    Compute $a_{11} = \frac{\partial \mathfrak{F}_x}{\partial x}(s,t)$, $a_{22} = \frac{\partial \mathfrak{F}_y}{\partial y}(s,t)$, $a_{12} = \frac{\partial \mathfrak{F}_x}{\partial y}(s,t)$ with precision p.
    $(s,t) \leftarrow (s,t) - (\mathfrak{F}(s,t) - (z,\tau)) \begin{pmatrix} a_{11} & a_{12} \\ 0 & a_{22} \end{pmatrix}^{-1}$
end while
$(a,b) \leftarrow \mathfrak{G}(1,s,1,t)$
$(a,b) \leftarrow (1/a, 1/b)$
$(s,t) \leftarrow (sa, tb)$
Return (a,s,b,t).

- The value of P_0 has to be large enough that Newton's method converges. We note that, in general, a lower bound on P_0 may depend on the arguments; for instance, [Dup11] experimentally finds $4.53\,\mathrm{Im}(\tau)$ to be a suitable lower bound for P_0 when computing theta-constants. However, we outline in the next section a better algorithm which only uses the present algorithm for z,τ within a compact set; hence, P_0 can be chosen to be a constant, and we use in practice $P_0 = 30000$.

We do not provide a full analysis for Algorithm 14; we outline in the next section a better algorithm, which uses this algorithm as a subroutine, and we will provide a full analysis at that time. We can however sketch the proof of the following result:

Proposition 6.3.5. *For (z,τ) with absolute precision P and satisfying the hypotheses of Proposition 6.3.2, Algorithm 14 computes $\theta_0^2(z,\tau), \theta_1^2(z,\tau), \theta_0^2(0,\tau), \theta_1^2(0,\tau)$ with absolute precision P in time bounded by $k\mathcal{M}(P)\log P$, where k is a constant in P but a function of z,τ.*

Proof (sketch). The computation of $\mathfrak{G}$ at precision p is done in time $O(\mathcal{M}(p)\log p)$ using Algorithm 12; this running time depends on z,τ, since it depends on the bounds C, c that one can write for $|x_n|, |y_n|, |z_n|, |t_n|$. Hence, the cost of evaluating $\mathfrak{F}$ at precision p is $O(\mathcal{M}(p)\log p)$ bit operations, and the fact that we double the working precision at every step means that the algorithm is as costly as the last iteration (see e.g. the proof of Theorem 0.3.9).

Furthermore, one should choose $\mathcal{P}'$ so that the final result is accurate with absolute precision P. This means compensating the loss of absolute precision incurred during the computation of $\mathfrak{F}$; in general, this only depends on $\mathrm{Im}(\tau)$ and linearly in $\log p$. In addition, recall (Corollary 0.3.8) that each step of Newton's iteration roughly doubles the accuracy of the result, i.e. gives approximations with absolute precision $2P - \delta$ from approximations with absolute precision P. In practice, we found $\delta = 4$ to be enough; but determining the number of bits lost at each step can be done in the same way as [ET14a, p. 19]: if $s^{(n-1)}$ and $s^{(n-2)}$ agree to k bits, and $s^{(n)}$ and $s^{(n-1)}$ agree to k' bits, the number of bits lost can be computed as $2k - k'$. In the end, working at precision $\mathcal{P}' = P + c\log P + d$, with c, d independent of P but functions of z,τ, is enough to compensate all the precision loss; this proves that the running time of this algorithm is asymptotically $O(\mathcal{M}(P)\log P)$. □

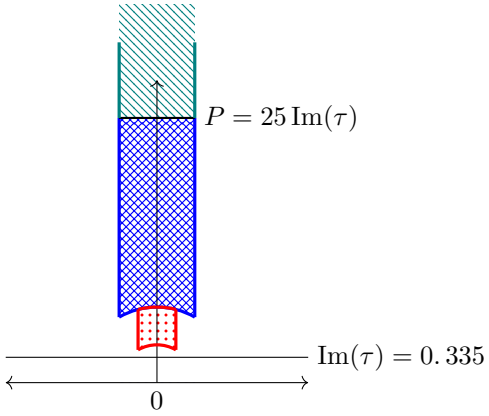

Figure 6.2: In Algorithm 15, τ can either be in the green zone (in which case the naive algorithm is used) or in the blue zone, in which case we divide τ by 2 until it is in the red zone, apply Algorithm 14, then use τ-duplication formulas to recover the result. Note that z, which is not represented here, also needs to be divided by a power of 2 to get $(z, \tau) \in \mathcal{A}$.

6.3.2 Computing $\theta(z, \tau)$ in uniform quasi-optimal time

We now show an algorithm with complexity bounded by $k\mathcal{M}(P)\log P$ bit operations, with k a constant independent in z and τ, which computes $\theta(z, \tau)$ for any (z, τ) satisfying conditions (2.5.11). We use the same strategy as [Dup11]; namely, we use the naive algorithm for large $\mathrm{Im}(\tau)$; and for smaller values, we divide τ by a power of 2 to get arguments within a compact set, and we also divide z by a power of 2 in order to have $(z, \tau) \in \mathcal{A}$. In fact, we divide τ, z by 2 until $1 \leq |\tau| < 2$, then use Note 6.3.3 and Proposition 6.2.3 to get $\left(\frac{z}{2^t}, \frac{\tau}{2^s}\right)$ in a compact set and belonging to $\mathcal{A}$. We then apply Algorithm 14 to these arguments; its running time is then uniform in the arguments since they are in a compact set. We then alternate between using Equation (2.5.4) to double the second argument and Equation (2.5.5) to double the first argument, until finally recovering $\theta(z, \tau)$. Figure 6.2 summarizes different domains τ can belong to in the algorithm.

A few notes on this algorithm:

- As we mentioned in Section 5.1, the computation of the complex exponentials in the naive algorithm must use the UniformExp subroutine (Prop 5.1.4) in order to get a uniform complexity for the naive algorithm.

- We note that, at several steps of the algorithm (e.g. Steps 9, 14, 16) we need to compute theta-constants from their square. The correct choice of signs is given by the proof of Theorem 6.2.6, which shows that $\mathrm{Re}(\theta_0(0, \tau)) \geq 0$ and $\mathrm{Re}(\theta_1(0, \tau)) \geq 0$; and furthermore, since $\mathrm{Re}(q^{1/4}) \geq |q|^{1/4}\cos(\pi/8)$, we also have $\mathrm{Re}(\theta_2(0, \tau)) \geq 0$.

- Taking $\tau_2 = \tau_1/2$ allows us to use the τ-duplication formulas for θ_2 (Equation (2.5.4)) in step 8, instead of using Equation (2.5.6) and Equation (2.5.7) to recover θ_2; this is more efficient and loses fewer bits.

- The knowledge of $\theta_2^2(2^{i-2}z_1, 2^i\tau)$ is enough for the z-duplication formulas of step 17, and it can be computed directly from θ_0 and θ_1 using the τ-duplication formulas for θ_2.

Algorithm 15 Compute $\theta_0(z,\tau), \theta_1(z,\tau), \theta_2(z,\tau), \theta_0(0,\tau), \theta_1(0,\tau), \theta_2(0,\tau)$ with absolute precision P.
Input: $\tau \in \mathcal{F}$ and z reduced, with absolute precision P.

1: **if** $P \leq 25\,\mathrm{Im}(\tau)$ **then**
2: Compute $\theta_{0,1,2}(z,\tau), \theta_{0,1,2}(0,\tau)$ with precision P using the naive method (Algorithm 7 and Algorithm 8).
3: **else**
4: Take $s \in \mathbb{N}$ such that $1 \leq |\tau|/2^s < 2$
5: Put $\tau_1 = \frac{\tau}{2^s}$ and $z_1 = \frac{z}{2^s}$, so that $\mathrm{Im}(z_1) \leq \mathrm{Im}(\tau_1)/2$.
6: Put $z_2 = z_1/4$ and $\tau_2 = \tau_1/2$.
7: Compute approximations of absolute precision $\mathcal{P}$ of $\theta_0^2(z_2,\tau_2), \theta_1^2(z_2,\tau_2)$, $\theta_0^2(0,\tau_2)$, and $\theta_1^2(0,\tau_2)$ using Algorithm 14.
8: Compute $\theta_0^2(z_2,\tau_1), \theta_1^2(z_2,\tau_1), \theta_0^2(0,\tau_1), \theta_1^2(0,\tau_1), \theta_2^2(z_2,\tau_1), \theta_2^2(0,\tau_1)$ using the τ-duplication formulas (Equation (2.5.4)).
9: Compute $\theta_{0,1,2}(0,\tau_1)$.
10: Compute $\theta_{0,1}(z_1/2,\tau_1)$ using the z-duplication formulas (Equation (2.5.5)).
11: **for** i = 1 .. s **do**
12: Compute $\theta_0^2(0,2^i\tau_1), \theta_1^2(0,2^i\tau_1)$ using the AGM.
13: Compute $\theta_0^2(2^{i-2}z_1,2^i\tau_1), \theta_1^2(2^{i-2}z_1,2^i\tau_1)$ using the τ-duplication formulas (Equation (2.5.4)).
14: If $i = s$, compute also $\theta_2^2(0,2^i\tau_1)$ using the τ-duplication formulas, then $\theta_2(0,2^i\tau_1)$ by taking the square root.
15: Compute $\theta_2^2(2^{i-2}z_1,2^i\tau_1)$ using the τ-duplication formulas.
16: Compute $\theta_{0,1}(0,2^i\tau_1)$.
17: Compute $\theta_0(2^{i-1}z_1,2^i\tau_1), \theta_1(2^{i-1}z_1,2^i\tau_1)$ using the z-duplication formulas.
18: **end for**
19: Compute $\theta_2^2(2^{s-1}z_1,2^s\tau_1)$ using Equation (2.5.7).
20: Compute $\theta_{0,1,2}(z,\tau)$ using the z-duplication formulas.
21: **end if**

- Computing $\theta_3(z,\tau)$ is also possible; one should use a partial summation if $P \leq 25\,\mathrm{Im}(\tau)$. In the other case, since all the z-duplication formulas for $\theta_3(z,\tau)$ involve a division by $\theta_2(0,\tau)$ [Mum83, p.22], it is just as efficient to simply use Equation (5.1.2) after Step 20, then extract the square root. The square root extraction loses $O(\mathrm{Im}(\tau)) = O(P)$ bits, and this also gives a quasi-optimal time algorithm.

6.3.3 Proving the correctness of the algorithm

This section is devoted to proving the following theorem:

Theorem 6.3.6. *For any τ, z with absolute precision P satisfying conditions* (2.5.11)*, Algorithm 15 with $\mathcal{P} = 2P$ computes $\theta_{0,1,2}(z,\tau), \theta_{0,1,2}(0,\tau)$ with absolute precision P in $c\mathcal{M}(P)\log P$ bit operations, where c is independent of z, τ.*

As we discussed in Section 2, this also gives an algorithm that computes $\theta(z,\tau)$ for any $(z,\tau) \in \mathbb{C} \times \mathcal{H}$; one simply needs to reduce τ to $\tau' \in \mathcal{F}$, then reduce z to z', and deduce $\theta(z,\tau)$ from $\theta(z',\tau')$ using Proposition 2.1.7 and Theorem2.4.4. This causes a loss of absolute precision which depends on z and τ, and this algorithm is no longer uniform.

We need to perform an analysis of the number of bits lost by the algorithm; once again, we use Theorem 0.3.3. For each step, we proceed as follows: we assume that the error on all the quantities is bounded by k (i.e. if $\hat{a}$ is our approximation of $a \in \mathbb{C}$, we have $|a - \hat{a}| < k$), and we determine a factor x such that the error on the quantities we get after the computation is bounded by xk. We then declare the number of bits lost in this step to be $\log_2 x$; this gives a very loose upper bound, but simplifies the process.

Finally, we also need to prove that the hypotheses made in Sections 6.2.4 and 6.2.5 are satisfied in Step 7 of the algorithm. This is necessary to prove that the sequence (λ_n) we consider is quadratically convergent, and that the number of bits lost is only $O(\log P)$. We prove this in Section 6.3.3, which then completes the proof that the running time is indeed uniform and quasi-optimal.

Invertibility of the Jacobian

Newton's method can only be applied if the Jacobian of the function we invert (here, $\mathfrak{F}$) is invertible. The following proposition establishes this:

Proposition 6.3.7. *The Jacobian of $\mathfrak{F}$ at $\left(\frac{\theta_1^2(z,\tau)}{\theta_0^2(z,\tau)}, \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right)$ is of the form $\begin{pmatrix} a & b \\ 0 & c \end{pmatrix}$ with $a, c \neq 0$.*

This proves that the Jacobian is invertible on a neighbourhood of $\left(\frac{\theta_1^2(z,\tau)}{\theta_0^2(z,\tau)}, \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right)$. In practice, numerical experiments indicate that for any z, τ in the compact we consider, the Jacobian of the system seems invertible on a ball of radius 10^{-P_0} with $P_0 = 30000$, and that this base precision is enough to make Newton's method converge.

Proof. We have

$$\begin{aligned} a &= \frac{\partial \mathfrak{F}_1}{\partial z_1}\left(\frac{\theta_1^2(z,\tau)}{\theta_0^2(z,\tau)}, \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right) \\ c &= \frac{\partial \mathfrak{F}_2}{\partial z_2}\left(\frac{\theta_1^2(z,\tau)}{\theta_0^2(z,\tau)}, \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right). \end{aligned}$$

Given the expression of the function F, where only the third and fourth argument influence the third and fourth coordinate, we have that $c = \frac{\partial f_\tau}{\partial z}\left(\frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right)$ where f_τ is the function defined in Section 6.1.1 such that $f_\tau\left(\frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right) = 0$. We then have $c \neq 0$ by [Dup06, Prop. 4.3, p. 102].

We prove $a \neq 0$ using the chain rule: define $u : (z,\tau) \mapsto \left(\frac{\theta_1^2(z,\tau)}{\theta_0^2(z,\tau)}, \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right)$. Then $(\mathfrak{F}\circ u)(z,\tau) = (z,\tau)$ and we thus have

$$\begin{aligned}
1 = \frac{\partial(\mathfrak{F}\circ u)_1}{\partial z}(z,\tau) &= a \times \frac{\partial u_1}{\partial z}(z,\tau) + \frac{\partial \mathfrak{F}_1}{\partial u_2}\left(\frac{\theta_1^2(z,\tau)}{\theta_0^2(z,\tau)}, \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right) \times \frac{\partial u_2}{\partial z}(z,\tau) \\
&= a \times \frac{\partial u_1}{\partial z}(z,\tau) \qquad \text{since } \frac{\partial u_2}{\partial z}(z,\tau) = 0. \\
\text{Hence} \qquad a &= \frac{\partial u_1}{\partial z}(z,\tau)^{-1} \\
&= 2\frac{\theta_1(z,\tau)}{\theta_0(z,\tau)}\left(\frac{\theta_1(z,\tau)}{\theta_0(z,\tau)}\right)' \\
&= 2\frac{\theta_1(z,\tau)}{\theta_0^3(z,\tau)}(\theta_1'(z,\tau)\theta_0(z,\tau) - \theta_0'(z,\tau)\theta_1(z,\tau)) \\
&= 2\frac{\theta_1(z,\tau)}{\theta_0^3(z,\tau)}\pi\theta_2^2(0,\tau)\theta_2(z,\tau)\theta_3(z,\tau) \neq 0
\end{aligned}$$

the last equality deriving from Formula 10 in [Web21, Section 23]. This finishes the proof. □

Naive algorithm

Since $P \leq 25\,\mathrm{Im}(\tau)$, we have $\sqrt{\frac{P+2}{\pi \log_2 e\,\mathrm{Im}(\tau)}} + 1 \leq 4$, and hence at most 4 terms are needed to compute an approximation with absolute precision P. Theorem 5.1.5 shows that at most 9 bits are lost during these computations.

The cost of the algorithm is then asymptotically dominated by the cost of computing π, q and v_1 with precision $\mathcal{P} = P + 10$. As we highlighted in Proposition 5.1.4 and Note 5.1.6, the cost of these steps is $O(\mathcal{M}(P)\log P)$ independently of z and τ; hence, the complexity of this step is quasi-linear and uniform in z, τ.

Square root extraction

Steps 9, 16 and 14 of Algorithm 15 require extracting square roots, which multiply the error by $\frac{\sqrt{2}}{\sqrt{|z|}}$ (cf. Theorem 0.3.3). We prove in the next subsection that $|\theta_{0,1}(0, 2^i\tau_1)| \geq 0.859$ for $i = [1 \ldots s]$. Hence, each extraction of square root loses at most 4 bits: step 9 loses 4 bits, and step 16 loses $4s \leq 4\log P$ bits.

Step 14 loses more bits since $\theta_2(0,\tau)$ is smaller; indeed, $|\theta_2(0,\tau)| \sim |q|^{1/4}$. This means the number of bits lost during this step is bounded by $\frac{\log|q|}{8} = \frac{\pi}{8}\,\mathrm{Im}(\tau)\log_2 e$.

Duplication formulas

Algorithm 15 uses both τ-duplication formulas and z-duplication formulas, and we need to analyse how many bits are lost for each application of those formulas.

The τ-duplication formulas are nothing more than applying F to $\theta_{0,1}^2(z,\tau)$ and $\theta_{0,1}^2(0,\tau)$. However, the analysis here is simpler than in section 6.2.5, because we do not need to compute the square roots of $\theta_{0,1}(z,\tau)$, since they are directly given by step 17. Hence we just need

to account for the error of the additions, subtractions and multiplications in Equation (6.2.1); since all the quantities are bounded, this means each step loses a constant number g of bits (our analysis, which is tedious and unilluminating, shows that $g \leq 10.48$). In the end, the τ-duplication formulas account for the loss of $g \times s \leq g \log P$ bits of precision.

As for the z-duplication formulas, we need to perform several analyses. Looking at the z-duplication formulas (Equation (2.5.5)), one needs to evaluate the fourth power of theta functions, then add them; then evaluate the third power of theta-constants, then perform a division. Computing the error using the formulas from Theorem 0.3.3 is rather straightforward when one has bounds on those quantities, which are given by the following theorem:

Lemma 6.3.8. *Assume* $\mathrm{Im}(\tau) > \sqrt{3}/2$. *Then*

$$0.859 \leq |\theta_{0,1}(0,\tau)| \leq 1.141, \qquad |\theta_2(0,\tau)| \leq 1.018$$

We also have:

- *Suppose that* $0 \leq \mathrm{Im}(z) \leq \frac{\mathrm{Im}(\tau)}{8}$. *Then* $|w|^{-2} \leq e^{\pi \,\mathrm{Im}(\tau)/4}$ *and*

$$0.8038 \leq |\theta_{0,1}(z,\tau)| \leq 1.1962 \qquad |\theta_2(z,\tau)| \leq 1.228$$

- *Suppose that* $0 \leq \mathrm{Im}(z) \leq \frac{\mathrm{Im}(\tau)}{4}$. *Then* $|w|^{-2} \leq e^{\pi \,\mathrm{Im}(\tau)/2}$ *and*

$$0.6772 \leq |\theta_{0,1}(z,\tau)| \leq 1.3228 \qquad |\theta_2(z,\tau)| \leq 1.543$$

Proof. The bounds on the theta-constants come from [Dup11, p. 5], who proves $|\theta_{0,1}(0,\tau)| \leq \frac{2|q|}{1-|q|}$. The techniques are the same as the proof of Proposition 6.2.4 or Theorem 5.1.1, that is to say we combine the triangle inequality with Lemma 5.1.2. This gives in the first case

$$\begin{aligned} |\theta_{0,1}(z,\tau) - 1| &\leq |q|^{3/4} + |q| + |q|^{7/2} + |q|^4 + |q|^{8.25} + |q|^9 + \frac{|q|^{15}}{1-|q|} \\ &\leq 0.1962 \text{ since } \mathrm{Im}(\tau) \geq \sqrt{3}/2 \\ |\theta_2(z,\tau) - q^{1/4}(w + w^{-1})| &\leq \frac{|q|^{15/8}}{1-|q|^{3/8}} \leq 0.009 \end{aligned}$$

so $|\theta_2(z,\tau)| \leq |q|^{1/4}(|w| + |w|^{-1}) + 0.009 \leq 1.228$. In the second case:

$$|\theta_{0,1}(z,\tau) - 1| \leq |q|^{1/2} + |q| + \frac{|q|^3}{1-|q|} \leq 0.3228$$

$$|\theta_2(z,\tau) - q^{1/4}(w + w^{-1})| \leq |q|^{5/4} + |q|^{9/4} + \ldots \leq \frac{|q|^{5/4}}{1-|q|} \leq 0.0357. \qquad \square$$

Combining these bounds with formulas from Theorem 0.3.3 gives the bounds

$$\begin{aligned} |\theta_0(z_1/2,\tau_1) - \theta_0(\tilde{z_1/2},\tau_1)| &\leq (20051 + 1819k_{\theta_1(z_2,\tau_1)} + 1967k_{\theta_2(z_2,\tau_1)} + 33516k_{\theta_0(0,\tau_1)})2^{-P} \\ |\theta_1(z_1/2,\tau_1) - \theta_1(\tilde{z_1/2},\tau_1)| &\leq (20051 + 1819k_{\theta_0(z_2,\tau_1)} + 1967k_{\theta_2(z_2,\tau_1)} + 33516k_{\theta_1(0,\tau_1)})2^{-P} \end{aligned}$$

which means losing at most 16 more bits of precision.

Step 19 causes a larger number of lost bits. We use Equation (2.5.7) instead of the third z-duplication formula, because dividing by $\theta_2(0,\tau)^2$ loses fewer bits than dividing by $\theta_2(0,\tau)^3$,

and we only need the knowledge of $\theta_2^2(2^{s-1}z, 2^s\tau)$ for the next step anyway. This amounts to computing:

$$\theta_2^2(z,\tau) = \frac{\theta_0^2(z,\tau)\theta_0^2(0,\tau) - \theta_1^2(z,\tau)\theta_1^2(0,\tau)}{\theta_2^2(0,\tau)}$$

Computing the numerator multiplies the error by a factor at most 60, and the norm of this numerator is bounded by 4.557; we then get from Theorem 0.3.3 that the error is bounded by $\frac{m}{|\theta_2(0,\tau)|^8} \sim m|q|^{-2}$, with $m \leq 1600$. In the end, we lose at most $2\pi(\log_2 e)\operatorname{Im}(\tau) + 11$ bits.

Finally, we also lose many bits during the last application of the z-duplication formulas in step 20, since the formula for $\theta_2(z,\tau)$ requires dividing by $\theta_2(0,\tau)^3$. The error is thus multiplied by $|q|^{-3}$ up to a constant factor; this means a loss of $3\pi \operatorname{Im}(\tau)\log_2 e$ bits, up to a constant.

In the end, we see that the number of lost bits is bounded by $(2\pi + \pi/4 + 3\pi)\operatorname{Im}(\tau)\ \log_2 e + c\log P + d$; given that $P \geq 25\operatorname{Im}(\tau)$, and that $5.25\pi\log_2 e \leq 23.8$, the number of bits lost is thus less than P.

Proof of correctness

Proposition 6.3.9. *For all $\tau \in \mathcal{F}$ and z reduced, with absolute precision P, Algorithm 15 computes $\theta_0(z,\tau), \theta_1(z,\tau), \theta_2(z,\tau), \theta_0(0,\tau), \theta_1(0,\tau), \theta_2(0,\tau)$.*

Proof. The trickiest part is checking that $\mathfrak{F}$ returns the right result, as in Proposition 6.3.2. We have that $1 \leq |\tau_1| < 2$ and $\operatorname{Im}(z_1) \leq \frac{\operatorname{Im}(\tau_1)}{2}$; conditions (2.5.11) and the condition $(z_2,\tau_2) \in \mathcal{A}$ hold, since we have:

$$|\operatorname{Re}(\tau_2)| \leq 1/2^{s+2} \leq 1/4, \qquad 0.335 \leq \frac{\sqrt{3}}{4} \leq \operatorname{Im}(\tau_2) \leq 1,$$
$$|\operatorname{Re}(z_2)| \leq 1/2^{s+3} \leq 1/8, \qquad 0 \leq \operatorname{Im}(z_2) \leq \frac{\operatorname{Im}(\tau_2)}{4}.$$

This means the choices of signs are always good, and hence our result is indeed theta functions and theta-constants.

Finally, collecting the number of lost bits in the previous subsections show that

$$\mathcal{P} = 1.952P + c\log P + d \leq 2P$$

is enough to get a result which is accurate to absolute precision P; this also means that we indeed never have an error k bigger than $2^{(2P)/2}$, which is necessary to apply Theorem 0.3.3. □

Proof of quasi-optimal running time

It remains to prove that the complexity is the right one. If $P \geq 25\operatorname{Im}(\tau)$, $\log_2 P > \log_2 \operatorname{Im}(\tau) + 4.7$, which means $s \leq \log P$ and the cost of Steps 11 to 18 is $O(\mathcal{M}(\mathcal{P})\log P)$; this running time is uniform in z, τ, as highlighted in Note 5.1.6.

We then need prove that there is a $C > 1$ such that, for all z_2, τ_2 that we consider,

$$\frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)} \leq C, \frac{\theta_2^2(0,\tau)}{\theta_0^2(0,\tau)} \leq C, \frac{\theta_1^2(z,\tau)}{\theta_0^2(z,\tau)} \leq C, \frac{\theta_2^2(z,\tau)}{\theta_0^2(z,\tau)} \leq C.$$

This is a direct consequence of the fact that z_2, τ_2 are within a compact set that does not contain any zero of $\theta(z,\tau)$; hence one can write (non-zero) lower and upper bounds for any of

the values of theta. One can be more precise using the same reasoning as in Lemma 6.3.8: since $\sqrt{3}/4 \leq \mathrm{Im}(\tau_2) \leq 1$:

$$|\theta_{0,1}(z_2,\tau_2) - 1| \leq |q|^{1/2} + |q| + \frac{|q|^3}{1-|q|} \leq 0.7859, \qquad |\theta_2(z_2,\tau_2)| \leq 1 + |q|^{1/4} + \frac{|q|^{5/4}}{1-|q|} \leq 1.958.$$

This gives $C \leq 83.64$ and $c \geq \frac{0.042^2}{1.7859^2} \simeq \frac{1}{1808}$. Furthermore, with a careful analysis, one can prove that $c_1 = 55$ is enough in Theorem 6.2.13.

The existence of c and C proves that (λ_n) is quadratically convergent. The last thing left to analyze is the number of bits lost in the computation, which is given by Section 6.2.5. We note that the fact that z_2, τ_2 are within a compact shows that the constants b, C', C'' exist and are independent of z, τ. (For implementation purposes, we were able to determine experimentally that $C'' = 2.05$; furthermore a rough analysis, which we do not detail here, showed that $\log C' \leq 11.65$ and $\log_2 b \leq 15.52$.) This makes the running time of Step 7 only dependent on P, which was the point of the uniform algorithm. In particular, the number of bits lost during the computation of $\mathfrak{G}$ or in $\mathfrak{F}$ can be written as $c_1 \log P + c_2$, with c_1, c_2 constants independent in z, τ. Hence, the number of bits that are lost in the whole of Step 7 is

$$\sum_{i=1}^{n} \delta + c_1 \log(p/2^i) + c_2 \leq G \log P + H$$

for some constants G, H, since the number n of steps in Newton's method is $O(\log P)$.

This means the computations in step 7 should be carried out at precision $\mathcal{P}' = 2P + G \log P + H$, so that the result is accurate to $2P$ bits. This gives a running time of $O(\mathcal{M}(P) \log P)$, independently of z and τ. All the other steps cost no more than $O(\mathcal{M}(P))$ bit operations, which indeed gives us a running time of $O(\mathcal{M}(P) \log P)$.

6.4 Implementation results

An implementation using the GNU MPC library [EGTZ12] for arithmetic on multiprecision complex numbers was developed; we compared our algorithm to our own implementation of Algorithm 7 using MPC[7]. The code is distributed under the GNU General public license, version 3 or any later version (GPLv3+); it is available at the address

`http://www.hlabrande.fr/pubs/fastthetas.tar.gz`

We compared those implementations to MAGMA's implementation of the computation of $\theta(z,\tau)$ (function `Theta`). Each of those implementations computed $\theta(z,\tau)$ at different precisions for $z = 0.123456789 + 0.123456789i$ and $\tau = 0.23456789 + 1.23456789i$; the computations took place on a computer with an Intel Core i5-4570 processor. The results are presented in Figure 6.3 and Table 6.1.

Our figures show that our algorithm outperforms Magma even for computations at relatively low precision[8] (i.e. 1000 decimal digits). Furthermore, it is faster than our implementation of Algorithm 7 for precisions greater than 260 000 decimal digits. Hence, a combined algorithm which uses the naive method for precisions smaller than 260 000 decimal digits, and our method for larger precision, will yield the best algorithm, and outperform Magma in all cases, as shown in Table 6.1.

[7] The naive algorithm which only computes $\theta(z,\tau)$ is only 5% faster; furthermore since Algorithm 15 computes these 4 values, it is fair to compare it to Algorithm 7.

[8] This is even though Magma only returns $\theta_0(z,\tau)$, when our algorithm returns more values.

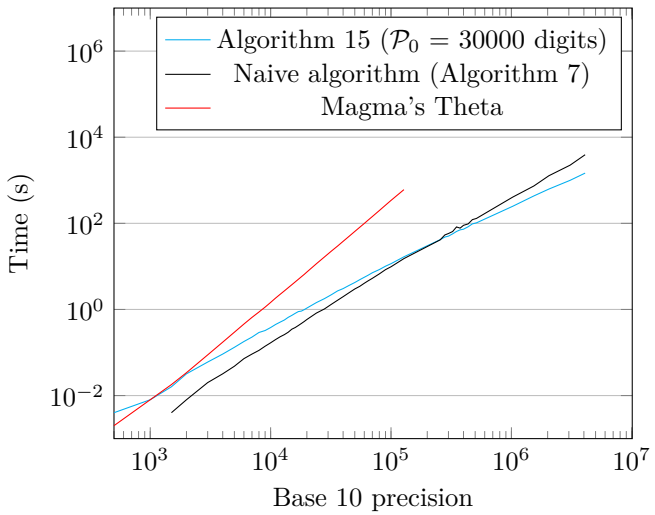

Figure 6.3: Timing results

Note that Algorithm 7 returns 4 values (the fundamental theta functions and theta-constants), while our quasi-linear time algorithm (Algorithm 15) returns in fact 6 values. Our implementation of a naive algorithm which computes all 6 values (i.e. combining Algorithm 7 and Algorithm 8) is slower than our quasi-linear time algorithm for $P = 30\ 000$ decimal digits; this is the reason why we chose to take $P_0 = 30\ 000$ in Algorithm 14, i.e. start the Newton iterations on an approximation of the quotients with precision 30 000 digits.

6.5 Batching computations of theta for different z

To finish this chapter, we consider the problem of batching the computation of θ at different z – that is to say, computing $\theta(z_k, \tau)$ for $z_1, ..., z_n \in \mathbb{C}$ faster than with simply n evaluations. We find speedups by a constant factor for both the naive algorithm and the quasi-linear algorithm; note that there does not seem to be a speedup if the value of τ is also different. These results are useful in a variety of settings, including in our case Chapter 9. However, we did not implement those speedups, since our main application for them (Algorithm 25) has asymptotic complexity which is worse than the state of the art anyway, which means that constant-factor speedups are not that interesting.

Remark 6.5.1. The naive algorithm in genus 2 (Section 5.2.2), as well as the generalization of the fast algorithm in genus 2, can also be sped up when batching computations of θ at different z, using techniques that are very similar to the ones presented here. We leave their precise description and implementation to future work.

Throughout this section, we assume that the z_k are reduced, i.e. $|\mathrm{Re}(z_k)| \leq \frac{1}{2}, 0 \leq \mathrm{Im}(z_k) < \mathrm{Im}(\tau)$.

6.5.1 Batch naive algorithm

We propose Algorithm 16, a variant on Algorithm 7 for batch computations of θ.

Prec (digits)	This work (Algorithm 15)	Naive (Algorithm 7)	Magma
1000	0.008	0	0.01120
2000	0.032	0.008	0.05520
4000	0.092	0.032	0.2790
8000	0.292	0.112	1.316
16000	0.812	0.380	7.692
32000	2.184	1.32	38.99
64000	6.152	4.55	158.2
128000	16.6	15.1	922.2
256000	41.8	41.3	
512000	102	130	
1024000	248	405	

Table 6.1: Timings (in s) of different methods

Algorithm 16 Compute $(\theta_{0,1}(z_k,\tau))_{k=1,\ldots,n}$ and $\theta_{0,1}(0,\tau)$, with absolute precision P.
Input: z_k, τ with absolute precision P, satisfying conditions (2.5.11).

Work with precision $\mathcal{P}$.
$B \leftarrow \left\lceil \sqrt{\frac{P+2}{\pi \operatorname{Im}(\tau) \log_2(e)}} \right\rceil + 1$
$\theta_{0,z} \leftarrow 1, \theta_{1,z} \leftarrow 1, \theta_{0,0} \leftarrow 1, \theta_{1,0} \leftarrow 1$
$q \leftarrow \text{UniformExp}(i\pi\tau), \quad q_1 \leftarrow q, \quad q_2 \leftarrow q$
for $k = 1, .., n$ **do**
    $v_1^{(k)} \leftarrow \text{UniformExp}(2i\pi(z_k + \tau/2)) + \text{UniformExp}(-2i\pi(z_k - \tau/2)), v^{(k)} \leftarrow v_1^{(k)}, v'^{(k)} \leftarrow 2$
end for
for $n = 1..B$ **do**
    For $k = 1, .., n$: $\theta_{0,z}^{(k)} \leftarrow \theta_{0,z}^{(k)} + v^{(k)}, \quad \theta_{1,z}^{(k)} \leftarrow \theta_{1,z}^{(k)} + (-1)^n \times v^{(k)}$
    $\theta_{0,0} \leftarrow \theta_{0,0} + 2q_2, \quad \theta_{1,0} \leftarrow \theta_{1,0} + (-1)^n \times 2q_2$
    $q_2 \leftarrow q_2 \times (q_1)^2 \times q; \quad q_1 \leftarrow q_1 \times q$
    For $k = 1, .., n$: temp $\leftarrow v^{(k)}, \quad v^{(k)} \leftarrow \left(q_1^2 \times v_1^{(k)}\right) \times v^{(k)} - q_1^4 \times v'^{(k)} \quad v'^{(k)} \leftarrow$ temp
end for

This algorithm computes n values of θ using $n+1$ exponentiations, n divisions, and $(3n+5)B$ multiplications. Algorithm 7 requires 2 exponentiations, 1 division and $8B$ multiplications. Hence, asymptotically, Algorithm 16 is about $\frac{8}{3} = 2.3$ times faster than running n times Algorithm 7.

6.5.2 Batch quasi-linear algorithm

We can also modify the quasi-optimal algorithm presented in this chapter to achieve a constant-factor speedup in the evaluation of n values of $\theta(z_k, \tau)$. Again, the strategy is to avoid recomputing theta-constants several times; instead, we compute the theta-constants once, using Algorithm 11 for instance, and store them for future use. Incidentally, this makes the problem only depend on z.

We can modify Algorithm 12 so that it computes a two-variable function $\mathfrak{G}_2$ such that $\mathfrak{G}_2(\lambda\theta_0(z,\tau)^2, \lambda\theta_1(z,\tau)^2) = \lambda$. Assuming the $\theta_i(0, 2^k\tau)$ are also computed once and stored, and putting N the number of steps (recall that $N = O(\log P)$), the computation of this function requires $2N$ multiplications and $2N$ square root computations, compared to $4N$ square roots and $3N$ multiplications. This represents an asymptotic factor of 1.75 if we assume that square roots and multiplications have the same cost; however, in practice, since square root extractions cost a few multiplications (experiments using GNU MPC showed a constant greater than 100), this asymptotic factor is probably closer to 2.

We can then modify Algorithm 13 so that it computes a function $\mathfrak{F}_2$ so that $\mathfrak{F}_2\left(\frac{\theta_1(z,\tau)^2}{\theta_0(z,\tau)^2}\right) = z$; such a modification does not yield important savings. We then use Newton's method on this function to compute $\frac{\theta_1(z,\tau)^2}{\theta_0(z,\tau)^2}$, and ultimately the values of θ. We compute the derivative of $\mathfrak{F}_2$ using finite differences, which requires 2 evaluations of $\mathfrak{F}_2$; this saves 33% over the algorithm we presented in this chapter, which requires 3 evaluations of $\mathfrak{F}$ to compute the 3 coefficients of the Jacobian.

Finally, since we assume we precomputed the $\theta_i(0, 2^k\tau)$, a few steps in Algorithm 15 are simplified. Taking a closer look at Steps 8 to 20, we find that this variant costs $6s + 11$ multiplications and 2 square root extractions, while Algorithm 15 costs $7s + 12$ multiplications and $2s+8$ square root extractions. If we assume multiplications and square roots have the same cost, this is a factor 1.5; however, in practice, the asymptotic factor is likely to be much bigger: if a square root costs 100 multiplications, this is a factor 34.

Putting it all together, we get a factor 2.3 in the execution of the naive algorithm (either for $P \leq 25\,\mathrm{Im}(\tau)$ or in the initial approximation in Algorithm 14), a factor 3 in Algorithm 14, and a large factor in Steps 8 to 20 of Algorithm 15. The resulting speedup factor is difficult to estimate; however since Algorithm 14 represents the bulk of the computation, one can expect a factor 3 to be saved.

Chapter 7

Computing the Riemann theta function in genus 2 and above

This chapter is dedicated to attempting to generalize the algorithm outlined in the previous chapter to higher genera. We succeed in making the algorithm fully explicit in the case of genus 2 theta functions, i.e. $\theta(z,\tau)$ with $z \in \mathbb{C}^2, \tau \in \mathcal{H}_2$: the running time is also of $O(\mathcal{M}(P)\log P)$ bit operations. We suppose that z and τ are reduced as explained in Section 2.3 and Section 2.4.

The generalization of the algorithm in genus g is not complete yet. We managed to generalize the function $\mathfrak{G}$, the function giving something depending only on z, τ from the quotients of theta functions and theta-constants; this function can be conjecturally computed with precision P in $O(2^g\mathcal{M}(P)\log P)$, although we managed to prove the result in genus 2. However, constructing $\mathfrak{F}$ from $\mathfrak{G}$ so that we get a function with an invertible Jacobian is harder: we managed to construct such a function in genus 2, although the invertibility of the Jacobian is conjectural; in genus g, we did not manage to provide an explicit definition which would have a good chance of being invertible. Hence, the algorithm is completely described in genus 2, and has even been implemented, and we also managed to generalize the definition of $\mathfrak{G}$ to genus g, but we did not manage to define $\mathfrak{F}$ in genus g.

As in the previous chapter, we start by studying the case of genus 2 theta-constants, which was first outlined in [Dup06] and used in [ET14a] for the computation of modular polynomials of record size. We then present our algorithm in genus 2, along with timing results, and discuss genus g; the presentation of these results is taken from an article with Emmanuel Thomé [LT16] which was published at the Twelfth Algorithmic Number Theory Symposium (ANTS-XII).

7.1 Preamble: genus 2 theta-constants

We discuss here the computation of theta-constants in genus 2 in $O(\mathcal{M}(P)\log P)$, using the algorithm outlined in [Dup06] and the refinements shown in [ET14a].

Recall first the naive algorithm for genus 2 theta functions, which we outlined in Algorithm 9. As we showed in Section 5.2.1, the cost of this algorithm is $O\left(\mathcal{M}(P)\frac{P}{\mathrm{Im}(\tau)}\right)$. A variant of this algorithm which evaluates only the theta-constants can be written easily; this gives an algorithm with a smaller constant in the O, but does not change its asymptotic running time. Note that the algorithms of [ET14a, Prop. 3] and [Dup06, Algorithme 15.] use an analysis which does not highlight the dependency in the size of $\mathrm{Im}(\tau_{1,1})$, and hence get an asymptotic complexity of

$O(\mathcal{M}(P)P)$; however, [ET14a, Prop. 3] shows how to use recurrence relations to compute the terms of the sum efficiently, an idea we borrowed for our naive algorithms.

Recall from Section 3.4 the definition of the Borchardt mean in genus 2:

Definition 7.1.1. The *Borchardt mean of four complex numbers*, denoted $\mathcal{B}_2(a_0, a_1, a_2, a_3)$, is defined as the common limit of the sequences $(a_0^{(n)}, a_1^{(n)}, a_2^{(n)}, a_3^{(n)})_{n\in\mathbb{N}}$ defined by

$$a_0^{(n+1)} = \frac{a_0^{(n)} + a_1^{(n)} + a_2^{(n)} + a_3^{(n)}}{4} \qquad a_1^{(n+1)} = \frac{\alpha_0\alpha_1 + \alpha_2\alpha_3}{2}$$
$$a_2^{(n+1)} = \frac{\alpha_0\alpha_2 + \alpha_1\alpha_3}{2} \qquad a_3^{(n+1)} = \frac{\alpha_0\alpha_3 + \alpha_1\alpha_2}{2}$$

where $\alpha_0^2 = a_0^{(n)}, \alpha_1^2 = a_1^{(n)}, \alpha_2^2 = a_2^{(n)}, \alpha_3^2 = a_3^{(n)}$, and where the choice of signs are *good*, i.e.

$$|\alpha_i - \alpha_j| < |\alpha_i + \alpha_j|$$

We previously noted that the Borchardt mean of 4 numbers cannot always be defined, as it is impossible to define a good choice of signs for some quadruplets (see Figure 3.2).

Recall also the τ-duplication formulas for the fundamental thetas:

$$\theta_{[0;b]}(0, 2\tau)^2 = \frac{1}{4} \sum_{\beta\in\frac{1}{2}\mathbb{Z}^2/\mathbb{Z}^2} \theta_{[0;\beta]}(0,\tau)\theta_{[0;b+\beta]}(0,\tau)$$

As we discussed in Section 3.4.3, this formula shows that $\left(\theta_0^2(0, 2^k\tau), \ldots, \theta_3^2(0, 2^k\tau)\right)_{k\in\mathbb{N}}$ is a Borchardt sequence.

The link between theta-constants and the Borchardt mean is the equivalent in genus 2 of the link between theta-constants and the AGM in genus 1. The genus 2 algorithm to compute theta-constants in quasi-linear time is then very similar to the one described in Section 6.1: the goal here is then to construct a function $\mathfrak{F}$ which returns τ from the knowledge of quotients of squares of theta-constants, and we do so once again by applying the Borchardt mean to different quotients. However, this requires first to study the domain for which good choices of square roots always coincide with the values $\theta_i(0, 2^k\tau)$, which is (Definition 3.4.8):

$$\mathcal{U}_2 = \{\tau \in \mathcal{H}_2 \mid \mathcal{B}_2(\theta_0^2(0,\tau), \ldots, \theta_3^2(0,\tau)) = 1\}.$$

We can determine sufficient conditions on τ so that $\tau \in \mathcal{U}_2$. We were able to prove a result which is stronger than [Dup06, Prop. 6.1]:

Proposition 7.1.2. *Let $\tau \in \mathcal{H}_2$ such that $\mathrm{Im}(\tau)$ is Minkowski-reduced and $\mathrm{Im}(\tau_1) \geq 0.594$. Then $\mathrm{Re}(\theta_i(0,\tau)) \geq 0$ for $i = 0, 1, 2, 3$, and furthermore $|\theta_0(0,\tau) - \theta_j(0,\tau)| < |\theta_0(0,\tau) + \theta_j(0,\tau)|$ for $j = 1, 2, 3$.*

Hence $\{\tau \in \mathcal{H}_2 \mid \mathrm{Im}(\tau)$ is Minkowski-reduced, $\mathrm{Im}(\tau_{1,1}) \geq 0.594\} \subset \mathcal{U}_2$. In particular, we have $\mathcal{F}_2' \subset \mathcal{U}_2$ and $\mathcal{F}_2 \subset \mathcal{U}_2$.

Proof. The first statement can be proven with the same proof as [Dup06, Prop. 6.1]. The proof of the other statement is tedious and unillumating; we do not reproduce it fully here. The method is similar to that of Proposition 6.2.4. We first take advantage of cancellations from the series defining the theta-constants; for instance:

$$\theta_0(0,\tau) + \theta_1(0,\tau) = 2 \sum_{\substack{n=(n_1,n_2)\\ n_1\in\mathbb{Z}, n_2 \text{ even}}} e^{i\pi^t n\tau n}, \qquad \theta_0(0,\tau) - \theta_1(0,\tau) = 2 \sum_{\substack{n=(n_1,n_2)\\ n_1\in\mathbb{Z}, n_2 \text{ odd}}} e^{i\pi^t n\tau n}$$

We then attempt to compute a bound on the quotient $\frac{|\theta_0(0,\tau)-\theta_1(0,\tau)|}{2-|\theta_0(0,\tau)+\theta_1(0,\tau)-2|}$, using the triangle inequality to bound each of the absolute value; when the quotient has an absolute value smaller than 1, the choice of signs is good. To get a rather fine bound, we compute explicitly terms with small (e.g. smaller than 4) value for n_1 or n_2 and bound them individually, then use Lemma 5.1.2 to bound the remainders. □

This result thus proves that, for τ verifying the hypotheses of Proposition 7.1.2, and using homogeneity of $\mathcal{B}_2$,

$$\mathcal{B}_2\left(1, \frac{\theta_1(0,\tau)^2}{\theta_0(0,\tau)^2}, \frac{\theta_2(0,\tau)^2}{\theta_0(0,\tau)^2}, \frac{\theta_3(0,\tau)^2}{\theta_0(0,\tau)^2}\right) = \frac{1}{\theta_0(0,\tau)^2}$$

which is a property very similar to the situation in genus 1 (Note 3.2.10).

We then take a look at the Borchardt mean of other quotients and determine some which allow the computation of coefficients of τ:

Proposition 7.1.3. *Define* $\mathfrak{J} = \begin{pmatrix} 0 & -I_2 \\ I_2 & 0 \end{pmatrix}$ *and* $\mathfrak{M}_i = \begin{pmatrix} I_2 & m_i \\ 0 & I_2 \end{pmatrix}$, *with* $m_1 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$, $m_2 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$, $m_3 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ *(as in* [Dup06, Chapter 6]*). Then*

$$\begin{aligned}
\theta_0\left(0, (\mathfrak{J}\mathfrak{M}_1)^2\cdot\tau\right)^2 &= -\tau_1\theta_8(z,\tau)^2, \\
\theta_0\left(0, (\mathfrak{J}\mathfrak{M}_2)^2\cdot\tau\right)^2 &= -\tau_2\theta_4(z,\tau)^2, \\
\theta_0\left(0, (\mathfrak{J}\mathfrak{M}_3)^2\cdot\tau\right)^2 &= (\tau_3^2-\tau_1\tau_2)\theta_0(z,\tau)^2.
\end{aligned}$$

This result is a direct consequence of Theorem 2.4.4[9]. Using Theorem 2.4.4 again for the numerators, and provided once again that good choices of sign correspond to values of theta, we have [Dup06, p.197]

$$\begin{aligned}
\mathcal{B}_2\left(1, \frac{\theta_9^2(0,\tau)}{\theta_8^2(0,\tau)}, \frac{\theta_0^2(0,\tau)}{\theta_8^2(0,\tau)}, \frac{\theta_1^2(0,\tau)}{\theta_8^2(0,\tau)}\right) &= \mathcal{B}_2\left(1, \frac{\theta_1^2(0,(\mathfrak{J}\mathfrak{M}_1)^2\cdot\tau)}{\theta_0^2(0,(\mathfrak{J}\mathfrak{M}_1)^2\cdot\tau)}, \frac{\theta_2^2(0,(\mathfrak{J}\mathfrak{M}_1)^2\cdot\tau)}{\theta_0^2(0,(\mathfrak{J}\mathfrak{M}_1)^2\cdot\tau)}, \frac{\theta_3^2(0,(\mathfrak{J}\mathfrak{M}_1)^2\cdot\tau)}{\theta_0^2(0,(\mathfrak{J}\mathfrak{M}_1)^2\cdot\tau)}\right) \\
&= \frac{1}{-\tau_1\theta_8^2(0,\tau)} \qquad (7.1.1) \\
\mathcal{B}_2\left(1, \frac{\theta_0^2(0,\tau)}{\theta_4^2(0,\tau)}, \frac{\theta_6^2(0,\tau)}{\theta_4^2(0,\tau)}, \frac{\theta_2^2(0,\tau)}{\theta_4^2(0,\tau)}\right) &= \mathcal{B}_2\left(1, \frac{\theta_1^2(0,(\mathfrak{J}\mathfrak{M}_2)^2\cdot\tau)}{\theta_0^2(0,(\mathfrak{J}\mathfrak{M}_2)^2\cdot\tau)}, \frac{\theta_2^2(0,(\mathfrak{J}\mathfrak{M}_2)^2\cdot\tau)}{\theta_0^2(0,(\mathfrak{J}\mathfrak{M}_2)^2\cdot\tau)}, \frac{\theta_3^2(0,(\mathfrak{J}\mathfrak{M}_2)^2\cdot\tau)}{\theta_0^2(0,(\mathfrak{J}\mathfrak{M}_2)^2\cdot\tau)}\right) \\
&= \frac{1}{-\tau_2\theta_4^2(0,\tau)} \qquad (7.1.2) \\
\mathcal{B}_2\left(1, \frac{\theta_8^2(0,\tau)}{\theta_0^2(0,\tau)}, \frac{\theta_4^2(0,\tau)}{\theta_0^2(0,\tau)}, \frac{\theta_{12}^2(0,\tau)}{\theta_0^2(0,\tau)}\right) &= \mathcal{B}_2\left(1, \frac{\theta_1^2(0,(\mathfrak{J}\mathfrak{M}_3)^2\cdot\tau)}{\theta_0^2(0,(\mathfrak{J}\mathfrak{M}_3)^2\cdot\tau)}, \frac{\theta_2^2(0,(\mathfrak{J}\mathfrak{M}_3)^2\cdot\tau)}{\theta_0^2(0,(\mathfrak{J}\mathfrak{M}_3)^2\cdot\tau)}, \frac{\theta_3^2(0,(\mathfrak{J}\mathfrak{M}_3)^2\cdot\tau)}{\theta_0^2(0,(\mathfrak{J}\mathfrak{M}_3)^2\cdot\tau)}\right) \\
&= \frac{1}{(\tau_3^2-\tau_1\tau_2)\theta_0^2(0,\tau)} \qquad (7.1.3)
\end{aligned}$$

However, note that the hypotheses of Proposition 7.1.2 are not necessarily satisfied by $(\mathfrak{J}\mathfrak{M}_1)^2\cdot\tau$, $(\mathfrak{J}\mathfrak{M}_2)^2\cdot\tau$ or $(\mathfrak{J}\mathfrak{M}_3)^2\cdot\tau$; in fact, these period matrices do not even necessarily have a Minkowski-reduced imaginary part. We were unable to find conditions on τ so that either of the propositions could be applied to these matrices, and were also unable to determine the shape of the set described by the $(\mathfrak{J}\mathfrak{M}_i)^2\cdot\tau$ for, say, $\tau\in\mathcal{F}_2'$ (which would have allowed us to attempt to prove

[9]Note that the result appears different from [Dup06] only because the tables for $\mathfrak{M}_1$ and $\mathfrak{M}_2$ (page 146) have been switched by mistake; and it differs from [ET14a] because their $\mathfrak{M}_2$ is our $\mathfrak{M}_3$, and vice-versa.

a more general version of Proposition 7.1.2). One could also think of looking at the sums one gets when considering $\theta_i(0,\tau) \pm \theta_j(0,\tau)$, but no obvious cancellations seem to occur, and then the triangle inequality does not seem like the right tool; we did not pursue this line of thought.

Note that in practice, all the quotients in Equations (7.1.1) to (7.1.3) appear to have a positive real part, and probably also satisfy the condition of the good choice of signs. Hence, the following conjecture is proposed in [Dup06, Conjecture 9.1] and [ET14a, Conjecture 9]:

Conjecture 7.1.4. *For $k = 1, 2, 3$ and $\tau \in \mathcal{F}_2$, we have $(\mathfrak{JM}_1)^2 \cdot \tau \in \mathcal{U}_2$.*

We show in Section 7.2 how we can circumvent the "good choice vs correct value of θ" problem using low-precision approximations.

Proposition 7.1.3 and Equations (7.1.1) to (7.1.3) allow the computation of the τ_i from the knowledge of the θ_i^2 for $0 \leq i \leq 15$; this is similar to what Equation (6.1.2) gave us in genus 1. Hence, we can use these equations to construct the function $\mathfrak{F}$, which returs τ from quotients of fundamental theta-constants; we will then invert $\mathfrak{F}$ using Newton's method. The algorithm is as follows: starting with $\frac{\theta^2_{1,2,3}(0,\tau)}{\theta_0^2(0,\tau)}$, we recover the individual fundamental theta-constants, by computing $\theta_0^2(0,\tau)$ as the inverse of the Borchardt mean of the quotients. We then need to compute the theta-constants $\theta_4, \theta_6, \theta_8, \theta_9, \theta_{12}$; once this is done, we apply Equations (7.1.1) to (7.1.3) to recover τ. Computing these theta-constants can be done in two ways: using some explicit formulas, as in [Dup06, Section 6.4], or by directly using the τ-duplication formulas to compute all the theta-constants at 2τ, then account for the factor 2 in the final result. We prefer to use the latter approach, as it is generalizable to any genus. We summarize this in Algorithm 17.

Algorithm 17 Compute τ from $\frac{\theta^2_{1,2,3}(0,\tau)}{\theta_0^2(0,\tau)}$, assuming Conjecture 7.1.4.

Input: $\frac{\theta^2_{1,2,3}(0,\tau)}{\theta_0^2(0,\tau)}$ with absolute precision P.
Output: τ with absolute precision P.

1: $t_0 \leftarrow \mathcal{B}_2\left(1, \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}, \frac{\theta_2^2(0,\tau)}{\theta_0^2(0,\tau)}, \frac{\theta_3^2(0,\tau)}{\theta_0^2(0,\tau)}\right)$.
2: $t_0 \leftarrow \frac{1}{t_0}$.
3: $t_1 \leftarrow t_0 \times \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}; t_2 \leftarrow t_0 \times \frac{\theta_2^2(0,\tau)}{\theta_0^2(0,\tau)}; t_3 \leftarrow t_0 \times \frac{\theta_3^2(0,\tau)}{\theta_0^2(0,\tau)}$.
4: $t_i \leftarrow \sqrt{t_i}$, choosing the square root corresponding to $\theta_i(0,\tau)$.
5: Apply the τ-duplication formula (Equation (2.2.1)) to get $t_i \leftarrow \theta_i^2(0,2\tau)$ for $i = 0 \dots 16$.
6: $\tau_1 = \mathcal{B}_2\left(1, \frac{t_9}{t_8}, \frac{t_0}{t_8}, \frac{t_1}{t_8}\right), \tau_2 = \mathcal{B}_2\left(1, \frac{t_0}{t_4}, \frac{t_6}{t_4}, \frac{t_2}{t_4}\right), \tau_3 = \mathcal{B}_2\left(1, \frac{t_8}{t_0}, \frac{t_4}{t_0}, \frac{t_{12}}{t_0}\right)$.
7: $\tau_1 \leftarrow \frac{-t_8}{2\tau_1}, \tau_2 \leftarrow \frac{-t_4}{2\tau_2}, \tau_3 = \sqrt{\frac{t_0}{4\tau_3} + \tau_1\tau_2}$.
8: **return** $\begin{pmatrix} \tau_1 & \tau_3 \\ \tau_3 & \tau_1 \end{pmatrix}$.

Computing theta-constants is then done by applying Newton's method to the function defined by Algorithm 17, which is a function $\mathbb{C}^3 \to \mathbb{C}^3$. The Jacobian appears to be invertible in practice, but no proof of this fact has been found; both [Dup06] and [ET14a] assume it is the case. One can compute the Jacobian either with a quadratically convergent algorithm which works conjecturally [Dup06], or with finite differences [ET14a]; both have the same quasi-optimal asymptotic running time, but [ET14a] finds that the first approach is 45% more expensive in practice. We refer to [Dup06] or [ET14a] for more details, and to [ET14b] for an implementation of this algorithm (i.e. Algorithm 17 and the three-dimensional Newton scheme which allows to compute theta-constants in quasi-optimal time).

The implementation of this algorithm [ET14b] was used in [ET14a] to compute the Igusa class polynomials corresponding to Jacobians of genus 2 curves, where the theta-constants are linked to the roots of such a polynomial, which is then reconstructed from the roots. The authors implemented the algorithm using the multiprecision arithmetic library GNU MPC [EGTZ12]; their implementation results show that this approach beats the naive algorithm for precisions as low as a few thousand bits. Using this algorithm, they were able to compute a class polynomial of record size, corresponding to a class number of 20016.

7.2 The algorithm

7.2.1 The function F

Recall the *τ-duplication formulas* (Prop. 2.2.1): for all $a, b \in \frac{1}{2}\mathbb{Z}^g/\mathbb{Z}^g$,

$$\theta_{[a;b]}(z,\tau)^2 = \frac{1}{2^g} \sum_{\beta \in \frac{1}{2}\mathbb{Z}^g/\mathbb{Z}^g} e^{-4i\pi\, {}^t a\beta} \theta_{[0;b+\beta]}\left(z, \frac{\tau}{2}\right) \theta_{[0;\beta]}\left(0, \frac{\tau}{2}\right). \tag{7.2.1}$$

This prompts us to define the following function, crafted so that Proposition 7.2.1 holds, as a direct consequence of the τ-duplication formula. The definition below is ambiguous (because of square roots), an issue we deal with in what follows.

$$\begin{aligned} F : \mathbb{C}^8 &\to \mathbb{C}^8 \\ a_{0\ldots3}, b_{0\ldots3} &\mapsto \left(\frac{\sqrt{a_0}\sqrt{b_0} + \sqrt{a_1}\sqrt{b_1} + \sqrt{a_2}\sqrt{b_2} + \sqrt{a_3}\sqrt{b_3}}{4}, \frac{\sqrt{a_0}\sqrt{b_1} + \sqrt{a_1}\sqrt{b_0} + \sqrt{a_2}\sqrt{b_3} + \sqrt{a_3}\sqrt{b_2}}{4}, \right. \\ &\quad \frac{\sqrt{a_0}\sqrt{b_2} + \sqrt{a_1}\sqrt{b_3} + \sqrt{a_2}\sqrt{b_0} + \sqrt{a_3}\sqrt{b_1}}{4}, \frac{\sqrt{a_0}\sqrt{b_3} + \sqrt{a_1}\sqrt{b_2} + \sqrt{a_2}\sqrt{b_1} + \sqrt{a_3}\sqrt{b_0}}{4}, \\ &\quad \left. \frac{b_0 + b_1 + b_2 + b_3}{4}, \frac{2\sqrt{b_0}\sqrt{b_1} + 2\sqrt{b_2}\sqrt{b_3}}{4}, \frac{2\sqrt{b_0}\sqrt{b_2} + 2\sqrt{b_1}\sqrt{b_3}}{4}, \frac{2\sqrt{b_0}\sqrt{b_3} + 2\sqrt{b_1}\sqrt{b_2}}{4} \right). \end{aligned}$$

Proposition 7.2.1. *For a suitable choice of square roots, we have*

$$F\left(\theta_{0,1,2,3}(z,\tau)^2, \theta_{0,1,2,3}(0,\tau)^2\right) = \left(\theta_{0,1,2,3}(z,2\tau)^2, \theta_{0,1,2,3}(0,2\tau)^2\right)$$

Bad, good, and correct choices of square roots We discuss what we mean above by suitable choice of square roots. Two different notions must be considered:

- "Good choices" in the sense of [Dup06, Cox84], i.e. such that $\mathrm{Re}\left(\frac{\sqrt{a_i}}{\sqrt{a_j}}\right)$, $\mathrm{Re}\left(\frac{\sqrt{b_i}}{\sqrt{b_j}}\right) \geq 0$. Note that not all tuples of complex numbers admit a set of "good" square roots. In genus 1, having infinitely many bad choices means the sequence converges (at least linearly) to zero; to avoid this case, we need them to all be good after a while, which in addition ensures we have quadratic convergence. This is key to our strategy to get a quasi-linear running time.
- The choice of signs that corresponds to θ, i.e. given two quadruples which are (proportional to) approximations of $\theta_{0,1,2,3}(z,\tau)^2$ and $\theta_{0,1,2,3}(0,\tau)^2$, the ones which approximate well the values $\theta_{0,1,2,3}(z,\tau)$ and $\theta_{0,1,2,3}(0,\tau)$. We call these the "correct" choice, which need not be a "good" choice. We need this in order to compute the right value of θ in the end.

Fortunately, the notions of "good" and "correct" choices overlap very often. In genus 1, results from [Cox84] (i.e. Proposition 3.2.9) shows that the correct choice for theta-constants is the good choice for τ within a large domain which includes the fundamental domain (see Figure 3.1); we

proved a similar result for theta functions in Theorem 6.2.6 (see Figure 6.1). In genus 2, we do not determine an explicit domain for which correct choices are good. Although one can try to improve on the approach of Section 5.2 to establish such a result, the mere requirement that τ be in $\mathcal{F}'_g$ is already too strict for our further use (in particular in Section 7.2.2), so that proofs are difficult to obtain.

Iterates of F Proposition 2.1.9 shows that $\lim_{k\to\infty} \frac{\theta_{[0;b]}(z,2^k\tau)}{\theta_{[0;b']}(z,2^k\tau)} = 1$, which easily implies that correct choices are good for large enough τ. Therefore, given an 8-uple X which approximates $(\theta_{0,1,2,3}(z,\tau)^2, \theta_{0,1,2,3}(z,\tau)^2)$, computing iterates $F^n(X)$ and making correct choices consistently is bound to coincide with good choices after a finite number of iterations. To ensure that the first few choices are indeed the correct ones, it suffices to rely on low-precision approximations of θ, so that we know the sign of either $\mathrm{Re}(\theta)$ or $\mathrm{Im}(\theta)$. The number of terms and the precision needed to achieve this do not asymptotically depend on P, but only in z and τ; since we neglect the dependency in z, τ in the complexity of our algorithm[10], determining the correct square root requires only a constant number of operations. We used this strategy in our implementation; furthermore, it generalizes easily to genus g.

Lemma 7.2.2. *Let $(a^{(0)}_{0,1,2,3}, b^{(0)}_{0,1,2,3}) \in \mathbb{C}^8$, and let $\left(a^{(n+1)}_{0,1,2,3}, b^{(n+1)}_{0,1,2,3}\right) = F\left(a^{(n)}_{0,1,2,3}, b^{(n)}_{0,1,2,3}\right)$ for any integer $n \in \mathbb{N}$. Assume that there exists $\alpha, \beta \in \mathbb{C}^*$ and $n_0 \in \mathbb{N}$ such that $\mathrm{Re}(a^{(n_0)}_i/\alpha) > 0$ and $\mathrm{Re}(b^{(n_0)}_i/\beta) > 0$ for all $i \in \{0,1,2,3\}$. Then there exists positive real constants c, C such that assuming all choices of square roots from iteration n_0 onwards are good, we have $\forall n \geq n_0,\ \forall i \in \{0,1,2,3\},\ c \leq |a^{(n)}_i|, |b^{(n)}_i| < C$.*

Proof. The upper bound result follows trivially from the definition. For the lower bound, let us assume without loss of generality that $|\alpha| = |\beta| = 1$, and let $c = \min(\mathrm{Re}(a^{(n_0)}_{0,1,2,3}/\alpha), \mathrm{Re}(b^{(n_0)}_{0,1,2,3}/\beta))$. For good choices of square roots and any i, j, we have

$$\mathrm{Re}(\sqrt{a^{(n_0)}_i/\alpha}\sqrt{b^{(n_0)}_j/\beta}) \geq \min(\mathrm{Re}(a^{(n_0)}_i/\alpha), \mathrm{Re}(b^{(n_0)}_j/\beta)) \geq c$$

(for a proof, see e.g. [Dup06, Lemme 7.3]). This implies from the definition that $\left|a^{(n_0+1)}_i\right| \geq \mathrm{Re}(a^{(n_0+1)}_i/\sqrt{\alpha\beta}) \geq c$, and similarly for $b^{(n_0+1)}_i$. The result follows by induction (with $\sqrt{\alpha\beta}$, of modulus 1, playing the role of α at the next iteration). □

An important remark is that the "low-precision" strategy described above is sufficient to ensure that conditions of Lemma 7.2.2 hold after a few steps.

A Karatsuba-like trick to compute F Proposition 7.2.3 computes F in 4 products and 4 squares instead of the 22 products in its definition. Section 7.4 extends this to genus g.

Proposition 7.2.3. *Put $\mathcal{H} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ and $\mathcal{H}_2 = \mathcal{H} \otimes \mathcal{H}$. Let ${}^t(m_{0,1,2,3}) = \mathcal{H}_2\, {}^t\left(\sqrt{a^{(n)}_{0,1,2,3}}\right)$ and ${}^t(s_{0,1,2,3}) = \mathcal{H}_2\, {}^t\left(\sqrt{b^{(n)}_{0,1,2,3}}\right)$. We have ${}^t(a^{(n+1)}_{0,1,2,3}) = \frac{1}{16}\mathcal{H}_2\, {}^t(m_{0,1,2,3} * s_{0,1,2,3})$ (* being the termwise product), and ${}^t(b^{(n+1)}_{0,1,2,3}) = \frac{1}{16}\mathcal{H}_2\, {}^t(s_{0,1,2,3} * s_{0,1,2,3})$.*

[10] To extend this work into an algorithm whose complexity is uniform in z, τ, one could follow the same approach as in genus 1 (see [Dup06], or Algorithms 11 and 15), since we have once again a naive algorithm whose complexity decreases as $\mathrm{Im}(\tau_1)$ increases.

7.2.2 Constructing and inverting the $\mathfrak{F}$ function

Proposition 7.2.4 (extension of Prop 7.1.3)**.** *Define the matrices* $\mathfrak{J}, \mathfrak{M}_1, \mathfrak{M}_2, \mathfrak{M}_3$ *as in* [Dup06, Chapter 6]. *Then*

$$\begin{aligned}
\theta_0\left((\mathfrak{J}\mathfrak{M}_1)^2\cdot z, (\mathfrak{J}\mathfrak{M}_1)^2\cdot\tau\right)^2 &= -\tau_1 e^{2i\pi z_1^2/\tau_1}\theta_8(z,\tau)^2,\\
\theta_0\left((\mathfrak{J}\mathfrak{M}_2)^2\cdot z, (\mathfrak{J}\mathfrak{M}_2)^2\cdot\tau\right)^2 &= -\tau_2 e^{2i\pi z_2^2/\tau_2}\theta_4(z,\tau)^2,\\
\theta_0\left((\mathfrak{J}\mathfrak{M}_3)^2\cdot z, (\mathfrak{J}\mathfrak{M}_3)^2\cdot\tau\right)^2 &= (\tau_3^2-\tau_1\tau_2)e^{2i\pi\frac{z_1^2\tau_2+z_2^2\tau_1-2z_1z_2\tau_3}{\det(\tau)}}\theta_0(z,\tau)^2.
\end{aligned}$$

This result is a direct consequence of Theorem 2.4.4. The next proposition looks at how the sequence of F^n behaves with respect to homogeneity; its proof is similar to Proposition 6.2.8, i.e. an induction.

Proposition 7.2.5. *Let*

$$\begin{aligned}
\left((a_i^{(n)})_{0\le i\le 3}, (b_i^{(n)})_{0\le i\le 3}\right) &= F^n\left(\left(\theta_{[0;b]}(z,\tau)^2\right)_{b\in\frac12\mathbb{Z}^2/\mathbb{Z}^2}, \left(\theta_{[0;b]}(0,\tau)^2\right)_{b\in\frac12\mathbb{Z}^2/\mathbb{Z}^2}\right),\\
\left((a_i'^{(n)})_{0\le i\le 3}, (b_i'^{(n)})_{0\le i\le 3}\right) &= F^n\left(\lambda\left(\theta_{[0;b]}(z,\tau)^2\right)_{b\in\frac12\mathbb{Z}^2/\mathbb{Z}^2}, \mu\left(\theta_{[0;b]}(0,\tau)^2\right)_{b\in\frac12\mathbb{Z}^2/\mathbb{Z}^2}\right).
\end{aligned}$$

Then we have $a_0'^{(n)} = \epsilon_n\lambda^{1/2^n}\mu^{1-1/2^n}a_0^{(n)}$ *(where* $\epsilon_n^{2^n}=1$*), and* $b_0'^{(n)} = \mu b_0^{(n)}$*, and we can compute* λ,μ *as* $\mu = \lim_{n\to\infty} b_0'^{(n)}$ *and* $\lambda = \lim_{n\to\infty}\left(\frac{a_0'^{(n)}}{b_0'^{(n)}}\right)^{2^n}\times\mu$.

We define $\mathfrak{G}$ as the function which computes these two quantities, that is to say such that $\mathfrak{G}\left(\lambda\left(\theta_{[0;b]}(z,\tau)^2\right)_{b\in\frac12\mathbb{Z}^2/\mathbb{Z}^2}, \mu\left(\theta_{[0;b]}(0,\tau)^2\right)_{b\in\frac12\mathbb{Z}^2/\mathbb{Z}^2}\right) = (\lambda,\mu)$. We prove in Section 7.2.3 that $\mathfrak{G}$ can be computed in $O(\mathcal{M}(P)\log P)$ operations.

We now build $\mathfrak{F}$ from $\mathfrak{G}$. The idea is to evaluate $\mathfrak{G}$ at quotients of theta functions after the action of $(\mathfrak{J}\mathfrak{M}_i)^2$; the λ and μ we recover are the inverse of the quantities in Proposition 7.2.4. Note that $(\mathfrak{J}\mathfrak{M}_i)^2\cdot\tau\notin\mathcal{F}_2'$, which prevents us from generalizing proofs which worked for genus 1 to make "good choices" and "correct choices" coincide. However it is still possible to determine the sign using low-precision approximations.

Evaluating the quotients after the action of $(\mathfrak{J}\mathfrak{M}_i)^2$ is actually simply evaluating different quotients of theta functions; for instance:

$$\frac{\theta_1\left((\mathfrak{J}\mathfrak{M}_1)^2\cdot z, (\mathfrak{J}\mathfrak{M}_1)^2\cdot\tau\right)^2}{\theta_0\left((\mathfrak{J}\mathfrak{M}_1)^2\cdot z, (\mathfrak{J}\mathfrak{M}_1)^2\cdot\tau\right)^2} = \frac{\theta_9(z,\tau)^2}{\theta_0(z,\tau)^2}.$$

Hence, we need to compute $\theta_{[a;b]}(z,\tau)$ for $a\neq 0$. The approach used in [Dup06] for theta-constants is to use explicit formulas linking the $\left(\theta_{[a;b]}(0,\tau)\right)$ to the $\left(\theta_{[0;b]}(0,\tau)\right)$. Instead, we use the approach of [ET14a], which is simpler and more generalizable. The τ-duplication formulas (Equation (2.2.1)) allow us to compute $\theta_{[a;b]}(z,2\tau)$ from the fundamental thetas, and we then compute λ and τ corresponding to the quotients at 2τ, instead of the ones corresponding to the same quotients at τ. This still gives two numbers that are a simple function of z and τ, which is all we need to use Newton's method.

Defining $\mathfrak{F}$ so that it is locally invertible, in order to use Newton's method, requires some care. In genus 1 (Chapter 6), we simply compute z and τ, which gives a $\mathbb{C}^2\to\mathbb{C}^2$ function; however in higher genus this approach leads to a function from $\mathbb{C}^{2^{g+1}-2}$ to $\mathbb{C}^{g(g+3)/2}$, and we cannot apply Newton's method to recover the quotients of thetas. In genus 2, the function is from $\mathbb{C}^6$ to $\mathbb{C}^5$; there are two ways to work around the issue:

1. A natural idea would be to add an extra equation, which would be the equation of a variety which the thetas lie on. For instance, we can take the equation of the Kummer surface, as described in [Gau07] (see Section 2.6.4), which links the fundamental theta functions and theta-constants. Evaluating this equation with $x = a = 1$ and (y, z, t, b, c, d) equal to the inputs of $\mathfrak{F}$ gives one more complex number, which makes $\mathfrak{F}$ from $\mathbb{C}^6$ to $\mathbb{C}^6$; this sixth complex number would be 0 if the inputs were exact quotients of squares of θ, hence Newton's method should strive to make it 0. This approach, however, does not appear to be easily generalizable to higher genus.

2. An approach which actually works just as well is to define $\mathfrak{F}$ so that it outputs a few values of λ and μ computed by different means, instead of z, τ. In our case, we modify the λ, μ slightly, so that

$$\mathfrak{F}\left(\frac{\theta_1(z,\tau)^2}{\theta_0(z,\tau)^2}, \frac{\theta_2(z,\tau)^2}{\theta_0(z,\tau)^2}, \frac{\theta_3(z,\tau)^2}{\theta_0(z,\tau)^2}, \frac{\theta_1(0,\tau)^2}{\theta_0(0,\tau)^2}, \frac{\theta_2(0,\tau)^2}{\theta_0(0,\tau)^2}, \frac{\theta_3(0,\tau)^2}{\theta_0(0,\tau)^2}\right)$$
$$= \left(e^{i\pi\frac{z_1^2}{\tau_{11}}}, e^{i\pi\frac{z_2^2}{\tau_{22}}}, e^{i\pi\frac{z_1^2\tau_2 + z_2^2\tau_1 - 2z_1z_2\tau_3}{\det(\tau)}}, \tau_1, \tau_2, \tau_3^2 - \tau_1\tau_2\right).$$

We use here the second approach, since it can be generalized to higher genera.

Our function $\mathfrak{F}$ is defined in Algorithm 18, using a formulation in terms of quotients of theta functions to make things clearer. The calculations involved are valid for any 6-uple of complex numbers, and a complete description is easy to obtain; we show it in Algorithm 19.

Algorithm 18 Compute $\mathfrak{F}\left(\frac{\theta_1^2}{\theta_0^2}(z,\tau), \frac{\theta_2^2}{\theta_0^2}(z,\tau), \frac{\theta_3^2}{\theta_0^2}(z,\tau), \frac{\theta_1^2}{\theta_0^2}(0,\tau), \frac{\theta_2^2}{\theta_0^2}(0,\tau), \frac{\theta_3^2}{\theta_0^2}(0,\tau)\right)$ with absolute precision P.
Input: $\left(\frac{\theta_1^2}{\theta_0^2}(z,\tau), \frac{\theta_2^2}{\theta_0^2}(z,\tau), \frac{\theta_3^2}{\theta_0^2}(z,\tau), \frac{\theta_1^2}{\theta_0^2}(0,\tau), \frac{\theta_2^2}{\theta_0^2}(0,\tau), \frac{\theta_3^2}{\theta_0^2}(0,\tau)\right)$ with absolute precision P.

Evaluate $\mathfrak{G}$ at $\left(1, \frac{\theta_1^2}{\theta_0^2}(z,\tau), \frac{\theta_2^2}{\theta_0^2}(z,\tau), \frac{\theta_3^2}{\theta_0^2}(z,\tau), 1, \frac{\theta_1^2}{\theta_0^2}(0,\tau), \frac{\theta_2^2}{\theta_0^2}(0,\tau), \frac{\theta_3^2}{\theta_0^2}(0,\tau)\right)$ to recover $\lambda = \frac{1}{\theta_0(z,\tau)^2}$ and $\mu = \frac{1}{\theta_0(0,\tau)^2}$.
Compute $\theta_{0,1,2,3}(z,\tau), \theta_{0,1,2,3}(0,\tau)$, using a low-precision approximation of $\theta(z,\tau)$ for the sign.
Compute the $\theta_i(z,2\tau)^2, \theta_i(0,2\tau)^2$ using Equation (2.2.1).
Compute $\lambda_1, \mu_1 = \mathfrak{G}\left(1, \frac{\theta_9^2}{\theta_8^2}(z,2\tau), \frac{\theta_0^2}{\theta_8^2}(z,2\tau), \frac{\theta_1^2}{\theta_8^2}(z,2\tau), 1, \frac{\theta_9^2}{\theta_8^2}(0,2\tau), \frac{\theta_0^2}{\theta_8^2}(0,2\tau), \frac{\theta_1^2}{\theta_8^2}(0,2\tau)\right)$.
Compute $\lambda_2, \mu_2 = \mathfrak{G}\left(1, \frac{\theta_0^2}{\theta_4^2}(z,2\tau), \frac{\theta_6^2}{\theta_4^2}(z,2\tau), \frac{\theta_2^2}{\theta_4^2}(z,2\tau), 1, \frac{\theta_0^2}{\theta_4^2}(0,2\tau), \frac{\theta_6^2}{\theta_4^2}(0,2\tau), \frac{\theta_2^2}{\theta_4^2}(0,2\tau)\right)$.
Compute $\lambda_3, \mu_3 = \mathfrak{G}\left(1, \frac{\theta_8^2}{\theta_0^2}(z,2\tau), \frac{\theta_4^2}{\theta_0^2}(z,2\tau), \frac{\theta_{12}^2}{\theta_0^2}(z,2\tau), 1, \frac{\theta_8^2}{\theta_0^2}(0,2\tau), \frac{\theta_4^2}{\theta_0^2}(0,2\tau), \frac{\theta_{12}^2}{\theta_0^2}(0,2\tau)\right)$.
$\mu_1 \leftarrow \theta_8^2(0,2\tau)/\mu_1, \mu_2 \leftarrow \theta_4^2/\mu_2, \mu_3 \leftarrow \theta_0^2/\mu_3$.
return $\left(\mu_1\theta_8^2(z,2\tau)/\lambda_1, \mu_2\theta_4^2/\lambda_2, \mu_3\theta_0^2/\lambda_3, \quad \mu_1, \mu_2, \mu_3\right)$

We conjecture the following, which holds in practice:

Conjecture 7.2.6. *The Jacobian of $\mathfrak{F}$ system is invertible.*

As described in Corollary 0.3.8, applying Newton's method allows us to compute an approximation of θ with precision $p - \delta$, where δ is a small constant, from an approximation with precision $p/2$. Instead of computing the Jacobian exactly, we compute an approximation of $\frac{\partial \mathfrak{F}_i}{\partial x_j}$ with precision p using finite differences, i.e. $\frac{\mathfrak{F}_i(x+\epsilon_j) - \mathfrak{F}_i(x)}{||\epsilon_j||}$, for ϵ_j a perturbation of 2^{-p} on the j-th coordinate; this does not modify the result of Corollary 0.3.8, but may require to increase

Algorithm 19 Computation of $\mathfrak{F}(a_{1,2,3}, b_{1,2,3})$ with absolute precision P.
Input: $a_{1,2,3}, b_{1,2,3} \in \mathbb{C}^6$, and a pair $z, \tau \in \mathbb{C}^2 \times \mathcal{H}_2$, with absolute precision P.
We assume that $a_{1,2,3}, b_{1,2,3}$ are approximations to $\left(\frac{\theta^2_{1,2,3}}{\theta^2_0}(z,\tau), \frac{\theta^2_{1,2,3}}{\theta^2_0}(0,\tau)\right)$ to some constant base precision. These coarse estimates serve as a guide to choose the correct signs of square roots.

$(\lambda_0, \mu_0) \leftarrow \mathfrak{G}(1, a_{1,2,3}, 1, b_{1,2,3})$, using (z, τ) to inform the choices of sign.
$x_0 \leftarrow \frac{1}{\lambda_0}, \qquad y_0 \leftarrow \frac{1}{\mu_0}$.
$x_{1,2,3} \leftarrow a_{1,2,3} \times x_0, \qquad y_{1,2,3} \leftarrow b_{1,2,3} \times y_0$.
for i = 0 to 3 **do**
    $x_{0,1,2,3} \leftarrow \pm\sqrt{x_{0,1,2,3}}$, using a low-precision approximation of $\theta_i(z,\tau)$ for the sign.
    $y_{0,1,2,3} \leftarrow \pm\sqrt{y_{0,1,2,3}}$, using a low-precision approximation of $\theta_i(0,\tau)$ for the sign.
end for
for i = 4 to 15 **do**
    $b \leftarrow i \pmod 4$, $a = \frac{i-b}{4}$.
    $x_i \leftarrow \frac{1}{4}\sum_{j=0}^{3}(-1)^{a\cdot j} x_{b\oplus j} y_j$.
    $y_i \leftarrow \frac{1}{4}\sum_{j=0}^{3}(-1)^{a\cdot j} y_{b\oplus j} y_j$.
end for
$(\lambda_1, \mu_1) \leftarrow \mathfrak{G}\left(1, \frac{x_{9,0,1}}{x_8}, 1, \frac{y_{9,0,1}}{y_8}\right)$, using $((\mathfrak{JM}_1)^2 \cdot z, (\mathfrak{JM}_1)^2 \cdot 2\tau)$ to inform the choices of sign.
$(\lambda_2, \mu_2) \leftarrow \mathfrak{G}\left(1, \frac{x_{0,6,2}}{x_4}, 1, \frac{y_{0,6,2}}{y_4}\right)$, using $((\mathfrak{JM}_2)^2 \cdot z, (\mathfrak{JM}_2)^2 \cdot 2\tau)$ to inform the choices of sign.
$(\lambda_3, \mu_3) \leftarrow \mathfrak{G}\left(1, \frac{x_{8,4,12}}{x_0}, 1, \frac{y_{8,4,12}}{y_0}\right)$, using $((\mathfrak{JM}_3)^2 \cdot z, (\mathfrak{JM}_3)^2 \cdot 2\tau)$ to inform the choices of sign.
$\mu_1 \leftarrow \frac{y_8}{\mu_1}, \mu_2 \leftarrow \frac{y_4}{\mu_2}, \mu_3 \leftarrow \frac{y_0}{\mu_3}$.
return $\left(\frac{\mu_1 x_8}{\lambda_1}, \frac{\mu_2 x_4}{\lambda_2}, \frac{\mu_3 x_0}{\lambda_3}, \mu_1/2, \mu_2/2, \mu_3/4\right)$

δ. We prove in the next section that computing $\mathfrak{F}$ with precision p costs $O(\mathcal{M}(p)\log p)$ for any arguments; this implies that applying one step of Newton's method costs $O(\mathcal{M}(p)\log p)$. Thus, as in [Dup06, ET14a] or Chapter 6, we can compute an approximation of θ with precision P_0 using the naive algorithm, then use Newton's method to refine it into a value of θ with precision P provided P_0 is large enough (as in Theorem 0.3.7). The total cost of this algorithm is $O(\mathcal{M}(P)\log P)$.

Remark 7.2.7. Note that the complexity of this algorithm depends on z, τ. However, one could build an algorithm with complexity uniform in z, τ using the same techniques as the ones we showed in genus 1 (Algorithm 11 or Algorithm 15). Namely, one uses the naive algorithm (Algorithm 9) for $P \leq c\,\mathrm{Im}(\tau_{1,1})$; recall that we noted at the end of Section 5.2.2 that the complexity was then uniform, as in genus 1. If $P \geq c\,\mathrm{Im}(\tau_{1,1})$, one uses τ-duplication formulas and z-duplication formulas so that the previous algorithm is called for z, τ in a compact. We leave the precise analysis of such an algorithm to future work.

7.2.3 Proof of quasi-optimal time

Theorem 7.2.8. *One can compute $\mathfrak{G}(a^{(0)}_{0,1,2,3}, b^{(0)}_{0,1,2,3}) = (\lambda, \mu)$ with precision P with complexity $O(\mathcal{M}(P)\log P)$ operations, assuming the choice of signs is always good.*

The result if the arguments are $\left(\lambda\theta^2_{0,1,2,3}(z,\tau), \mu\theta^2_{0,1,2,3}(0,\tau)\right)$ is merely a consequence of the quadratic convergence of $(\theta(z, 2^k\tau))_{k\in\mathbb{N}}$; however we need to prove the result for any arguments to apply it to the computation of the Jacobian. The proof is very similar to the proofs in Section 6.2.4.

Proof. By Lemma 7.2.2, we have $0 < c \leq |a^{(n)}_i|, |b^{(n)}_i| \leq C$ for any i, for n large enough (independent of P). The sequence $d_n = \max_{i,j}|b^{(n)}_i - b^{(n)}_j|$ converges quadratically to zero [Dup06, Prop. 7.1]. So μ can be computed in time $O(\mathcal{M}(P)\log P)$.

Now let $A > 0$ and n_1 be large enough so that $d_{n+1} \leq Ad_n^2$ for all $n \geq n_1$, and additionally that $d_{n_1} < \frac{1}{2A}$. This implies $d_n \leq \frac{1}{A}2^{-2^{n-n_1}}$ for any $n \geq n_1$. The $|a^{(n)}_i - a^{(n)}_j|$ can be linked to d_n. For instance, for any $n \geq n_1$ we have

$$\begin{aligned}
|a_0^{(n+1)} - a_1^{(n+1)}| &= \frac{|m_1 s_1 + m_3 s_3|}{2} \leq 2C(|s_1| + |s_3|) \quad \text{(using notations of Prop. 7.2.3)} \\
&\leq 4C(|\sqrt{b_0} - \sqrt{b_1}| + |\sqrt{b_2} - \sqrt{b_3}|) \\
&\leq 4C(\sqrt{|\sqrt{b_0} - \sqrt{b_1}|^2} + \sqrt{|\sqrt{b_2} - \sqrt{b_3}|^2}) \\
&\leq 4C(\sqrt{|\sqrt{b_0} - \sqrt{b_1}||\sqrt{b_0} + \sqrt{b_1}|} + \sqrt{|\sqrt{b_2} - \sqrt{b_3}||\sqrt{b_2} + \sqrt{b_3}|}) \\
&\leq 4C(\sqrt{|b_0 - b_1|} + \sqrt{|b_2 - b_3|}) \\
&\leq 8C\sqrt{d_n}.
\end{aligned}$$

Calculus also shows that

$$\begin{aligned}
|a_0^{(n+1)} - \sqrt{a_0^{(n)} b_0^{(n)}}| &= \frac{1}{8}\left|\sum_{i=1}^{3}(\sqrt{a_i} - \sqrt{a_0})\left(\sqrt{b_i} + \sqrt{b_0}\right) + (\sqrt{a_i} + \sqrt{a_0})\left(\sqrt{b_i} - \sqrt{b_0}\right)\right| \\
&\leq K\sqrt{d_n} \text{ for some explicit constant } K.
\end{aligned}$$

Superscripts $^{(n)}$ have been omitted from the right-hand sides above for brevity. For both inequalities, we used the fact that choices of roots are good.

We now show that $\lambda_n = (a_0^{(n)}/b_0^{(n)})^{2^n}$ converges quadratically. Let $q_n = \frac{(a_0^{(n+1)}/b_0^{(n+1)})^2}{a_0^{(n)}/b_0^{(n)}}$, so that $\lambda_{n+1} = \lambda_n q_n^{2^n}$.

It is relatively easy to check that $|q_{n+1} - 1|$ is also bounded by $K'\sqrt{d_n}$ for an explicit constant K', given the bounds established above, as well as the inequality $|\mu - b_0^{(n+m)}| \leq \frac{2}{A}(A\eta)^{2^m}$, which is proven as follows:

$$\begin{aligned} |b_0^{(n+m+1)} - b_0^{(n+m)}| &\leq d_{n+m} \qquad \text{since } b_0^{(n+m+1)} \text{ is the arithmetic mean of the } b_j^{(n+m)} \\ |b_0^{(n+m+l)} - b_0^{(n+m)}| &\leq \sum_{i=0}^{l-1} d_{n+m+i} \leq \frac{1}{A}\sum_{i=0}^{l-1}(A\eta)^{2^{i+m}} \leq \frac{2}{A}(A\eta)^{2^m}. \end{aligned}$$

It follows, from an unsurprising calculation similar to that of Theorem 6.2.13, that the sequence λ_n also converges quadratically. This concludes the proof that only a logarithmic number of steps are needed to compute the values taken by $\mathfrak{G}$, and hence by $\mathfrak{F}$, to precision P. □

7.3 Implementation results

Our Magma implementation of Algorithm 9 and our quasi-linear time algorithm is at

`http://www.hlabrande.fr/pubs/fastthetasgenus2.m`

We compared with Magma's general-purpose `Theta` function. Assuming the latter computes each term by exponentiation, its complexity would be $O(\mathcal{M}(P)P\log P)$. However, practice reveals it behaves much worse. Table 7.1 show that for precision above 1 000 decimal digits our algorithm, which outputs 8 values, is faster than one call to Magma's `Theta` function, which only computes $\theta(z,\tau)$. Furthermore, it is also faster than Algorithm 9 for precisions greater than 3 000 decimal digits. This cut-off is much lower than in genus 1, which is expected since the complexity of the naive algorithm is $O(\mathcal{M}(P)\sqrt{P})$ in genus 1 and $O(\mathcal{M}(P)P)$ in genus 2. Our results are consistent with the situation for theta constants, studied in [ET14a][11]

Prec (decimal digits)	Magma	Algorithm 9	This work
1000	0.42	0.38	0.38
2000	2.58	1.86	1.86
4000	18.4	9.51	6.65
8000	128	53.9	13.2
16000	889	303	25
32000	6368	1535	50
64000	46566	8798	120

Table 7.1: Timings (in s) of different methods

7.4 Computing theta functions of higher genera

This section outlines ideas for extending the previous strategy to the case $g > 2$. The complexity of such an algorithm will certainly be exponential (or worse) in g; we do not make any attempt at lowering this complexity, and in fact we do not even evaluate it fully. However, the complexity in P would still be $O(\mathcal{M}(P)\log P)$, which is desirable.

[11] Compared to [ET14a], we compute more, and we do it in Magma, not in C. Hence the slower timings.

7.4.1 The function F

We use once again the τ-duplication formula (Equation (2.2.1)) where $a = 0$:

$$\theta_i(z, 2\tau)^2 = \frac{1}{2^g} \sum_{k \in \{0,\dots,2^g-1\}} \theta_{i\oplus k}(z,\tau)\theta_k(0,\tau), \tag{7.4.1}$$

where $\oplus$ is the bitwise XOR. This gives us a function

$$\begin{array}{rcl} F : \mathbb{C}^{2^{g+1}} & \to & \mathbb{C}^{2^{g+1}} \\ \left(\left(\theta_{[0;b]}(z,\tau)\right), \left(\theta_{[0;b]}(0,\tau)\right)\right) & \mapsto & \left(\left(\theta_{[0;b]}(z,2\tau)\right), \left(\theta_{[0;b]}(0,2\tau)\right)\right). \end{array}$$

Just like in genus 2, we solve the problem of determining the correct square root by using low-precision approximations of θ, which only require a number of terms and a precision that are independent of P. Furthermore, Proposition 2.1.9 proves that $\lim_{n\to\infty} \theta_{[0;b]}(z, 2^n\tau) = 1$, which shows that, after a finite number of steps, correct choices of sign correspond to good choices of sign, i.e. the ones corresponding to $\mathrm{Re}\left(\frac{\sqrt{a_i}}{\sqrt{a_j}}\right) > 0$.

The Karatsuba-like trick we used in Proposition 7.2.3 can be generalized here for the evaluation of F. Sums involving bitwise XORs as the one in Equation (2.2.1) are called *dyadic convolutions* in [Mak75], which also gives an algorithm to compute them with an optimal number of multiplications. The method is exactly the one we used in Proposition 7.2.3, using this time $\mathcal{H}_g = \mathcal{H} \otimes \cdots \otimes \mathcal{H}$ (g times). This means we only need 2^{g+1} multiplications, instead of the 2^{2g+1} multiplications that appear in the definition of F.

7.4.2 Extending the quasi-linear time algorithm

Iterates of F

We then define the sequence of iterates of F as:

$$\left(a_0^{(n+1)}, \dots, a_{2^g-1]}^{(n+1)}, b_0^{(n+1)}, \dots, b_{2^g-1}^{(n+1)}\right) = F\left(a_0^{(n)}, \dots, a_{2^g-1]}^{(n)}, b_0^{(n)}, \dots, b_{2^g-1}^{(n)}\right)$$

and we denote $F^\infty(a_0^{(0)}, \dots, a_{2^g-1}^{(0)}, b_0^{(0)}, \dots, b_{2^g-1}^{(0)})$ as the limit of this sequence.

Note that the choice of signs in this sequence are taken to correspond to the correct choices of signs with respect to values $\theta_i(z, 2^n\tau)$. Hence, by definition, we have $F\left(\theta^2_{0,\dots,2^g-1}(z,\tau), \theta^2_{0,\dots,2^g-1}\right) = 1$, since the choices are always chosen to be correct. Furthemore, Proposition 2.1.9 proves that there is only a finite number of bad choices of signs. Finally, note that the operations on the $b_i^{(n)}$ are exactly those of the Borchardt mean; Theorem 3.4.6 then proves that the convergence of the $b_i^{(n)}$ is quadratic.

The following generalizes Lemma 7.2.2; its proof is essentially the same.

Lemma 7.4.1. *Let* $\left(a^{(0)}_{0,\dots,2^g-1}, b^{(0)}_{0,\dots,2^g-1}\right) \in \mathbb{C}^{2^{g+1}}$, *and define* $a_i^{(n)}, b_i^{(n)}$ *for all* $n > 0$ *as previously. Assume that there exists* $\alpha, \beta \in \mathbb{C}^*$ *and* $n_0 \in \mathbb{N}$ *such that* $\mathrm{Re}(a_i^{(n_0)}/\alpha) > 0$ *and* $\mathrm{Re}(b_i^{(n_0)}/\beta) > 0$ *for all* $i \in \{0, \dots, 2^g - 1\}$. *Then there exists positive real constants* c, C *such that assuming all choices of square roots from iteration* n_0 *onwards are good, we have* $\forall n \geq n_0,\ \forall i \in \{0, \dots, 2^g - 1\},\ c \leq |a_i^{(n)}|, |b_i^{(n)}| < C$.

Homogeneity

We take a look at how the sequence of iterates of F behaves with respect to homogeneity. The following proposition is a straightforward generalization of Propositions 6.2.8 and 7.2.5

Proposition 7.4.2. *Let*

$$\begin{aligned}\left((a_i^{(n)})_{0\leq i\leq 2^g-1},(b_i^{(n)})_{0\leq i\leq 2^g-1}\right) &= F^n\left(\left(\theta_{[0;b]}(z,\tau)^2\right)_{b\in\frac{1}{2}\mathbb{Z}^g/\mathbb{Z}^g},\left(\theta_{[0;b]}(0,\tau)^2\right)_{b\in\frac{1}{2}\mathbb{Z}^g/\mathbb{Z}^g}\right),\\ \left((a_i'^{(n)})_{0\leq i\leq 2^g-1},(b_i'^{(n)})_{0\leq i\leq 2^g-1}\right) &= F^n\left(\lambda\left(\theta_{[0;b]}(z,\tau)^2\right)_{b\in\frac{1}{2}\mathbb{Z}^g/\mathbb{Z}^g},\mu\left(\theta_{[0;b]}(0,\tau)^2\right)_{b\in\frac{1}{2}\mathbb{Z}^g/\mathbb{Z}^g}\right).\end{aligned}$$

Then we have $a_0'^{(n)} = \epsilon_n\lambda^{1/2^n}\mu^{1-1/2^n}a_0^{(n)}$ *(where* $\epsilon_n^{2^n}=1$*), and* $b_0'^{(n)} = \mu b_0^{(n)}$*, and we can compute* λ,μ *as* $\mu=\lim_{n\to\infty} b_0'^{(n)}$ *and* $\lambda=\lim_{n\to\infty}\left(\frac{a_0'^{(n)}}{b_0'^{(n)}}\right)^{2^n}\times\mu$.

Call $\mathfrak{G}$ the function which computes λ,μ with precision P by applying the formulas above; by definition, we then have

$$\mathfrak{G}\left(1,\frac{\theta_1^2(z,\tau)}{\theta_0^2(z,\tau)},\ldots,\frac{\theta_{2^g-1}^2(z,\tau)}{\theta_0^2(z,\tau)},1,\frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)},\ldots,\frac{\theta_{2^g-1}^2(0,\tau)}{\theta_0^2(0,\tau)}\right)=\left(\frac{1}{\theta_0^2(z,\tau)},\frac{1}{\theta_0^2(0,\tau)}\right). \quad (7.4.2)$$

Once again, the $a_i^{(n)}$ do not necessarily converge quadratically (for instance, take $a_i^{(n)}=2$). However, note that μ can be computed in $O(\log P)$ steps, since the Borchardt mean converges quadratically. We then propose the following conjecture:

Conjecture 7.4.3. *If all the choices of sign are good,* λ *and* μ *can be computed with absolute precision* P *using only* $O(\log P)$ *iterations of* F.

We proved this conjecture in genus 1 (Theorem 6.2.13) and in genus 2 (Theorem 7.2.8). We believe that the proof carries to genus g, using Lemma 7.4.1; however, due to the technical nature of the proof, we were unable to generalize it in a rigourous and satisfying manner. This conjecture means that $\mathfrak{G}$ can be evaluated in $O(2^g\mathcal{M}(P)\log P)$ operations.

Defining $\mathfrak{F}$

We now wish to define $\mathfrak{F}$, a function which can be evaluated in complexity quasi-linear in the precision, which takes simple values at arguments which are quotients of fundamental theta functions and of theta constants, and which can be inverted.

As we mentioned in genus 2 (Section 7.2.2), Newton's method cannot be applied as straightforwardly as in genus 1. We are considering 2^g-1 quotients $\frac{\theta_i^2(z,\tau)}{\theta_0^2(z,\tau)}$ of fundamental theta-functions and as many quotients of fundamental theta-constants, while there are only g coordinates in z and $\frac{g(g+1)}{2}$ in τ. Hence, using $\mathfrak{G}$ to compute the z_i and $\tau_{i,j}$ is not a process which can be inverted using Newton's method, as it leads to a function from $\mathbb{C}^{2^{g+1}-2}$ to $\mathbb{C}^{g(g+3)/2}$.

We considered two possible workarounds in the genus 2 case. The first one was to take a look at the equations defining the variety defined by the theta functions and the theta-constants, such as the Kummer variety in genus 2; however, we are not aware of an explicit description of these equations in genus g. The other possibility is simpler, and we believe that it still yields to a correct algorithm, which we express in a conjecture:

Conjecture 7.4.4. *There is a set of* 2^g-1 *matrices* $M_0,\ldots,M_{2^g-1}$ *of* $\mathrm{Sp}_{2g}(\mathbb{Z})$*, containing* I_{2g}*, such that, if we write*

$$\theta_0(0,M_i\cdot\tau)^2=\mu_i\theta_{n_i}(0,\tau)^2,\qquad \theta_0(M_i\cdot z,M_i\cdot\tau)^2=\mu_i\lambda_i\theta_{n_i}(z,\tau)^2$$

and we define the function $\mathfrak{F} : \mathbb{C}^{2^{2g-2}} \to \mathbb{C}^{2^{2g-2}}$ *as in Algorithm 20, then the Jacobian of* $\mathfrak{F}$ *at* $\left(\frac{\theta^2_{1,\dots,2^g-1}(z,\tau)}{\theta^2_0(z,\tau)}, \frac{\theta^2_{1,\dots,2^g-1}(0,\tau)}{\theta^2_0(0,\tau)}\right)$ *is invertible.*

Algorithm 20 Compute $\mathfrak{F}\left(\frac{\theta_{1,\dots,2^g-1}(z,\tau)^2}{\theta_0(z,\tau)^2}, \frac{\theta_{1,\dots,2^g-1}(0,\tau)^2}{\theta_0(0,\tau)^2}\right)$ with precision P.

Input: $\left(\frac{\theta_{1,\dots,2^g-1}(z,\tau)^2}{\theta_0(z,\tau)^2}, \frac{\theta_{1,\dots,2^g-1}(0,\tau)^2}{\theta_0(0,\tau)^2}\right)$ with absolute precision P.

1: Compute $\lambda_0 = \frac{1}{\theta_0(z,\tau)^2}, \frac{1}{\theta_0(0,\tau)^2} = \mathfrak{G}\left(1, \frac{\theta_{1,\dots,2^g-1}(z,\tau)^2}{\theta_0(z,\tau)^2}, 1, \frac{\theta_{1,\dots,2^g-1}(0,\tau)^2}{\theta_0(0,\tau)^2}\right)$.
2: Compute the individual $\theta_i(z,\tau), \theta_i(z,\tau)$ for $i \in \{0, ..., 2^g - 1\}$.
3: Use Equation (2.2.1) to compute $\theta_i(z, 2\tau)^2, \theta_i(0, 2\tau)^2$ for $i \in \{0, ..., 2^{2g} - 1\}$.
4: **for** i = 1 to $2^g - 1$ **do**
5: Compute $\lambda_i, \mu_i = \mathfrak{G}\left(1, \frac{\theta_{1,\dots,2^g-1}(M_i\cdot z, M_i\cdot 2\tau)^2}{\theta_0(M_i\cdot z, M_i\cdot 2\tau)^2}, 1, \frac{\theta_{1,\dots,2^g-1}(0, M_i\cdot 2\tau)^2}{\theta_0(0, M_i\cdot 2\tau)^2}\right)$
6: **end for**
7: **return** $(\lambda_0, \dots, \lambda_{2^g-1}, \mu_0, \dots, \mu_{2^g-1})$.

In genus 2, Conjecture 7.2.6 is simply that the set $\{(\mathfrak{JM}_1)^2, (\mathfrak{JM}_2)^2, (\mathfrak{JM}_2)^2\}$ is the one needed in Conjecture 7.4.4.

The final algorithm

Conjecture 7.4.4 simply expresses that there is a set of symplectic matrices such that considering their action on quotients of theta-constants yields a function on which Newton's method can be applied. Furthermore, note that the shape of λ_i, μ_i is given by Theorem 2.4.4, and that they are simple functions of z and τ.

Hence, provided that the conjecture is true, one can simply compute the λ_i, μ_i at full precision, compute a low-precision approximation of the fundamental theta functions and theta constants, then refine this approximation using Newton's method on $\mathfrak{F}$. The total complexity of this method is the same as the complexity of evaluating $\mathfrak{F}$ at full precision, since Newton's method (when doubling the working precision at each step) does not add any asymptotic complexity. The complexity of the evaluation of $\mathfrak{F}$ is $O(4^g \mathcal{M}(P) \log P)$ bit operations. Although this is exponential in the genus g, this is quasi-linear in the precision P; hence, as it was the case between genus 1 and 2, we expect the precision for which our algorithm is better than a naive approach to be smaller as the genus grows.

Remark 7.4.5. A similar algorithm can be constructed to compute the theta-constants. In this case, the function $\mathfrak{G}$ is simply the Borchardt mean, which converges quadratically and is homogeneous. However, there is still the problem of determining symplectic matrices so that one can apply Newton's method to the function which outputs the corresponding set of μ_i, which is also unresolved.

We were able to write a prototype implementation for the computation of genus 3 theta-constants using this method. In that particular case, there are 7 quotients $\frac{\theta_i^2}{\theta_0^2}(0,\tau)$, while there are 6 coefficients for τ: this requires adding one equation to the output, just as in the case of $\theta(z,\tau)$ in genus 2 (Section 7.2.2). To solve this problem, we considered the action of the 6 symplectic matrices $(\mathfrak{JM}_1)^2, (\mathfrak{JM}_2)^2, (\mathfrak{JM}_3)^2, (\mathfrak{JM}_{1,2})^2, (\mathfrak{JM}_{1,3})^2, (\mathfrak{JM}_{2,3})^2$ as defined by Dupont (see e.g. Section 8.3.1), along with the action of the matrix $\mathfrak{J}$; the 7 μ_i one gets are not easily linked together, which may justify why the Jacobian of the system appears to be invertible, although we still do not have any proof of this invertibility.

Preliminary results confirm that the method appears to work, and that the Jacobian appears to be invertible. The algorithm appears to be much faster than the naive algorithm for precisions greater than 450 decimal digits, and is also much faster than Magma's `Theta`. The implementation has been made available publicly; however, a more thorough analysis of the algorithm is still needed at this point.

Chapter 8

Fast computation of the Abel-Jacobi map and its inverse in genus 1, 2 and above

This chapter uses the algorithms of Chapter 6 and Chapter 7 to compute the Abel-Jacobi map. We show that the Abel-Jacobi map can be computed in quasi-optimal time in genus g; as for its inverse, its computation of quasi-linear time requires the existence of a quasi-linear time algorithm for θ, which means solving the problems outlined in Section 7.4.

8.1 In genus 1

We first show in this section how to compute the inverse of the Abel-Jacobi map, as doing so establishes a result one can use for the computation of the Abel-Jacobi map. Computing the inverse of the Abel-Jacobi map requires solving two problems: computing the equation of the curve given the periods, and computing the image of points by the inverse map. The latter problem can be solved using a formula linking between $\wp$ and θ; we compare the resulting algorithm to our algorithm using Landen transform to compute $\wp$ (Algorithm 6). The last subsection shows how to compute the Abel-Jacobi map, and in particular shows an algorithm which uses the link between $\wp$ and θ.

8.1.1 Computing the equation of the curve

Suppose that we have an elliptic curve $E = \mathbb{C}/\Lambda$ in its analytic representation, with $\Lambda = \mathbb{Z}\omega_1 + \mathbb{Z}\omega_2$. We put $\tau = \omega_2/\omega_1$ and suppose that $\mathrm{Im}(\tau) > 0$. We would like to compute two complex numbers a, b such that E is isomorphic to the curve defined by $y^2 = 4x^3 - ax - b$.

The equation of the curve is given by the differential equation satisfied by $\wp$ (Proposition 1.3.18):

Theorem 8.1.1. *For all $\omega_1, \omega_2 \in \mathbb{C}$, the Weierstrass-$\wp$ function satisfies the differential equation*

$$\wp'(z, [\omega_1, \omega_2])^2 = 4\wp(z, [\omega_1, \omega_2])^3 - g_2\wp(z, [\omega_1, \omega_2]) - g_3$$

where

$$\begin{aligned} g_2 &= 60G_4(\tau) = 60 \times \frac{2\zeta(4)}{\omega_1^4} E_4(\tau) = \frac{1}{12}\left(\frac{2\pi}{\omega_1}\right)^4 E_4(\tau) \\ g_3 &= 140G_6(\tau) = 140 \times \frac{2\zeta(6)}{\omega_1^6} E_6(\tau) = \frac{1}{216}\left(\frac{2\pi}{\omega_1}\right)^6 E_6(\tau), \end{aligned}$$

and where

$$E_4(\tau) = \sum_{\omega \in \mathbb{Z}+\tau\mathbb{Z}, \omega \neq 0} \frac{1}{\omega^4}, \qquad E_6(\tau) = \sum_{\omega \in \mathbb{Z}+\tau\mathbb{Z}, \omega \neq 0} \frac{1}{\omega^6}$$

are the normalized Eisenstein series of weight 2 and 3.

The computation of such Eisenstein series can be done using their expansion as Lambert series; we study such expansions in Section 8.4. For now, we just mention the expansions for E_4 and E_6 (see, e.g., [Coh93, Prop. 7.4.1]):

$$E_4(\tau) = 1 + 240 \sum_{n \geq 1} \frac{n^3 q^n}{1-q^n}, \qquad E_6(\tau) = 1 - 504 \sum_{n \geq 1} \frac{n^5 q^n}{1-q^n}$$

The terms of those series are eventually decreasing to zero in a geometric fashion, which means computing E_4 and E_6 with precision P using those expansions requires $O(P)$ terms and a final complexity of $O(\mathcal{M}(P)P)$ bit operations. We refer to Section 8.4 for a more thorough analysis.

We now show how to evaluate g_2, g_3 (and hence E_4, E_6) with precision P in quasi-optimal time $O(\mathcal{M}(P)\log P)$, using connections with theta-constants. Recall the *Thomae formulas* (Theorem 1.3.19), which link theta-constants and the polynomial defining the elliptic curve:

Theorem 8.1.2. *Let $P = 4X^3 - g_2X - g_3 = 4(X-e_1)(X-e_2)(X-e_3)$. Then*

$$e_1 - e_2 = \left(\frac{\pi}{\omega_1}\right)^2 \theta_0^4(0,\tau), \quad e_1 - e_3 = \left(\frac{\pi}{\omega_1}\right)^2 \theta_1^4(0,\tau), \quad e_3 - e_2 = \left(\frac{\pi}{\omega_1}\right)^2 \theta_2^4(0,\tau)$$

We showed in Chapter 6 (Algorithm 11) a way to compute theta-constants in $O(\mathcal{M}(P)\log P)$ bit operations. We then use the following proposition to compute g_2 and g_3:

Proposition 8.1.3. *We have:*

$$\begin{aligned} g_2 &= \frac{2}{3}\left(\frac{\pi}{\omega_1}\right)^4 (\theta_0(0,\tau)^8 + \theta_1(0,\tau)^8 + \theta_2(0,\tau)^8) \\ g_3 &= \frac{4}{27}\left(\frac{\pi}{\omega_1}\right)^6 (\theta_0(0,\tau)^4 + \theta_1(0,\tau)^4)(\theta_0(0,\tau)^4 + \theta_2(0,\tau)^4)(\theta_1(0,\tau)^4 - \theta_2(0,\tau)^4) \end{aligned}$$

We digress briefly to discuss other formulas for g_2 and g_3 found in the literature. Note that the first formula is rather well-known and appears in many references; for instance it corresponds to the theta function linked to the lattice E_8, and a proof using this fact can be found in [CS93, Prop. 8.1, Chapter 4]. The second formula is not mentioned in that form as often; we derived it from similar-looking formulas in [Cle80], and it is implicit in [AS64] (one just needs to combine Equations 18.10.9 to 18.10.11 with Equation 18.10.16). Other formulas for g_3 are sometimes given, such as:

$$\begin{aligned} g_3 &= \frac{4}{27}\left(\frac{\pi}{\omega_1}\right)^6 \sqrt{\frac{(\theta_0(0,\tau)^8 + \theta_1(0,\tau)^8 + \theta_2(0,\tau)^8)^3 - 54(\theta_0(0,\tau)\theta_1(0,\tau)\theta_2(0,\tau))^8}{2}} \\ &= \frac{1}{6^3}\left(\frac{\pi}{\omega_1}\right)^6 \left(\theta_2(0,\tau)^{12} - \frac{3}{2}\theta_2(0,\tau)^8\theta_0(0,\tau)^4 - \frac{3}{2}\theta_2(0,\tau)^4\theta_0(0,\tau)^8 + \theta_0(0,\tau)^{12}\right) \end{aligned}$$

These formulas can be proven, for instance, by looking at the first few coefficients of the Laurent series and showing that they agree, and thus that the difference between the functions is an elliptic holomorphic function which is 0 at 0[12]. The first one is usually derived from the expression of the discriminant of the curve Δ, as a function of theta-constants using Thomae's formulas (see e.g [Cha85, p.34 and proof of Cor. 1 of Theorem 5, Chap V]) and the equation $\Delta = g_2^3 - 27g_3^2$. The formula of Proposition 8.1.3 is simpler, as one does not need to extract a square root (and hence worrying about picking the right sign) and requires fewer multiplications.

Proof of Prop. 8.1.3. We prove the proposition using relations between the coefficients of a polynomial and its roots:

$$e_1 + e_2 + e_3 = 0, \qquad e_1e_2e_3 = \frac{g_3}{4}, \qquad e_1e_2 + e_1e_3 + e_2e_3 = -\frac{g_2}{4}$$

Now,

$$e_1^2 + e_2^2 + e_3^2 = (e_1 + e_2 + e_3)^2 - 2(e_1e_2 + e_1e_3 + e_2e_3) = \frac{g_2}{2}$$

Hence : $$(e_1 - e_2)^2 + (e_1 - e_3)^2 + (e_2 - e_3)^2 = 2(e_1^2 + e_2^2 + e_3^2) - 2(e_1e_2 + e_1e_3 + e_2e_3) = \frac{3g_2}{2}$$

This proves the first formula. The second one is a consequence of $e_1 = -e_2 - e_3$ and so on, which means

$$27e_1e_2e_3 = (e_1 - e_2 + e_1 - e_3)(e_1 - e_2 + e_3 - e_2)(e_1 - e_3 + e_2 - e_3). \qquad \square$$

These formulas, combined with Algorithm 11, proves that one can compute the coefficients g_2, g_3 of the equation of the algebraic representation of the elliptic curve with precision P with only $O(\mathcal{M}(P)\log P)$ bit operations. This also provides a $O(\mathcal{M}(P)\log P)$ algorithm to compute E_4, E_6. We show in Section 8.4 how this can be used to compute any E_{2k} faster than with the naive method.

8.1.2 Computing Weierstrass's $\wp$ function using the θ function

Suppose that we have an elliptic curve $E = \mathbb{C}/(\mathbb{Z}\omega_1 + \mathbb{Z}\omega_2)$. Given a $z \in E$, we would like to compute its image $P_z \in E(\mathbb{C})$ by the inverse of the Abel-Jacobi map. Recall that Proposition 1.3.18 proves that $P_z = (\wp(z, [\omega_1, \omega_2]), \wp'(z, [\omega_1, \omega_2]))$; hence, we are interested in this section in the computation of $\wp$ and its derivative.

Some references (e.g. [Coh93, Section 7.4]) compute those quantities by using their series expansion; an algorithm of Coquereaux et al. [CGL90] seems to claim a $O(\mathcal{M}(P)P)$ complexity, by computing $\wp$ at $\frac{z}{2^N}$ (with $N = O(P)$ for reasons of accuracy) using the series expansion, then using N duplication formulas for $\wp$. We outlined in Section 4.3 an algorithm with complexity $O(\mathcal{M}(P)\log P)$ to compute $\wp(z, \tau)$. We provide another one here, based on the results of Chapter 6.

The connection between $\wp(z, \tau)$ and $\theta(z, \tau)$ is well-known: the formula allowing one to compute $\wp$ from θ appears in numerous references, e.g. [Mum83, BM88][13]. We show a proof of this result, which is Theorem 8.1.4.

Adding to this, we prove Theorem 8.1.6, which gives a formula allowing the computation of $\wp'$ from the knowledge of θ. Combining both these theorems and the algorithm from Chapter 6 then show that the inverse of the Abel-Jacobi map can be computed in $O(\mathcal{M}(P)\log P)$.

[12] Michael Somos, in a personal communication, indicated that many identities related to theta-constants and Eisenstein series that he added to the OEIS were found this way.

[13] We find in [Coh93, p. 397] the recommendation to "use the formulas that link elliptic functions with the [Weierstrass] σ function, since the latter are theta series and so can be computed efficiently", which amounts to the same proof.

Theorem 8.1.4 ([Mum83, p. 26 & p. 73], [BM88]). *Suppose $E = \mathbb{C}/\Lambda$, with $\Lambda = \mathbb{Z}\omega_1 + \mathbb{Z}\omega_2$. Define:*

$$\tilde{\wp}(z,\tau) = \frac{1}{\omega_1^2}\wp(z,\Lambda) = \frac{1}{z^2} + \sum_{\omega\in\mathbb{Z}+\tau\mathbb{Z}} \frac{1}{(z-\omega)^2} - \frac{1}{\omega^2}$$

We then have:

$$\tilde{\wp}(z,\tau) = \frac{\pi^2}{3}(\theta_2^4(0,\tau) - \theta_1^4(0,\tau)) - \pi^2\theta_1^2(0,\tau)\theta_2^2(0,\tau)\frac{\theta_0^2(z,\tau)}{\theta_3^2(z,\tau)} \tag{8.1.1}$$

The explicit determination of the constant term (i.e. the one independent in z) is often left to the reader; hence we present the full proof here.

Proof. We start from the equation [Mum83, p. 25]

$$\widetilde{\wp}(z,\tau) = -\frac{d^2}{dz^2}\log\theta_3(z,\tau) + c$$

This classical equation can be proven using the theory of elliptic functions, for instance by outlining the connections between $\wp$, θ and the Weierstrass functions ζ and σ; we refer to e.g. [Cha85, Sections IV.1, IV.2, V.1]. We then look at the addition formula (A_{10}) of [Mum83, p.26]:

$$\theta_3(x+u,\tau)\theta_3(x-u,\tau)\theta_0^2(0,\tau) = \theta_3^2(x,\tau)\theta_0^2(u,\tau) - \theta_0^2(x,\tau)\theta_3^2(u,\tau)$$

Some terms simplify using $\theta_3(0,\tau) = 0$ and $\theta_0'(0,\tau) = 0$, which are consequences of the parity of these functions. This gives:

$$\widetilde{\wp}(z,\tau) = c - \frac{\theta_3'(0,\tau)^2}{\theta_0(0,\tau)^2} \times \frac{\theta_0(z,\tau)^2}{\theta_3(z,\tau)^2}$$

where $\theta_3'(0,\tau)$ is the derivative in z of $\theta_3(z,\tau)$ in $z = 0$. We then use Jacobi's derivative formula [Mum83, p.64]:

$$\theta_3'(0,\tau) = -\pi\theta_0(0,\tau)\theta_1(0,\tau)\theta_2(0,\tau)$$

which means:

$$\begin{aligned}\widetilde{\wp}(z) &= c - \frac{(-\pi\theta_0(0,\tau)\theta_1(0,\tau)\theta_2(0,\tau))^2}{\theta_0(0)^2} \times \frac{\theta_0(z,\tau)^2}{\theta_3(z,\tau)^2} \\ &= c - \pi^2\theta_1(0,\tau)^2\theta_2(0,\tau)^2 \times \frac{\theta_0(z,\tau)^2}{\theta_3(z,\tau)^2}.\end{aligned}$$

Note that the equation can also be rewritten as

$$\wp(z,\tau) = c - \pi^2\exp(-2\pi iz)\theta(1/2,\tau)^2\theta(\tau/2,\tau)^2\frac{\theta(z,\tau)^2}{\theta(z+(\tau+1)/2,\tau)^2}.$$

All that remains is determining c. We have $\theta\left(\frac{1+\tau}{2},\tau\right) = \theta_3(0,\tau) = 0$ since θ_3 is odd. This means that $c = \widetilde{\wp}(\frac{\tau+1}{2}) = \omega_1^2\wp(\frac{\omega_1+\omega_2}{2})$. Thomae's formula gives

$$\left(\frac{\pi}{\omega_1}\right)^2(\theta_2^4(0,\tau) - \theta_1^4(0,\tau)) = 2e_3 - e_1 - e_2 = 3e_3$$

since the sum of the roots is 0. Hence

$$c = \omega_1^2\wp\left(\frac{\omega_1+\omega_2}{2}\right) = \frac{\pi^2}{3}(\theta_2^4(0,\tau) - \theta_1^4(0,\tau)).$$

□

Note 1.3.16 showed that the value of $\wp$ is not changed after reduction of τ (so that $\tau \in \mathcal{F}$) and z (so that $0 \leq \mathrm{Im}(z) < \mathrm{Im}(\tau)/2$). Those are exactly the conditions we required in Algorithm 15 to compute θ. Hence, one can directly use the value of θ at reduced arguments to compute $\wp$, without even needing to compute the exponential factor which appears in Proposition 2.1.7.

Remark 8.1.5. The algorithm we presented in Section 6 did not compute $\theta_3(z,\tau)$; however Equation (2.5.7) and the equation (cf. [Mum83, p. 22])

$$\theta_3^2(z,\tau)\theta_0^2(0,\tau) = \theta_1^2(z,\tau)\theta_2^2(0,\tau) - \theta_2^2(z,\tau)\theta_1^2(0,\tau)$$

can be combined to prove that

$$\tilde{\wp}(z,\tau) = \frac{\pi^2}{3}(\theta_2^4(0,\tau)+\theta_1^4(0,\tau)) - \pi^2\theta_1^2(0,\tau)\theta_2^2(0,\tau)\frac{\theta_1^2(z,\tau)\theta_1^2(0,\tau)+\theta_2^2(z,\tau)\theta_2^2(0,\tau)}{\theta_1^2(z,\tau)\theta_2^2(0,\tau)-\theta_2^2(z,\tau)\theta_1^2(0,\tau)}$$

which only uses quantities that Algorithm 15 computes.

We now prove a second formula, linking this time $\wp'$ to θ. A similar formula providing this link, which we were not aware of, seems to be already known [AS64, Equation 18.10.6]. The formula is a bit different from ours, suggesting a different method or proof has been used; we do not know of a proof for that formula.

Theorem 8.1.6.

$$\widetilde{\wp}'(z,\tau) = -\pi^3\theta_0^3(0,\tau)\theta_1^3(0,\tau)\theta_2^3(0,\tau)\frac{\theta_3(2z,\tau)}{\theta_3^4(z,\tau)} \tag{8.1.2}$$

Proof. We start from the relation $\widetilde{\wp}(z,\tau) = c - \frac{\theta_3'(0,\tau)^2}{\theta_0(0,\tau)^2} \times \frac{\theta_0(z,\tau)^2}{\theta_3(z,\tau)^2}$, which we proved while proving the previous theorem. Taking the derivative in z:

$$\widetilde{\wp}'(z,\tau) = -\frac{\theta_3'(0,\tau)^2}{\theta_0(0,\tau)^2} \times \frac{2\theta_3^2(z,\tau)\theta_0'(z,\tau)\theta_0(z,\tau) - 2\theta_0^2(z,\tau)\theta_3'(z,\tau)\theta_3(z)}{\theta_3^4(z,\tau)}$$

Furthermore Equation (A_{10}) in [Mum83, p.26] is:

$$\theta_3(x+u,\tau)\theta_3(x-u,\tau)\theta_0^2(0,\tau) = \theta_3^2(x,\tau)\theta_0^2(u,\tau) - \theta_0^2(x,\tau)\theta_3^2(u,\tau)$$

Take the derivative in u of the latter:

$\theta_3'(x+u,\tau)\theta_3(x-u,\tau)\theta_0^2(0,\tau) - \theta_3(x+u,\tau)\theta_3'(x-u,\tau)\theta_0^2(0,\tau) = 2\theta_3^2(x,\tau)\theta_0'(u,\tau)\theta_0(u,\tau) - 2\theta_0^2(x,\tau)\theta_3'(u,\tau)\theta_3(u,\tau)$

Taking $x = u = z$, we notice that this right-hand side is the numerator which appears in the derivative of $\tilde{\wp}$; hence we have:

$$\widetilde{\wp}'(z,\tau) = -\frac{\theta_3'(0,\tau)^2}{\theta_0(0,\tau)^2} \times \frac{\theta_3'(2z,\tau)\theta_3(0,\tau)\theta_0^2(0,\tau) - \theta_3(2z,\tau)\theta_3'(0,\tau)\theta_0^2(0,\tau)}{\theta_3^4(z)}$$

Using the parity of θ_3 and Jacobi's formula finally gives:

$$\widetilde{\wp}'(z,\tau) = -\pi^3\theta_0^3(0,\tau)\theta_1^3(0,\tau)\theta_2^3(0,\tau)\frac{\theta_3(2z,\tau)}{\theta_3^4(z,\tau)}. \qquad \square$$

This formula shows that one can compute $\wp'$ in the same running time as $\wp$, that is to say $O(\mathcal{M}(P)\log P)$. However, if one has already computed $\wp(z,\Lambda)$ and g_2, g_3, it is more efficient to use the differential equation satisfied by $\wp$:

$$\wp'(z,\Lambda)^2 = 4\wp(z,\Lambda)^3 - g_2\wp(z,\Lambda) - g_3.$$

This gives $\wp'(z,\Lambda)^2$ directly from the knowledge of $\wp(z,\tau)$. The correct sign for $\wp'$ can then be determined by computing low-precision (e.g. 10 significant digits) approximations of θ, then using the formula in Theorem 8.1.6 to compute a low-precision approximation of $\wp'$, which gives the correct sign.

8.1.3 Comparing methods for the computation of $\wp$

We compare here Algorithm 6, which computes $\wp$ using the Landen transform, and the algorithm consisting in using Theorem 8.1.4 with Algorithm 15, which computes $\wp$ via θ. We discuss the precision loss and compare timings.

Precision loss

Recall that we analyzed the precision loss in Algorithm 15, and showed how working with internal precision $2P$ gave the value of θ up to 2^{-P}. However, applying Equation (8.1.1) can cause a potentially large loss of precision. For instance, the fact that $\theta_3(z,\tau) = 0$ means that θ_3 gets arbitrarily close to 0 as z nears the corners of the fundamental parallelogram; and Theorem 0.3.3 shows that division causes a loss of absolute precision proportional to minus the logarithm of the denominator (provided the number of bits lost is less than half of the total number of bits). We make this more precise by determining an equivalent of θ_3 near $z = 0$, using the product expansion of θ (see e.g. [Mum83, Section 14, p. 69]) to show that:

$$\theta_3(z,\tau) \sim -2e^{i\pi\tau/4}\sin(\pi z) \sim -2\pi e^{i\pi\tau/4}z.$$

Hence, we expect a loss of precision proportional to $-\log z$. Similarly, if ω_1 is small, computing $\wp$ from $\tilde{\wp}$ (cf. Theorem 8.1.4) causes a loss of precision proportional to the logarithm of $|\omega_1|$. Hence, for the extreme case $z = 2^{-P}$ and $\omega_1 = 2^{-P}$, we expect a loss of precision proportional to P, which means one would need $O(P)$ guard bits.

As for Algorithm 6, note that, because of Theorem 4.3.2, the algorithm could require computing $\frac{1}{\sin^2(z)}$ for $z \simeq 0$, and $\frac{1}{\omega_1^2}$ for $\omega_1 \simeq 0$. Hence, we expect the algorithm to also lose precision for z close to 0 as well as for small ω_1, which is very similar to the situation for Equation (8.1.1). Again, at worst, we could have $z = 2^{-P}$ and $\omega_1 = 2^{-P}$; this tends to show that only $O(P)$ guard bits are needed. There may also be a loss of precision in the computation of a denominator in the series, which could be in theory close to 0; however, the denominator is small only when $z \simeq \omega_2$, and this condition is easily checked at the beginning of the algorithm: we assume we can avoid this case easily. Overall, a more refined analysis would be needed to confirm the heuristics on precision loss we outline here; one could think for example of looking at the precision loss analysis of Miller's algorithm for Bessel's function [BZ10, p. 153], as they both rely on backwards induction.

We performed experiments to attempt to compare the precision loss in both algorithms by determining the precision lost in the cases mentioned above, i.e. z or ω_1 small. Using Magma, we attempted to compute $\wp(\pi 2^{-t}, [1, i])$ with decimal precision P, then $\wp(\pi, [2^{-t}, 2^t i])$, and $\wp(\pi 2^{-t}, [2^{-t}, 2^t i])$. We determined the loss of precision by comparing the output of an algorithm when run at precision P to the output of the same algorithm ran at a much larger precision. The results are presented in Table 8.1; the computations were performed at precision 10000, and we stopped at $t = 4096$ to avoid the computation of results with a minority of correct digits.

Given the results shown in Table 8.1, we formulate the following conjecture:

Conjecture 8.1.7. *For z, ω_1, ω_2 known with absolute precision P, working with internal precision $2P$ (resp. $3P$) in Algorithm 6 (resp. using Equation (8.1.1)) is enough to guarantee that $\wp(z, [\omega_1, \omega_2])$ is correct up to 2^{-P}.*

Hence, both Algorithm 6 and the algorithm which uses Equation (8.1.1) and Algorithm 15 have complexity $O(\mathcal{M}(P)\log P)$, even when taking the precision loss into account.

t	$\wp(\pi 2^{-t}, [1, i])$		$\wp(\pi, [2^{-t}, 2^t i])$		$\wp(\pi 2^{-t}, [2^{-t}, 2^t i])$	
	Algorithm 6	Equation (8.1.1)	Algorithm 6	Equation (8.1.1)	Algorithm 6	Equation (8.1.1)
16	6	16	15	15	12	12
32	17	35	29	29	22	21
64	37	73	61	60	41	41
128	75	150	117	116	80	79
256	152	304	233	232	157	156
512	307	613	463	463	311	310
1024	615	1227	926	925	619	619
2048	1231	2462	1851	1850	1236	1235
4096	2464	4928	3702	3701	2469	2468

Table 8.1: Precision loss (in decimal digits) when computing $\wp(2^{-t}, [1, i])$.

Timings

We implemented Algorithm 6 in MAGMA, and we also write a MAGMA script which calls our MPC implementation of Algorithm 15 and applies Equation (8.1.1). We then measured the time taken by the computation of $\wp(0.123456789 + 0.123465789i, [1, 0.23456789 + 1.23456789i])$ at various precisions P, using internal precision of P (i.e. without attempting to compensate for the potential precision loss by working at precision $2P$ or $3P$). The time needed by Magma to parse the results given by the MPC program implementing Algorithm 15 is not counted in the running times.

This comparison is somewhat biased against Algorithm 6, as it is implemented in Magma (which has more overhead than MPC) and we do not take into account the fact that the working precision one needs to ensure a correct result up to 2^{-P} is smaller for this algorithm than for the other one. Regardless, the timings presented in Table 8.2 show that Algorithm 6 is faster than the other method.

To introduce a point of comparison with currently used methods, we also measured timings of the PARI/GP [The14] function `ellwp`, by calling ellwp($[1, 0.23456789+1.23456789i], 0.123456789+0.123465789i$); we used PARI/GP because we did not find a function which would perform this computation in MAGMA. The function performing the computation in the library is called `ellwpnum_all`, and performs summation of the series described in [Coh93, Prop. 7.4.4]; this seems to give a $O(\mathcal{M}(P)P)$ algorithm. As Pari is implemented in C, the comparison is once again biased against our Magma implementation of Algorithm 6; however, this algorithm is still faster than the Pari one, even if we were to use internal precision $2P$ to compensate the potential loss of precision. It is likely that an implementation of Coqueraux et al.'s algorithm [CGL90] would yield similar results, since its asymptotic complexity is also greater than the one of our algorithms.

Remark 8.1.8. Both algorithms can be modified to compute n different values $\wp(z_k, [\omega_1, \omega_2])$ faster than by just applying the algorithm n times. Indeed, both algorithms require the computation of theta-constants, which can then be cached and reused. However, the amount of computation saved is hard to estimate, and we do not know which algorithm would end up being faster, as we did not investigate this further.

Prec (decimal digits)	Algorithm 6	Equation (8.1.1) + Algorithm 15	Pari's `ellwp`
2000	0.01	0.03	0.15
4000	0.04	0.10	0.86
8000	0.11	0.29	4.82
16000	0.36	0.82	25.49
32000	1.1	2.20	135
64000	3.35	6.19	677
128000	10	16.7	
256000	25.3	41.8	
512000	61.5	103.5	
1024000	155	243	

Table 8.2: Timings (in s) of different methods to compute $\wp$.

8.1.4 Computing the Abel-Jacobi map

We now consider the problem of computing the Abel-Jacobi map, i.e. given a curve $E(\mathbb{C})$ defined with the equation $y^2 = x^3 + ax + b$, compute:

- a lattice $\Lambda \subset \mathbb{C}$ such that $E(\mathbb{C}) \simeq \mathbb{C}/\Lambda$;
- for any $Q = (x, y) \in E(\mathbb{C})$, its elliptic logarithm z_Q.

Using the Landen isogeny

Both of these problems can be solved directly using the Landen isogeny, as we mentioned in Chapter 4. The paper [BM88] gives relevant formulas, theorems, and algorithms in the case where the roots of the polynomial $4x^3 + ax + b$ are real; the general case is explained in [CT13]. We briefly recall the methods outlined in Chapter 4 for the sake of completeness.

Recall that the Landen isogeny can be used to provide a change of variables in the elliptic integrals that appear in the Abel-Jacobi maps – both to compute the periods (complete elliptic integrals) and the elliptic logarithm (incomplete elliptic integrals). However, recall that the Landen isogeny is defined using a certain numbering of the roots of the polynomial defining the elliptic curve; one needs to pick the correct numbering (ultimately, the right square root) to see the connection with quadratically convergent AGM sequences (e.g. optimal ones, in the sense defined in Section 3.2) and get a sequence of elliptic integrals that is quadratically convergent.

This reduces the computation of the periods to the computation of optimal AGM sequences, as per Theorem 4.2.12, while the computation of the elliptic logarithm is basically the computation of an optimal AGM sequence while keeping track of the integration bounds (see Algorithm 5). The cost of both methods is quasi-optimal, i.e. $O(\mathcal{M}(P) \log P)$.

Dupont's algorithm

Another algorithm, outlined in [Dup06, p. 195], computes τ associated to a given elliptic curve; we discuss it here, also using [Sil86, p.50] to make the link with Thomae's formulas explicit. This algorithm generalizes well to higher genera (cf. Section 8.2 and Section 8.3).

Rewriting the equation defining E as $y^2 = (x - e_1)(x - e_2)(x - e_3)$, we have that E is isomorphic to a curve of the form

$$E' : y^2 = x(x - 1)(x - \lambda)$$

with $\lambda \neq 0, 1$: this is called the *Legendre form*. In fact, following the proof of [Sil86, p.50] shows that

$$\lambda = \frac{e_3 - e_1}{e_2 - e_1}.$$

However, note that this value of λ depends on a choice in the numbering of the roots of the polynomial $x^3 + ax + b$. The values corresponding to the other numberings of the roots are

$$\left\{\lambda, \frac{1}{\lambda}, 1-\lambda, \frac{1}{1-\lambda}, \frac{\lambda}{\lambda-1}, \frac{\lambda-1}{\lambda}\right\}.$$

In Section 4.2, the choice of a numbering of the roots determined the beginning of the short chain of lattices (i.e. the lattices $\Lambda_2 \subset \Lambda_1 \subset \Lambda_0$) or the first two isogenies of the isogeny chain. Each of those led to different periods being computed, and in general led to a different value of τ; there are in general 6 different values of τ one can compute from the elliptic curve E. The link between the 6 values of τ and the 6 values of λ is given by Thomae's formulas (see Theorem 1.3.19), which give $\lambda = \frac{\theta_1(0,\tau)^4}{\theta_0(0,\tau)^4}$.

The algorithm then consists in the following steps:

1. Compute a low-precision approximation of τ corresponding to the curve using an algorithm that evaluates complex integrals, such as Gaussian quadrature (such algorithms are discussed e.g. in [Dup06, Section 9.2.1]);
2. Reduce τ so that it is in the fundamental domain;
3. Evaluate $\frac{\theta_1(0,\tau)^4}{\theta_0(0,\tau)^4}$ with low precision, then compare this value to λ in order to determine the correct numbering of the roots: this gives a high-precision approximation of $\lambda = \frac{\theta_1(0,\tau)^4}{\theta_0(0,\tau)^4}$ with $\tau \in \mathcal{F}$;
4. Use the fact that $\mathrm{Re}\left(\frac{\theta_1(0,\tau)^2}{\theta_0(0,\tau)^2}\right) \geq 0$ (Theorem 6.2.6) to obtain $\frac{\theta_1(0,\tau)^2}{\theta_0(0,\tau)^2}$ with high precision;
5. Use the formula $\tau = \frac{i\,\mathrm{AGM}\left(1, \frac{\theta_1(0,\tau)^2}{\theta_0(0,\tau)^2}\right)}{\mathrm{AGM}\left(1, \sqrt{1-\frac{\theta_1(0,\tau)^4}{\theta_0(0,\tau)^4}}\right)}$, valid for τ in the fundamental domain, to recover τ with high precision.

This algorithm runs in quasi-linear time. It requires two evaluations of the AGM and two square root extractions; this is comparable to the algorithm for computing periods using the Landen transform (Theorem 4.2.12), which requires 3 square roots but outputs ω_1, ω_2.

Computation of the elliptic logarithm

The previous algorithm exploited the explicit link between the curve equation and the theta-constants, then used the AGM (in fact, the function f_τ, on which Newton's method is applied in order to get the theta-constants) to compute τ. We show how a similar idea can be applied to the computation of z, i.e. of the elliptic logarithm. This algorithm can also be generalized to higher genera; see Section 8.2 and Section 8.3.

Let $P = (x, y) \in E(\mathbb{C})$, and denote by z its elliptic logarithm. Theorem 8.1.4 shows that

$$\frac{x}{\omega_1^2} = \frac{\wp(z, [\omega_1, \omega_2])}{\omega_1^2} = \frac{\pi^2}{3}(\theta_2^4(0,\tau) - \theta_1^4(0,\tau)) - \pi^2\theta_1^2(0,\tau)\theta_2^2(0,\tau)\frac{\theta_0^2(z,\tau)}{\theta_3^2(z,\tau)}$$

Hence

$$\frac{\theta_3^2(z,\tau)}{\theta_0^2(z,\tau)} = \frac{\pi^2\theta_1^2(0,\tau)\theta_2^2(0,\tau)}{\frac{\pi^2}{3}(\theta_2^4(0,\tau)-\theta_1^4(0,\tau))-x/\omega_1^2}$$

Furthermore, Equations (2.5.7) and (2.5.8) can be rewritten as

$$\begin{aligned}\frac{\theta_3^2(z,\tau)}{\theta_0^2(z,\tau)} &= \frac{\theta_1^2(z,\tau)}{\theta_0^2(z,\tau)}\frac{\theta_2^2(0,\tau)}{\theta_0^2(0,\tau)} - \frac{\theta_2^2(z,\tau)}{\theta_0^2(z,\tau)}\frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\\ 1 &= \frac{\theta_1^2(z,\tau)}{\theta_0^2(z,\tau)}\frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)} + \frac{\theta_2^2(z,\tau)}{\theta_0^2(z,\tau)}\frac{\theta_2^2(0,\tau)}{\theta_0^2(0,\tau)}\end{aligned}$$

Solving this gives

$$\begin{aligned}\frac{\theta_1^2(z,\tau)}{\theta_0^2(z,\tau)} &= \frac{\frac{\theta_2^2(0,\tau)}{\theta_0^2(0,\tau)}\frac{\theta_3^2(z,\tau)}{\theta_0^2(z,\tau)} + \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}}{\frac{\theta_2^4(0,\tau)}{\theta_0^4(0,\tau)} + \frac{\theta_1^4(0,\tau)}{\theta_0^4(0,\tau)}} = \frac{\theta_2^2(0,\tau)}{\theta_0^2(0,\tau)}\frac{\theta_3^2(z,\tau)}{\theta_0^2(z,\tau)} + \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\\ \frac{\theta_2^2(z,\tau)}{\theta_0^2(z,\tau)} &= \frac{\frac{\theta_2^2(0,\tau)}{\theta_0^2(0,\tau)} - \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\frac{\theta_3^2(z,\tau)}{\theta_0^2(z,\tau)}}{\frac{\theta_2^4(0,\tau)}{\theta_0^4(0,\tau)} + \frac{\theta_1^4(0,\tau)}{\theta_0^4(0,\tau)}} = \frac{\theta_2^2(0,\tau)}{\theta_0^2(0,\tau)} - \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\frac{\theta_3^2(z,\tau)}{\theta_0^2(z,\tau)}\end{aligned}$$

We thus get the quotients $\frac{\theta_1^2(z,\tau)}{\theta_0^2(z,\tau)}$; since the problem is to compute z, a natural idea is then to apply the $\mathfrak{F}$ function that we defined in Section 6.3.1 (Algorithm 13), which gives precisely this. We summarize the algorithm in Algorithm 21.

Algorithm 21 Quasi-linear algorithm to compute the elliptic logarithm of $P \in E(\mathbb{C})$ with absolute precision P, using $\mathfrak{F}$.
Input: $P \in E(\mathbb{C})$, $\tau = \frac{\omega_2}{\omega_1}$, with absolute precision P.

1: Compute ω_1 using the AGM (Theorem 4.2.12).
2: Compute $\frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}$, using e.g. the previous section for a cost of $O(\mathcal{M}(P))$ operations.
3: Compute $\frac{1}{\theta_0^2(0,\tau)} = \mathrm{AGM}\left(1, \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right)$, and deduce $\theta_0^2(0,\tau), \theta_1^2(0,\tau)$.
4: Compute $\theta_2^2(0,\tau)$ using Jacobi's formula (Equation 2.5.6).
5: $q_3 \leftarrow \frac{\pi^2\theta_1^2(0,\tau)\theta_2^2(0,\tau)}{\frac{\pi^2}{3}(\theta_2^4(0,\tau)-\theta_1^4(0,\tau))-x/\omega_1^2}$
6: $q_1 \leftarrow \frac{\theta_2^2(0,\tau)}{\theta_0^2(0,\tau)}q_3 + \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}$
7: $(z,\tau) \leftarrow \mathfrak{F}\left(1, q_1, 1, \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right)$
8: Compute a low-precision approximation of $\wp'(z,[\omega_1,\omega_2])$, using e.g. Theorem 8.1.6; if $\wp'(z,[\omega_1,\omega_2]) = -y$, set $z \leftarrow -z$.
9: **return** z.

The cost of this algorithm is roughly the cost of applying two AGMs, then applying $\mathfrak{F}$. On the other hand, [CT13, Algorithm 28] (i.e. Algorithm 5 of Section 4.2.4) requires an Arctan computation (performed using the AGM), as well as an extra inversion and square root extraction (and a few multiplications) for every AGM iteration. We implemented Algorithm 21 and [CT13, Algorithm 28] in MAGMA; some timings are presented in Table 8.3, and show that the method presented in this section is slower than the method of [CT13] by a factor of roughly 2.5.

Prec (decimal digits)	[CT13, Algorithm 28] (Algorithm 5 of Section 4.2.4)	Algorithm 21
2000	0.02	0.03
4000	0.03	0.06
8000	0.07	0.19
16000	0.24	0.56
32000	0.74	1.74
64000	2.7	5.3
128000	6	16
256000	15	39
512000	37	101
1024000	98	256

Table 8.3: Timings (in s) of different methods to compute the elliptic logarithm

8.2 In genus 2

Fast computation of the Abel-Jacobi map in genus 2 is attainable given the state of the art and the algorithms presented in this manuscript. As for the computation of its inverse, we proved in Chapter 7 that there was a $O(\mathcal{M}(P)\log P)$ algorithm for the computation of $\theta(z,\tau)$ with precision P, provided Conjecture 7.2.6 holds; we show how one can use this, and obtain a $O(\mathcal{M}(P)\log P)$ algorithm under Conjecture 7.2.6.

8.2.1 Computing the Abel-Jacobi map

Going from the algebraic representation to the analytic representation involves computing periods and computing the genus 2 hyperelliptic logarithm. This corresponds to the problem of evaluating hyperelliptic integrals of the form

$$\int_x^y \frac{dx}{\sqrt{P(x)}}, \qquad \deg P = 6$$

This problem is discussed in the case where P has real roots in [BM88]: much as in the genus 1 case (see Chapter 4), there is a change of variables which corresponds to an isogeny between hyperelliptic curves, and which is associated to quadratically convergent sequences. Hence, after $O(\log P)$ steps, the integral can be evaluated easily, which gives a $O(\mathcal{M}(P)\log P)$ algorithm to compute complete integrals (and hence periods), and a similar one to compute incomplete integrals (and hence the hyperelliptic logarithm). However, to the best of our knowledge, this algorithm has not been generalized to the general case, unlike in genus 1 where [CT13] extended the algorithms of [BM88] to the complex case.

An easier way could be to proceed as in Section 8.1.4. This method has the advantage of being generalizable to higher genus; we outline the method here, and refer to the genus g algorithms (Algorithm 22 and Algorithm 23) for the full method.

First of all, as described in [Dup06, Chapitre 9], one can compute the genus 2 periods by computing theta-constants using Thomae's formulas and the Borchardt mean, then applying the Borchardt mean to appropriately chosen quotients; in fact, we can use the same ones as in Section 7.2.2. As in genus 1, the ordering of the roots has its importance in Thomae's formula, but this does not affect the running time. This allows one to recover τ from the equation of the curve in $O(\mathcal{M}(P)\log P)$ provided theta-constants can be computed in this amount of time (i.e.

provided the Jacobian of the system is invertible, which is covered by Conjecture 7.2.6). We refer to [Dup06, Chapitre 9] for a more precise exposition of the algorithm in this case.

As for the computation of the hyperelliptic logarithm, we can use the work of Cosset [Cos11, Chapter 5], which allows one to go from theta coordinates to Mumford coordinates, using explicit formulas which are valid in any genus but require the computation of theta-constants. We can thus retrieve the value of quotients of theta function corresponding to the Mumford coordinates of the divisor, then apply the function $\mathfrak{F}$ we defined in Section 7.2.2 to appropriately chosen quotients, which allows us to compute λ_1 and λ_2, and in the end z. All in all, this gives a $O(\mathcal{M}(P)\log P)$ algorithm to compute the periods and the hyperelliptic logarithm, provided Conjecture 7.2.6 holds.

8.2.2 Computing the inverse of the Abel-Jacobi map

Given the periods of the lattice representing the Jacobian, computing the coefficients of a hyperelliptic equation corresponding to the Jacobian seems feasible using Thomae's formula, or, more precisely, formulas derived from them sometimes called the "Umemura forumas" or the "reverse Thomae formulas"; these formulas express quotients $\frac{a_k-a_i}{a_k-a_j}$ as a function of the (squares of) theta constants of characteristic 2 associated to the curve. We refer to [Mum84, IIIc] or [Cos11, Theorem 3.1.20 and p.151] for a discussion of these formulas. In genus 2, it is possible to compute a model of the curve in several different ways; see [CDSLY14] for a summary of these models. For instance, the model outlined in [Gau07] or [Cos11] gives:

$$\begin{aligned} &C : y^2 = x(x-1)(x-\lambda_1)(x-\lambda_2)(x-\lambda_3) \\ \text{with} \quad &\lambda_1 = \frac{\theta_0^2(0,\tau)\theta_2^2(0,\tau)}{\theta_3^2(0,\tau)\theta_1^2(0,\tau)}, \qquad \lambda_2 = \frac{\theta_2^2(0,\tau)\theta_{12}^2(0,\tau)}{\theta_1^2(0,\tau)\theta_{15}^2(0,\tau)}, \qquad \lambda_3 = \frac{\theta_0^2(0,\tau)\theta_{12}^2(0,\tau)}{\theta_3^2(0,\tau)\theta_{15}^2(0,\tau)} \end{aligned}$$

Note that the denominators are never zero since, in genus 2, we have $\theta_i(0,\tau) \neq 0 \Leftrightarrow \theta_i$ is even, which is true of $\theta_1, \theta_3, \theta_{15}$.

One can also, given a point z in the lattice corresponding to the Jacobian, compute the Mumford coordinates of the divisor which corresponds to it by the Abel-Jacobi map. The method is simply to evaluate the theta functions at z and τ, then use [Cos11, Chapter 5] to recover the Mumford coordinates of the divisor. The evaluation of θ can be done in $O(\mathcal{M}(P)\log P)$, provided Conjecture 7.2.6 holds; hence the inverse of the Abel-Jacobi map can be computed in quasi-linear time under this conjecture.

8.3 Extending the strategy to higher genus

Finally, we give a brief outline of how one could extend the strategy to the computation of the Abel-Jacobi map in genus g.

8.3.1 Computing the Abel-Jacobi map

We look at the problem of computing the period matrix τ and hyperelliptic logarithms of points of a hyperelliptic curve given by an equation $y^2 = f(x)$. We follow and generalize the approach of [Dup06, Section 9.2.3], using for instance the same matrices.

Define $\mathfrak{J} = \begin{pmatrix} 0 & -I_g \\ I_g & 0 \end{pmatrix} \in Sp_{2g}(\mathbb{Z})$ and

$$\mathfrak{M}_j = \begin{pmatrix} I_g & \delta_{j,j} \\ 0 & I_g \end{pmatrix} \in \mathrm{Sp}_{2g}(\mathbb{Z}), \qquad \mathfrak{M}_{j,k} = \begin{pmatrix} I_g & \delta_{j,k}+\delta_{k,j} \\ 0 & I_g \end{pmatrix} \in \mathrm{Sp}_{2g}(\mathbb{Z})$$

then $$(\mathfrak{J}\mathfrak{M}_j)^2 = \begin{pmatrix} -I_g & -\delta_{j,j} \\ \delta_{j,j} & \delta_{j,j}-I_g \end{pmatrix}, \qquad (\mathfrak{J}\mathfrak{M}_{j,k})^2 = \begin{pmatrix} -I_g & -\delta_{j,k}-\delta_{k,j} \\ \delta_{j,k}+\delta_{k,j} & \delta_{j,j}+\delta_{k,k}-I_g \end{pmatrix}$$

We then have by Theorem 2.4.4:

$$\begin{aligned} \theta_0^2\left((\mathfrak{J}\mathfrak{M}_j)^2\cdot z, (\mathfrak{J}\mathfrak{M}_j)^2\cdot\tau\right) &= \zeta_j\tau_{j,j}e^{i\pi(-1)^g 2z_j^2/\tau_{j,j}}\theta^2_{\sigma_j(0)}(z,\tau) \\ \theta_0^2\left((\mathfrak{J}\mathfrak{M}_{j,k})^2\cdot z, (\mathfrak{J}\mathfrak{M}_j)^2\cdot\tau\right) &= \zeta_{j,k}\left(\tau_{j,k}^2-\tau_{j,j}\tau_{k,k}\right)f(z,\tau)\theta^2_{\sigma_{j,k}(0)}(z,\tau) \end{aligned} \tag{8.3.1}$$

with ζ_j and ζ_k are fourth roots of unity.

For $z=0$ we find the equations of [Dup06, p. 202]:

$$\begin{aligned} \theta_0^2\left(0,(\mathfrak{J}\mathfrak{M}_j)^2\cdot\tau\right) &= \zeta_j\tau_{j,j}\theta^2_{\sigma(0)}(0,\tau) \\ \theta_0^2\left(0,(\mathfrak{J}\mathfrak{M}_{j,k})^2\cdot\tau\right) &= \zeta_{j,k}\left(\tau_{j,k}^2-\tau_{j,j}\tau_{k,k}\right)\theta^2_{\sigma'(0)}(0,\tau). \end{aligned}$$

These formulas allows one to recover the coefficients of τ following the algorithm presented in [Dup06]. This is Algorithm 22, which computes the $\frac{g(g+1)}{2}$ coefficients of τ from the 2^g-1 quotients of squares of theta-constants.

Since the Borchardt mean converges quadratically, Algorithm 22 requires $O(\mathcal{M}(P)\log P)$ operations per coefficient.

Computing the hyperelliptic logarithm can be done using only Equation (8.3.1), i.e. the action of $(\mathfrak{J}\mathfrak{M}_j)^2$ on the values of the theta functions. The algorithm is similar to the previous one; we present it in Algorithm 23.

Algorithm 22 Compute the period matrix τ associated to a hyperelliptic curve $\mathcal{C}$ of genus g.
Input: the equation of a curve $\mathcal{C} : y^2 = \prod_{i=1}^{2g+2}(x - x_i)$, and more precisely the roots x_i with absolute precision P.
Output: $\tau \in \mathcal{H}_g$ with absolute precision P.

1: Compute τ with low precision (e.g. a few dozen bits) by evaluating the complete hyperelliptic integral with low precision.
2: Use Thomae's formulas (see [Cos11, Théorème 3.1.19] or [Mum84, Section 8]) to compute the fourth powers of quotients of the squares of theta-constants associated to the curve.
3: Use the low-precision approximation of τ to extract the correct square root and compute $\left(\frac{\theta_1(0,\tau)^2}{\theta_0(0,\tau)^2}, \ldots, \frac{\theta_{2^g-1}(0,\tau)^2}{\theta_0(0,\tau)^2}\right)$.
4: Compute low-precision approximations of $\theta_i(0, 2^k\tau)$; use these to pick the correct sign (the one corresponding to the values of theta-constants) in the Borchardt mean of $\left(\frac{\theta_1(0,\tau)^2}{\theta_0(0,\tau)^2}, \ldots, \frac{\theta_{2^g-1}(0,\tau)^2}{\theta_0(0,\tau)^2}\right)$. This gives $\frac{1}{\theta_0(0,\tau)^2}$.
5: For $i = 1, \ldots, 2^g - 1$, compute $\theta_i^2(0,\tau) = \frac{\theta_i^2(0,\tau)}{\theta_0^2(0,\tau)} \times \theta_0(0,\tau)^2$.
6: Use τ-duplication formulas to compute $\theta_i^2(0, 2\tau)$ for $i > 2^g - 1$.
7: **for** $j = 1...g$ **do**
8: Compute $\mathcal{B}_g\left(1, \frac{\theta^2_{\sigma_j(1)}(0,2\tau)}{\theta^2_{\sigma_j(0)}(0,2\tau)}, \ldots, \frac{\theta^2_{\sigma_j(2^g-1)}(0,2\tau)}{\theta^2_{\sigma_j(0)}(0,2\tau)}\right) = \frac{1}{\theta_0\left(0,(\mathfrak{Im}_j)^2\cdot 2\tau\right)}$; retrieve $\tau_{j,j}$.
9: **end for**
10: **for** $j, k = 1...g$ **do**
11: Compute $\mathcal{B}_g\left(1, \frac{\theta^2_{\sigma_{j,k}(1)}(0,2\tau)}{\theta^2_{\sigma_{j,k}(0)}(0,2\tau)}, \ldots, \frac{\theta^2_{\sigma_{j,k}(2^g-1)}(0,2\tau)}{\theta^2_{\sigma_{j,k}(0)}(0,2\tau)}\right) = \frac{1}{\theta_0\left(0,(\mathfrak{Im}_j)^2\cdot 2\tau\right)}$; retrieve $\tau_{j,k}$.
12: **end for**
13: **return** τ.

Algorithm 23 Compute the hyperelliptic logarithm of a divisor on a genus g hyperelliptic curve.
Input: the equation of a curve $\mathcal{C} : y^2 = \prod_{i=1}^{2g+2}(x - x_i)$, the Mumford coordinates of the divisor D, the period matrix τ associated to $\mathcal{C}$. All these quantities are with absolute precision P.
Output: $z \in \mathbb{C}^g$ with absolute precision P.

1: Compute the quotients of squares of theta-constants using Thomae's formulas, as in Algorithm 22.
2: Use these quotients and the formulas in [Cos11, Section 5.3] to compute the quotients of the theta functions corresponding to D, and in particular $\left(\frac{\theta_1(z,\tau)^2}{\theta_0(z,\tau)^2}, \ldots, \frac{\theta_{2^g-1}(z,\tau)^2}{\theta_0(z,\tau)^2}\right)$.
3: Compute z with low precision (e.g. a few dozen bits) by evaluating the incomplete hyperelliptic integral with low precision.
4: Compute low-precision approximations of $\theta_i(z, 2^k\tau)$; use these to pick the correct sign (the one corresponding to the values of theta functions) in the computation of $\mathfrak{G}\left(1, \frac{\theta_1(z,\tau)^2}{\theta_0(z,\tau)^2}, \ldots, \frac{\theta_{2^g-1}(z,\tau)^2}{\theta_0(z,\tau)^2}, 1, \frac{\theta_1(0,\tau)^2}{\theta_0(0,\tau)^2}, \ldots, \frac{\theta_{2^g-1}(0,\tau)^2}{\theta_0(0,\tau)^2}\right)$. This gives $\frac{1}{\theta_0(z,\tau)^2}$.
5: For $i = 1, \ldots, 2^g - 1$, compute $\theta_i^2(z,\tau) = \frac{\theta_i^2(z,\tau)}{\theta_0^2(z,\tau)} \times \theta_0(z,\tau)^2$.
6: Use τ-duplication formulas to compute $\theta_i^2(z, 2\tau)$ for $i > 2^g - 1$.
7: **for** $j = 1...g$ **do**
8: Compute z_j by computing $\mathfrak{G}\left(1, \frac{\theta^2_{\sigma_j(1),\ldots,\sigma_j(2^g-1)}}{\theta^2_{\sigma_j(0)}}(z, 2\tau), 1, \frac{\theta^2_{\sigma_j(1),\ldots,\sigma_j(2^g-1)}}{\theta^2_{\sigma_j(0)}}(0, 2\tau)\right)$, which gives $\frac{1}{\theta_0^2\left((\mathfrak{J}\mathfrak{M}_j)^2\cdot z, (\mathfrak{J}\mathfrak{M}_j)^2\cdot 2\tau\right)}$.
9: **end for**
10: **return** z.

Theorem 8.3.1. *The Abel-Jacobi map can be computed in $O(\mathcal{M}(P)\log P)$ bit operations.*

Proof. The complexity of the computation is dominated by the cost of evaluating $\mathfrak{G}$. Note that $\mathfrak{G}$ is always evaluated at quotients of theta functions, which means that the sequences $a_i^{(n)}$ necessarily converge quadratically, and hence that the evaluation requires only $O(\mathcal{M}(P)\log P)$ bit operations. This means that each coefficient can be computed in $O(\mathcal{M}(P)\log P)$. □

8.3.2 Computing the inverse of the Abel-Jacobi map

Going from the analytical representation (points on the torus) to the algebraic representation (Mumford coordinates of the corresponding divisor) can, once again, be done *via* the theta functions; the formulas in [Cos11] allow one to compute Mumford coordinates from the theta coordinates (i.e. values of quotients of theta functions). The problem thus reduces to that of computing genus g theta functions. As we outlined in Section 7.4, the strategy which gives quasi-linear time evaluation of theta functions in genus 1 and 2 may be generalizable to arbitrary genera, but there are a few obstacles which prevent this generalization to be completely straightforward.

The first obstacle is that we need a proof that the $\mathfrak{G}$ function can be evaluated in quasi-linear time *for any arguments*. Note that, by the arguments of Proposition 2.1.9, the sequences $\theta(z, 2^k\tau)^{2^k}$ converge quadratically, which means that $\mathfrak{G}\left(1, \frac{\theta^2_{1,\ldots,2^g-1}(z,\tau)}{\theta_0^2(z,\tau)}, 1, \frac{\theta^2_{1,\ldots,2^g-1}(0,\tau)}{\theta_0^2(0,\tau)}\right)$ can be evaluated in quasi-linear time. We simply need to prove that this running time holds for any arguments, or at least for arguments which are at a distance ϵ from quotients of thetas; this is needed to prove that the evaluation of the Jacobian matrix using finite differences can be done in quasi-linear time. We think is a reasonable assumption to make, the biggest obstacle to a

Genus	Number of arguments of $\mathfrak{F}$	Number of pairs (λ, μ)	
1	2	1	✓
2	6	3	✓
3	14	6	✗
g	$2^{g+1}-2$	$\frac{g(g+1)}{2}$	✗

Table 8.4: Table highlighting the number of arguments $\mathfrak{F}$ (and $\mathfrak{G}$) takes (i.e. the number of quotients $\frac{\theta_i^2}{\theta_0^2}(z,\tau)$, $\frac{\theta_i^2}{\theta_0^2}(0,\tau)$ of fundamental thetas), versus the number of λ, μ one can compute with $\mathfrak{G}$ by using the action of $(\mathfrak{JM}_j)^2$ and $(\mathfrak{JM}_{j,k})^2$ on the quotients. If both numbers do not match, one needs to consider the action of more matrices on the quotients.

proof being that our proof in genus 1 (Section 6.2.4) or 2 (Section 7.2.3) is very technical and thus tricky to generalize to arbitrary genus.

The biggest obstacle is to manage to define a function $\mathfrak{F} : \mathbb{C}^{2^{g+1}-2} \to \mathbb{C}^{2^{g+1}-2}$ with an invertible Jacobian from the function $\mathfrak{G}$ such that Equation (7.4.2) holds. In genus 1 and 2, this was achieved by computing the value of $\mathfrak{G}$ at several different quotients, corresponding to the value of fundamental theta functions under the action of different matrices: we considered the action of S in genus 1, and the action of $(\mathfrak{JM}_1)^2, (\mathfrak{JM}_2)^2$ and $(\mathfrak{JM}_{1,2})^2$ in genus 2. Note that we did not manage to prove that this approach yielded invertible Jacobians in genus 2. In higher genus, the approach of using the action of $(\mathfrak{JM}_j)^2$ and $(\mathfrak{JM}_{j,k})^2$ to define $\mathfrak{F}$ is not sufficient, as highlighted by Table 8.4; we need to consider a larger set of matrices, so that the dimensions match.

Another approach was to include the equations defining the theta variety to the parameters on which Newton's method should be applied – i.e. make the results output by $\mathfrak{F}$ contain $f_i(X)$ for each equation $f_i(X) = 0$ defining the theta variety. For this method to be effective, one has to find these equations explicitly; and once again, this does not appear to yield straighforwardly a proof that the Jacobian is invertible.

Assuming these problems can be solved, we would get a $O(\mathcal{M}(P)\log P)$ algorithm for the computation of the inverse of the genus g Abel-Jacobi map.

8.4 Interlude: a faster algorithm to compute $E_{2k}(\tau)$

This section is unrelated to the rest of this thesis; however, it describes a result that does not appear to be known yet. Recall the definition of the normalized Eisenstein series of weight $2k$:

$$E_{2k}(\tau) = \frac{1}{2\zeta(2k)} \sum_{\substack{\omega \in \mathbb{Z}+\tau\mathbb{Z} \\ \omega \neq 0}} \frac{1}{\omega^{2k}}$$

We look here at the problem of computing $E_{2k}(\tau)$ for $\tau \in \mathcal{F}$ with absolute precision P.

8.4.1 Naive algorithm for the Eisenstein series

Putting $q = e^{i\pi\tau}$, we can start from the expression of Eisenstein series as a function of the divisor function (see e.g. [Mum83, Section 15] or [Cha85, Section VI.2]) and rewrite the sum as

a Lambert series (e.g. [AS64, Section 24.3.3]). This yields:

$$E_{2k}(\tau) = 1 + \frac{2}{\zeta(1-2k)} \sum_{n \geq 0} \frac{n^{2k-1} q^{2n}}{1-q^{2n}}$$

A naive algorithm to compute $E_{2k}(\tau)$ with absolute precision P is to evaluate $\zeta(1-2k)$ and the series with sufficient precision.

Note that $\zeta(1-2k) = -\frac{B_{2k}}{2k}$, and

$$\frac{2k}{B_{2k}} \sim \frac{(\pi e)^{2k}}{2\sqrt{\pi} k^{2k-1/2}}.$$

This estimate will be useful later.

As for the series, we write

$$\begin{aligned} |\sum_{n \geq B} \frac{n^{2k-1} q^{2n}}{1-q^{2n}}| &\leq \sum_{n \geq B} \frac{n^{2k-1} |q|^{2n}}{|1-q^{2n}|} \\ &\leq \sum_{n \geq B} \frac{n^{2k-1} |q|^{2n}}{1-|q|^{2n}} \leq \frac{1}{1-|q|^{2B}} \sum_{n \geq B} n^{2k-1} |q|^{2n} \\ &\leq 2 \sum_{n \geq B} e^{(2k-1)\log n - 2n\pi \operatorname{Im}(\tau)} \end{aligned}$$

For τ, k fixed, the function $f : n \mapsto \frac{2}{\zeta(1-2k)} n^{2k-1} |q|^{2n}$ is at first increasing, then decreases to 0. We look at its derivative:

$$(2k-1) n^{2k-2} |q|^{2n} - n^{2k-1} 2 \operatorname{Im}(\tau) \pi |q|^{2n} = 0 \Leftrightarrow (2k-1) = n 2 \operatorname{Im}(\tau) \pi$$

Put $N_0 = \frac{2k-1}{2\pi \operatorname{Im}(\tau)}$; then f is increasing when $n < N_0$, and decreasing for $n > N_0$. Its maximal value is then

$$\begin{aligned} f(N_0) &= \frac{8k}{B_{2k}} e^{(2k-1)(\log\left(\frac{2k-1}{2\pi \operatorname{Im}(\tau)}\right)-1)} \\ &\leq e^{2k\log(\pi)+2k+2\log 2-\frac{1}{2}\log\pi-(2k-1/2)\log k+(2k-1)\log(k/\pi \operatorname{Im}(\tau))} \\ &\leq e^{2k-(2k-1)\log(\operatorname{Im}(\tau))} \leq e^{(2-\sqrt{3})k} \end{aligned}$$

if we suppose that $\tau \in \mathcal{F}$. This also means that, in order for the result to be accurate up to 2^{-P}, we need to work with $O(P+k)$-bit numbers, to take into account the maximal size of the integral part and the precision loss.

The convergence of the sequence $(f(n))_{n \in \mathbb{N}}$ is somewhat geometric; the ratio rule gives

$$|q|^2 \left(1 + \frac{1}{n}\right)^{2k-1} \to_{n \to \infty} |q|^2 < 1$$

Suppose that $(f(n))_{n > N_0}$ decreases in a geometric fashion, of a factor $|q|^2$ for each term (which is better than what happens in practice). We can then estimate the number of terms needed before $f(n) \leq 2^{-P}$: the function rises for $n < N_0$, then decreases to 1, then decreases to 2^{-P}. This means that $f(n) \leq 2^{-P}$ for

$$n = O(N_0 + \log(f(N_0))/\operatorname{Im}(\tau) + P/\operatorname{Im}(\tau)) = O\left(\frac{k\left(\frac{1}{\pi} + 2 - \sqrt{3}\right)}{\operatorname{Im}(\tau)} + \frac{P}{\operatorname{Im}(\tau)}\right) = O\left(\frac{P+k}{\operatorname{Im}(\tau)}\right)$$

We can actually bound the size of the remainder of the series by bounding the ratio between two terms. We use

$$(1+1/n)^k = e^{k\log(1+1/n)} \leq e^{k/n} \leq e^{k/B}$$

and put $x = |q|^2 e^{(2k-1)/B}$, so that

$$\begin{aligned} |\sum_{n\geq B} \frac{n^{2k-1}q^{2n}}{1-q^{2n}}| &\leq \frac{1}{1-|q|^{2B}} \sum_{n\geq B} n^{2k-1}|q|^{2n} \\ &\leq \frac{B^{2k-1}|q|^{2B}}{1-|q|^{2B}}(1+x+x^2+...) \\ &\leq \frac{B^{2k-1}|q|^{2B}}{1-|q|^{2B}} \frac{1}{1-x} \end{aligned}$$

We can suppose that $B > 2N_0 = \frac{2k-1}{\pi \operatorname{Im}(\tau)}$, which means that $x \leq |q|$. Since $e^{-t}-1 = -t+\frac{t^2}{2!}-... \geq -t$, we have $\frac{1}{1-x} \leq \frac{1}{1-|q|} \leq 1.08$, since we suppose that $\tau \in \mathcal{F}$. Hence

$$\frac{2}{\zeta(1-2k)}|\sum_{n\geq B} \frac{n^{2k-1}q^{2n}}{1-q^{2n}}| \leq 2f(B)$$

We can then put everything together:

$$\begin{aligned} \text{Approximate } E_{2k}(\tau) \text{ up to } 2^{-P} &\Leftarrow \frac{2}{|\zeta(1-2k)|}|\sum_{n\geq B} \frac{n^{2k-1}q^{2n}}{1-q^{2n}}| \leq 2^{-P} \\ &\Leftarrow f(B) \leq 2^{-P-1} \\ &\Leftarrow B = O\left(\frac{P+k}{\operatorname{Im}(\tau)}\right). \end{aligned}$$

The total cost of the method breaks down as follows: compute B_{2k}, then compute the first B terms of the series. Recall that we need to work with numbers of size $O(P+k)$ bits in order to get a result that is accurate to 2^{-P}; hence, each multiplication has a cost of $O(\mathcal{M}(P+k))$. We estimate the cost of each of those steps.

Computing the Bernoulli numbers

Note that the Bernoulli numbers B_{2k} are fractions with numerator and denominators of size $O(k\log k)$. Hence, we do not need to work with $O(P+k)$-bit approximations of those numbers: it is more efficient (provided $P \gg k\log k$) to compute the numerator and denominators, using operations on numbers of size $O(k\log k)$ (e.g. with integral part of maximal size $2k\log k$ and fractional part of maximal size $2k\log k$), then compute a P-bit approximation of the fraction (if needed) with simply a division, of cost $O(\mathcal{M}(P+k))$.

The first way to compute B_{2k} is to use the techniques described in [BZ10, Section 4.7.2]. The *scaled Bernoulli numbers* $C_{2k} = \frac{B_{2k}}{(2k)!}$ can be evaluated with absolute precision P using a recurrence relation [BZ10, Eq. 4.60], for a cost of $O(\mathcal{M}(P+\log k)k^2)$ bit operations; here, this gives a cost of $O(\mathcal{M}(k\log k)k^2)$ bit operations. Recovering the B_{2k} then requires the evaluation of a factorial, which only takes $O(k)$ multiplications of numbers of maximal size $O(k\log k)$ (using the well-known equivalent $\log k! \simeq k\log k$); this step has negligible cost. The number of guard bits one must take to compensate this multiplication by a factorial is $O(k\log k)$; this still gives a final cost of $O(\mathcal{M}(k\log k)k^2) = O(k^{3+\epsilon})$.

An improvement over this algorithm is outlined in [BH13] (or alternatively in [BZ10, Exercise 4.41]); it uses an equality between the generating function for Bernoulli numbers and the power series corresponding to $\frac{x}{e^x-1}$, then uses fast algorithms on power series to compute the first k terms of the generating function. The cost of this method is $O(k^2 \log^{2+\epsilon} k) = O(k^{2+\epsilon})$ bit operations.

Finally, we note that Harvey gave in [Har14] an algorithm which has the currently best running time to recover B_{2k}; the algorithm requires $O(k^{4/3+\epsilon})$ bit operations to compute a single B_{2k}.

Note that the first (resp. the second) method actually computes the $\frac{B_{2k}}{(2k)!}$ (resp. $\frac{B_{2k}}{k!}$) for all $k' \leq k$. Hence, given that we likely compute factorials using an iterative algorithm, we can easily transform these methods in an algorithm that computes $B_{2k'}$ for $k' \leq k$ in the same amount of time. This gives, for the second method, an amortized cost of $O(k^{1+\epsilon})$, which is better than using the algorithm [Har14].

Computing the terms of the series

We can compute q^n inductively, and hence the computation of each $\frac{q^{2n}}{1-q^{2n}}$ costs only $O(\mathcal{M}(P+k))$ bit operations. However, the computation of n^{2k-1} is more costly: it requires $O(k)$ multiplications with a naive method, or $O(\log k)$ multiplications with fast exponentiation. Furthermore, the multiplication $n^{2k-1} \times \frac{q^{2n}}{1-q^{2n}}$ induces a loss of precision, since n^{2k-1} is large; for the result to still be accurate to 2^{-P}, we need $O(k \log B) = O\left(k \log\left(\frac{P}{\mathrm{Im}(\tau)}\right)\right)$ guard bits. Hence, the computation of each term of the series requires $O(\mathcal{M}\left(P + k \log\left(\frac{P}{\mathrm{Im}(\tau)}\right)\right) \log k)$ bit operations.

Total cost

Hence, the total cost of the naive algorithm to compute $E_{2k}(\tau)$ with absolute precision P is

$$O\left(\mathcal{M}\left(P + k \log\left(\frac{P}{\mathrm{Im}(\tau)}\right)\right)\left(\frac{P+k}{\mathrm{Im}(\tau)}\right) \log k + k^{4/3+\epsilon}\right) = O\left((P+k)^{2+\epsilon} \log k\right).$$

8.4.2 An algorithm based on the coefficients of the series expansion of $\wp$

We now show a faster algorithm to compute $E_{2k}(\tau)$ with high precision, based on the link between the Laurent series expansion of $\wp$ and the values E_{2k}. We have:

Theorem 8.4.1 ([Sil86, Theorem VI.3.5, p. 169])**.** *Write*

$$\wp(z) = \frac{1}{z^2} + \sum_{n=0}^{\infty} b_n z^{2n}$$

Then we have

$$b_n = (2n+1) \sum_{\omega \in L} \frac{1}{\omega^{2n+2}} = \frac{2n+1}{2\zeta(2n)} E_{2n+2}(\tau)$$

Furthermore, there is an algorithm to compute b_n knowing g_2 and g_3, the coefficients in the differential equation of $\wp$:

Theorem 8.4.2 ([BMSS08, Theorem 1]). *There is an algorithm that computes the first n coefficients of the Laurent series of $\wp$ from the knowledge of g_2, g_3; this algorithm performs $O(\mathcal{M}(n))$ operations.*

Section 8.1.1 shows how to compute g_2, g_3 from the theta-constants; Chapter 6 gives an algorithm for theta-constants which complexity is $O(\mathcal{M}(P)\log P)$. Hence, it is possible to compute the first k coefficients of the Laurent series of $\wp$ with precision P in $O(\mathcal{M}(P)\log P + \mathcal{M}(k))$.

As for the computation of $\zeta(2k)$, one can use for instance the FEE method of Karatsuba; as described in [Kar95], one value of $\zeta(2k)$ can be computed with precision P in $O(\mathcal{M}(P)\log^2 P) = O(P^{1+\epsilon})$. However, we were not able to determine the dependency in k. A possibly better way would be to use the connection with Bernoulli numbers

$$\zeta(2k) = \frac{(-1)^{k+1}(2\pi)^{2k}}{2(2k)!}B_{2k}$$

then use the results mentioned in Section 8.4.1: this gives a $O(k\mathcal{M}(P) + k^{4/3+\epsilon})$ algorithm to compute one value of $\zeta(2k)$, and a $O(k\mathcal{M}(P) + k^{2+\epsilon})$ algorithm to compute all values $\zeta(2k')$ for $k' \leq k$ (which is useful in order to recover all $E_{2k'}$ from the first k coefficients of the Laurent series of $\wp$). If we assume that P is larger than k, or even than $k\log k$, this gives a $O(\mathcal{M}(P)(\log P + k))$ algorithm for either task.

In the end, we propose Algorithm 24, which condenses two algorithms: one to compute only $E_{2k}(\tau)$, another one to compute $E_{2k'}(\tau)$ for all $k' \leq k$.

Algorithm 24 Compute Eisenstein series with absolute precision P.
Input: $k \in \mathbb{N}$, $\tau \in \mathcal{F}$ with absolute precision P.
Output: $E_{2k}(\tau)$ up to 2^{-P} [respectively $E_{2k'}(\tau)$ up to 2^{-P} for all $k' \leq k$].

1: Compute $\theta_0(0,\tau), \theta_1(0,\tau), \theta_2(0,\tau)$ using Algorithm 11.
2: Compute g_2, g_3 using Proposition 8.1.3.
3: Compute the first k coefficients of the Laurent series of $\wp$ using the algorithm of [BMSS08].
4: Compute $\zeta(2k)$, using [Har14] [respectively, compute all $\zeta(2k')$ for $k' \leq k$ using [BH13]].
5: Return $E_{2k}(\tau)$ [respectively, return $E_{2k'}(\tau)$ for $k' \leq k$].

If one uses this algorithm to compute one value of $E_{2k}(\tau)$, the running time one can achieve is $O(\mathcal{M}(P)(\log P + \mathcal{M}(k)) + k^{4/3+\epsilon})$; if it is used to compute all values $E_{2k'}(\tau)$ for $k' \leq k$, its running time is $O(\mathcal{M}(P)(\log P + \mathcal{M}(k)) + k^{2+\epsilon})$.

8.4.3 Comparison

We assume that $\tau \in \mathcal{F}$, and distinguish whether we want to compute one value $E_{2k}(\tau)$ of the Eisenstein series, or all the values $E_{2k'}(\tau), k' \leq k$. Since both algorithms require the computation of Bernoulli numbers B_{2k} with precision P, we do not include the complexity of computing them in those running times, so that only the core running times are compared; recall (see Section 8.4.1) that computing one Bernoulli number costs $O(\mathcal{M}(P) + k^{4/3+\epsilon})$, while computing the first k Bernoulli numbers costs $O(\mathcal{M}(P)k + k^{2+\epsilon})$.

Computing one value

The naive algorithm is able to compute one value $E_{2k}(\tau)$ with precision P using

$$O\left(\mathcal{M}\left(P + k\log\left(\frac{P}{\mathrm{Im}(\tau)}\right)\right)\left(\frac{P+k}{\mathrm{Im}(\tau)}\right)\log k\right) = O\left((P+k)^{2+\epsilon}\log k\right)$$

bit operations. Our algorithm can output this value using

$$O(\mathcal{M}(P)(\log P + \mathcal{M}(k))) = O(P^{1+\epsilon}k^{1+\epsilon})$$

bit operations, which is a better asymptotic complexity in most cases.

Computing all the values

Our algorithm computes all $E_{2k'}(\tau), k' \leq k$ in time

$$O(\mathcal{M}(P)(\log P + \mathcal{M}(k)) = O(P^{1+\epsilon}k^{1+\epsilon})$$

As for the naive algorithm, computing each of the k series requires to compute all of the $n^{2k'-1}$, which means one should use the naive, iterative powering algorithm. Hence, the naive algorithm can compute all $E_{2k'}(\tau), k' \leq k$ in time

$$O\left(\mathcal{M}\left(P + k\log\left(\frac{P}{\mathrm{Im}(\tau)}\right)\right)\left(\frac{P+k}{\mathrm{Im}(\tau)}\right)k\right) = O\left((P+k)^{2+\epsilon}k\right).$$

In both cases, our algorithm outperforms asymptotically the naive algorithm. We leave the implementation of both algorithms and their comparison for future work.

Chapter 9

Computing isogenous curves in genus 1

Let E be an elliptic curve defined over a field K and given in short Weierstrass form, i.e.

$$E/K : y^2 = x^3 + ax + b.$$

Suppose we are also given a point Q of ℓ-torsion. We then look at Problem 1.1.16 – that is to say, we want to find an ℓ-isogenous curve E' and an ℓ-isogeny $\phi : E \to E'$ such that $\phi(Q) = 0$. As we explained in Chapter 1, any isogeny can be written as the composition of isogenies of prime degree; hence, we only consider here ℓ-isogenies with ℓ prime. Recall that we proved in Section 1.1.3 that, in the case of Weierstrass curves, the shape of our isogeny is

$$\phi(x, y) = \left(\frac{N(x)}{D(x)}, y \left(\frac{N(x)}{D(x)} \right)' \right)$$

with $\deg N = \ell, \deg D = \ell - 1$.

Note that this problem is already solved by the use of Vélu's formulas [Vél71]. In this chapter, we outline a different algorithm to solve this problem. The algorithm works on curves defined over $\mathbb{C}$, or over a number field, or over $\mathbb{F}_p$; it uses the evaluation of the Abel-Jacobi map for complex elliptic curves, which the previous chapter showed has quasi-linear complexity.

The asymptotic complexity of the algorithm is worse than that of Vélu's formulas, and is also slower in practice. However, this algorithm is still interesting because it is generalizable to curves of higher genera. More precisely, the fact that the Abel-Jacobi map could potentially be computed in quasi-linear running time in higher genus (cf. Chapter 8) could be used to generalize this algorithm into one which solves this problem in genus g, at least for jacobians of curves defined over $\mathbb{C}$ or K.

9.1 Computing isogenous curves over $\mathbb{C}$

We start by giving an algorithm which solves the problem for elliptic curves defined over $\mathbb{C}$ which is different from the one stemming from Vélu's formulas [Vél71], and has worse overall complexity. We assume we are given a point $Q \in E[\ell]$, which generates the kernel of the isogeny. We want to compute the equation of the isogenous curve, as well as the expression of the isogeny as a rational function.

Recall that, in the case of short Weierstrass forms, we have [BMSS08]:

$$\phi(x,y) = \left(\frac{N(x)}{D(x)}, y \left(\frac{N(x)}{D(x)} \right)' \right)$$

with $\deg N = \ell, \deg D = \ell - 1$. This means one only needs to compute the rational function giving the x-coordinate of the isogeny; doing so entails significant savings (by a constant factor) in the algorithm.

The strategy we follow is that of an evaluation-interpolation; more precisely, we wish to take advantage of the fact that the isogeny between tori is easy to compute, using the quasi-optimal algorithms to go from the curve to the torus and back. This can be summarized with the diagram:

$$\begin{array}{ccc} \mathbb{C}/\Lambda & \to & \mathbb{C}/\Lambda' \\ \uparrow & & \downarrow \\ E/\mathbb{C} & & E'/\mathbb{C} \end{array}$$

9.1.1 Determining the isogenous curve

The first step of the algorithm is to determine the equation of the isogenous curve we are looking for. For this, we adopt the following strategy: we compute the periods using the AGM (as in Chapter 4), then use the elliptic logarithm of P to compute the isogenous periods, and finally we compute the coefficients of the isogenous curve as in Section 8.1.1.

Computing the isogenous periods is done as follows. There are $\ell + 1$ possibilities for the period lattice; they are given by the lattices:

$$\{\mathbb{Z}\omega_1 + \mathbb{Z}\omega_2', \quad \omega_2' = \frac{\omega_2}{\ell} + k\frac{\omega_1}{\ell} \quad (0 \leq k \leq l-1)\} \cup \{\mathbb{Z}\frac{\omega_1}{\ell} + \mathbb{Z}\omega_2\}$$

The isogenous period lattice corresponds to one for which the elliptic logarithm of Q is mapped to 0, since Q generates $\operatorname{Ker}\phi$. Hence, we first compute the elliptic logarithm of Q using the algorithm of [CT13]; this gives us a point $\frac{a\omega_1 + b\omega_2}{\ell}$. We then have to determine which of the $\ell+1$ lattices contains the elliptic logarithm of Q. The procedure is as follows:

- if $b = 0$, the lattice $\mathbb{Z}\frac{\omega_1}{\ell} + \mathbb{Z}\omega_2$ is the one we want;
- if not, the point $\frac{a\omega_1 + b\omega_2}{\ell} \times (b^{-1} \pmod{\ell})$ is the elliptic logarithm of a point in $\operatorname{Ker}\phi$, and hence also generates $\operatorname{Ker}\phi$, so the lattice $\mathbb{Z}\omega_1 + \mathbb{Z}\left(\frac{\omega_2}{\ell} + (ab^{-1} \pmod{\ell})\frac{\omega_1}{\ell}\right)$ is the one we want.

9.1.2 Evaluating the isogeny

Once again, the strategy is to travel through the analytic representation of the elliptic curve in order to evaluate the isogeny at 2ℓ points, so that we can retrieve the rational function via interpolation. For this, we compute the elliptic logarithm (see Chapter 4) of 2ℓ points, determine the image of those logarithms in the isogenous torus, then find the image point on the isogenous curve that we are looking for. Finally, we use rational function interpolation to compute the isogeny; note that we do not need to compute $\wp'$ nor the y-coordinate of the isogenous curve, which saves a constant factor.

Recall that we outlined two algorithms to compute $\wp(z, \Lambda)$ in quasi-optimal time: the first one, based on the fast computation of θ, was presented in this chapter (Section 8.1.2, and more precisely Note 8.1.5); the second one, based on the Landen transform, was presented in

Section 4.3. As outlined in Section 8.1.3, the second algorithm is around twice faster than the first one; both require working at precision $O(P)$ to compensate rounding errors, which are especially large when z is close to the corners of the parallelogram. However, note that we are free to pick the 2ℓ points aribitrarily, discarding problematic points if need be. This would limit precision losses, and we could then potentially reduce the working precision in the algorithm for $\wp$, thus gaining a constant factor; on the other hand, throwing away each "bad" z means wasting an elliptic logarithm computation, which introduces a potential slowdown.

We implemented a version of the Landen-based algorithm in Magma which reuses computations of the $\theta_{0,1,2}(0, 2^k\tau)^2$, which amounts to a batched version of the algorithm (as described in Note 4.3.1); the theta-constants are computed by our MPC implementation of Algorithm 11. A further speedup could be obtained by writing the full algorithm in MPC. We did not implement the batched versions of the algorithms for $\theta(z, \tau)$ which are described in Section 6.5; however, we believe that the first algorithm with batched computations of θ would still be slower that the batched Landen-based algorithm, at least for the precisions we are considering here.

Determining the image in the other parallelogram

The method to reduce z modulo Λ' is simple. Even though $\wp(z, \Lambda') = \wp(z \bmod \Lambda', \Lambda')$, we still need to reduce z since our algorithm for $\wp$ depends on computing $\theta(z, \tau)$, and the bounds and the running time we showed in Chapter 6 require that z is reduced. The reduction we have in mind is not the one in the fundamental parallelogram, but rather the reduction in $[-\omega_1/2, \omega_1/2] \times [-\omega_2/2, \omega_2/2]$.

The reduction is performed by determining the coordinates of $z \in \mathbb{C}$ with respect to the periods (ω'_1, ω'_2) of the isogenous lattice. The technique is the same as the one described in Note 2.3.1; write

$$(z, \overline{z}) = (x, y) \begin{pmatrix} \omega'_1 & \overline{\omega'_1} \\ \omega'_2 & \overline{\omega'_2} \end{pmatrix}$$

with $x, y \in \mathbb{R}$, which we can compute easily by inverting the 2×2 matrix. We then subtract $\lfloor x \rceil \omega'_1 + \lfloor y \rceil \omega'_2$ to z to get the result.

Interpolation

We use Cauchy interpolation [VZGG13, p.119] to recover the rational fraction; that is to say, given n equations $f(u_i) = v_i$ and a parameter k, solve the problem

$$t(u_i) \neq 0 \text{ and } \frac{r(u_i)}{t(u_i)} = v_i \quad \forall i \in \{0, ..., n\}, \text{ with } \deg r < k, \quad \deg t \leq n - k$$

Here, given the shape of the isogeny (given in Note 1.1.15), we have $k = \ell + 1$ and $n = \ell + 1 + 2\frac{\ell-1}{2} = 2\ell$; hence, the image of 2ℓ points are needed to interpolate the correct rational fraction. Note here that using the general equation giving the shape of an isogeny (Note 1.1.15) instead would require to interpolate the y-coordinate of the isogeny, which requires 3ℓ points instead, and a second Cauchy interpolation.

The outline of this algorithm is as follows. We first find a polynomial g of the right degree that interpolates those values, then we compute an extended GCD to solve the problem $r \equiv tg \pmod m$ where $m = (x - u_0)...(x - u_{n-1})$; to be more precise, we run an extended GCD algorithm on m and g, and we interrupt the algorithm when the GCD we get is of degree $< k$. The coefficient multiplying g is t, and the GCD is r; we have a solution if $gcd(r, t) = 1$. We refer to [VZGG13, p.119] for details.

The complexity of this algorithm depends on the complexity of the computation of the GCD of polynomials. We find in [VZGG13, p.313] that, if polynomials are of degree n, the complexity is $O(\mathcal{M}(n)\log n)$ field operations. In our case, those operations are multiplications of precision P, and $n = O(\ell)$; we find a running time of

$$O(\mathcal{M}(P)\mathcal{M}(\ell)\log \ell)$$

bit operations.

9.1.3 Description of the algorithm and complexity

We summarize the final algorithm in Algorithm 25.

Algorithm 25 Compute an ℓ-isogenous curve and an isogeny with a given kernel, over $\mathbb{C}$.
Input: $E(\mathbb{C}) : y^2 = x^3 + ax + b$ (with $a, b \in \mathbb{C}$), $Q \in E[\ell]$.
Output: $E'(\mathbb{C}) : y^2 = x^3 + a'x + b'$ (with $a', b' \in \mathbb{C}$), ℓ-isogenous to E; the rational function defining the ℓ-isogeny $\phi : E \to E'$ such that $\phi(Q) = 0$.

1: Compute the periods $\omega_1, \omega_2 \in \mathbb{C}$ corresponding to E.
2: Compute the elliptic logarithm of Q, i.e. $z \in \mathbb{C}/(\mathbb{Z}\omega_1 + \mathbb{Z}\omega_2)$.
3: Determine the periods $\omega_1', \omega_2' \in \mathbb{C}$ of the isogenous curve (Section 9.1.1).
4: Compute the coefficients a', b' of E' using theta-constants (Section 8.1.1)
5: **for** k = 1 to 2ℓ **do**
6: Take a point $P_k \in E$ – for instance, $P_k = (k, \sqrt{k^3 + ak + b})$.
7: Compute the elliptic logarithm z_k of P_k.
8: Compute z_k', the image of z_k by the isogeny (Section 9.1.2).
9: Compute $\wp(z_k', [\omega_1', \omega_2'])$, the x-coordinate of the corresponding point (Section 8.1.3).
10: **end for**
11: Use rational function interpolation (Section 9.1.2) to recover $\frac{g(x)}{h(x)^2}$.
12: Return E' and $\phi : (x, y) \mapsto \left(\frac{g(x)}{h(x)^2}, y\left(\frac{g(x)}{h(x)^2}\right)'\right)$.

Proposition 9.1.1. *Assuming $\ell \ll P$, Algorithm 25 has a complexity of $O(\mathcal{M}(P)(\ell \log P + \mathcal{M}(\ell)\log \ell)) = O(\mathcal{M}(P)\log P\mathcal{M}(\ell)\log \ell)$.*

Proof. Steps 7 and 9, i.e. the computation of the Abel-Jacobi map and its inverse, each cost $O(\mathcal{M}(P)\log P)$; hence Steps 5 to 10 cost $O(\ell\mathcal{M}(P)\log P)$. Step 11 costs $O(\mathcal{M}(P)\mathcal{M}(\ell)\log \ell)$. All the other steps have negligible cost. □

We described in Chapter 1 a few algorithms used to compute isogenies, noting that their asymptotic cost was roughly $O(\mathcal{M}(P)\mathcal{M}(\ell)\log \ell)$ – which is just the running time of the rational function interpolation in our method. Hence, our algorithm does not provide a better running time; in fact, since $\ell \ll P$, it is much slower, because of the $\log P$ factor.

We provide a few timings for our algorithm. Since Magma does not suppose Vélu's formulas over $\mathbb{C}$, we were not able to get such timings for comparison with our algorithm; however, it is clear that such a comparison would show that Vélu's formulas are much faster than our algorithm.

ℓ	P	Time (s)
11	357	1.5
11	380	1.7
11	720	3.1
11	907	3.8
17	1113	4
23	2037	152

9.2 Computing isogenous curves over a number field

We now outline a method to compute an isogeny between two curves defined over a number field K of degree n, using the method described in the previous section. Once again, we summarize the algorithm with the diagram

$$\begin{array}{ccc} \mathbb{C}/\Lambda & \to & \mathbb{C}/\Lambda' \\ \uparrow & & \downarrow \\ E/\mathbb{C} & & E'/\mathbb{C} \\ \uparrow\uparrow\uparrow & & \downarrow\downarrow\downarrow \\ E/K & & E'/K \end{array}$$

The algorithm consists in computing the curves that are images of E by all the embeddings of K in $\mathbb{C}$, then use the previous algorithm to compute an isogenous curve over $\mathbb{C}$; we then use interpolation to compute the isogeny over K, using continued fractions to recognize the rational numbers. This will be outlined in Algorithm 26.

One important issue in the algorithm is the precision at which we wish to work in $\mathbb{C}$; this has a big impact on the overall complexity of the algorithm. Unfortunately, we do not have a definite answer to this question, which we mention in Section 9.2.5.

9.2.1 Computing embeddings

The first step of the algorithm is to compute every embedding of the number field in $\mathbb{C}$, with sufficient precision, in order to apply the previous section; we will then combine all the complex isogenies we obtain to find the isogeny over K. We discuss the complexity of this step.

Root-finding methods

There are a wealth of methods dedicated to computing complex approximations of roots of polynomials; this problem is a fundamental problem in analysis. We refer the reader to [MP13] for an overview of a great number of methods, from methods using companion matrices to those based on Newton's iterations. We assume that (at the cost of computing $\gcd(f, f')$ with high enough precision) we are trying to compute the roots with precision P of a polynomial f of degree n.

An interesting family of algorithms is the *splitting methods*. The concept is to first perform a crude splitting of the complex plane, determining regions of $\mathbb{C}$ that contain some, but not all, of the roots of f; this can be used to determine factors of the polynomial, and we can apply the procedure recursively. Note that, at some point in the algorithm, it is faster to refine the approximation of the roots using Newton's method than to continue with this strategy. A notable algorithm following this pattern is the splitting circle method, designed by Schönhage in 1982 [Sch82]. It claims a complexity of $O(n^3 \log n + n^2 P)$ up to logarithmic factors. The same

concept was used and refined by Pan [Pan01], which claims a quasi-optimal bit complexity of $O(n \log^2 n(\log^2 n + \log P)\mathcal{M}(P))$ to find all the roots.

We do not know if this algorithm has been implemented anywhere; [Pan01] notes that it would be quite challenging. However, note that Schönhage's algorithm has been implemented in Magma by Xavier Gourdon; an in-depth discussion of the implementation of Schönhage's algorithm can be found in his thesis [Gou96]. Regardless, we assume that one can use Pan's complexity of $O(n \log^2 n(\log^2 n + \log P)\mathcal{M}(P))$.

Total complexity

The complexity estimates in Pan's algorithm assume that the roots of the polynomial have norm bounded by 1, since several classical scaling techniques allow one to reduce the general problem to this particular case. Hence, in order to get a final result accurate up to 2^{-P}, we need to compute such roots with absolute precision $P + s$, where s is the maximum size of the roots of our polynomial. In all generality, Rouché's theorem shows that the norm of any root of the polynomial is bounded by $1 + \max|a_i|$ (where a_i are the coefficients of the polynomial).

Hence, the complexity of computing all embeddings of K in $\mathbb{C}$ is

$$O(n \log^2 n(\log^2 n + \log(P + \log(1 + \max|a_i|)))\mathcal{M}(P + \log(1 + \max|a_i|)))$$

If we assume that $\max|a_i| = O(2^P)$, i.e. that the coefficients of the polynomial defining K can be stored with absolute precision P in $O(P)$ bits, the complexity becomes

$$O(n \log^2 n(\log^2 n + \log P)\mathcal{M}(P))$$

9.2.2 Using complex conjugation

The following result is not very hard to prove, but is nonetheless important for an efficient implementation.

Proposition 9.2.1. *Let $K = \mathbb{Q}[X]/(f)$ with $f \in \mathbb{Z}[X]$, and let $\alpha \in \mathbb{C} \setminus \mathbb{R}$ be a root of f. Let Q_K be an ℓ-torsion point of $E(K)$. Denote E_α (resp. Q_α) the image of E_K (resp Q_K) by the embedding $X \mapsto \alpha$; let E'_α be the ℓ-isogenous curve over $\mathbb{C}$ such that the isogeny $\phi_\alpha : E_\alpha \to E'_\alpha$ is such that* $\operatorname{Ker} \phi_\alpha =< P_\alpha >$.
Then $E_{\overline{\alpha}}$ is ℓ-isogenous to $E'_{\overline{\alpha}}$ and the isogeny $\phi_{\overline{\alpha}}$ is such that $\operatorname{Ker} \phi_{\overline{\alpha}} =< P_{\overline{\alpha}} >$.
That is to say, the coefficients of the curves we obtain when using the embedding $X \to \overline{\alpha}$ are the complex conjugates of the coefficients of the curves we obtain when using the embedding $X \to \alpha$, and the same goes for the coefficients of the isogenies.

As for the proof, one can for instance take a look at Vélu's formulas (Section 1.1.3 and propagate the complex conjugation. This result means that one only needs to compute one isogeny per pair of complex conjugate roots of f, since the other (complex conjugate) isogeny is simply deduced using complex conjugation of the coefficients. This induces at best a factor 2 speedup in the algorithm, which is not negligible; however, we did not implement this in our script by lack of time.

9.2.3 Multi-evaluation and fast interpolation

We now take a look at the problem of computing the coefficients of the isogenous curve over K and the isogeny. What we computed so far are n complex values for each of those coefficients,

which correspond to the image of each coefficient by each of the n embeddings from K to $\mathbb{C}$; that is to say, for each coefficient v we have n complex values $v_1, \ldots, v_n$ such that

$$v_1 = \sum_{i=0}^{n-1} c_i \alpha_1^i, \quad \ldots, \quad v_n = \sum_{i=0}^{n-1} c_i \alpha_n^i \qquad \text{with } v = \sum_{i=0}^{n-1} c_i X^i \in K = \mathbb{Q}[X]/(f) \text{ and } f(\alpha_i) = 0$$

Hence, for each coefficient, we have what amounts to n values of a polynomial with rational coefficients of degree $n-1$: we can thus use interpolation to recover complex approximations of the coefficients, then recognize the rational coefficients using techniques described in the next section (Section 9.2.4).

Note that we need to recover the $O(\ell)$ coefficients of the isogeny by interpolating at the, the α_i (roots of the polynomial defining the number field). We can use a fast algorithm to perform these interpolations; we refer to [VZGG13, chap. 10] for all the details, but we sketch the algorithm here to show how the $O(\ell)$ interpolations can be sped up.

We start with the problem of multi-evaluation at $n = 2^k$ values, which is the inverse of the problem at hand, but is needed in the fast interpolation algorithm. Let f be a polynomial of degree $< 2^k$, and $(u_0, ..., u_{2^k})$ complex numbers. We can compute $(f(u_0), \ldots, f(u_{2^k}))$ faster than applying Horner's scheme 2^k times, using the remark that

$$f(u_i) = f \pmod{(X - u_i)}$$

The Euclidean remainders are computed using *remainder trees*. Define

$$M_{i,j} = \prod_{l=i2^j}^{(i+1)2^j-1} (X - u_l)$$

which corresponds to the subproducts one gets when splitting the interval $[1, n]$ in slices of length 2^j. The idea is then to compute $f \pmod{M_{i,j}}$ for all i, j, starting with $j = k$ and going down (towards the leaves). Computing the remaindering tree is not hard. Start with the leaves, i.e. $M_{i,0} = (X - u_i)$, then build the nodes by multiplying together the values found in the two children of the node:

$$M_{i,j} = M_{2i,j-1} \times M_{2i+1,j-1}$$

Each step of the formula is the multiplication of 2^{k-j} polynomials of degree less than 2^j, so it costs less than $O(\mathcal{M}(n))$; there are $\log n$ levels/steps in the tree, so the total cost is $O(\mathcal{M}(n) \log n)$. Once we have this remaindering tree, we want to compute $f \pmod{\text{leaf}}$, starting with $f \pmod{M_{0,k}}$. The computation of $f \pmod{g}$ with $\deg f = 2n, \deg g = n$ can be done in $5\mathcal{M}(n)$ operations [VZGG13, Theorem 9.6]. Hence, going down one level in the tree is 2^i euclidean divisions of polynomials of degree 2^{k-i} by polynomials of degree 2^{k-i-1}, hence the complexity at each step is bounded by $O(\mathcal{M}(n))$; the total cost is then $O(\mathcal{M}(n) \log n)$.

The algorithm for fast interpolation also uses the remainder tree. The idea is as follows: the Lagrange interpolation of the relations $P(u_i) = v_i, i \in [1..n]$ corresponds to

$$P = \sum_{i=1}^{n} v_i \prod_{j \neq i} \frac{X - u_j}{u_i - u_j} = \sum_{i=1}^{n} v_i s_i \frac{m}{X - u_i}$$

with $s_i = \prod j \neq i \frac{1}{u_i - u_j}$, and $m = \prod_{k=1}^{n} (X - u_k)$. But we also have:

$$s_i = 1/m'(u_i)$$

Hence evaluating all the s_i is just a multi-evaluation of a polynomial, and we use the algorithm we just described. The next step is to evaluate $\sum_i v_i s_i \frac{m}{x-u_i}$ quickly; we use a similar method of splitting the sum in half:

$$\begin{aligned}\sum_{i\in[1..2^k]} v_i s_i \frac{m}{x-u_i} &= \sum_{i\in[1..2^k]} v_i s_i \frac{M_{0,k}\times M_{1,k}}{x-u_i} \\ &= M_{1,k}\sum_{i\in[1..2^{k-1}]} v_i s_i \frac{M_{0,k}}{x-u_i} + M_{0,k}\sum_{i\in[2^{k-1}+1,2^k]} v_i s_i \frac{M_{1,k}}{x-u_i} \\ &= \dots\end{aligned}$$

At the leaves, we just need to evaluate $v_i s_i$. Hence the cost of the algorithm $T(n)$ satisfies

$$T(n) \le 2T(n/2) + 2M(n/2) + n \le 2T(n/2) + M(n) + n$$

which is $O(M(n)\log n)$.

In our case here, we are looking to interpolate $O(\ell)$ coefficients knowing their values at $\alpha_1, \dots, \alpha_n$ – hence, anything that depends only on the u_i, which is to say the computation of the remainder trees, of m and of the s_i, can be computed once for all the computations. However, in the computation of the interpolation (the recursive algorithm that splits the sums in half), we do not seem to obtain any savings, since the values in the leaves are different. Hence, the caching of remainder trees and of the s_i gives a speedup in practice compared to the strategy which simply applies the interpolation algorithm for each coefficient, although this does not improve the asymptotic complexity. We did not implement this in our script for lack of time.

Hence, the total running time of interpolating $O(\ell)$ elements at the α_i is $O(\ell\mathcal{M}(n)\log n)$ operations.

9.2.4 Recovering coefficients as rationals

We now discuss how to convert the interpolated coefficients, which are written as complex numbers of precision P, into rational numbers. Of course, there are infinitely many approximations of a real number by rational numbers, including the representation $\frac{a}{2^P}$ that the interpolation algorithms outputted; we are looking for a fraction $\frac{p}{q}$ with $q < 2^P$. In addition, we could also enforce a bound on the denominator if we happen to know one. We will simply assume that P has been chosen such that 2^P is a bound on the denominators we are looking for; we discuss this condition in the next section.

To solve this problem, we use the continued fraction expansion of our rational number, which in this case simply amounts to a Euclidean algorithm. Denote by a_n the successive quotients when running Euclid's algorithm on a and 2^P; we can then build a sequence $\frac{h_n}{k_n}$ of fractions approximating $\frac{a}{2^P}$ using the relations:

$$\begin{aligned} h_n &= a_n h_{n-1} + h_{n-2} \\ k_n &= a_n k_{n-1} + k_{n-2} \\ h_{-1} = 1 &\qquad h_{-2} = 0 \\ k_{-1} = 0 &\qquad k_{-2} = 1 \end{aligned}$$

The approximation we are looking for is simply the last one with $k_n < 2^P$. In the worst case, this algorithm is asymptotically as costly as the whole Euclidean algorithm, which is $O(\mathcal{M}(n)\log n)$, where n is the size of the numerator/denominator; here, this gives $O(\mathcal{M}(P)\log P)$.

9.2.5 Description of the algorithm

We outline here the algorithm to compute an isogeny over K.

Algorithm 26 Compute an ℓ-isogenous curve and an isogeny with a given kernel, over K.
Input: $K = \mathbb{Q}(X)/(f)$ a number field of degree n; $E(K) : y^2 = x^3+ax+b$ $(a, b \in K)$, $Q_K \in E[\ell]$.
Output: $E'(K) : y^2 = x^3 + a'x + b$ $(a', b' \in K)$, ℓ-isogenous to E; the rational function defining the ℓ-isogeny $\phi : E \to E'$.

1: Compute the n embeddings of K in $\mathbb{C}$ – that is to say, compute approximations of the roots of f in $\mathbb{C}$ with absolute precision P.
2: **for** i = 1 to n **do**
3: **if** $\alpha_i = \overline{\alpha_j}$ for $j < i$ **then**
4: Put $E'_i = \overline{E'_j}, \phi_{i,x} = \overline{\phi_{j,x}}$.
5: **else**
6: Compute $a_i, b_i \in \mathbb{C}$, the image of a, b by the ith complex embedding of K in $\mathbb{C}$. Put $E_i(\mathbb{C}) : y^2 = x^3 + a_i x + b_i$.
7: Compute also $Q_i \in E_i(C)$, the image of Q_K by the ith embedding of K in $\mathbb{C}$.
8: Use Algorithm 25 to compute E'_i and $\phi_{i,x}$ (the rational function giving the x-coordinate of the isogeny).
9: **end if**
10: **end for**
11: Perform interpolation (Section 9.1.2) on the coefficients of E'_i and $\phi_{i,x}$ to recover a curve E' and a rational function ϕ_x, both of which will have coefficients in $\mathbb{C}[X]$ (but the coefficients are actually rational)
12: Recognize the coefficients, i.e. write them as fractions (Section 9.2.4). This makes E' and ϕ have coefficients in K.
13: Return E' and $\phi = (\phi_x, y\phi'_x)$.

Proposition 9.2.2. *Algorithm 26 requires*

$$O((n\log^4 n + n\log^2 n \log P + n\mathcal{M}(\ell)\log \ell \log P + \ell\mathcal{M}(n)\log n)\mathcal{M}(P)) = O(n^{1+\epsilon}\ell^{1+\epsilon}P^{1+\epsilon})$$

bit operations.

Proof. Assume that $\max|a_i| = O(2^P)$, i.e. that the coefficients of the polynomial defining the number field can be stored with absolute precision P in $O(P)$ bits. The overall complexity is then:

- $O(n\log^2 n(\log^2 n + \log P)\mathcal{M}(P))$ for Step 1;
- $O(n\mathcal{M}(\ell)\log \ell\mathcal{M}(P)\log P)$ for the For loop;
- $O(\ell\mathcal{M}(n)\log n\mathcal{M}(P))$ for the interpolation;
- $O(\ell\mathcal{M}(P)\log P)$ for the conversion to rational numbers.

This gives the claimed total complexity. □

We can compare this algorithm to Vélu's formula applied over K:

Proposition 9.2.3. *Assuming that the integers (e.g. the denominators) which are computed when applying Vélu's formulas are bounded by $O(2^P)$, Vélu's formulas applied over K have a cost of $O(\mathcal{M}(\ell)\log \ell\mathcal{M}(n)\log n\mathcal{M}(P))$.*

Proof. We evaluated in Section 1.1.3 the cost of Vélu's formulas, which is here of $O(\mathcal{M}(\ell)\log\ell)$ operations in the number field. If the number field is represented as $K = \mathbb{Q}[X]/(f)$ with f a polynomial of degree n, the cost of arithmetic operations in K is $O(\mathcal{M}(n)\log n)$ integer operations, using the fast GCD algorithms [VZGG13, Chapter 11]. If we assume that the maximal denominator is smaller than 2^P and the maximal numerator is also $O(2^P)$, we get a total cost bounded by $O(\mathcal{M}(\ell)\log\ell\mathcal{M}(n)\log n\mathcal{M}(P))$ bit operations. □

This is a lower complexity than the one of Algorithm 26, at least by logarithmic factors. Furthermore, there is a fundamental difference between the two algorithms: our algorithm requires a choice of P, then performs essentially only operations on P-bit integers, which means it is very sensitive to our choice of P; on the contrary, Vélu's formulas will in general handle integers which can be much smaller than 2^P, which yields a better complexity. Obtaining the best running time for our algorithm requires to know the smallest value of P for which Algorithm 26 returns the right value, i.e. recognizes the correct elements of K (and hence the correct rational numbers).

Discussion on P

The total complexity depends on P, which is the precision we choose to work with when computing the isogenies over the complex numbers. This precision must be big enough to allow the recognition of the coefficients of the isogeny and of the isogenous curve as elements of K; in particular, we must have that the maximal denominator in the rational numbers that appear in each coefficient must be smaller than 2^P.

We did not manage to obtain any result tying the height of the rational numbers appearing in the coefficients of the isogeny or of the isogenous curve to anything, such as the height of the coefficients of the original curve, or the coefficients of the polynomial f defining the number field. The determination of P would allow us to make the complexities above more precise. However, in a specific case, which we outline in the next section, we managed to formulate some heuristics on P, finding that it was roughly the same size as the largest coefficient of the polynomial defining the number field.

We carried out an experiment and attempted to determine the smallest P such that the recognition of the coefficients of the isogeny and the isogenous curve succeded, in order to compare our algorithm to Vélu's formulas. This value of P was determined by running the algorithm at several different precisions and seeing if the interpolation succeded and gave the correct result. Generalizing this amounts to assuming we have an oracle giving us the correct value of P. The following table hence shows the fastest running time possible of our algorithm on several examples, and compare it to Vélu's formulas as implemented in Magma; we stress that the comparison is unfair and biased towards our algorithm.

ℓ	n	P required	Our algorithm (s)	Vélu in Magma (s)
23	132	650	460	174
29	210	1390	3665	2111

Recall that, as mentioned previously, our implementation of Algorithm 26 lacks several interesting optimizations that we have mentioned previously, among which a batched algorithm to compute $\theta(z,\tau)$, the use of complex conjugation to reduce the number of embeddings for which the full computations have to be carried out, and the caching of the remaindering tree in the fast interpolation algorithm. We can hope to achieve a factor 4 speedup by combining these optimizations, which would make our algorithm within the same realm as Vélu's formulas over a number field, at least for these examples; hence, further investigation is needed before discounting our algorithm.

9.3 Computing isogenous curves over $\mathbb{F}_p$

The idea of this algorithm is, once again, to use the previous algorithms to compute the isogeny. More specifically, we *lift* the curve and the torsion point on a number field K, then use Algorithm 26 to compute an isogenous curve over K, and deduce from this the isogenous curve over $\mathbb{F}_p$. We summarize the algorithm with the diagram

$$\begin{array}{ccc} \mathbb{C}/\Lambda & \to & \mathbb{C}/\Lambda' \\ \uparrow & & \downarrow \\ E/\mathbb{C} & & E'/\mathbb{C} \\ \uparrow\uparrow\uparrow & & \downarrow\downarrow\downarrow \\ E/K & & E'/K \\ \uparrow & & \downarrow \\ E/\mathbb{F}_p & & E'/\mathbb{F}_p \end{array}$$

Going from the curve E'/K to $E'/\mathbb{F}_p$ requires taking the quotient of K by a maximal ideal which we determine in the next subsection. The elements of K – that is to say, the coefficients of the curve and those of the isogeny – are then sent to the right elements of $\mathbb{F}_p$ (i.e. those we were looking for) provided we picked the maximal ideal which sends the coefficients of E/K to $E/\mathbb{F}_p$.

In all that follows, we suppose that ℓ is an odd prime greater than 3. Our algorithm does not seem to be directly generalizable to the case $\ell = 2$; however, note that in this particular case, Vélu's formulas are particularly easy to compute.

9.3.1 Global torsion lifting

Transforming the problem over $\mathbb{F}_p$ in a problem over K requires solving the *lifting problem*:

Definition 9.3.1 ([Sil09])**.** Let e/k be an elliptic curve and $q \in e(k)$. The *lifting problem* for (k, e, p) is the problem of finding the following quantities:

- a field K with subring R;
- a maximal ideal $\mathfrak{p}$ of R satisfying $R/\mathfrak{p} \simeq k$;
- an elliptic curve E/K satisfying $E \pmod{\mathfrak{p}} \simeq e$;
- a point $Q \in E(K)$ satisfying $Q \pmod{\mathfrak{p}} = q$.

As described in [Sil09], who looked at this lifting problem in the context of the ECDLP, there are essentially four ways to solve the lifting problem. The lift is called a *global lift* (resp. local lift) if K is a global field such as $\mathbb{Q}$ or a number field (resp. K is a local field such as $\mathbb{Q}_p$); it is called a *torsion lift* if q is a torsion point, and a non-torsion lift if it is not. However, in the context of [Sil09], none of those approaches seem to yield any speedup for the ECDLP.

We show here how to perform a *global torsion lifting* from $\mathbb{F}_p$ to a number field K. There are two reasons for this: first, as noted in [Sil09, Table 1], this is the only way to move the problem to $\mathbb{C}$, where we can use our algorithm based on the evaluation of the Abel-Jacobi map; furthermore, the idea seems natural as we are given an ℓ-torsion point $Q \in E/\mathbb{F}_p$ as input for the problem of finding an isogenous curve. Hence, assuming we are given $E/\mathbb{F}_p : y^2 = x^3 + a_{\mathbb{F}_p}x + b_{\mathbb{F}_p}$ and $Q = (x, y)$ an ℓ-torsion point, we are looking to determine a number field $K = \mathbb{Q}[\alpha]$, a maximal ideal $\mathfrak{p}$ such that $K/\mathfrak{p} = \mathbb{F}_p$, an elliptic curve $E_K/K : y^2 = x^3 + a_K x + b_K$ such that (a_K, b_K) reduce to $(a_{\mathbb{F}_p}, b_{\mathbb{F}_p})$, and a point $Q_K = (x_K, y_K)$ of ℓ-torsion on E_K such that Q_K reduces to Q.

We start by choosing to lift Q using the map

$$\mathbb{F}_p \to \mathbb{Z}, \qquad x \mapsto x_K \in \left\{-\frac{p+1}{2}, ..., \frac{p-1}{2}\right\} \text{ such that } x \equiv x_K \pmod{p}.$$

We then consider the resulting integers x_K, y_K as elements of K. This lift can be done regardless of the number field K, since any number field contains $\mathbb{Z}$.

Secondly, we choose the equation of the elliptic curve E_K, imposing that $a_K = \alpha$. The condition $Q_K \in E_K$ then imposes that

$$b_k = y_K^2 - x_K^3 - x_K\alpha.$$

This imposes the conditions $x_K, y_K, \alpha \pmod{\mathfrak{p}} = x, y, a_{\mathbb{F}_p}$.

We now determine the number field K using the condition that Q_K should be an ℓ-torsion point. This condition can be translated in terms of *generic ℓ-division polynomials* ψ_ℓ; we introduce those polynomials in the next section (Section 9.3.2). The condition "Q_K is of ℓ-torsion on E_K" translates as

$$\psi_\ell(x_K, \alpha, y_K^2 - x_K^3 - \alpha x_K) = \Phi_\ell(\alpha) = 0.$$

where Φ_ℓ is a polynomial with integer coefficients[14]. We study Φ_ℓ in Section 9.3.3; experiments show this polynomial to be irreducible, but we did not manage to prove it in the general case. Assuming this polynomial is irreducible, it is the minimal polynomial of α, and we can define the corresponding number field K as $\mathbb{Q}[X]/(\Phi_\ell)$.

Furthermore, we have

$$\begin{aligned}
\overline{\Phi_\ell}(a_{\mathbb{F}_p}) &= \psi_\ell(x_K, a_{\mathbb{F}_p}, y_K^2 - x_K^3 - a_{\mathbb{F}_p}) \pmod{p} \\
&= \psi_\ell(x, a_{\mathbb{F}_p}, y^2 - x^3 - a_{\mathbb{F}_p}) \pmod{p} \qquad \text{since } x_K \equiv x \pmod{p} \\
&= 0 \qquad\qquad\qquad \text{since } (x, y) \in E[\ell]
\end{aligned}$$

This means that $X - a_{\mathbb{F}_p} | \overline{\Phi_\ell}$ (in $\mathbb{F}_p[X]$). We thus put

$$\mathfrak{p} = (p, \alpha - a_{\mathbb{F}_p})$$

Note that $K/\mathfrak{p} = \mathbb{F}_p$; this indeed defines a maximal ideal such that $E_K \pmod{\mathfrak{p}} = E$ and $P_K \pmod{\mathfrak{p}} = P$. Reducing an element $u = r(\alpha) \in K$ into an element of $\mathbb{F}_p$ simply requires reducing the coefficients of r modulo p, then evaluating the resulting polynomial in $a_{\mathbb{F}_p}$.

This procedure solves the lifting problem, and lifts the curve and the ℓ-torsion point to a number field, where Algorithm 26 can be applied. We then recover the $\mathbb{F}_p$-isogeny by reducing the coefficients of the curve and of the isogeny modulo $\mathfrak{p}$. In the next two sections, we study the polynomials ψ_ℓ (Section 9.3.2) and Φ_ℓ (Section 9.3.3) and determine some of their properties; this is crucial in order to determine the complexity of the computation of the lift and of the reduction, as well as the precision needed when embedding K in $\mathbb{C}$.

9.3.2 Generic division polynomials

We define and study the generic ℓ-division polynomials and establish a few of their properties. The results from this part are taken in part from [BSS99] and [McK94].

[14] Despite the notation Φ_ℓ, one should not confuse this with modular polynomials.

Definition 9.3.2. Consider the generic complex curve given by the short Weierstrass equation

$$E_{A,B} : y^2 = x^3 + Ax + B$$

We define the *generic ℓ-division polynomial* as the polynomial $\psi_\ell \in \mathbb{Z}[x, A, B]$ such that for any $P = (x, y) \in E_{A,B}$,

$$[\ell]P = \left(\frac{Q_\ell(x, A, B)}{\psi_\ell(x, A, B)^2}, y \frac{R_\ell(x, A, B)}{\psi_\ell(x, A, B)^3} \right),$$

or, equivalently, as the polynomial $\overline{f_\ell}$ such that $P \in E_{A,B}[\ell] \Leftrightarrow \overline{f_\ell}(x, A, B) = 0$.

A key point here is that this polynomial has integer coefficients and only depends on x, A, B. Hence, we can lift the condition "Q is an ℓ-torsion point on $E/\mathbb{F}_p$", i.e. $\psi_\ell(x, a, b) = 0$, into the condition "$Q_K = (x_K, y_K)$, the lift of Q in the number field, is an ℓ-torsion point of the lift over K of the curve E", which is exactly $\psi_\ell(x_K, a_K, b_K) = 0$.

The generic ℓ-division polynomials have not been discussed in many papers besides [McK94]. Hence, this section provides a few results on those polynomials, which will be of help later.

Computing ψ_ℓ

One can compute the polynomial $\psi_\ell(x, A, B)$ using recurrence relations [BSS99][15]:

Theorem 9.3.3. *We have*

$$\begin{aligned}
\psi_0 &= 0, \qquad \psi_1 = 1, \qquad \psi_2 = 1 \\
\psi_3 &= 3x^4 + 6Ax^2 + 12Bx - A^2 \\
\psi_4 &= 2x^6 + 10Ax^4 + 40Bx^3 - 10A^2x^2 - 8ABx - 16B^2 - 2A^3 \\
\psi_{2m+1} &= \begin{cases} \psi_{m+2}\psi_m^3 - 16(x^3 + Ax + B)^2\psi_{m-1}\psi_{m+1}^3 & \text{if } m \text{ odd} \\ 16(x^3 + Ax + B)^2\psi_{m+2}\psi_m^3 - \psi_{m-1}\psi_{m+1}^3 & \text{if } m \text{ even.} \end{cases} \\
\psi_{2m} &= (\psi_{m+2}\psi_{m-1}^2 - \psi_{m-2}\psi_{m+1}^2)\psi_m
\end{aligned}$$

We find in [McK94] a partial analysis of the complexity of using those relations to compute $\psi_\ell(1, A, B)$: the costliest step is the last one, where two polynomials of degree $O(\ell^2)$ in A and B (hence with $O(\ell^4)$ terms) are multiplied together. This gives a cost for that last step of $O(\ell^8)$ integer multiplications using a naive method, and $O(\ell^4 \log^2 \ell)$ integer multiplications using an FFT method. Hence, we obtain a bound on the total complexity of $O(\ell^{5+\epsilon})$ integer multiplications.

The second method is the purpose of [McK94], which fixes ℓ and considers recurrence relations between the coefficients in front of the different monomials. This method requires $O(\ell^6)$ multiplications of integers (with $O(\ell^2)$ digits), which is worse asymptotically; however, we presumably compute ψ_ℓ for ℓ small enough that the FFT methods are slower than the naive ones, and hence the worse asymptotic complexity is not much of a problem. Experimental data in [McK94] shows that this algorithm can be faster than the last step of the induction alone, for $\ell \geq 23$; presumably, it is also faster than the whole algorithm for some smaller values of ℓ.

[15] The recurrence relations in [McK94] are essentially the same, except for a constant factor (a power of two) in the ψ_{2k}.

Degrees of ψ_ℓ

Proposition 9.3.4. *Assign a weight 1 to x, weight 2 to A and weight 3 to B. Then all the monomials appearing in the generic division polynomial $\psi_\ell(x, A, B)$ have the same weight, equal to*

$$\chi_\ell = \begin{cases} \frac{\ell^2-1}{2} & \text{if } \ell \text{ is odd} \\ \frac{\ell^2-4}{2} & \text{if } \ell \text{ is even} \end{cases}$$

Proof. This proposition is stated by [McK94], but is not proven. The property can be readily verified on ψ_i for $i < 4$. We use the recurrence relations for the general case:

1. Case $2m+1$ with m odd:

   $$\psi_{2m+1} = \psi_{m+2}\psi_m^3 - 16(x^3 + Ax + B)^2\psi_{m-1}\psi_{m+1}^3$$

   Using the induction hypothesis, the monomials of ψ_{m+2} have weight $\frac{m^2+4m+4-1}{2}$ and those from ψ_m have weight $\frac{m^2-1}{2}$: hence the monomials of $\psi_{m+2}\psi_m^3$ have weight

   $$\frac{m^2+4m+4-1}{2} + 3\frac{m^2-1}{2} = \frac{(2m+1)^2-1}{2} = \chi(2m+1)$$

   Then, note that the weight of each monomial of $(x^3 + Ax + B)^2$ is 6. The second product is then made of the product of 3 monomials of weight $\frac{m^2+2m+1-4}{2}$ by one monomial of weight $\frac{m^2-2m+1-4}{2}$ and one of weight 6, which gives monomials of weight $\frac{4m^2+4m-12+12}{2} = \chi(2m+1)$.

2. Case $2m+1$ with m even:

   $$\psi_{2m+1} = 16(x^3 + Ax + B)^2\psi_{m+2}\psi_m^3 - \psi_{m-1}\psi_{m+1}^3$$

   The calculations are then:

   $$\begin{aligned} 6 + \frac{(m+2)^2-4}{2} + 3\frac{m^2-4}{2} &= \frac{4m^2+4m-3\times 4+12}{2} \\ \frac{(m-1)^2-1}{2} + 3\frac{(m+1)^2-1}{2} &= \frac{4m^2+2m+2m+1-1+3\times(1-1)}{2} \end{aligned}$$

   which give in both cases $\chi(2m+1)$.

3. Case $2m$:

   $$\psi_{2m} = (\psi_{m+2}\psi_{m-1}^2 - \psi_{m-2}\psi_{m+1}^2)\psi_m$$

   Put $(c, c') = (1, 4)$ if m is odd, $(4, 1)$ if m is even. Then:

   $$\begin{aligned} \frac{(m+2)^2-c}{2} + 2\frac{(m-1)^2-c'}{2} + \frac{m^2-c}{2} &= \frac{4m^2+4m+4-4m+2-2(c+c')}{2} \\ \frac{(m-2)^2-c}{2} + 2\frac{(m+1)^2-c'}{2} + \frac{m^2-c}{2} &= \frac{4m^2-4m+4+4m+2-2(c+c')}{2} \end{aligned}$$

   In both cases we get $\chi(2m)$.

□

This means we can write:

$$\psi_\ell(x, A, B) = \sum \alpha_{i,j} A^i B^j x^{\chi(\ell)-2i-3j} \tag{9.3.1}$$

Theorem 9.3.5.
- $\deg_x \psi_\ell(x, A, B) = \chi(\ell)$ – *i.e. the monomial* $x^{\chi(\ell)}$ *appears in* ψ_ℓ.
- $\deg_A \psi_\ell(x, A, B) = \chi(\ell)/2$ – *i.e. the monomial* $A^{\chi(\ell)/2}$ *appears in* ψ_ℓ.

Proof. Both of those statements can be proven using induction, in the same way as the previous theorem. We outline a very similar proof in Section 9.3.3, and hence refer to it for full details. □

Size of coefficients of ψ_ℓ

The following proposition gives some information on the size of the $\alpha_{i,j}$:

Proposition 9.3.6 ([McK94])**.** *We have*

$$|\alpha_{i,j}(\ell)| \leq \frac{\ell^{\ell^2}(\ell^2 - 1/2)!}{[((\ell^2-1)/2)!]^2(\ell^2/2+1)!}$$

We simplify the shape of this bound using the following variants on Stirling's formula [Rob55]:

$$\sqrt{2\pi} n^{n+1/2} e^{-n} \leq \sqrt{2\pi} n^{n+1/2} e^{-n} e^{\frac{1}{12n+1}} \leq n! \leq \sqrt{2\pi} n^{n+1/2} e^{-n} e^{\frac{1}{12n}}$$

which gives

$$\begin{aligned}
|\alpha_{i,j}(\ell)| &\leq \frac{\ell^{\ell^2}(\ell^2 - 1/2)!}{[((\ell^2-1)/2)!]^2(\ell^2/2+1)!} \\
&\leq \frac{\ell^{\ell^2} \times (\ell^2 - 1/2)^{\ell^2} e^{-\ell^2+1/2+\frac{1}{12n}}}{2\pi \left(\frac{\ell^2-1}{2}\right)^{\ell^2} e^{-\ell^2+1} \times \left(\frac{\ell^2}{2}+1\right)^{\ell^2/2+3/2} e^{-\ell^2/2-1}} \\
&\leq \frac{\ell^{3\ell^2} \times e^{\ell^2/2+1/2+\frac{1}{12n}}}{2\pi \left(\frac{\ell^2-1}{2}\right)^{\ell^2} \times \left(\frac{\ell^2}{2}+1\right)^{\ell^2/2+3/2}} \\
&\leq e^{\ell^2/2+1/2+\frac{1}{12n}} 2^{(3\ell^2+1)/2} \frac{\ell^{3\ell^2}}{\pi (\ell^2-1)^{\ell^2} \times (\ell^2+2)^{\ell^2/2+3/2}} \\
&\leq \frac{e^{1/2+\frac{1}{12}}}{\pi \ell^3} e^{\ell^2/2} 2^{(3\ell^2+1)/2} \frac{\ell^{3\ell^2}}{(\ell^2-1)^{\ell^2} \times (\ell^2+2)^{\ell^2/2}} \\
&\leq \frac{e^{\ell^2/2} 2^{(3\ell^2+1)/2}}{\ell^3}
\end{aligned}$$

since the fraction on the right is bounded by 1.41 for $\ell \geq 2$. This upper bound is slightly worse (by a factor π) than the equivalent given in [McK94], but this is not a problem for our purposes.

Hence, we have

$$\log|\alpha_{r,s}| \leq \frac{\ell^2}{2} + \frac{3\ell^2+1}{2} \log 2 - 3 \log \ell \leq 2 \leq 2\ell^2$$

which makes the estimate [McK94, Corollary 1] more precise.

9.3.3 A univariate polynomial

The condition "the lift Q_K of Q is an ℓ-torsion point on E_K" can be rephrased as "the generic ℓ-division polynomial is 0 when evaluated at (x_K, α, b_K)". We use this condition to find the minimal polynomial of α, which we then use to define explicitly the number field K. This means we only need to look at the univariate polynomials

$$\Phi_\ell(A) = \psi_\ell(x_K, A, y_K^2 - x_K^3 - Ax_K).$$

The recurrence relations defining ψ_ℓ can be instantiated for Φ_ℓ:

$$\begin{aligned}
\Phi_0 &= 0, \Phi_1 = 1, \Phi_2 = 1 \\
\Phi_3 &= -A^2 - 6x_K^2 A + (12x_K y_K^2 - 9x_K^4) \\
\Phi_4 &= -2A^3 - 18x_K^2 A^2 + (24x_K y_K^2 - 54x_K^4)A + (72y_K^2 x_K^3 - 54x_K^6 - 16y_K^4) \\
\Phi_{2m+1} &= \begin{cases} \Phi_{m+2}\Phi_m^3 - 16y_K^4 \Phi_{m-1}\Phi_{m+1}^3 & \text{if } m \text{ odd} \\ 16y_K^4 \Phi_{m+2}\Phi_m^3 - \Phi_{m-1}\Phi_{m+1}^3 & \text{if } m \text{ even.} \end{cases} \\
\Phi_{2m} &= (\Phi_{m+2}\Phi_{m-1}^2 - \Phi_{m-2}\Phi_{m+1}^2)\Phi_m
\end{aligned}$$

Finally, using Equation (9.3.1), we can also write our polynomial as:

$$\Phi_\ell(A) = \sum_{n+2i+3j=\chi(\ell)} \alpha_{i,j}(\ell) x_K^n A^i (d - x_K A)^j$$

with $d = y_K^2 - x_K^3$.

The remainder of this section will be dedicated to proving some properties of this polynomial.

Degree and coefficient of highest degree

Theorem 9.3.7. *We have* $\deg \Phi_\ell = \frac{\chi(\ell)}{2}$*, and the coefficient* c_ℓ *of the monomial* $A^{\deg \Phi_\ell}$ *is:*

$$c_\ell = \begin{cases} (-1)^m & \text{if } \ell = 2m+1 \\ (-1)^{m+1} m & \text{if } \ell = 2m \end{cases}$$

Proof. The intuition is that replacing x by the value of x_K and B by $y_K^2 - x_K^3 - Ax_K$ decreases the weight of the monomials in which x and B appear, which means that $\alpha_{\chi(\ell)/2,0} A^{\chi(\ell)/2}$ is the only monomial with weight $\chi(\ell)/2$ after the substitution. We give a more formal proof by induction, which is needed anyway to compute the coefficient of highest degree of Φ_ℓ. The cases are as follows:

1. For Φ_{2m+1}, with m odd :

$$\begin{aligned}
\frac{\chi(m+2)}{2} + 3\frac{\chi(m)}{2} &= \frac{(m+2)^2 - 1 + 3(m^2-1)}{4} = \frac{4m^2+4m+4-4}{4} \\
&= \frac{(2m+1)^2 - 1}{4} = \frac{\chi(2m+1)}{2} \\
\frac{\chi(m-1)}{2} + 3\frac{\chi(m+1)}{2} &= \frac{(m-1)^2 - 4 + 3((m+1)^2 - 4)}{4} = \frac{4m^2+4m-12}{4} \\
&< \frac{4m^2+4m}{4} = \frac{\chi(2m+1)}{2}
\end{aligned}$$

Hence, $\deg(\Phi_{m+2}\Phi_m^3) > \deg(\Phi_{m-1}\Phi_{m+1}^3)$. This proves that $\deg \Phi_{2m+1} = \frac{\chi(2m+1)}{2}$. Furthermore, the monomial with highest degree of Φ_{2m+1} is simply the monomial of highest degree of $\Phi_{m-1}\Phi_{m+1}^3$, which means

$$c_{2m+1} = c_{m+2}c_m^3 = (-1)^{(m+1)/2+3(m-1)/2} = (-1)^{-1} = (-1)^m$$

2. For Φ_{2m+1} with m even:

$$\begin{aligned}
\frac{\chi(m+2)}{2} + 3\frac{\chi(m)}{2} &= \frac{(m+2)^2 - 4 + 3(m^2-4)}{4} \\
&< \frac{4m^2+4m}{4} = \frac{\chi(2m+1)}{2} \\
\frac{\chi(m-1)}{2} + 3\frac{\chi(m+1)}{2} &= \frac{(m-1)^2 - 1 + 3((m+1)^2-1)}{4} \\
&= \frac{m^2 - 2m + 3m^2 + 6m}{4} = \frac{\chi(2m+1)}{2}
\end{aligned}$$

This time, $\deg(\Phi_{m+2}\Phi_m^3) < \deg(\Phi_{m-1}\Phi_{m+1}^3)$; this also proves that $\deg \Phi_{2m+1} = \frac{\chi(2m+1)}{2}$, and furthermore:

$$c_{2m+1} = -c_{m-1}c_{m+1}^3 = (-1)^{(m-2)/2+3(m/2)+1} = 1 = (-1)^m$$

3. For Φ_{2m} :

$$\begin{aligned}
\frac{\chi(m)}{2} + \frac{\chi(m+2)}{2} + 2\frac{\chi(m-1)}{2} &= \frac{m^2 - 4 + (m+2)^2 - 4 + 2((m-1)^2-1)}{4} \\
&= \frac{4m^2 - 4 + 4m - 4m}{4} = \frac{\chi(2m)}{2} \\
\frac{\chi(m)}{2} + \frac{\chi(m-2)}{2} + 2\frac{\chi(m+1)}{2} &= \frac{m^2 - 4 + (m-2)^2 - 4 + 2((m+1)^2-1)}{4} \\
&= \frac{4m^2 - 4}{4} = \frac{\chi(2m)}{2}
\end{aligned}$$

Those polynomials have the same degree; hence we have to show that we do not have cancellation of the highest-degree coefficients. We look at the two subcases:

(a) If m is even:

$$\begin{aligned}
\text{leading coeff.}(\Phi_{m+2}\Phi_{m-1}^2\Phi_m) &= (-1)^{\frac{m+2}{2}+1}\frac{m+2}{2} \times (-1)^{\frac{m}{2}+1}\frac{m}{2} = -\frac{m^2+2m}{4} \\
\text{leading coeff.}(\Phi_{m-2}\Phi_{m+1}^2\Phi_m) &= (-1)^{\frac{m-2}{2}+1}\frac{m-2}{2} \times (-1)^{\frac{m}{2}+1}\frac{m}{2} = -\frac{m^2-2m}{4}
\end{aligned}$$

Hence the coefficient of degree $\frac{\chi(2m)}{2}$ is $-\frac{m^2+2m-m^2+2m}{4} = -m \neq 0$.

(b) If m is odd:

$$\begin{aligned}
\text{leading coefficient}(\Phi_{m+2}\Phi_m\Phi m - 1^2) &= -\frac{(m-1)^2}{4} \\
\text{leading coefficient}(\Phi_{m-2}\Phi_m\Phi_{m+1}^2) &= -\frac{(m+1)^2}{4}
\end{aligned}$$

Hence the coefficient of degree $\frac{\chi(2m)}{2}$ is $-\frac{m^2-2m+1-m^2-2m-1}{4} = m \neq 0$, which completes the induction.

□

Remark 9.3.8. Our algorithm for isogeny computation over $\mathbb{F}_p$ we are discussing only considers isogenies of degree ℓ an odd prime number. Hence, in that case, the number field we define is of degree $\frac{\chi(\ell)}{2}$, and is defined by a monic polynomial.

Irreducibility of the polynomial defining the number field

We wish to use the polynomial Φ_ℓ to define a number field $K = \mathbb{Q}[X]/(\Phi_\ell)$; however, for K to be a number field, we need to have Φ_ℓ irreducible.

We conducted a small empirical study on several dozens of cases, each time choosing an odd prime ℓ such that $3 \leq \ell \leq 29$, an odd prime p such that $11 \leq p \leq 1789$, and a curve $E/\mathbb{F}_p$ at random until we found a curve with ℓ-torsion points. For each of these examples, we computed Φ_ℓ, and verified that it was an irreducible polynomial. However, a proof of this fact has so far eluded us.

We can bypass this difficulty by patching the procedure should Φ_ℓ happen to be reducible. Recall from the discussion in Section 9.3.1 that $\Phi_\ell(a_{\mathbb{F}_p}) = 0 \pmod p$. Let A_ℓ be an irreducible factor of Φ_ℓ in $\mathbb{Z}$ such that $A_\ell(a_{\mathbb{F}_p}) = 0 \pmod p$, and define $K = \mathbb{Q}[X]/(A_\ell)$. We then follow the same procedure as in Section 9.3.1, and we then still have that $\psi_\ell(x_K, \alpha, y_K^2 - x_K^3 - \alpha x_K) = 0$; furthermore, going from K to $\mathbb{F}_p$ can still be done, by replacing α by $a_{\mathbb{F}_p}$. Hence, the global torsion lifting procedure can still be carried out with an irreducible factor of Φ_ℓ, which means that K is indeed a number field.

The estimates for the size of the coefficients and of the complex roots of Φ_ℓ that we prove in the rest of this section are still valid for an irreducible factor of Φ_ℓ. In fact, these sizes are actually even lower, as well as the degree, which speeds up the rest of the procedure. However, we also have to add the cost of determining the right irreducible factor of Φ_ℓ to the total cost of the procedure; this amounts to factoring Φ_ℓ over $\mathbb{Z}[X]$. According to [VZGG13, Thm. 16.23], this costs $O(\ell^{20} + \ell^{16} \log^2 A)$ where A is a bound on the absolute value of the coefficients of Φ_ℓ. We prove in a subsequent section that $\log A = O(\ell^2 \log p)$, which gives a total complexity of $O(\ell^{20} \log^2 p)$. Although polynomial-time, this complexity is not good at all, and dominates the cost (determined in Section 9.3.5) of all the other steps.

In all that follows, we will assume that Φ_ℓ is irreducible over $\mathbb{Z}[X]$; this is true generically, as the probability of a random polynomial to be irreducible is 1, but should this not be the case, one should replace Φ_ℓ with an irreducible factor of Φ_ℓ for which $a_{\mathbb{F}_p}$ is a root modulo p.

Coefficient size

We discuss the size of the coefficients of the polynomial Φ_ℓ. We can make explicit the terms that appear after replacing B by $(d - x_K A)$:

$$\begin{aligned}\Phi_\ell(A) &= \sum_{n+2i+3j=\chi(\ell)} \sum_{k=0}^{j} (-1)^j \alpha_{i,j}(\ell) x_K^n A^i \binom{j}{k} x_K^k A^k (-d)^{j-k} \\ &= \sum_{n+2i+3j=\chi(\ell)} \sum_{k=0}^{j} (-1)^j \alpha_{i,j}(\ell) \binom{j}{k} x_K^{n+k} (-d)^{j-k} A^{i+k}\end{aligned}$$

We then use the triangle inequality, and the fact that $|d| = |y_K^2 - x_K^3| \leq \frac{p^3}{2}$:

$$\begin{aligned}\sum_{n+2i+3j=\chi(\ell)} \sum_{k=0}^{j} (-1)^j |\alpha_{i,j}(\ell) \binom{j}{k} x_K^{n+k} (-d)^{j-k} A^{i+k}| &\leq \sum_{n+2i+3j=\chi(\ell)} \sum_{k=0}^{j} \alpha_{i,j}(\ell) 2^j x_K^{n+k} |d|^{j-k} A^{i+k} \\ &\leq \sum_{n+2i+3j=\chi(\ell)} \sum_{k=0}^{j} \alpha_{i,j}(\ell) 2^j p^{n+k} \left(\frac{p^3}{2}\right)^{j-k} A^{i+k}\end{aligned}$$

We now take a closer look at the coefficient $\alpha_{i,j}(\ell) 2^j p^{n+k} \left(\frac{p^3}{2}\right)^{j-k}$. This coefficient is maximized when $k = 0$, which means the coefficients are bounded by

$$\alpha_{i,j}(\ell) p^{n+3j} = \alpha_{i,j}(\ell) p^{\chi(\ell)-2i}$$

The biggest of those coefficients is when $i = 0$, which in the end gives the bound

$$|\alpha_{i,j}(\ell) 2^j p^{n+k} \left(\frac{p^3}{2}\right)^{j-k}| \leq e^{2\ell^2} p^{\chi(\ell)}$$

Since there are at most $\chi(\ell)^2$ monomials, this gives a bound on the coefficients of the polynomial Φ_ℓ:

$$\chi(\ell)^2 e^{2\ell^2} p^{\chi(\ell)}$$

Hence the coefficients have a size bounded by

$$2\ell^2 + \chi(\ell) \log p + 2 \log \chi(\ell) \leq \chi(\ell)(\log p + 4) + 2 \log \chi(\ell) + 1.$$

Root size

We compute an upper bound on the size of the complex roots of this polynomial; this will be useful to bound the running time of the root-finding algorithm we will use to compute the embeddings from K to $\mathbb{C}$ explicitly.

Theorem 9.3.9. *Let $z \in \mathbb{C}$ such that $\Phi_\ell(z) = 0$. Then*

$$|z| \leq 2\chi(\ell)^2 e^{2\ell^2} p^2$$

Proof. Our proof is heavily inspired by the one for Cauchy's theorem on bounds of roots of polynomials. Define $Q = X^d - \sum_{i=0}^{d-1} |a_i| X^i$, where the a_i denote coefficients of Φ_ℓ. Then write

$$\frac{Q(x)}{x^d} = 1 - f(x)$$

with $f : \mathbb{R}_+^* \to \mathbb{R}$ a continuous, strictly decreasing function, which is $+\infty$ at 0 and 0 at $+\infty$. Hence this function is equal to 1 only once, which proves that Q has only one positive root. Call r this positive root; since $Q(0) < 0$, we have that Q is negative on $[0, r]$ and positive on $[r, +\infty[$.

Using the same arguments as in the previous section, but in a different order, one can prove that a bound on the coefficient of A^i in Φ_ℓ is

$$\chi(\ell)^2 e^{2\ell^2} p^{\chi(\ell)-2i}$$

Put $g = \chi(\ell)^2 \exp(2\ell^2)$ and $h = p^2$; then the coefficient of $A^{\chi(\ell)/2-i}$ in Φ_ℓ is bounded by gh^i. Putting $d = \chi(\ell)/2$, we have

$$\begin{aligned}\sum_{i=0}^{d-1}|a_i|(2gh)^i &\leq \sum_{i=0}^{d-1} h^{d-i}h^i 2^i g^{i+1} \\ &= gh^d\frac{(2g)^d-1}{2g-1} \leq (2gh)^d\frac{g}{2g-1} < (2gh)^d\end{aligned}$$

This proves that $Q(2gh) > 0$, and hence that $r \leq 2gh$.

Finally, the triangle inequality gives

$$|P(z) - z^n| \leq \sum_{i=0}^{n-1}|a_i||z|^i$$

Hence for any root $\zeta \in \mathbb{C}^*$ we have $|\zeta^n| \leq \sum_{i=0}^{n-1}|a_i||\zeta|^i$, hence $Q(|\zeta|) \leq 0$. Thus $f(|\zeta|) \geq 1 = f(r)$; since f is strictly decreasing, this proves that $|\zeta| \leq r$. This proves the result. □

Hence we can write $|z| \leq p^2\chi(\ell)^2 2^{2.89\ell^2+1}$, and hence $\log_2|z| \leq 2.89\ell^2 + 2\log_2 p + 4\log_2 \ell + 1 = O(\ell^2 + \log p)$. Note that this is a tighter bound than a bound given by Rouché's theorem, which would be of the form $r = 1 + \max_{i=0,\ldots,n-1}\{|a_i|\}$, and give an upper bound of size $O(\ell^2 \log p)$.

Computation of the polynomial

One strategy to compute $\Phi_\ell(A)$ is simply to use its definition; this means computing the generic ℓ-division polynomial ψ_ℓ, then evaluating it at $x = x_K$ and $B = d - x_K A$. The evaluation requires computing each monomial individually; assuming the powers of x and B are precomputed and cached, each monomial only requires a constant number of multiplications, which gives a total cost of $O(\ell^4)$ multiplications of integers. This is dominated by the cost of computing ψ_ℓ; in the end we get a complexity of $O(\ell^{5+\epsilon})$ integer multiplications.

A much faster way is to use the recurrence relations for Φ_ℓ, which are the recurrence relations defining ψ_ℓ instantiated for $x = x_K$ and $B = d - x_K A$. Those are recurrence relations between univariate polynomials; using them limits the number of terms and of operations compared with the case of the tri-variate polynomials ψ_ℓ. The complexity analysis of this method is exactly the same as the one for the computation of the division polynomial of an explicit elliptic curve in short Weierstrass. This is not surprising, since the former amounts to computing $\psi_\ell(x, A, B)$ knowing x, B, and the latter, knowing A, B. Hence

Proposition 9.3.10. *Computing Φ_ℓ requires $O(\ell\mathcal{M}(\ell^2))$ arithmetic operations. Since the size of the coefficients of Φ_ℓ is $O(\ell^2 \log p)$, this gives a bit complexity of $O(\ell\mathcal{M}(\ell^2)\mathcal{M}(\ell^2 \log p))$.*

Our implementation of the latter method (i.e. univariate recurrence relations) in Magma shows that the computation of Φ_ℓ takes 0.02s for $\ell = 23$ and 0.07s for $\ell = 37$, while the first strategy (compute the generic ℓ-division polynomial then substitute x and B by some values) took respectively 2s and 47s.

Empirical observations on the coefficients

We finish this section by outlining some empirical observations on Φ_ℓ. These assertions seem to hold for any polynomial we obtained when running the algorithm described in this section on several dozen examples; the parameters were $\ell \leq 29$ and $p \leq 1789$. Proving these propositions is likely to be hard, most of all because we do not know much about the coefficients $\alpha_{r,s}$ of the generic division polynomial.

Conjecture 9.3.11. *The following assertions, from weakest to strongest, hold:*

1. *The coefficient with the largest absolute value is the constant coefficient $|a_0|$.*
2. *The coefficients have decreasing absolute value, ie $|a_0| \geq |a_1| \geq |a_2| \geq \ldots \geq |a_{\chi(\ell)/2}|$.*
3. *We have $|a_0| \geq \sum_{i=1}^{\chi(\ell)/2} |a_i|$.*

Any of these assertions would yield the bound $\chi(\ell)e^{2\ell^2}2^{\chi(\ell)/3}p^{\chi(\ell)}$ on the coefficients of Φ_ℓ, which saves a factor $\chi(\ell)2^{\chi(\ell)/3}$ compared to our bound; that saving would carry over to the bound on its roots. However, this does not seem to change the asymptotic running time of our algorithm.

9.3.4 Precision required

Recall that Section 9.2.4 established that we need to choose P such that the largest denominator in the coefficients of the elements in K is smaller than 2^P, so that the conversion of complex coefficients into rational numbers can be done. Hence, we need to determine a value for P, or at least its asymptotic behavior as a function of ℓ and p. This task seems difficult, and we did not manage to prove anything rigourously; it may involve taking a closer look at Vélu's formulas and keep track of the height of the coefficients. However, we propose a conjecture, which seems to hold in practice.

Recall that a root-finding algorithm will find all d roots with precision P in time

$$O(n \log^2 n(\log^2 n + \log(P + \log S))\mathcal{M}(P + \log S))$$

with S a bound on the size of the roots. Judging by the previous section, we have

$$S \leq 2\chi(\ell)^2 \exp(2\ell^2)p^2 \leq p^2\chi(\ell)^2 2^{2.89\ell^2+1}$$

which means $\log S = 2.89\ell^2 + 2\log p + 4\log \ell + 1$. Hence, we can take $P = O(\ell^2 + \log p)$ without it changing the asymptotic complexity of our algorithm. If we had just used Rouché's theorem, and our assumption that the constant coefficient is the biggest one (which we did not manage to prove), we would have gotten the bound $\chi(\ell)(\log p + 4.47) + 2\log \chi(\ell) + 1 = O(\ell^2 \log p)$, which is a bit worse.

In practice, our experiments showed that taking $P = \ell^2 \log p$ always gave us more than enough precision for the rational reconstruction to succeed; indeed, the size of the denominators was often much smaller than this bound. Hence, we put forward the following conjecture:

Conjecture 9.3.12. *Taking $P = O(\ell^2 \log p)$ is enough to recognize the rational coefficients; in fact, $P = \ell^2 \log p$ is enough.*

We verified this conjectures on curves up to $\ell = 29$ and on fields up to $p = 1789$.

9.3.5 Description of the algorithm

We now state the final algorithm to compute an isogenous curve over $\mathbb{F}_p$ with given kernel. Recall from the discussion in Section 9.3.3 that we assume that Φ_ℓ is an irreducible polynomial over $\mathbb{Z}[X]$, which seems true in practice and is true generically with probability 1.

Proposition 9.3.13. *Algorithm 27 has an asymptotic complexity of*

$$O\left((\ell^3 + \ell^3 \log P + \ell\mathcal{M}(\ell^2)\log \ell)\mathcal{M}(P) + \ell\mathcal{M}(\ell^2)\mathcal{M}(\ell^2 \log p) + \mathcal{M}(\log p)\log\log p\right)$$

bit operations.

Algorithm 27 Compute an ℓ-isogenous curve and an isogeny with a given kernel, over $\mathbb{F}_p$.
Input: $E(\mathbb{F}_p) : y^2 = x^3 + ax + b$ $(a, b \in \mathbb{F}_p)$, $P \in E[\ell]$.
Output: $E'(\mathbb{F}_p) : y^2 = x^3 + a'x + b$ $(a', b' \in K)$, ℓ-isogenous to E; the rational function defining the ℓ-isogeny $\phi : E \to E'$.

1: Put $P_k = (x_K, y_K)$, where $-p/2 \leq x_K, y_K \leq p/2$.
2: Compute the polynomial Φ_ℓ; define $K = \mathbb{Q}[X]/(\Phi_\ell)$.
3: Put $a_K = X$ and $b_K = (y_K^2 - x_K^3 - a_K x_K)$ and define $E_K : y^2 = x^3 + a_K x + b_K$.
4: Use Algorithm 26 to compute $E'_K : y^2 = x^3 + a'_K x + b'_K$, and the ℓ-isogeny over K, ϕ_K.
5: Compute the elements in $\mathbb{F}_p$ corresponding to each coefficient of the curve and of the rational function, using Section 9.3.1; return the result.

Note that this complexity is much lower than the cost of factoring Φ_ℓ over $\mathbb{Z}[X]$ (which is $O(\ell^{20} \log^2 p)$ as analyzed in Section 9.3.3); hence, patching the algorithm in the event that Φ_ℓ has a cost far superior to the rest of the algorithm.

Proof. Recall that the total cost of Algorithm 26 is

$$O((n \log^4 n + n \log^2 n \log P + n\ell \log P + \ell \mathcal{M}(n) \log n)\mathcal{M}(P))$$

In our case, we have $n = \deg K = O(\ell^2)$, which gives a running time of

$$O((\ell^3 \log P + \ell \mathcal{M}(\ell^2) \log \ell)\mathcal{M}(P)).$$

Step 2 requires, according to Proposition 9.3.10, $O(\ell \mathcal{M}(\ell^2)\mathcal{M}(\ell^2 \log p))$ bit operations.

We determine the complexity of Step 5 by first determining the complexity of reducing one element, $u = r(\alpha) \in K$, into an element of $\mathbb{F}_p$. Reducing the $O(\ell^2)$ coefficients of r, which are rational numbers, requires reducing $O(\ell^2)$ numerators and denominators modulo p, and computing the inverse of the denominators. If we suppose that the size of each numerator and denominator is $O(P)$, one can then determine the quotient of the division of one of those integers by p by computing $\frac{1}{p}$ with precision P, then multiply the integer by $\frac{1}{p}$ and take the integral part; this requires $O(\mathcal{M}(P))$ operations per coefficient. Computing the inverse of the denominators is done by computing the inverse of the product and recover each inverse, which requires $O(\mathcal{M}(\log p)(\ell^2 + \log \log p))$ bit operations. Hence the cost of this step is $O(\ell^2 \mathcal{M}(P) + \mathcal{M}(\log p) \log \log p)$. Finally, evaluating $\overline{r}$ in $a_{\mathbb{F}_p}$ can be done using Horner's rule, and requires $O(\ell^2)$ operations in $\mathbb{F}_p$, which is negligible. We thus get a total cost of

$$O(\ell^2 \mathcal{M}(P) + \mathcal{M}(\log p) \log \log p)$$

bit operations for the reduction of a coefficient in $\mathbb{F}_p$, and hence a cost of

$$O(\ell^3 \mathcal{M}(P) + \mathcal{M}(\log p) \log \log p)$$

for reducing all the coefficients of an ℓ-isogeny. This gives the claimed complexity. □

If we suppose that the estimate $P = O(\ell^2 \log p)$ is correct, we get an overall complexity of

$$O\left((\ell^3 + \ell^3 \log \log p + \ell \mathcal{M}(\ell^2) \log \ell)\mathcal{M}(\ell^2 \log p)\right) = O(\ell^{5+\epsilon} \log^{1+\epsilon} p).$$

Vélu's formulas require $O(\mathcal{M}(\ell) \log \ell)$ field operations, each costing $O\left(\log^{1+\epsilon} p\right)$; this complexity is much better than our algorithm's.

9.4 Extending this idea to other settings

The algorithms we presented in this chapter relied on the idea of using the algorithms for computing Abel-Jacobi map quickly in order to reduce the problem to the easy case of isogeny computation on complex tori. We show here how this idea could be adapted to other settings.

Extending the algorithm to higher genera

We start by noting that there is no straightforward generalization of Vélu's formulas to genus g; hence, other methods may need to be used to solve the hyperelliptic equivalent of Problem 1.1.16.

It should be possible to generalize Algorithm 25 (i.e. the one working over $\mathbb{C}$) to curves of higher gernus in a rather straightforward manner. Indeed, in higher genera, evaluating isogenies on complex tori is also computationally easy (see Note 1.4.12), and much simpler than evaluating isogenies on algebraic representations. Our algorithms in Chapter 8 can then provide the link between analytic and algebraic representations in quasi-linear time, at least for genus 2. Hence, the strategy of going to the complex torus to evaluate the isogeny, then come back to the algebraic representation and interpolate the isogeny can be generalized to higher genus. We leave the details of this generalization to future work.

Generalizing the algorithm to K would then be very straightforward: the same method of computing embeddings to $\mathbb{C}$ to a high enough precision, solving the problem on $\mathbb{C}$ then interpolating the coefficients of the rational function over K would work in exactly the same way.

However, note that the generalization of the algorithm to $\mathbb{F}_p$ is not as straightforward. In particular, the generalization of the global torsion lifting procedure is unclear, as it requires the computation of (generic) ℓ-division polynomials in genus 1; the generalization of such polynomials to genus 2 and their properties have to be studied further in order to show that a similar global torsion lifting procedure could be used.

Solving other problems

Finally, we note that it may be possible to use a very similar idea to solve other problems related to isogenies. This section is inspired by an idea of [VW00], who uses a similar idea to find an isogeny between two genus 2 curves.

In particular, it seems like the problem of finding ℓ-isogenies between two given curves (Problem 1.1.19) may be solvable using this idea. This would first require the computation of the two periods associated to each of the two given curves; once this is done, one can attempt to compute the ℓ-isogeny between the tori, by computing an $\alpha \in \mathbb{C}$ which sends Λ_1 onto Λ_2. This is potentially not very hard, as we can reduce the periods so that their quotient is in the fundamental domain, then look for relations of the form $\tau_2 = \frac{m\tau_1+n}{\ell}$. Once α has been found, the isogeny can easily be computed.

This algorithm would extend to K very straightforwardly. Extending it to $\mathbb{F}_p$ would require finding a lifting procedure which would preserve the isogeny and lift both curves to curves defined over the same number field; we do not know how to solve this problem.

Finally, we note that [VW00] uses very similar ideas to find isogenies between genus 2 curves, without knowing the degree of the isogeny beforehand. The idea is to use the LLL algorithm to attempt to find integer relations between the periods defining each lattice; this idea seems rather natural, and may lead to an algorithm to solve Problem 1.1.20 over $\mathbb{C}$ and K. As before, the generalization of the lifting procedure in a way that preserves the isogeny is unclear, and the algorithm does not seem to generalize straightforwardly to $\mathbb{F}_p$.

Bibliography

[AS64] Milton Abramowitz and Irene A. Stegun. *Handbook of mathematical functions: with formulas, graphs, and mathematical tables*, volume 55 of *Applied Mathematics Series*. National Bureau of Standards, 1964.

[BB87] Jonathan M. Borwein and Peter B. Borwein. *Pi and the AGM: a study in the analytic number theory and computational complexity*. Canadian Mathematical Society series of monographs and advanced texts. Wiley-Interscience, 1987.

[BCHL16] Joppe W. Bos, Craig Costello, Huseyin Hisil, and Kristin Lauter. Fast cryptography in genus 2. *Journal of Cryptology*, 29(1):28–60, 2016. Springer.

[BCSS97] Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and real computation*. Springer Science & Business Media, 1997.

[Bel61] Richard Bellman. *A brief introduction to theta functions*. Athena Series Selected Topics in Mathematics. Holt, Rinehart and Winston, 1961.

[Bel10] Fabrice Bellard. Computation of 2700 billion decimal digits of pi using a desktop computer. `http://bellard.org/pi/pi2700e9/pipcrecord.pdf`, February 2010. 4th revision.

[BH13] Richard P. Brent and David Harvey. Fast computation of bernoulli, tangent and secant numbers. In *Computational and Analytical Mathematics*, pages 127–142. Springer, 2013.

[BJS14] Jean-François Biasse, David Jao, and Anirudh Sankar. A quantum algorithm for computing isogenies between supersingular elliptic curves. In *Progress in Cryptology–INDOCRYPT 2014*, pages 428–442. Springer, 2014.

[BK10] Joppe W. Bos and Marcelo E. Kaihara. PlayStation 3 computing breaks 2^{60} barrier – 112-bit prime ECDLP solved, 2010. `http://lacal.epfl.ch/112bit_prime`.

[BL13] Daniel Bernstein and Tanja Lange. Two grumpy giants and a baby. *The Open Book Series*, 1(1):87–111, 2013. Mathematical Sciences Publishers.

[BM88] Jean-Benoît Bost and Jean-François Mestre. Moyenne arithmético-géométrique et périodes des courbes de genre 1 et 2. *Gaz. Math*, 38:36–64, 1988.

[BMSS08] Alin Bostan, François Morain, Bruno Salvy, and Éric Schost. Fast algorithms for computing isogenies between elliptic curves. *Mathematics of Computation*, 77(263):1755–1778, 2008. American Mathematical Society.

[Bor78] Carl-Wilhelm Borchardt. Theorie des arithmetisch-geometrisches Mittels aux vier Elementen. *Mathematische Abhandlungen der Akademie der Wissenschaften zu Berlin*, page 33–96, 1878.

[Bor76] Carl-Wilhelm Borchardt. Ueber das arithmetisch-geometrische Mittel aus vier Elementen. *Monatsbericht der Akademie der Wissenschaften zu Berlin*, page 611–621, November 1976.

[Bre76] Richard P. Brent. The complexity of multiple-precision arithmetic. *The Complexity of Computational Problem Solving*, pages 126–165, 1976.

[BSS99] Ian F. Blake, Gadiel Seroussi, and Nigel Smart. *Elliptic Curves in Cryptography*, volume 265 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 1999.

[BZ10] Richard P. Brent and Paul Zimmerman. *Modern Computer Arithmetic*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2010.

[CC89] David V. Chudnovsky and Gregory V. Chudnovsky. The computation of classical constants. *Proceedings of the National Academy of Sciences of the United States of America*, 86(21):8178–8182, 1989. National Acad Sciences.

[CCS15] Ping Ngai Chung, Craig Costello, and Benjamin Smith. Fast, uniform, and compact scalar multiplication for elliptic curves and genus 2 Jacobians with applications to signature schemes. preprint, October 2015.

[CDSLY14] Craig Costello, Alyson Deines-Schartz, Kristin Lauter, and Tonghai Yang. Constructing abelian surfaces for cryptography via Rosenhain invariants. *LMS Journal of Computation and Mathematics*, 17(A):157–180, 2014. Cambridge University Press.

[CFA+10] Henri Cohen, Gerhard Frey, Roberto Avanzi, Christophe Doche, Tanja Lange, Kim Nguyen, and Frederik Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Discrete mathematics and its applications. CRC press, 2010.

[CGL90] Robert Coquereaux, Alex Grossmann, and Benny E. Lautrup. Iterative method for calculation of the Weierstrass elliptic function. *IMA journal of numerical analysis*, 10(1):119–128, 1990. Oxford University Press.

[Cha85] Komaravolu Chandrasekharan. *Elliptic functions*, volume 281 of *Grundlelaren tier mathematischen Wissenschaften*. Springer, 1985.

[CJS14] Andrew Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *Journal of Mathematical Cryptology*, 8(1):1–29, 2014. De Gruyter.

[Cle80] Charles H. Clemens. *A scrapbook of complex curve theory*, volume 55 of *The University Series in Mathematics*. American Mathematical Soc., 1980.

[CLN16] Craig Costello, Patrick Longa, and Michael Naehrig. Efficient algorithms for supersingular isogeny Diffie-Hellman. In *Advances in Cryptology – CRYPTO 2016*. Springer, 2016.

[Coh93] Henri Cohen. *A Course in Computational Algebraic Number Theory*, volume 138 of *Graduate Texts in Mathematics*. Springer, 1993.

[Cos11] Romain Cosset. *Applications des fonctions thêta à la cryptographie sur courbes hyperelliptiques*. PhD thesis, Université Henri Poincaré-Nancy I, 2011.

[Cou94] Jean-Marc Couveignes. *Quelques calculs en théorie des nombres*. PhD thesis, Université de Bordeaux I, 1994.

[Cou96] Jean-Marc Couveignes. Computing ℓ-isogenies using the p-torsion. In *Algorithmic Number Theory Symposium – ANTS-II*, Lecture Notes in Computer Science 1122, pages 59–65. Springer, 1996.

[Cou06] Jean-Marc Couveignes. Hard Homogeneous Spaces. *Cryptology ePrint Archive*, 2006. IACR ePrint/2006/291.

[Cox84] David A. Cox. The arithmetic-geometric mean of Gauss. *L'Enseignement Mathématique*, 30(2):275–330, 1984.

[CR15] Romain Cosset and Damien Robert. Computing (ℓ, ℓ)-isogenies in polynomial time on jacobians of genus 2 curves. *Mathematics of Computation*, 84(294):1953–1975, 2015. American Mathematical Society.

[CS93] John H. Conway and Neil J. A. Sloane. Sphere packings, lattices and groups, 1993.

[CT13] John E. Cremona and Thotsaphon Thongjunthug. The complex AGM, periods of elliptic curves over $\mathbb{C}$ and complex elliptic logarithms. *Journal of Number Theory*, 133(8):2813–2841, August 2013. Elsevier.

[DF11] Luca De Feo. Fast algorithms for computing isogenies between ordinary elliptic curves in small characteristic. *Journal of Number Theory*, 131(5):873–893, 2011. Elsevier.

[DFHPS16] Luca De Feo, Cyril Hugounenq, Jérôme Plût, and Éric Schost. Explicit isogenies in quadratic time in any characteristic. *arXiv preprint arXiv:1603.00711*, 2016.

[DFJP14] Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Journal of Mathematical Cryptology*, 8(3):209–247, 2014. De Gruyter.

[DG16] Christina Delfs and Steven D. Galbraith. Computing isogenies between supersingular elliptic curves over $\mathbb{F}_p$. *Designs, Codes and Cryptography*, 78(2):425–440, 2016. Springer.

[DHB+04] Bernard Deconinck, Matthias Heil, Alexander Bobenko, Mark Van Hoeij, and Marcus Schmies. Computing Riemann theta functions. *Mathematics of Computation*, 73(247):1417–1442, 2004. American Mathematical Society.

[DIK06] Christophe Doche, Thomas Icart, and David R. Kohel. Efficient scalar multiplication by isogeny decompositions. In *Public Key Cryptography – PKC 2006*, Lecture Notes in Computer Science 3958, pages 191–206. Springer, 2006.

[DT08] Claus Diem and Emmanuel Thomé. Index calculus in class groups of non-hyperelliptic curves of genus three. *Journal of Cryptology*, 21(4):593–611, 2008. Springer.

[Dup06] Régis Dupont. *Moyenne arithmético-géométrique, suites de Borchardt et applications*. PhD thesis, École polytechnique, Palaiseau, 2006.

[Dup11] Régis Dupont. Fast evaluation of modular functions using Newton iterations and the AGM. *Mathematics of Computation*, 80(275):1823–1847, 2011. American Mathematical Society.

[EG02] Andreas Enge and Pierrick Gaudry. A general framework for subexponential discrete logarithm algorithms. *Acta Arithmetica*, 102:83–103, 2002. Instytut Matematyczny PAN.

[EGT11] Andreas Enge, Pierrick Gaudry, and Emmanuel Thomé. An L(1/3) discrete logarithm algorithm for low degree curves. *Journal of Cryptology*, 24(1):24–41, 2011. Springer.

[EGTZ12] Andreas Enge, Mickaël Gastineau, Philippe Théveny, and Paul Zimmerman. GNU MPC – *A library for multiprecision complex arithmetic with exact rounding*. INRIA, September 2012. Release 1.0.1, `http://mpc.multiprecision.org/`.

[Eng09] Andreas Enge. The complexity of class polynomial computation via floating point approximations. *Mathematics of Computation*, 78(266):1089–1107, 2009. American Mathematical Society.

[ET14a] Andreas Enge and Emmanuel Thomé. Computing class polynomials for abelian surfaces. *Experimental Mathematics*, 23(2):129–145, 2014. Taylor & Francis.

[ET14b] Andreas Enge and Emmanuel Thomé. CMH — *Computation of Igusa Class Polynomials*, December 2014. Version 1.0, `http://cmh.gforge.inria.fr/`.

[ETZ] Andreas Enge, Philippe Théveny, and Paul Zimmerman. MPC: Algorithms and Error Analysis. draft available in the MPC repository (`https://gforge.inria.fr/projects/mpc`).

[Für09] Martin Fürer. Faster integer multiplication. *SIAM Journal on Computing*, 39(3):979–1005, 2009. SIAM.

[Gal99] Steven D. Galbraith. Constructing isogenies between elliptic curves over finite fields. *LMS Journal of Computation and Mathematics*, 2(1):118–138, 1999. Cambridge University Press.

[Gal12] Steven D. Galbraith. *Mathematics of public key cryptography*. Cambridge University Press, 2012.

[Gau00] Pierrick Gaudry. An algorithm for solving the discrete log problem on hyperelliptic curves. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 19–34. Springer, 2000.

[Gau07] Pierrick Gaudry. Fast genus 2 arithmetic based on Theta functions. *Journal of Mathematical Cryptology*, 1(3):243–265, 2007. De Gruyter.

[Gep28] Harald Geppert. Zur Theorie des arithmetisch-geometrischen Mittels. *Mathematische Annalen*, 99(1):162–180, 1928. Springer.

[GG16] Steven Galbraith and Pierrick Gaudry. Recent progress on the elliptic curve discrete logarithm problem. *Designs, Codes and Cryptography*, 78(1):51–72, 2016. Springer Verlag.

[GHS02a] Steven D. Galbraith, Florian Heß, and Nigel P. Smart. Extending the GHS Weil descent attack. In *Advances in Cryptology—EUROCRYPT 2002*, pages 29–44. Springer, 2002.

[GHS02b] Pierrick Gaudry, Florian Hess, and Nigel P. Smart. Constructive and destructive facets of Weil descent on elliptic curves. *Journal of Cryptology*, 15(1):19–46, 2002. Springer.

[Got59] Erhard Gottschling. Explizite Bestimmung der Randflächen des Fundamentalbereiches der Modulgruppe Zweiten Grades. *Mathematische Annalen*, 138(2):103–124, 1959. Springer.

[Gou96] Xavier Gourdon. *Combinatoire, algorithmique et géométrie des polynômes*. PhD thesis, École Polytechnique, 1996.

[Gra88] David Grant. A generalization of jacobi's derivative formula to dimension two. *J. reine angew. Math*, 392:125–136, 1988.

[GS13] Steven Galbraith and Anton Stolbunov. Improved algorithm for the isogeny problem for ordinary elliptic curves. *Applicable Algebra in Engineering, Communication and Computing*, 24(2):107–131, 2013. Springer.

[GTTD07] Pierrick Gaudry, Emmanuel Thomé, Nicolas Thériault, and Claus Diem. A double large prime variation for small genus hyperelliptic index calculus. *Mathematics of Computation*, 76(257):475–492, 2007. American Mathematical Society.

[GWZ15] Steven D. Galbraith, Ping Wang, and Fangguo Zhang. Computing elliptic curve discrete logarithms with improved baby-step giant-step algorithm. *Cryptology ePrint Archive*, 2015. IACR ePrint/2015/605.

[Har14] David Harvey. A subquadratic algorithm for computing the n-th Bernoulli number. *Mathematics of Computation*, 83(289):2471–2477, 2014. American Mathematical Society.

[Hel85] Bettina Helfrich. Algorithms to construct Minkowski reduced and Hermite reduced lattice bases. *Theoretical Computer Science*, 41:125–139, 1985. Elsevier.

[Hig02] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.

[Igu72] Jun-Ichi Igusa. *Theta functions*, volume 194 of *Die Grundlehren der mathematischen Wissenschaften*. Springer, 1972.

[Igu80] Jun-ichi Igusa. On Jacobi's derivative formula and its generalizations. *American Journal of Mathematics*, pages 409–446, 1980.

[Jar08] Frazer Jarvis. Higher genus arithmetic-geometric means. *The Ramanujan Journal*, 17(1):1–17, 2008. Springer.

[JMV05] David Jao, Stephen D. Miller, and Ramarathnam Venkatesan. Do all elliptic curves of the same order have the same difficulty of discrete log? In *Advances in Cryptology-ASIACRYPT 2005*, pages 21–40. Springer, 2005.

[KAF+10] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K. Lenstra, Emmanuel Thomé, Joppe W. Bos, Pierrick Gaudry, Alexander Kruppa, Peter L. Montgomery, Dag Arne Osvik, et al. Factorization of a 768-bit RSA modulus. In *Advances in Cryptology – CRYPTO 2010*, pages 333–350. Springer, 2010.

[Kar95] Ekatherina A. Karatsuba. Fast calculation of the Riemann zeta function $\zeta(s)$ for integer values of the argument s. *Problemy Peredachi Informatsii*, 31(4):69–80, 1995. Russian Academy of Sciences, Branch of Informatics, Computer Equipment and Automatization.

[Kli90] Helmut Klingen. *Introductory lectures on Siegel modular forms*, volume 20 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1990.

[Kob84] Neal Koblitz. *Introduction to elliptic curves and modular forms*, volume 97 of *Graduate Texts in Mathematics*. Springer, 1984.

[Koh96] David R. Kohel. *Endomorphism rings of elliptic curves over finite fields*. PhD thesis, University of California at Berkeley, 1996.

[Lab15] Hugo Labrande. Computing Jacobi's θ in quasi-linear time. `http://arxiv.org/abs/1511.04248`, 2015.

[Ler97] Reynald Lercier. *Algorithmique des courbes elliptiques dans les corps finis*. PhD thesis, École polytechnique, Palaiseau, 1997.

[LO94] Wolfram Luther and Werner Otten. Computation of Standard Interval Functions in Multiple-Precision Interval Arithmetic. *Interval Compputations*, 4(2(6)):78–99, 1994.

[LO98] Wolfram Luther and Werner Otten. Reliable computation of elliptic functions. *Journal of Universal Computer Science*, 4(1):25–33, 1998.

[LT16] Hugo Labrande and Emmanuel Thomé. Computing theta functions in quasi-linear time in genus 2 and above. In University of Kaiserslautern, editor, *Twelfth Algorithmic Number Theory Symposium (ANTS-XII)*, 2016.

[Mak75] O. M. Makarov. The connection between algorithms of the fast Fourier and Hadamard transformations and the algorithms of Karatsuba, Strassen, and Winograd. *USSR Computational Mathematics and Mathematical Physics*, 15(5):1–11, 1975. Elsevier.

[McK94] James McKee. Computing division polynomials. *Mathematics of computation*, 63(208):767–771, 1994. American Mathematical Society.

[MP13] John M. McNamee and Victor Pan. *Numerical methods for roots of polynomials*, volume 16 of *Series in Computational Mathematics*. Newnes, 2013.

[MTW04] Alfred Menezes, Edlyn Teske, and Annegret Weng. Weak fields for ECC. In *Cryptographers' Track at the RSA Conference*, pages 366–386. Springer, 2004.

[Mum83] David Mumford. *Tata lectures on Theta, Vol. I.* Modern Birkhäuser Classics. Birkhäuser, Boston, 1983.

[Mum84] David Mumford. *Tata lectures on Theta, Vol. II.* Modern Birkhäuser Classics. Birkhäuser, Boston, 1984.

[NS04] Phong Q. Nguyen and Damien Stehlé. Low-dimensional lattice basis reduction revisited. In *Algorithmic Number Theory, 6th International Symposium, ANTS-VI*, pages 338–357. Springer, 2004.

[Pan01] Victor Y. Pan. Univariate polynomials: nearly optimal algorithms for factorization and rootfinding. In *Proceedings of the 2001 international symposium on Symbolic and algebraic computation*, pages 253–267. ACM, 2001.

[Rob55] Herbert Robbins. A remark on Stirling's formula. *The American Mathematical Monthly*, 62(1):26–29, 1955. JSTOR.

[Sch82] Arnold Schönhage. The fundamental theorem of algebra in terms of computational complexity. Technical report, Univ. of Tübingen, Germany, 1982.

[Sch95] René Schoof. Counting points on elliptic curves over finite fields. *Journal de théorie des nombres de Bordeaux*, 7(1):219–254, 1995.

[Sie89] Carl Ludwig Siegel. *Topics in Complex Function Theory: Abelian Functions and Modular Functions of Several Variables.* Number 25 in Interscience tracts in pure and applied mathematics. John Wiley & Sons, 1989.

[Sil86] Joseph H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Graduate Texts in Mathematics.* Springer, 1986.

[Sil09] Joseph H. Silverman. Lifting and Elliptic Curve Discrete Logarithms. In *Selected Areas in Cryptography*, pages 82–102. Springer, 2009.

[SK82] Tateaki Sasaki and Yasumasa Kanada. Practically fast multiple-precision evaluation of $\log(x)$. *J. Inf. Process*, 5:247–250, 1982.

[Smi09] Benjamin Smith. Isogenies and the Discrete Logarithm Problem in Jacobians of Genus 3 Hyperelliptic Curves. *Journal of Cryptology*, 22(4):505–529, 2009. Springer.

[Sto12] Anton Stolbunov. *Cryptographic schemes based on isogenies.* PhD thesis, Norwegian University of Science and Technology (NTNU), 2012.

[Str14] Marco Streng. Computing Igusa class polynomials. *Math. Comp.*, 83(285), 2014.

[Sut13] Andrew Sutherland. Isogeny volcanoes. *The Open Book Series*, 1(1):507–530, 2013. Mathematical Sciences Publishers.

[SvS12] Jeroen Spandaw and Duco van Straten. Hyperelliptic integrals and generalized arithmetic–geometric mean. *The Ramanujan Journal*, 28(1):61–78, 2012. Springer.

[Tes06] Edlyn Teske. An elliptic curve trapdoor system. *Journal of Cryptology*, 19(1):115–133, 2006. Springer.

[The14] The PARI Group, Bordeaux. *PARI/GP version* `2.7.2`, 2014. available from `http://pari.math.u-bordeaux.fr/`.

[Vél71] Jacques Vélu. Isogénies entre courbes elliptiques. *CR Acad. Sci. Paris Sér. AB*, 273:A238–A241, 1971.

[VV09] Brigitte Vallée and Antonio Vera. Probabilistic analyses of lattice reduction algorithms. In *The LLL Algorithm*, pages 71–143. Springer, 2009.

[VW99] Paul Van Wamelen. Proving that a genus 2 curve has complex multiplication. *Mathematics of Computation*, 68(228):1663–1677, 1999. American Mathematical Society.

[VW00] Paul Van Wamelen. Poonen's question concerning isogenies between Smart's genus 2 curves. *Mathematics of Computation*, 69(232):1685–1697, 2000. American Mathematical Society.

[VZGG13] Joachim Von Zur Gathen and Jürgen Gerhard. *Modern computer algebra*. Cambridge University Press, 2013.

[Was08] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography*. Discrete mathematics and its applications. CRC press, 2008.

[Web21] Heinrich Weber. *Lehrbuch der Algebra*, volume 3. Druck und verlag Fr. Vieweg & Sohn, 1921.

[WW27] Edmund T. Whittaker and George N. Watson. *A course of modern analysis*. Cambridge University Press, 1927.

[YK11] Alexander Yee and Shigeru Kondo. 10 trillion digits of pi: A case study of summing hypergeometric series to high precision on multicore systems. Technical report, University of Illinois Urbana-Champaign, `http://hdl.handle.net/2142/28348`, 2011.

Appendix A

Absolute loss of precision in elementary fixed-point operations

This appendix outlines bounds on the absolute error that can arise when computing elementary operations in fixed point arithmetic with numbers in precision P. It is heavily inspired by the approach and the methods of [ETZ]; our results are adapted to the case of fixed-point arithmetic and, in some cases, give smaller error bounds than what can be obtained from [ETZ].

We write $z_k = x_k + iy_k$, and $\tilde{z_k} = \tilde{x_k} + i\tilde{y_k}$ its approximation with fixed precision $P \geq 1$. We suppose that

$$|z_j - \tilde{z_j}| \leq k_j 2^{-P}$$

which implies that $|x_j - \tilde{x_j}| \leq k_j 2^{-P}, |y_j - \tilde{y_j}| \leq k_j 2^{-P}$.

We recall Theorem 0.3.3:

Theorem A.0.1. *For $j = 1, 2$, let $z_j = x_j + iy_j \in \mathbb{C}$ and $\tilde{z_j} = \tilde{x_j} + i\tilde{y_j}$ its approximation. Suppose that:*

- $|z_j - \tilde{z_j}| \leq k_j 2^{-P}$;
- $k_j \leq 2^{P/2}$ *(which means that at least the majority of the bits are correct);*
- $k_j 2^{-P} \leq |z_j| \leq 2^{P/2-3}$.

Then

1. $|\operatorname{Re}(z_1 + z_2) - \operatorname{Re}(\tilde{z_1} + \tilde{z_2})| \leq (k_1 + k_2)2^{-P}$
2. $|\operatorname{Re}(z_1 z_2) - \operatorname{Re}(\tilde{z_1}\tilde{z_2})| \leq (2 + 2k_1|z_2| + 2k_2|z_1|)2^{-P}$
3. $|\operatorname{Re}(z_1^2) - \operatorname{Re}(\tilde{z_1}^2)| \leq (2 + 4k_1|z_1|)2^{-P}$

and the same bounds apply to imaginary parts as well; and

4. $|e^{z_1} - e^{\tilde{z_1}}| \leq |e^{z_1}| \frac{7k_1+8.5}{2} 2^{-P}$.

Furthermore if $|z_j| \geq 2k_j 2^{-P}$,

5. $|\operatorname{Re}\left(\frac{z_1}{z_2}\right) - \operatorname{Re}\left(\frac{\tilde{z_1}}{\tilde{z_2}}\right)| \leq \left(\frac{6(2+2k_1|z_2|+2k_2|z_1|)}{|z_2|^2} + \frac{2(4+8k_2|z_2|)(2|z_1||z_2|+1)+2}{|z_2|^4}\right) 2^{-P}$

and the same bound applies to the imaginary part, and

6. $|\sqrt{z_1} - \sqrt{\tilde{z_1}}| \leq \frac{k_1}{\sqrt{|z_1|}} 2^{-P}$.

We prove this theorem over the next few subsections.

Addition

If $z = z_1 + z_2$ we have

$$\begin{aligned} |x_1 + x_2 - (\tilde{x_1} + \tilde{x_2})| &\leq |x_1 - \tilde{x_1}| + |x_2 - \tilde{x_2}| \leq (k_1 + k_2)2^{-P} \\ |y_1 + y_2 - (\tilde{y_1} + \tilde{y_2})| &\leq |y_1 - \tilde{y_1}| + |y_2 - \tilde{y_2}| \leq (k_1 + k_2)2^{-P} \end{aligned}$$

Predictably, "the errors add up".

Multiplication

If $z = z_1 z_2$ we have $x = x_1 x_2 - y_1 y_2$ and $y = x_1 y_2 + x_2 y_1$. We write

$$\begin{aligned} |x_1 x_2 - \tilde{x_1}\tilde{x_2}| &\leq \frac{1}{2}|(x_1 - \tilde{x_1})(x_2 + \tilde{x_2}) + (x_1 + \tilde{x_1})(x_2 - \tilde{x_2})| \\ &\leq \frac{1}{2}\left(|x_1 - \tilde{x_1}|(|x_2| + |\tilde{x_2}|) + (|x_1| + |\tilde{x_1}|)|x_2 - \tilde{x_2}|\right) \\ &\leq \frac{1}{2}\left(k_1(|x_2| + |\tilde{x_2}|) + k_2(|x_1| + |\tilde{x_1}|)\right)2^{-P} \\ &\leq (k_1|x_2| + k_2|x_1|)2^{-P} + k_1 2^{-P} k_2 2^{-P} \end{aligned}$$

Since $k_1 \leq 2^{P/2}$ and $k_2 \leq 2^{P/2}$,

$$|x_1 x_2 - \tilde{x_1}\tilde{x_2}| \leq (k_1|x_2| + k_2|x_1| + 1)2^{-P}$$

Similarly we have

$$\begin{aligned} |y_1 y_2 - \tilde{y_1}\tilde{y_2}| &\leq (k_1|y_2| + k_2|y_1| + 1)2^{-P} \\ |x_1 y_2 - \tilde{x_1}\tilde{y_2}| &\leq (k_1|y_2| + k_2|x_1| + 1)2^{-P} \\ |y_1 x_2 - \tilde{y_1}\tilde{x_2}| &\leq (k_1|x_2| + k_2|y_1| + 1)2^{-P} \end{aligned}$$

Hence

$$\begin{aligned} |x - \tilde{x}| &\leq (2 + k_1|x_2| + k_2|x_1| + k_1|y_2| + k_2|y_1|)2^{-P} \\ |y - \tilde{y}| &\leq (2 + k_1|y_2| + k_2|x_1| + k_1|x_2| + k_2|y_1|)2^{-P} \end{aligned}$$

Bounding the norms of real and imaginary parts by the norm of the number itself:

$$\begin{aligned} |x - \tilde{x}| &\leq (2 + 2k_1|z_2| + 2k_2|z_1|)2^{-P} \\ |y - \tilde{y}| &\leq (2 + 2k_1|z_2| + 2k_2|z_1|)2^{-P} \end{aligned}$$

Squaring

The above formulas can be simplified in the case $z = z_1^2$:

$$\begin{aligned} |x_1^2 - \tilde{x_1}^2| &\leq (2k_1|x_1| + 1)2^{-P} \\ |y_1^2 - \tilde{y_1}^2| &\leq (2k_1|y_1| + 1)2^{-P} \\ |x_1 y_1 - \tilde{x_1}\tilde{y_1}| &\leq (k_1|y_1| + k_1|x_1| + 1)2^{-P} \end{aligned}$$

which gives the bounds

$$|x - \tilde{x}| \leq (2 + 2k_1(|x_1| + |y_1|))2^{-P}$$
$$|y - \tilde{y}| \leq (2 + 2k_1(|x_1| + |y_1|))2^{-P}$$

Bounding the norms of real and imaginary parts by the norm of the number itself gives:

$$|x - \tilde{x}| \leq (2 + 4k_1|z_1|)2^{-P}$$
$$|y - \tilde{y}| \leq (2 + 4k_1|z_1|)2^{-P}$$

Norm

We wish to compute $z = |z_1|^2$. We use previous estimates to write:

$$\begin{aligned} ||z|^2 - |\tilde{z}|^2| &\leq |x_1^2 - \tilde{x_1}^2| + |y_1^2 - \tilde{y_1}^2| \\ &\leq (4 + 4k_1|x_1| + 4k_1|y_1|)2^{-P} \end{aligned}$$

Again, bounding the norms of real and imaginary parts by the norm of the number itself:

$$||z|^2 - |\tilde{z}|^2| \leq (4 + 8k_1|z_1|)2^{-P}$$

Interlude: Division of a real by a positive real

Let us assume $|a_1 - \tilde{a_1}| \leq k_1 2^{-P}$ and $|a_2 - \tilde{a_2}| \leq k_2 2^{-P}$, with $a_2 > 0$. In addition, assume that $a_2 > k_2 2^{-P}$; this means that the sign of a_2 is known and that we are not at risk of dividing by 0. Then we have

$$\begin{aligned} \left|\frac{a_1}{a_2} - \frac{\tilde{a_1}}{\tilde{a_2}}\right| &\leq |\frac{a_1\tilde{a_2} - a_2\tilde{a_1}}{a_2\tilde{a_2}}| \\ &\leq |\frac{(a_1 - \tilde{a_1})(a_2 + \tilde{a_2}) - (a_1a_2 - \tilde{a_1}\tilde{a_2})}{a_2\tilde{a_2}}| \\ &\leq \frac{a_2 + \tilde{a_2}}{a_2\tilde{a_2}}k_1 2^{-P} + \frac{k_1|a_2| + k_2|a_1| + 1}{a_2\tilde{a_2}}2^{-P} \\ &\leq \left(\frac{2a_2 + \tilde{a_2}}{a_2\tilde{a_2}}k_1 + \frac{k_2|a_1| + 1}{a_2\tilde{a_2}}\right)2^{-P} \\ &\leq \left(\frac{3a_2 + k_2 2^{-P}}{a_2\tilde{a_2}}k_1 + \frac{k_2|a_1| + 1}{a_2\tilde{a_2}}\right)2^{-P} \\ &\leq \left(\frac{3k_1}{a_2 - k_2 2^{-P}} + \frac{k_2(|a_1| + k_1 2^{-P}) + 1}{a_2(a_2 - k_2 2^{-P})}\right)2^{-P} \end{aligned}$$

To further simplify, we assume that $a_2 \geq 2k_2 2^{-P}$; otherwise, we might end up in a case where not even the high bit of $\tilde{a_2}$ is correct (for instance if $a_2 = 2k_2 2^{-P}$ and $\tilde{a_2} = k_2 2^{-P}$). This means that $a_2 - k_2 2^{-P} \geq a_2/2$ and helps with denominators:

$$\left|\frac{a_1}{a_2} - \frac{\tilde{a_1}}{\tilde{a_2}}\right| \leq \left(\frac{6k_1}{a_2} + \frac{2k_2(|a_1| + k_1 2^{-P}) + 2}{a_2^2}\right)2^{-P}$$

Finally, we assume that $|a_1| \geq k_1 2^{-P}$ to further simplify:

$$\left|\frac{a_1}{a_2} - \frac{\tilde{a_1}}{\tilde{a_2}}\right| \leq \left(\frac{6k_1}{a_2} + \frac{4k_2|a_1| + 2}{a_2^2}\right)2^{-P}$$

Complex division

We write $\frac{z_1}{z_2} = \frac{z_1\overline{z_2}}{|z_2|^2}$ and then chain the results. To simplify, we bound $|x_i|, |y_i|$ by the norm of the number itself right away. Hence:

$$\begin{aligned} ||z_2|^2 - |\tilde{z_2}|^2| &\leq (4 + 8k_2|z_2|)2^{-P} \\ |\operatorname{Re}(z_1\overline{z_2}) - \operatorname{Re}(\tilde{z_1}\tilde{\overline{z_2}})| &\leq (2 + 2k_1|z_2| + 2k_2|z_1|)2^{-P} \\ |\operatorname{Im}(z_1\overline{z_2}) - \operatorname{Im}(\tilde{z_1}\tilde{\overline{z_2}})| &\leq (2 + 2k_1|z_2| + 2k_2|z_1|)2^{-P} \end{aligned}$$

and we use the previous part:

$$\begin{aligned} |\operatorname{Re}\left(\frac{z_1}{z_2}\right) - \operatorname{Re}\left(\frac{\tilde{z_1}}{\tilde{z_2}}\right)| &\leq (\frac{6(2 + 2k_1|z_2| + 2k_2|z_1|)}{|z_2|^2} \\ &\quad +2(4 + 8k_2|z_2|) \times \frac{(|\operatorname{Re}(\tilde{z_1}\tilde{\overline{z_2}})| + (2 + 2k_1|z_2| + 2k_2|z_1|)2^{-P})}{|z_2|^4} + \frac{2}{|z_2|^4})2^{-P} \\ &\leq (\frac{6(2 + 2k_1|z_2| + 2k_2|z_1|)}{|z_2|^2} \\ &\quad +2(4 + 8k_2|z_2|) \times \frac{(|\tilde{x_1}\tilde{x_2}| + |\tilde{y_1}\tilde{y_2}|) + (2 + 2k_1|z_2| + 2k_2|z_1|)2^{-P}}{|z_2|^4} + \frac{2}{|z_2|^4})2^{-P} \\ &\leq (\frac{6(2 + 2k_1|z_2| + 2k_2|z_1|)}{|z_2|^2} \\ &\quad +2(4 + 8k_2|z_2|) \times \frac{2|z_1||z_2| + 2(1 + 2k_1|z_2| + 2k_2|z_1|)2^{-P} + 2k_1k_2 2^{-2P}}{|z_2|^4} \\ &\quad + \frac{2}{|z_2|^4})2^{-P} \\ &\qquad \text{(using } |\tilde{x_1}| \leq |x_1| + k_1 2^{-P} \leq |z_1| + k_1 2^{-P}) \end{aligned}$$

$$\begin{aligned} |\operatorname{Im}\left(\frac{z_1}{z_2}\right) - \operatorname{Im}\left(\frac{\tilde{z_1}}{\tilde{z_2}}\right)| &\leq (\frac{6(2 + 2k_1|z_2| + 2k_2|z_1|)}{|z_2|^2} \\ &\quad +2(4 + 8k_2|z_2|) \times \frac{|\operatorname{Im}(\tilde{z_1}\tilde{\overline{z_2}})| + (2 + 2k_1|z_2| + 2k_2|z_1|)2^{-P}}{|z_2|^4} + \frac{2}{|z_2|^4})2^{-P} \\ &\leq (\frac{6(2 + 2k_1|z_2| + 2k_2|z_1|)}{|z_2|^2} \\ &\quad +2(4 + 8k_2|z_2|) \times \frac{(|\tilde{x_1}\tilde{y_2}| + |\tilde{x_2}\tilde{y_1}|) + (2 + 2k_1|z_2| + 2k_2|z_1|)2^{-P}}{|z_2|^4} + \frac{2}{|z_2|^4})2^{-P} \\ &\leq (\frac{6(2 + 2k_1|z_2| + 2k_2|z_1|)}{|z_2|^2} \\ &\quad +2(4 + 8k_2|z_2|) \times \frac{2|z_1||z_2| + 2(1 + 2k_1|z_2| + 2k_2|z_1|)2^{-P} + 2k_1k_2 2^{-2P}}{|z_2|^4} \\ &\quad + \frac{2}{|z_2|^4})2^{-P} \end{aligned}$$

Now since $k_j \leq 2^{P/2}$ and $|z_j| \leq 2^{P/2-3}$,

$$\begin{aligned} 2(1 + 2k_1|z_2| + 2k_2|z_1|)2^{-P} + 2k_1k_2 2^{-2P} &\leq 1 \\ 2(1 + 2k_1|z_2| + 2k_2|z_1|)2^{-P} + 2k_1k_2 2^{-2P} &\leq 1 \end{aligned}$$

We can then simplify:

$$\begin{aligned} |\operatorname{Re}\left(\frac{z_1}{z_2}\right) - \operatorname{Re}\left(\frac{\tilde{z_1}}{\tilde{z_2}}\right)| &\leq (\frac{6(2 + 2k_1|z_2| + 2k_2|z_1|)}{|z_2|^2} + \frac{2(4 + 8k_2|z_2|)(2|z_1||z_2| + 1) + 2}{|z_2|^4})2^{-P} \\ |\operatorname{Im}\left(\frac{z_1}{z_2}\right) - \operatorname{Im}\left(\frac{\tilde{z_1}}{\tilde{z_2}}\right)| &\leq (\frac{6(2 + 2k_1|z_2| + 2k_2|z_1|)}{|z_2|^2} + \frac{2(4 + 8k_2|z_2|)(2|z_1||z_2| + 1) + 2}{|z_2|^4})2^{-P} \end{aligned}$$

Square root

We write

$$\begin{aligned} |\sqrt{z_1} - \sqrt{\tilde{z_1}}| &= \frac{|z_1 - \tilde{z_1}|}{|\sqrt{z_1} + \sqrt{\tilde{z_1}}|} \\ &\leq \frac{k_1}{|\sqrt{z_1} + \sqrt{\tilde{z_1}}|} 2^{-P} \end{aligned}$$

If we suppose that z_1 and $\tilde{z_1}$ are in the same quadrant, which is true if we suppose $|x_1| > k_1 2^{-P}$ and $|y_1| > k_1 2^{-P}$, then $\sqrt{z_1}$ and $\sqrt{\tilde{z_1}}$ are in the same quadrant (since the angle is just divided by 2). This means that $|\sqrt{z_1} + \sqrt{\tilde{z_1}}| \geq |\sqrt{z_1}|$. Hence

$$|\sqrt{z_1} - \sqrt{\tilde{z_1}}| \leq \frac{k_1}{\sqrt{|z_1|}} 2^{-P}$$

Exponential

Starting with real numbers: we have $|e^x - e^{\tilde{x}}| \leq e^t |x - \tilde{x}|$ with t in the interval bounded by (but not containing) x and $\tilde{x}$, using Taylor-Lagrange with order 1 (or Rolle's theorem). Hence

$$\begin{aligned} |e^x - e^{\tilde{x}}| &\leq e^t |x - \tilde{x}| \\ &\leq \max(e^x, e^{\tilde{x}}) k_x 2^{-P} \\ &\leq e^x e^{k_x 2^{-P}} k_x 2^{-P} \end{aligned}$$

Since $k_x 2^{-P} \leq 2^{-P/2} \leq 1/2$, we have

$$e^{k_x 2^{-P}} \leq 1 + k_x 2^{-P} + \frac{(k_x 2^{-P})^2}{2} \frac{1}{1 - k_x 2^{-P}} \leq 1 + k_x 2^{-P} + (k_x 2^{-P})^2 \leq 1 + 2(k_x 2^{-P})$$

Hence

$$\begin{aligned} |e^x - e^{\tilde{x}}| &\leq e^x (1 + 2k_x 2^{-P}) k_x 2^{-P} \\ &\leq e^x (k_x + 2) 2^{-P} \end{aligned}$$

Now for complex numbers:

$$\begin{aligned} |e^{x+iy} - e^{\tilde{x}+i\tilde{y}}| &\leq |e^x - e^{\tilde{x}+i(\tilde{y}-y)}| \\ &\leq \sqrt{(e^x - e^{\tilde{x}} \cos(\tilde{y} - y))^2 + e^{2\tilde{x}} \sin^2(\tilde{y} - y)} \end{aligned}$$

Since for positive numbers $a + b \leq a + b + 2\sqrt{a}\sqrt{b}$, $\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}$ and

$$\begin{aligned} |e^{x+iy} - e^{\tilde{x}+i\tilde{y}}| &\leq |e^x - e^{\tilde{x}} \cos(\tilde{y} - y)| + e^{\tilde{x}} |\sin(\tilde{y} - y)| \\ &\leq |e^x - e^{\tilde{x}}| + e^{\tilde{x}} (|1 - \cos(\tilde{y} - y)| + |\sin(\tilde{y} - y)|) \\ &\leq e^x (k_x + 2) 2^{-P} + e^x (1 + 2k_x 2^{-P}) (|1 - \cos(\tilde{y} - y)| + |\sin(\tilde{y} - y)|) \end{aligned}$$

For $x > 0$ we have $\sin(x) \leq x$ and $|1 - \cos(x)| \leq \frac{x^2}{2}$ (since $\cos x = 1 - 2\sin^2(x/2)$ or the theorem for alternate series), hence

$$\begin{aligned} |e^{x+iy} - e^{\tilde{x}+i\tilde{y}}| &\leq e^x (k_x + 2) 2^{-P} + e^x (1 + 2k_x 2^{-P}) ((k_x 2^{-P})^2/4 + k_x 2^{-P}) \\ &\leq (e^x (k_x + 2)) 2^{-P} + e^x (1 + 2k_x 2^{-P}) (1/4 + k_x) 2^{-P} \end{aligned}$$

because as always we suppose $k_x 2^{-P} \leq 2^{-P/2}$; hence

$$|e^z - e^{\tilde{z}}| \leq e^x (7/2k_x + 4.25)2^{-P}$$

Résumé de la thèse en français

Cette thèse s'intéresse à l'évaluation rapide de fonctions complexes en précision arbitraire, et plus précisément de fonctions liées aux courbes elliptiques et hyperelliptiques définies sur les nombres complexes. Nos résultats, en particulier ceux sur l'évaluation de la fonction θ en temps quasi-linéaire en la précision voulue, ont une portée plus générale et peuvent être réutilisés dans d'autres contextes.

Nous décrivons dans les deux premières parties les fonctions pour lesquelles nous donnons des algorithmes rapides dans cette thèse ; nous montrons également une méthode utilisée par [Dup06] pour calculer certaines valeurs de la fonction theta en temps quasi-linéaire. Cette méthode est celle que nous généralisons pour obtenir nos algorithmes rapides : nous décrivons dans un premier temps une méthode pour calculer $\theta(z, \tau)$ en genre 1 en temps quasi-linéaire, puis généralisons cette méthode une nouvelle fois pour donner un algorithme calculant $\theta(z, \tau)$ en genre 2. De plus, il semblerait que cette méthode pourrait se généraliser à des genres plus grands, mais quelques problèmes surviennent alors et nous n'avons pas trouvé de façon satisfaisante de les règler. L'application principale que nous faisons de nos résultats est le calcul rapide de l'application d'Abel-Jacobi et de son inverse en temps quasi-linéaire. Ce résultat nous permet de plus de décrire un nouvel algorithme de calcul d'isogénies de noyau donné ; l'algorithme sur $\mathbb{C}$ utilise directement l'évaluation rapide de l'application d'Abel-Jacobi, et nous montrons comment utiliser cet algorithme pour le calcul d'isogénies de noyau donné sur un corps de nombre, puis sur un corps fini, en utilisant un procédé de relèvement de la courbe.

1 Courbes elliptiques et application d'Abel-Jacobi

Les courbes elliptiques, ainsi que les courbes hyperelliptiques, leur généralisation en genre supérieur, sont étudiées depuis des siècles par les mathématiciens ; c'est à la fin des années 1980 que l'idée de les utiliser en cryptographie a fait surface, et avec elle la question de concevoir des algorithmes efficaces pour calculer des objets reliés à ces courbes. Nous ne traiterons ici que le cas des courbes elliptiques, mais beaucoup de propriétés, ainsi que certains algorithmes, se généralisent au genre supérieur.

Une courbe elliptique sur un corps K peut être définie comme l'ensemble des points vérifiant l'équation dite "forme courte de Weierstrass" suivante:

$$y^2 = x^3 + ax + b$$

avec $a, b \in K$, auquel on adjoint un point dit "à l'infini". On peut définir une loi de groupe commutatif sur l'ensemble des points d'une courbe elliptique ; le procédé est bien connu, puisque Fermat l'utilisait déjà, et a une interprétation géométrique simple sur les nombres réels. Ainsi, on munit l'ensemble $E(K)$ des points d'une courbe d'une loi de groupe calculable complètement algébriquement.

La sécurité d'un système de chiffrement basé sur les courbes elliptiques repose sur le problème difficile dit du *logarithme discret sur les courbes elliptiques*:

$$\text{Étant donnés } P \in E(\mathbb{F}_p) \text{ et } Q = [n]P, \text{ trouver } n.$$

Ce problème est l'instanciation du problème du logarithme discret dans le groupe $E(\mathbb{F}_p)$. À l'heure actuelle, aucun algorithme de complexité meilleure que l'algorithme rho de Pollard, un algorithme générique en $O(\sqrt{r})$ opérations où r est le cardinal du groupe, n'est connu ; le problème du logarithme discret sur les courbes elliptiques offre ainsi une sécurité relativement forte, puisque la taille des clés nécessaire à atteindre un certain niveau de sécurité est grosso modo égale à la racine cubique d'un clé RSA de sécurité équivalente. Cette petite taille de clés permet de compenser le fait que l'arithmétique est plus complexe que pour le système RSA ; les courbes elliptiques sont ainsi standardisées et utilisées dans un nombre croissant d'applications.

Une *isogénie* est un morphisme d'une courbe elliptique à une autre, c'est à dire une application telle que

$$\phi(P+Q) = \phi(P) + \phi(Q), \qquad \text{pour tous } P, Q.$$

Ainsi, le problème du logarithme discret est transporté d'une courbe à une autre par une isogénie. Il est donc possible, en théorie, de ramener une instance de ce problème à une autre où il est plus simple à résoudre. On ne connaît actuellement pas d'algorithmes pour exploiter ceci, car le calcul d'isogénies est un problème complexe et les algorithmes pour le résoudre n'ont pour l'instant pas une complexité satisfaisante (de sorte que certains cryptosystèmes basés sur le calcul d'isogénies ont été proposés). Cependant, en genre 3, un article [Smi09] a montré que le problème du logarithme discret sur une courbe hyperelliptique pouvait parfois se réduire en un problème sur une courbe non-hyperelliptique, qui est un problème résoluble plus facilement ; il s'agit ainsi d'une direction de recherche intéressante.

Une courbe elliptique complexe possède une autre représentation, différente de sa représentation algébrique (par une équation courte de Weierstrass). En effet, on a le théorème suivant, dit d'Abel-Jacobi :

Theorem A.1.1. *Soit $E/\mathbb{C}$ une courbe elliptique complexe définie par sa forme de Weierstrass. Alors il existe ω_1, ω_2 (les* périodes *de la courbe elliptique) tels que $E/\mathbb{C} \simeq \mathbb{C}/(\mathbb{Z}\omega_1 + \mathbb{Z}\omega_2)$. De plus, l'isomorphisme est donné par l'*application d'Abel-Jacobi*, définie par*

$$E/\mathbb{C} \to \mathbb{C}/(\mathbb{Z}\omega_1 + \mathbb{Z}\omega_2)$$
$$P \to \int_{\mathcal{O}}^{P} \frac{dx}{\sqrt{x^3 + ax + b}}$$

On parle alors de *représentation analytique* d'une courbe elliptique, ou du *tore complexe* associé à une courbe, pour parler de l'ensemble $\mathbb{C}/(\mathbb{Z}\omega_1 + \mathbb{Z}\omega_2)$. Le calcul de cette application est possible par des algorithmes d'évaluation d'intégrales elliptiques ; nous renvoyons à la section ci-dessous sur l'application d'Abel-Jacobi. L'intérêt de cette application dans le contexte des isogénies est qu'une isogénie entre tores complexes $\mathbb{C}/\Lambda_1 \to \mathbb{C}/\Lambda_2$ est toujours de la forme $z \mapsto \alpha z$ pour un certain α : l'évaluation de l'isogénie est ainsi très aisée. Nous détaillons dans une section ci-dessous un algorithme de calcul d'isogénies qui tire parti de cette propriété.

Afin de calculer l'application d'Abel-Jacobi, ainsi que son inverse, de façon asymptotiquement rapide, nous avons étudié une fonction liée à cette application, la fonction θ de Jacobi. Nous détaillons le résultat le plus important de ce manuscrit, qui est un algorithme pour l'évaluation de θ en complexité quasi-linéaire en la précision demandée ; notre méthode s'inspire d'une méthode similaire pour les theta-constantes, qui sont des valeurs spéciales de cette fonction, étudiée dans [Dup06].

2 Fonction theta et calcul rapide de theta-constantes

Fonction theta

En genre 1, la fonction θ est définie par

$$\begin{aligned}\theta \ : \ & \mathbb{C} \times \mathcal{H} \to \mathbb{C} \\ & (z,\tau) \mapsto \sum_{n\in\mathbb{Z}} e^{i\pi n^2 \tau} e^{2i\pi n\tau}.\end{aligned}$$

avec $\mathcal{H}$ le demi-plan supérieur. On définit plusieurs fonctions, les *fonctions theta avec caractéristique*:

$$\begin{aligned}\theta_0(z,\tau) &= \theta(z,\tau) & \theta_2(z,\tau) &= e^{i\pi\tau/4+i\pi z}\theta\left(z+\frac{\tau}{2},\tau\right) \\ \theta_1(z,\tau) &= \theta\left(z+\frac{1}{2},\tau\right) & \theta_3(z,\tau) &= e^{i\pi\tau/4+i\pi(z+1/2)}\theta\left(z+\frac{1+\tau}{2},\tau\right)\end{aligned}$$

Les fonctions θ_0, θ_1 sont parfois appelées *fonctions thêta fondamentales*. Les *theta-constantes* sont simplement les valeurs en $z = 0$ de ces fonctions ; notons que $\theta_3(0,\tau) = 0$.

Le groupe $\mathrm{SL}_2(\mathbb{Z})$ agit sur les valeurs de θ pour donner la formule suivante:

$$\theta\left(\frac{z}{c\tau+d}, \frac{a\tau+b}{c\tau+d}\right) = \zeta\sqrt{c\tau+d}e^{i\pi\frac{z^2}{c\tau+d}}\theta_i(z,\tau)$$

pour ζ une racine huitième de l'unité, et pour un certain i. Cette propriété permet d'opérer une réduction d'argument : si l'on dénote $\mathcal{F}$ le domaine fondamental de l'action $\tau \mapsto \frac{a\tau+b}{c\tau+d}$ de $\mathrm{SL}_2(\mathbb{Z})$ sur le demi-plan supérieur, cette formule permet de déduire la valeur finale de la valeur de $\theta(z,\tau)$ avec $\tau \in \mathcal{F}$. De plus, une propriété de la fonction thêta permet d'effectuer également une réduction d'argument en z:

$$\theta(z+a\tau+b,\tau) = e^{-i\pi a^2\tau - 2\pi i a z}\theta(z,\tau).$$

Ainsi, on peut supposer que $\mathrm{Im}(z) \leq \frac{\mathrm{Im}(\tau)}{2}$ et $|\mathrm{Re}(\tau)| \leq \frac{1}{2}$. Cependant, notons que déduire les valeurs finales de $\theta(z',\tau')$ avec z',τ' réduits nécessite de calculer des facteurs exponentiels, dont la taille peut être grande ; ainsi, la complexité de la réduction d'argument dépend fortement des arguments, alors que nous verrons plus tard que l'algorithme calculant $\theta(z',\tau')$ a une complexité indépendante des arguments.

La généralisation de cette fonction au genre supérieur n'est pas très compliquée ; pour la fonction θ, la définition se fait au moyen d'une formule similaire, mais avec $z \in \mathbb{C}^g$ et τ une matrice $g \times g$ de partie imaginaire définie positive. On définit alors 2^g fonctions thêta fondamentales, et 2^{2g} fonctions theta avec caractéristiques au total ; toutes les propriétés que nous avons énoncé dans cette section se généralisent également au genre supérieur. La seule exception concerne la généralisation du domaine fondamental au genre supérieur, qui n'est pas aussi simple qu'elle n'y paraît, car le domaine que l'on obtient est caractérisé par des inéquations qui n'ont pas été déterminées explicitement ; cependant, il est possible de considérer deux réductions moins strictes, pour lesquelles un algorithme explicite existe, et qui semblent quand même avoir des propriétés désirables pour la réduction d'arguments pour θ.

Moyenne arithmético-géométrique

Parmi les nombreuses formules vérifiées par les valeurs de la fonction thêta, on peut citer les *formules de τ-duplication* des thêta-constantes:

$$\theta_0^2(0,2\tau) = \frac{\theta_0^2(0,\tau) + \theta_1^2(0,\tau)}{2}, \qquad \theta_1^2(0,2\tau) = \theta_0(0,\tau)\theta_1(0,\tau)$$

Ces formules correspondent à celles que l'on trouve dans la définition de la *moyenne arithmético-géométrique*:

Definition A.2.1. Soient deux nombres complexes a et b. On pose $a_0 = a, b_0 = b$, et pour tout $n \geq 0$

$$a_{n+1} = \frac{a_n + b_n}{2}, \qquad b_{n+1} = \sqrt{a_n b_n}$$

où la racine carrée est choisie de telle sorte que

$$\mathrm{Re}\left(\frac{b_{n+1}}{a_{n+1}}\right) > 0 \quad \text{ou } \mathrm{Re}\left(\frac{b_{n+1}}{a_{n+1}}\right) = 0 \text{ et } \mathrm{Im}\left(\frac{b_{n+1}}{a_{n+1}}\right) > 0$$

ce que l'on appelle un *bon choix de racines*. La suite ainsi définie, où tous les choix de racines sont bons, converge quadratiquement ; sa limite est appelée la *moyenne arithmético-géométrique de a et b*, notée $\mathrm{AGM}(a,b)$.

Une convergence quadratique signifie que le nombre de chiffres exacts à une étape est (quasiment) le double du nombre de chiffres exacts à l'étape précédente ; ainsi, une approximation de la limite avec P chiffres exacts peut être obtenue en calculant $O(\log P)$ termes de la suite. Dans le cas de l'AGM, cela signifie que $\mathrm{AGM}(a,b)$ peut être calculée à précision P en $O(\mathcal{M}(P)\log P)$ opérations, où $\mathcal{M}(P)$ représente le temps nécessaire pour multiplier deux nombres de taille P bits.

Pour relier formellement les thêta-constantes à la moyenne arithmético-géométrique, il convient de déterminer si la condition que les choix de signes soient bons est remplie pour certaines valeurs de τ :

Proposition A.2.2 ([Cox84, Lemma 2.8])**.** *Soit le domaine*

$$\mathcal{F}_{k'} = \{\tau \in \mathcal{H} \mid |\mathrm{Re}(\tau)| \leq 1, |\tau \pm 1/4| > 1/4, |\tau \pm 3/4| > 1/4\}.$$

Pour $\tau \in \mathcal{F}_{k'}$, le choix de signe est bon ; en particulier, on a pour tout $\tau \in \mathcal{F}_{k'}$

$$\mathrm{AGM}(\theta_0^2(0,\tau), \theta_1^2(0,\tau)) = 1.$$

Calcul rapide de theta-constantes

Nous décrivons maintenant l'algorithme, exposé dans [Dup06, Dup11], qui calcule les theta-constants rapidement à partir de l'AGM. La proposition précédente se réécrit en utilisant l'homogénéité de l'AGM:

$$\mathrm{AGM}\left(1, \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right) = \frac{1}{\theta_0^2(0,\tau)}.$$

On peut ensuite utiliser l'action de $\mathrm{SL}_2(\mathbb{Z})$, qui donne le résultat suivant:

$$\mathrm{AGM}\left(1, \frac{\theta_2^2(0,\tau)}{\theta_0^2(0,\tau)}\right) = \frac{1}{-i\tau\theta_0^2(0,\tau)}.$$

L'équation de Jacobi $\theta_0^4(0,\tau) = \theta_1^4(0,\tau) + \theta_2^4(0,\tau)$ permet de montrer que la fonction

$$f_\tau \ : x \mapsto \frac{\mathrm{AGM}(1,z)}{\mathrm{AGM}(1,\sqrt{1-z^2})} + i\tau$$

vérifie $f_\tau\left(\frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right) = 0$.

L'algorithme consiste alors à appliquer la méthode de Newton à la fonction f_τ, afin de calculer une approximation du quotient des carrés des thêta-constantes fondamentales ; on peut ensuite récupérer chacune d'entre elles en appliquant l'AGM, et la troisième en utilisant l'équation de Jacobi. Il importe d'appliquer la méthode de Newton de façon efficace, afin de préserver une complexité quasi-optimale : on part ainsi d'une approximation du résultat à basse précision, puis on applique chaque étape de la méthode de Newton en doublant la précision à laquelle on travaille. En effet, la méthode de Newton étant auto-correctrice, on peut se contenter de travailler à précision réduite et de doubler la précision requise à chaque étape ; l'analyse asymptotique permet de démontrer que la complexité totale est la même que la dernière étape, qui travaille à pleine précision. Ici, l'on obtient une complexité de $O(\mathcal{M}(P)\log P)$ opérations.

Enfin, notons que l'on peut aussi utiliser cet algorithme pour obtenir un algorithme qui calcule $\theta(0,\tau)$, pour $\tau \in \mathcal{F}$, avec une complexité bornée par $c\mathcal{M}(P)\log P$, avec c une constante indépendante de τ – la complexité est ainsi uniforme en τ. L'idée est d'utiliser également l'algorithme naïf, qui approxime θ en calculant une somme partielle avec assez de termes : la complexité de cet algorithme est en $O\left(\mathcal{M}(P)\sqrt{\frac{P}{\mathrm{Im}(\tau)}}\right)$, i.e. elle s'améliore lorsque $\mathrm{Im}(\tau)$ grossit, alors que la complexité de l'algorithme ci-dessus devient de pire en pire (il faut calculer le nombre de bits perdus par les erreurs d'approximation pour s'en convaincre). On peut ainsi utiliser l'algorithme naïf pour des valeurs de $\mathrm{Im}(\tau)$ relativement grosses par rapport à la précision voulue, et l'algorithme rapide pour les valeurs de $\mathrm{Im}(\tau)$ plus petites ; on peut montrer que cet algorithme a la bonne complexité. Cette stratégie d'uniformisation de la complexité sur le domaine fondamental peut aussi être appliquée dans nos algorithmes, comme nous allons le voir.

3 Calcul de la fonction thêta de Jacobi

Nous allons maintenant décrire comment nous avons généralisé cette méthode au calcul de $\theta(z,\tau)$, pour tout z et τ réduits, en temps quasi-linéaire. Les résultats de cette section sont tirés d'un article accepté pour publication dans le journal *Mathematics of Computation*.

L'ingrédient nécessaire est la généralisation des formules de τ-duplication aux fonctions thêta:

$$\theta_0^2(z,2\tau) = \frac{\theta_0(z,\tau)\theta_0(0,\tau) + \theta_1(z,\tau)\theta_1(0,\tau)}{2}, \qquad \theta_1^2(z,2\tau) = \frac{\theta_0(z,\tau)\theta_1(0,\tau) + \theta_1(z,\tau)\theta_0(0,\tau)}{2}.$$

On peut ainsi définir une fonction de 4 variables

$$F(x,y,z,t) = \left(\frac{\sqrt{x}\sqrt{z} + \sqrt{y}\sqrt{t}}{2}, \frac{\sqrt{x}\sqrt{t} + \sqrt{y}\sqrt{z}}{2}, \frac{z+t}{2}, \sqrt{z}\sqrt{t}\right)$$

avec la condition que le choix des racines carrées vérifie

$$\mathrm{Re}(\sqrt{x}) \geq 0, \quad \mathrm{Re}(\sqrt{z}) \geq 0, \qquad\qquad \mathrm{Re}\left(\frac{\sqrt{y}}{\sqrt{x}}\right) > 0, \mathrm{Re}\left(\frac{\sqrt{t}}{\sqrt{z}}\right) > 0.$$

Nous sommes parvenu à démontrer le résultat suivant:

Proposition A.3.1. *Pour z, τ vérifiant $\mathrm{Im}(z) \leq \frac{\mathrm{Im}(\tau)}{4}$ et $\mathrm{Im}(\tau) > 0.345$, on a $\mathrm{Re}(\theta_0(z,\tau)) > 0$ et $\mathrm{Re}\left(\frac{\theta_1(z,\tau)}{\theta_0(z,\tau)}\right) > 0$; ainsi, la suite des itérés de $(\theta_0(z,\tau), \theta_1(z,\tau), \theta_0(0,\tau), \theta_1(0,\tau))$ par la fonction F correspond aux valeurs où τ a été multiplié par 2, et la suite converge vers $(1,1,1,1)$.*

Notons que la condition sur $\mathrm{Im}(z)$ est un peu plus stricte que celle obtenue simplement par réduction du premier argument ; ainsi, l'application de ce théorème nécessite d'utiliser les *formules de z-duplication*, qui donnent $\theta(2z,\tau)$ en fonction de $\theta_i(z,\tau)$.

Cependant, la suite définie dans la proposition précédente ne généralise pas exactement l'AGM, car elle ne converge pas nécessairement quadratiquement ; on peut le voir en prenant par exemple $x = y, z = t$. La difficulté est en réalité contournable dans notre cas : ce que nous recherchons en réalité est la définition d'une fonction $\mathfrak{F}$ telle que l'on ait par exemple (tout comme pour l'AGM)

$$\mathfrak{F}\left(\frac{\theta_1^2(z,\tau)}{\theta_0^2(z,\tau)}, \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right) = \left(\frac{1}{\theta_0^2(z,\tau)}, \frac{1}{\theta_0^2(0,\tau)}\right).$$

De plus, il faut que la fonction $\mathfrak{F}$ soit calculable en temps quasi-linéaire. Dans le cas de l'AGM, la proprété découlait directement de la propriété d'homogénéité de l'AGM – c'est à dire $\mathrm{AGM}(\lambda a, \lambda b) = \lambda\,\mathrm{AGM}(a,b)$. Nous avons ainsi eu l'idée d'étudier la limite de la suite des itérés de F démarrant avec $(\lambda x, \lambda y, \mu z, \mu t)$, avec l'idée que nous pourrions ensuite prendre $\lambda = \frac{1}{\theta_0^2(z,\tau)}, \mu = \frac{1}{\theta_0^2(0,\tau)}$. Nous sommes arrivés à prouver la proposition suivante :

Proposition A.3.2. *Si (x_n, y_n, z_n, t_n) est la suite des itérés de F démarrant avec (x,y,z,t), et (x'_n, y'_n, z'_n, t'_n) celle démarrant avec $(\lambda x, \lambda y, \mu z, \mu t)$, on a*

$$\mu = \lim_{n\to\infty} \frac{z'_n}{z_n}, \qquad \lambda = \lim_{n\to\infty} \frac{\left(\frac{x'_n}{z'_n}\right)^{2^n} z'_n}{\left(\frac{x_n}{z_n}\right)^{2^n} z_n}.$$

De plus, les suites du membre droit de ces égalités convergent quadratiquement, i.e. on peut calculer μ, λ avec précision P en $O(\mathcal{M}(P)\log P)$.

La preuve de cette proposition est technique. Notons que, si x, y sont les fonctions thêta en z et en τ, et z, t les thêta-constantes, $\lim x_n = \lim z_n = 1$ et les membres droits se simplifient. On peut ainsi définir $\mathfrak{F}$ comme la fonction qui calcule la limite des suites qui apparaissent dans cette proposition.

Tout comme pour les thêta-constantes, on utilise l'action de $\mathrm{SL}_2(\mathbb{Z})$ pour faire ressortir les paramètres : on a ainsi que

$$\theta_2^2\left(\frac{z}{\tau}, \frac{-1}{\tau}\right) = -i\tau e^{2i\pi z^2/\tau}\theta_1^2(z,\tau), \quad \theta_0^2\left(\frac{z}{\tau}, \frac{-1}{\tau}\right) = -i\tau e^{2i\pi z^2/\tau}\theta_0^2(z,\tau).$$

Ainsi

$$\mathfrak{F}\left(\frac{\theta_2^2(z,\tau)}{\theta_0^2(z,\tau)}, \frac{\theta_2^2(0,\tau)}{\theta_0^2(0,\tau)}\right) = \left(\frac{e^{-2i\pi z^2/\tau}}{-i\tau\theta_0^2(z,\tau)}, \frac{1}{-i\tau\theta_0^2(0,\tau)}\right)$$

pour peu que les choix de signes soient bons pour $\theta\left(\frac{z}{\tau}, \frac{-1}{\tau}\right)$. Afin de s'en assurer, on impose les conditions $|\mathrm{Re}(z)| \leq \frac{1}{8}, \mathrm{Im}(z) \leq \frac{\mathrm{Im}(\tau)}{2}, 0.345 \leq \mathrm{Im}(\tau) < 1$, ce qui assure que les choix de signes sont bons pour θ à la fois en (z,τ) et en $\left(\frac{z}{\tau}, \frac{-1}{\tau}\right)$. Ceci contraint z, τ à des valeurs particulières, à l'intérieur d'un compact ; cependant, on peut toujours se ramener à ce compact à

partir des conditions obtenues après réduction des deux arguments, en utilisant des formules de z-duplication et de τ-duplication. Il est ensuite aisé de construire une fonction $f_{z,\tau}$ qui s'annule en $\frac{\theta_1^2(z,\tau)}{\theta_0^2(z,\tau)}$; puis, on utilise la méthode de Newton pour calculer ce quotient, ce qui (après une ultime application de $\mathfrak{F}$) donne $\theta(z,\tau)$. Le coût de cet algorithme est $O(\mathcal{M}(P)\log P)$, mais la complexité dépend a priori de z et de τ (en particulier, elle dépend du nombre de formules de τ- et de z-duplications que l'on doit appliquer) ; nous sommes parvenu à donner une version de l'algorithme qui a une complexité uniforme en z et τ sur tout le domaine fondamental, et pour lequel les pertes de précision ne sont jamais plus grandes que P chiffres (ainsi, travailler à précision $2P$ suffit pour obtenir le résultat exact).

Nous avons implanté l'algorithme ci-dessus en utilisant la bibliothèque MPC [EGTZ12], une bibliothèque C qui permet d'effectuer des calculs sur des nombres complexes de précision arbitraire. Nous avons également implanté une version optimisée de l'algorithme naïf afin d'obtenir une comparaison adéquate ; enfin, nous avons également comparé notre algorithme à la fonction `Theta` de MAGMA. Les temps de calcul sont résumés dans la Table 6.1 ; ils montrent que les deux algorithmes sont toujours plus rapides que MAGMA, et que notre algorithme est plus rapide que l'algorithme naïf pour des précisions supérieures à 260 000 chiffres décimaux (ce qui nécessite environ une minute de calcul).

4 Généralisation de l'algorithme au genre supérieur

Nous décrivons dans cette section les résultats du chapitre 7, qui sont tirés d'un article co-écrit avec Emmanuel Thomé et publié à l'*Algorithmic Number Theory Symposium XII* ; il s'agit de montrer comment l'algorithme décrit dans la section précédente peut se généraliser au genre supérieur.

Notons d'abord que l'algorithme naïf (par sommation partielle de la série) est plus difficile à analyser dans le cas général du genre g. En genre 1, l'algorithme nécessite $O\left(\mathcal{M}(P)\sqrt{\frac{P}{\mathrm{Im}(\tau)}}\right)$ opérations pour le calcul de θ à précision P ; nous avons réalisé une analyse similaire, rendue plus technique par le nombre plus grand de variables, en genre 2, qui donne un algorithme en $O\left(\mathcal{M}(P)\frac{P}{\mathrm{Im}(\tau_{1,1})}\right)$ opérations. Cependant, une telle preuve ne peut se généraliser en genre g très facilement : nous avons obtenu une complexité en $O\left(\mathcal{M}(P)\left(\frac{P}{\lambda}\right)^{g/2}\right)$, où λ est la plus petite valeur propre de $\mathrm{Im}(\tau)$, mais le lien avec $\mathrm{Im}(\tau_{1,1})$, bien que pressenti par une conjecture, n'a pu être formellement établi. Notons cependant qu'une approche tout aussi naturelle est décrite en d'autres termes par [DHB+04] ; notre analyse montre une complexité de $O\left(\mathcal{M}(P)P^{g/2}\right)$, mais nous n'avons pas pu déterminer la dépendance en τ de l'algorithme.

Revenons maintenant au cas du genre 2. Un algorithme similaire à celui existant pour les theta-constantes en genre 1 est décrit dans [Dup06] ; la généralisation de l'AGM qui est utilisée est la *moyenne de Borchardt*, définie par les relations de récurrence suivantes :

$$a_{n+1} = \frac{a_n + b_n + c_n + d_n}{4}, \quad b_{n+1} = \frac{\sqrt{a_n}\sqrt{b_n} + \sqrt{c_n}\sqrt{d_n}}{2},$$
$$c_{n+1} = \frac{\sqrt{a_n}\sqrt{c_n} + \sqrt{b_n}\sqrt{d_n}}{2}, \quad d_{n+1} = \frac{\sqrt{a_n}\sqrt{d_n} + \sqrt{b_n}\sqrt{c_n}}{2}.$$

Là encore, la notion de *bons choix de signes* est cruciale ; elle est définie par $\mathrm{Re}\left(\frac{\sqrt{b_n}}{\sqrt{a_n}}\right) > 0$, ainsi que les mêmes inéquations obtenues en permutant les quatre nombres. On a le théorème suivant :

Proposition A.4.1 ([Dup06, Théorème 7.1])**.** *Une suite de Borchardt où tous*[16] *les bons choix de signes sont bons converge quadratiquement ; la limite* $\mathfrak{B}(a,b,c,d)$ *peut donc être calculée en* $O(\mathcal{M}(P)\log P)$*.*

La moyenne de Borchardt correspond exactement aux formules de τ-duplication des thêta-constantes en genre 2 ; ainsi, on se retrouve à étudier $\mathrm{Re}\left(\frac{\theta_1(0,\tau)}{\theta_0(0,\tau)}\right)$. Nous sommes parvenu à étendre [Dup06, Prop. 6.1] et montrer que les choix de signes étaient bons dès que $\mathrm{Im}(\tau)$ est réduite par la réduction de Minkowksi et $\mathrm{Im}(\tau_{1,1}) > 0.594$. Ainsi, par homogénéité de la moyenne de Borchardt, on obtient pour τ vérifiant ces conditions :

$$\mathfrak{B}\left(1, \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}, \frac{\theta_2^2(0,\tau)}{\theta_0^2(0,\tau)}, \frac{\theta_3^2(0,\tau)}{\theta_0^2(0,\tau)}\right) = \frac{1}{\theta_0^2(0,\tau)}.$$

Il s'agit ensuite, comme en genre 1, d'utiliser l'action de $\mathrm{Sp}_4(\mathbb{Z})$ sur les valeurs de θ pour calculer les coefficients de τ : en considérant trois matrices bien choisies (données dans [Dup06], ou dans Proposition 7.1.3), on obtient un moyen de calculer les 3 coefficients de τ, et d'appliquer la méthode de Newton, pour obtenir un algorithme en temps quasi-linéaire (cf. Section 7.1). Cependant, il faut aussi démontrer que les choix de signes sont bons pour les thêta-constantes que l'on considère en regardant l'action de ces matrices ; malheureusement, il semble très difficile de prouver un tel résultat, qui reste à l'état de conjecture.

La généralisation de cet algorithme s'opère de la même façon qu'en genre 1. (Dans le reste de cette section, nous utilisons la notation $\theta_{i_1,i_2,\ldots,i_n}$ pour signifier $\theta_{i_1},\theta_{i_2},\ldots,\theta_{i_n}$.) On considère les formules de τ-duplication des fonctions thêta en genre 2 :

$$\theta_i^2(z,2\tau) = \sum_{j\in\{0,\ldots,2^g-1\}} \theta_j(z,\tau)\theta_{i\oplus j}(0,\tau)$$

où $\oplus$ dénote le XOR bit à bit. On peut ensuite définir une fonction F de 8 variables telle que

$$F(\theta_{0,1,2,3}^2(z,\tau), \theta_{0,1,2,3}^2(0,\tau)) = \left(\theta_{0,1,2,3}^2(z,2\tau), \theta_{0,1,2,3}^2(0,2\tau)\right).$$

Cette fonction nécessite l'extraction de racines carrées, ce qui nécessite l'étude de $\mathrm{Re}\left(\frac{\theta_{1,2,3}(z,\tau)}{\theta_0(z,\tau)}\right)$. Cependant, nous ne sommes pas arrivés à déterminer une condition suffisante sur τ pour garantir que ces parties réelles soient positives. Nous nous sommes alors tournés vers une autre solution, qui est celle de calculer une approximation à petite précision du quotient afin de déterminer quelle racine carrée correspond au quotient de thêtas voulu – ce que nous appelons un *choix correct* de racines, car c'est celui qui donne la valeur correcte pour la limite de la suite (qui est 1). Nous conjecturons que les *bons choix de racines* et les *choix corrects de racines* correspondent au moins pour $\tau \in \mathcal{F}_2$ et z réduit, ce qui se vérifie expérimentalement.

De même qu'en genre 1, la suite des itérés de F ne converge pas nécessairement quadratiquement ; mais, comme en genre 1, étudier $F(\lambda a_{0,1,2,3}, \mu b_{0,1,2,3})$ fait apparaître des suites très similaires. Nous avons réussi à montrer que ces suites convergent quadratiquement, en généralisant la même preuve qu'en genre 1 ; l'obstacle principal à la généralisation aux genres supérieurs est essentiellement la technicité de la démonstration. Ainsi, nous arrivons à définir une fonction $\mathfrak{F}$, calculable en temps $O(\mathcal{M}(P)\log P)$, telle que

$$\mathfrak{F}\left(\frac{\theta_{1,2,3}^2(z,\tau)}{\theta_0^2(z,\tau)}, \frac{\theta_{1,2,3}^2(0,\tau)}{\theta_0^2(0,\tau)}\right) = \left(\frac{1}{\theta_0^2(z,\tau)}, \frac{1}{\theta_0^2(0,\tau)}\right).$$

[16] En réalité, on peut même tolérer un nombre fini de mauvais choix.

Les trois matrices de $\mathrm{Sp}_4(\mathbb{Z})$ utilisées pour le calcul des coefficients de τ peuvent aussi être utilisées ici, et permettent de calculer les coefficients de z. Cependant, l'application de la méthode de Newton n'est pas aussi directe qu'en genre 1 ; en effet, nous avons ici une fonction de 6 variables, et seulement 5 coefficients (2 pour z et 3 pour τ). Deux solutions sont possibles :

- Utiliser l'équation de la variété de Kummer, une équation qui lie les thêta-fonctions et les thêta-constantes que nous considérons ici, pour en quelque sorte abaisser la dimension de l'espace de départ ;
- Définir une fonction dont la valeur en les 6 quotients considérés ici est un sextuplet dépendant uniquement de z et τ (on prend par exemple les λ et μ obtenus en considérant l'action des trois matrices mentionnées plus haut), en espérant que la matrice jacobienne de la fonction soit inversible.

Nous utilisons la deuxième approche, qui semble fonctionner en pratique : ceci correspond à rajouter une sixième équation qui, bien qu'elle ne dépende que de 5 paramètres calculés par la fonction, donne une jacobienne inversible. Nous n'avons pas réussi à trouver une explication convaincante qui justifierait cette inversibilité, qui est donc conjecturelle. En tout cas, modulo cette conjecture, la méthode de Newton s'applique, et l'on obtient un algorithme en $O(\mathcal{M}(P)\log P)$ pour calculer θ en genre 2.

Nous avons réalisé une implantation de cet algorithme, ainsi qu'une implantation optimisée de l'algorithme naïf ; les deux implantations ont été réalisées en MAGMA et ont été comparées à la fonction `Theta` de ce logiciel. Les résultats sont présentés dans la Table 7.1 ; notre algorithme est plus rapide que l'algorithme naïf pour des précisions supérieures à 3 000 chiffres décimaux, ce qui est encore mieux qu'en genre 1, et est sans doute dû à la complexité de l'algorithme naïf, plus grande qu'en genre 1.

Enfin, nous avons détaillé des pistes pour généraliser l'algorithme ci-dessus en genre g. La clé est une fois encore les formules de τ-duplication pour les fonctions thêta, qui sont valides en genre g ; on peut ainsi s'en servir pour définir une fonction F de 2^{g+1} variables, qui double τ quand on l'applique aux carrés des fonctions thêta et des thêta-constantes en τ. Là encore, les choix de signes doivent être guidés par des approximations à faible précision du résultat, qui donnent les valeurs correctes de θ. L'étude de l'homogénéité de la suite des itérés de F fait apparaître, pour le calcul de λ et μ, les mêmes suites qu'en genre 1 ou 2 ; nous pensons qu'il est possible de prouver qu'elles convergent quadratiquement, même si la technicité de notre preuve ne nous permet pas de la généraliser en genre supérieur. Au final, nous parvenons à définir une fonction $\mathfrak{F}$ telle que

$$\mathfrak{F}\left(\frac{\theta^2_{1,\ldots,2^g-1}(z,\tau)}{\theta^2_0(z,\tau)}, \frac{\theta^2_{1,\ldots,2^g-1}(0,\tau)}{\theta^2_0(0,\tau)}\right) = \left(\frac{1}{\theta^2_0(z,\tau)}, \frac{1}{\theta^2_0(0,\tau)}\right).$$

et qui, modulo une généralisation de notre preuve quant à la convergence de λ et μ, peut être calculée en temps $O(\mathcal{M}(P)\log P)$.

L'application de la méthode de Newton n'est pas aisée ; en effet, nous obtenons une fonction $\mathbb{C}^{2^{g+1}} \to \mathbb{C}^t$ où $t = g + \frac{g(g+1)}{2}$. Il s'agit d'adapter les stratégies que nous avons mis en place en genre 2 ; nous pourrions tenter d'abaisser la dimension de l'espace de départ, ou calculer plus de valeurs de $\mathfrak{F}$ en des points dictés par l'action de matrices de $\mathrm{Sp}_{2g}(\mathbb{Z})$. Il faut de plus assurer que la jacobienne du système est inversible, ce qui semble compliqué ; en réalité, nous ne sommes pas parvenus à déterminer une famille de matrices de $\mathrm{Sp}_{2g}(\mathbb{Z})$ qui consituterait une bonne candidate à cette étape de l'algorithme. Ainsi, le cas du genre g est plus complexe, et nécessite de régler plusieurs problèmes ; cependant, notons qu'il semblerait que notre approche ait de bonnes chances d'aboutir.

5 Calcul de l'application d'Abel-Jacobi

Nous montrons ensuite comment utiliser les algorithmes pour le calcul rapide de θ pour calculer l'application d'Abel-Jacobi ainsi que son inverse ; ceci est expliqué dans le chapitre 8 de ce manuscrit.

Application d'Abel-Jacobi

En genre 1, l'application d'Abel-Jacobi peut se calculer à l'aide de la *transformation de Landen*, un changement de variables sur les intégrales elliptiques complexes qui peut également s'interpréter comme une 2-isogénie entre courbes elliptiques. La méthode générale est exposée dans [BM88] dans le cas réel: répéter ce changement de variables donne une chaîne de 2-isogénies qui tend vers une courbe limite dont l'intégrale elliptique associée est particulièrement simple à calculer. On obtient ainsi un algorithme pour le calcul des intégrales elliptiques complètes (calcul des périodes) et incomplètes (calcul de l'application d'Abel-Jacobi); de plus, la suite ayant des liens très forts avec la moyenne arithmético-géométrique, le temps de calcul est de $O(\mathcal{M}(P)\log P)$ opérations. Cet algorithme est généralisé dans le cas complexe par un article récent [CT13], qui traite notamment du choix de signes dans les itérations.

Nous proposons un autre algorithme, basé sur notre travail sur la fonction thêta et inspiré par un algorithme de [Dup06, Chap. 9], pour calculer le logarithme elliptique d'un point. Le lien entre la fonction θ et la fonction $\wp$ donne l'équation suivante:

$$\frac{\theta_3^2(z,\tau)}{\theta_0^2(z,\tau)} = \frac{\pi^2\theta_1^2(0,\tau)\theta_2^2(0,\tau)}{\frac{\pi^2}{3}\left(\theta_2^4(0,\tau)-\theta_1^4(0,\tau)\right)-\frac{x}{\omega_1}}.$$

Les thêta-constantes peuvent se calculer en temps quasi-optimal, ce qui fait que $\frac{\theta_3^2}{\theta_0^2(z,\tau)}$ peut être calculé à partir du point; on peut ensuite utiliser l'équation de la variété pour calculer $\frac{\theta_1^2(z,\tau)}{\theta_0^2(z,\tau)}$. On peut ensuite utiliser la fonction $\mathfrak{F}$, qui est à la base de notre algorithme rapide pour le calcul de θ ; plus précisément, on utilise le fait que

$$\mathfrak{F}\left(\frac{\theta_1^2(z,\tau)}{\theta_0^2(z,\tau)}, \frac{\theta_1^2(0,\tau)}{\theta_0^2(0,\tau)}\right) = (z,\tau)$$

pour récupérer z, le logarithme elliptique. Ceci donne un deuxième algorithme pour calculer le logarithme elliptique en temps quasi-linéaire ; cependant, nos expériences montrent que cet algorithme est environ 2 fois plus lent que l'algorithme basé sur la transformation de Landen.

Cependant, cet algorithme peut se généraliser au genre supérieur, alors que l'algorithme basé sur la transformation de Landen ne semble pas se généraliser aussi facilement (le cas réel en genre 2 est esquissé dans [BM88], mais le lien avec les suites de Borchardt n'est pas vraiment explicite). Dans un premier temps, il s'agit de calculer les quotients de carrés de fonctions thêta à partir des coordonnées de Mumford du diviseur voulu ; ceci s'effectue en utilisant un algorithme pour calculer les thêta-constantes, ainsi que les formules données dans [Cos11], qui permettent de passer des coordonnées Mumford aux coordonnées thêta et vice-versa. Ensuite, il suffit d'appliquer une généralisation en genre g de la fonction $\mathcal{F}$, qui permet d'extraire les coefficients de z et de τ en considérant certains quotients de fonctions thêta bien précis. Nous formulons la conjecture que les suites considérées pendant ce calcul convergent quadratiquement, ainsi que celle que les thêta-constantes peuvent être calculées en temps quasi-optimal ; si ces deux conjectures sont vérifiées, nous obtenons un algorithme quasi-linéaire pour le calcul de l'application d'Abel-Jacobi.

Inverse de l'application d'Abel-Jacobi

Aucun algorithme quasi-linéaire pour le calcul de l'inverse de l'application d'Abel-Jacobi ne semble avoir été proposé ; nous montrons comment effectuer ce calcul en genre 1 et 2, et comment il pourrait être effectué en genre supérieur.

En genre 1, ce calcul revient à l'évaluation rapide de la fonction $\wp$ de Weierstrass. Le lien entre $\wp$ et θ est donné par la formule classique suivante:

$$\frac{\wp(z,[\omega_1,\omega_2])}{\omega_1^2} = \frac{\pi^2}{3}(\theta_2^4(0,\tau)-\theta_1^4(0,\tau)) - \pi^2\theta_1^2(0,\tau)\theta_2^2(0,\tau)\frac{\theta_0^2(z,\tau)}{\theta_3^2(z,\tau)}$$

A similar formula can be proven for $\wp'$. Ainsi, nos algorithmes pour le calcul rapide de θ donnent un algorithme pour le calcul de $\wp$ en temps quasi-linéaire.

Nous avons également trouvé un deuxième algorithme rapide pour calculer $\wp$, qui se base cette fois-ci sur la transformation de Landen. Celle-ci s'exprime comme une 2-isogénie entre courbes, et le lien entre les périodes des deux courbes est explicite ; ainsi, le changement de variables peut se réécrire comme une relation entre des valeurs de $\wp$ en des périodes différentes. En utilisant les formules de Thomae, qui permettent d'écrire les coefficients de la courbe en fonction des theta-constantes, on obtient la relation

$$\wp(z,[\omega_1,\omega_2]) = \wp(z,[\omega_1,2\omega_2]) + \frac{\left(\frac{\pi}{\omega_1}\right)^4\theta_0(2\tau)^4\theta_2(2\tau)^4}{\wp(z,[\omega_1,2\omega_2]) + \left(\frac{\pi}{\omega_1}\right)^2\frac{\theta_0(2\tau)^4+\theta_2(2\tau)^4}{3}}$$

Appliquer la transformation de Landen plusieurs fois de suite revient à étudier la suite $\wp(z,[\omega_1,2^k\omega_2])$ pour k tendant l'infini ; or on a

$$\lim_{k\to\infty}\wp(z,[\omega_1,2^k\omega_2]) = \left(\frac{\pi}{\omega_1}\right)^2\left(\frac{1}{\sin^2(z\pi/\omega_1)}-\frac{1}{3}\right).$$

Une stratégie est alors de calculer un rang N à partir duquel $\wp(z,[\omega_1,2^N\omega_2])$ est égal à cette limite, à 2^{-P} près ; nous prenons N comme le rang à partir duquel $|\theta_2(0,2^N\tau)| \leq 2^{-P}$, mais nous n'avons pu prouver que ce choix était le bon. Une fois que N a été choisi, on peut appliquer la relation de récurrence N fois afin de récupérer $\wp(z,[\omega_1,\omega_2])$. Puisque $N = O(\log P)$, ceci donne un algorithme en $O(\mathcal{M}(P)\log P)$ opérations.

La comparaison entre ces deux algorithmes montre que les pertes de précision sont à peu près les mêmes ; dans les deux cas, travailler avec P (voire $2P$) bits de garde permet de retrouver le bon résultat. Quant à la vitesse de chacun de ces algorithmes en pratique, une comparaison entre leurs implantations en MAGMA montre que le deuxième algorithme est environ 3 fois plus rapide que celui basé sur le calcul rapide de θ.

Cependant, le premier algorithme peut se généraliser aisément en genre supérieur ; en effet, il existe des formules, données dans [Cos11], qui permettent de relier les coordonnées de Mumford d'un diviseur à des quotients de fonctions thêta. Ainsi, le problème de l'évaluation de l'inverse de l'application d'Abel-Jacobi en genre supérieur se réduit au problème du calcul de θ ; comme nous l'avons indiqué précédemment, des algorithmes en temps quasi-linéaire existent en genre 1 et en genre 2, mais la généralisation en genre g nécessite d'abord de régler quelques problèmes qui surviennent lors de l'utilisation de la méthode de Newton.

6 Calcul d'isogénies de noyau donné

Une isogénie est un morphisme de groupe de courbes elliptiques, i.e. une fonction rationnelle telle que $\phi(P+Q) = \phi(P)+\phi(Q)$. Une isogénie transporte donc le problème du logarithme discret

sur une courbe elliptique vers une autre courbe elliptique ; le risque est alors que ce problème soit plus facile sur une courbe que sur une autre, ce qui aurait des conséquences sur la sécurité d'un cryptosystème basé sur une de ces courbes. On s'intéresse ainsi au problème de calculer des isogénies rapidement.

Nous considérons le problème suivant:

Problème. *Étant donnée une courbe elliptique E et un point P de ℓ-torsion (avec ℓ premier), calculer l'unique courbe E' et l'unique isogénie ϕ telles que* $\operatorname{Ker}\phi$ *est le sous-groupe d'ordre ℓ engendré par P.*

Ce problème est résolu par l'utilisation des *formules de Vélu* [Vél71], qui nécessitent $O(\mathcal{M}(\ell)\log\ell)$ opérations élémentaires sur le corps. Il s'agit du problème sur les isogénies le plus simple; d'autres problèmes, tels que "étant données deux courbes et un entier ℓ premier, calculer une ℓ-isogénie entre les courbes" ou "étant données deux courbes isogènes, calculer une isogénie", peuvent être résolus par des algorithmes de complexité plus importante, qui d'ailleurs font bien souvent appel aux formules de Vélu lors d'une de leurs étapes.

Nous proposons un algorithme différent, se basant sur le théorème suivant:

Théorème ([Sil86, Theorem VI.4.1])**.** *Une isogénie entre deux courbes elliptiques en représentation analytique, i.e. $\mathbb{C}/\Lambda_1$ et $\mathbb{C}/\Lambda_2$, est de la forme $\phi(z) = \alpha z \pmod{\Lambda_2}$ pour un certain $\alpha \in \mathbb{C}$.*

Calculer des isogénies entre tores complexes est ainsi très simple ; l'idée est d'utiliser l'application d'Abel-Jacobi pour étendre l'algorithme aux représentations algébriques de courbes elliptiques. Nous étudions cet algorithme dans la section 9.1 (Algorithme 25). Il consiste en une évaluation de l'isogénie en $O(\ell)$ points, en utilisant l'application d'Abel-Jacobi pour ramener le problème à une évaluation d'isogénie entre tores complexes, puis l'inverse de l'application d'Abel-Jacobi pour trouver la valeur de l'isogénie en le point ; une fois ces images calculées, il suffit d'interpoler pour trouver la fraction rationnelle définissant l'isogénie. La complexité de l'algorithme est de $O(\mathcal{M}(P)\log P\mathcal{M}(\ell)\log\ell)$ opérations sur les bits, ce qui est plus grand d'un facteur $\log P$ que la complexité des formules de Vélu.

L'algorithme peut se généraliser pour résoudre le même problème sur un corps de nombres $K = \mathbb{Q}[X]/(f)$. En effet, si f est de degré n, il existe n plongements de K dans $\mathbb{C}$, qui sont les applications qui évaluent les éléments du corps de nombre (considérés comme des polynômes) en une racine de f. On peut ainsi calculer les n plongements dans $\mathbb{C}$ de la courbe définie sur K, et appliquer l'algorithme pour calculer l'isogénie correspondant à chacun de ses plongements ; il est ensuite nécessaire d'interpoler les n plongements pour retrouver l'isogénie sur K. La complexité de cet algorithme est $O(n^{1+\epsilon}\ell^{1+\epsilon}P^{1+\epsilon})$; l'analyse détaillée montre que la complexité est une fois encore un peu moins bonne (i.e. avec quelques facteurs logarithmiques de plus) que celle des formules de Vélu sur K, même si la comparaison sur quelques exemples a montré qu'il était possible que cet algorithme soit compétitif en pratique.

Enfin, nous avons également généralisé l'algorithme pour résoudre le problème sur $\mathbb{F}_p$, en utilisant une procédure de *relèvement global par torsion* pour transformer le problème en un problème sur des courbes elliptiques sur un corps de nombres. Nous avons étudié la procédure de relèvement, et plus particulièrement le polynôme qui définit le corps de nombre que l'on obtient ; nous avons réussi à donner une borne sur la taille de ses racines. En utilisant ce dernier résultat, nous avons conjecturé que la précision à laquelle les plongements de K dans $\mathbb{C}$ devaient être calculés pour que l'algorithme fonctionne était en $O(\ell^2\log p)$, ce qui nous a permis d'obtenir la complexité finale de l'algorithme, en $O(\ell^{5+\epsilon}\log^{1+\epsilon}p)$. Cette complexité est bien pire (d'un facteur $O(\ell^4)$) que celle des formules de Vélu sur $\mathbb{F}_p$, et cela se vérifie en pratique.

Il serait intéressant de pousser plus loin l'étude de cette méthode. En particulier, il semblerait que l'algorithme se généralise relativement bien en genre supérieur, pour des courbes hyperelliptiques définies sur $\mathbb{C}$ ou sur K, en utilisant nos méthodes pour calculer l'application d'Abel-Jacobi ; en revanche, la procédure de relèvement se généralise mal en genre supérieur, car elle implique de calculer des polynômes de division, qui sont mal connus en genre g. Enfin, cette méthode pourrait peut-être permettre de résoudre un problème plus complexe, celui de déterminer une ℓ-isogénie entre deux courbes données, en s'aidant d'un travail similaire [VW00].

7 Conclusion

Durant cette thèse, nous avons mis au point plusieurs algorithmes de complexité quasi-linéaire pour calculer des fonctions en précision arbitraire. L'étude d'un algorithme existant pour les theta-constantes, utilisant les liens avec l'AGM et la méthode de Newton, nous a permis de trouver une généralisation de cet algorithme, en étudiant des suites dérivées des formules de τ-duplication de theta et en combinant leur évaluation à la méthode de Newton. Nous sommes parvenu à généraliser l'algorithme en genre 2, modulo une conjecture d'inversibilité de la matrice jacobienne du système ; une généralisation en genre g est sans doute possible, mais l'utilisation de la méthode de Newton dans ce cas-là n'est pas aussi directe. Grâce à ces algorithmes, il est possible de calculer l'application d'Abel-Jacobi en temps quasi-linéaire, de même que son inverse en genre 1 et 2, grâce à des algorithmes que nous décrivons ici. Une application concerne le calcul d'isogénies de courbes elliptiques : nous utilisons nos résultats pour construire un algorithme calculant une isogénie de noyau donné entre courbes complexes, que nous généralisons également pour des courbes définies sur des corps de nombres et des corps finis. Cet algorithme est de complexité plus grande que l'état de l'art, mais contrairement à celui-ci pourrait se généraliser plus facilement au genre supérieur.

Résumé

L'application d'Abel-Jacobi fait le lien entre la forme de Weierstrass d'une courbe elliptique définie sur $\mathbb{C}$ et le tore complexe qui lui est associé. Il est possible de la calculer en un nombre d'opérations quasi-linéaire en la précision voulue, c'est à dire en temps $O(\mathcal{M}(P)\log P)$. Son inverse est donné par la fonction $\wp$ de Weierstrass, qui s'exprime en fonction de θ, une fonction importante en théorie des nombres. L'algorithme naturel d'évaluation de θ nécessite $O(\mathcal{M}(P)\sqrt{P})$ opérations, mais certaines valeurs (les thêta-constantes) peuvent être calculées en $O(\mathcal{M}(P)\log P)$ opérations en exploitant les liens avec la moyenne arithmético-géométrique (AGM).

Dans ce manuscrit, nous généralisons cet algorithme afin de calculer θ en $O(\mathcal{M}(P)\log P)$. Nous exhibons une fonction $\mathfrak{F}$ qui a des propriétés similaires à l'AGM. D'une façon similaire à l'algorithme pour les thêta-constantes, nous pouvons alors utiliser la méthode de Newton pour calculer la valeur de θ. Nous avons implanté cet algorithme, qui est plus rapide que la méthode naïve pour des précisions supérieures à 260 000 chiffres décimaux.

Nous montrons comment généraliser cet algorithme en genre supérieur, et en particulier comment généraliser la fonction $\mathfrak{F}$. En genre 2, nous sommes parvenus à prouver que la même méthode mène à un algorithme qui évalue θ en $O(\mathcal{M}(P)\log P)$ opérations ; la même complexité s'applique aussi à l'application d'Abel-Jacobi. Cet algorithme est plus rapide que la méthode naïve pour des précisions plus faibles qu'en genre 1, de l'ordre de 3 000 chiffres décimaux. Nous esquissons également des pistes pour obtenir la même complexité en genre quelconque.

Enfin, nous exhibons un nouvel algorithme permettant de calculer une isogénie de courbes elliptiques de noyau donné. Cet algorithme utilise l'application d'Abel-Jacobi, car il est facile d'évaluer l'isogénie sur le tore ; il est sans doute possible de le généraliser au genre supérieur.

Mots-clés: fonctions thêta, application d'Abel-Jacobi, multiprécision, temps quasi-linéaire, courbes elliptiques, isogénies, courbes hyperelliptiques, cryptographie

Abstract

The Abel-Jacobi map links the short Weierstrass form of a complex elliptic curve to the complex torus associated to it. One can compute it with a number of operations which is quasi-linear in the target precision, i.e. in time $O(\mathcal{M}(P)\log P)$. Its inverse is given by Weierstrass's $\wp$ function, which can be written as a function of θ, an important function in number theory. The natural algorithm for evaluating θ requires $O(\mathcal{M}(P)\sqrt{P})$ operations, but some values (the theta-constants) can be computed in $O(\mathcal{M}(P)\log P)$ operations by exploiting the links with the arithmetico-geometric mean (AGM).

In this manuscript, we generalize this algorithm in order to compute θ in $O(\mathcal{M}(P)\log P)$. We give a function $\mathfrak{F}$ which has similar properties to the AGM. As with the algorithm for theta-constants, we can then use Newton's method to compute the value of θ. We implemented this algorithm, which is faster than the naive method for precisions larger than 260 000 decimal digits.

We then study the generalization of this algorithm in higher genus, and in particular how to generalize the $\mathfrak{F}$ function. In genus 2, we managed to prove that the same method leads to a $O(\mathcal{M}(P)\log P)$ algorithm for θ; the same complexity applies to the Abel-Jacobi map. This algorithm is faster than the naive method for precisions smaller than in genus 1, of about 3 000 decimal digits. We also outline a way one could reach the same complexity in any genus.

Finally, we study a new algorithm which computes an isogeny of elliptic curves with given kernel. This algorithm uses the Abel-Jacobi map because it is easy to evaluate the isogeny on the complex torus; this algorithm may be generalizable to higher genera.

Keywords: theta functions, Abel-Jacobi map, multiprecision, quasi-linear time, elliptic curves, isogenies, hyperelliptic curves, cryptography