LMS J. Comput. Math. 19 (Special issue A) (2016) 163–177 © 2016 Authors
doi:10.1112/S1461157016000309

Computing theta functions in quasi-linear time in genus two and above

Hugo Labrande and Emmanuel Thomé

Abstract

We outline an algorithm to compute $\theta(z,\tau)$ in genus two in quasi-linear time, borrowing ideas from the algorithm for theta constants and the one for $\theta(z,\tau)$ in genus one. Our implementation shows a large speed-up for precisions as low as a few thousand decimal digits. We also lay out a strategy to generalize this algorithm to genus g.

1. *Introduction*

The θ function is important to several fields of mathematics, such as the resolution of some non-linear differential equations or the study of complex Riemann surfaces of any genus, including the important case of complex elliptic and hyperelliptic curves. The numerous properties of this function underline its connection to deep topics. We refer to [**3**, **10**, **16**, **18**] and references therein for the uses of θ in various settings. Specific values of θ, called *theta constants*, are also of interest, for instance in the study of modular forms [**14**, **16**]. The problem we consider in this paper is the multi-precision computation of θ, that is, finding a fast algorithm for computing any number P of exact bits of θ. This problem has applications in the case of theta constants [**6**, **9**]; in the more general case of theta-functions, it allows us to compute the Abel–Jacobi map with large precision, thereby making the algebraic–analytic link effective. Such a link offers, for example, an alternative way of computing isogenies using embeddings to the complex numbers.

In the case of genus one, the theta constants exhibit a deep link with the arithmetic–geometric mean (AGM) [**2**]. Using the homogeneity of the AGM gives a function which takes a simple value at the theta constants; Newton's method can then be used to compute them (see, for example, [**5**]). This method has also been generalized in [**4**] to genus two theta constants, using the connection to the *Borchardt mean*; hints of a generalization to genus g are also given. Both algorithms have a *quasi-linear* asymptotic running time, that is, they compute the first P bits of the theta constants in $O(\mathcal{M}(P)\log P)$ operations, where $\mathcal{M}(P)$ is the cost of multiplying two P-bit numbers. An implementation of the algorithm has been released in the CMH package [**8**] and used to compute class polynomials of record size [**9**].

In [**15**], we used a similar approach to design an algorithm that computes $\theta(z,\tau)$ in genus one, for any arguments z,τ, also in asymptotic quasi-linear time. This required the design of a function, inspired by the arithmetic–geometric mean, that takes a special value when evaluated at $\theta(z,\tau)$ and at the theta constants, and could be evaluated in quasi-linear time. The quasi-linear complexity beats that of the naive algorithm, which is $O(\mathcal{M}(P)\sqrt{P})$; in practice, our algorithm beats an optimized version of the naive algorithm for precision above a hundred thousand decimal digits.

Received 21 February 2016.

2010 Mathematics Subject Classification 14K25 (primary), 14H42, 32G81 (secondary).

Contributed to the Twelfth Algorithmic Number Theory Symposium (ANTS-XII), Kaiserslautern, Germany, 29 August–2 September 2016.

In this article, we generalize this strategy to theta functions of any genus. We provide a careful analysis in the case of genus two, by finding a function similar to the Borchardt mean that can also be computed with precision P in $O(\mathcal{M}(P)\log P)$ and then inverting it using Newton's method. The algorithm achieves a quasi-linear complexity in P, neglecting the dependency in z and τ. Numerical experiments show that our implementation is faster than the naive algorithm for precisions as low as 3000 decimal digits. For higher genera, we outline a way that one could generalize this algorithm to genus g, with a complexity exponential in g but quasi-linear in P. Throughout the paper, we deliberately omit dealing with the precision losses; the full analysis in genus one in [15] shows that, given the argument reduction strategies, the loss of precision is not significant asymptotically in any of the building blocks ($O(\log P)$ for the most part, at most cP with $c < 1$). Such a result is expected to hold, for similar reasons, in genus two and in genus g.

This article is organized as follows. Section 2 lays out the background on genus g theta functions and algorithms to compute them. We then detail, in § 3, our algorithm for genus two, while § 4 shows how it could be generalized to arbitrary genus.

Notation 1.1. Throughout the paper, we use the following notation. For a sequence $(a_i)_i$, we denote by $a_{i_0,\dots,i_{n-1}}$ the n-uple (or n-vector) $(a_{i_0},\dots,a_{i_{n-1}})$.

2. *Background on genus g theta functions*

2.1. *Definitions*

Definition 2.1 [16, § II.1]. The *Siegel upper-half space* $\mathcal{H}_g$ is the set of symmetric $g\times g$ complex matrices τ whose imaginary part is positive definite. We write $\tau = (\tau_{i,j})$, except in genus two, where we write $\tau = \left(\begin{smallmatrix}\tau_1 & \tau_3\\ \tau_3 & \tau_2\end{smallmatrix}\right)$.

Definition 2.2 (θ function). Let $z\in\mathbb{C}^g$ and $\tau\in\mathcal{H}_g$. The θ function and the associated *theta functions with characteristics* are defined as

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

$$\text{For } a,b\in\frac{1}{2}\mathbb{Z}^g/\mathbb{Z}^g,\quad \theta_{[a;b]}(z,\tau) = \sum_{n\in\mathbb{Z}^g}\exp(i\pi{}^t(n+a)\tau(n+a))\exp(2i\pi{}^t(n+a)(z+b)).$$

Finally, *theta constants* are the values in $z=0$ of the functions $\theta_{[a;b]}$.

As in [1, 4, 9], we will often write $\theta_{[a;b]}$ as θ_i for the integer $i = 2(b_0+2b_1+\ldots+2^{g-1}b_{g-1}) + 2^{g+1}(a_0+2a_1+\ldots+2^{g-1}a_{g-1})$ whose binary expansion is $(2a\|2b)$. We call the *fundamental theta functions* $\theta_0,\dots,\theta_{2^g-1}$, that is, the ones with $a=0$.

Proposition 2.3 (Quasi-periodicity; [16, pp. 120–123]). *For all* $m\in\mathbb{Z}^g$,

$$\theta_{[a;b]}(z+m,\tau) = \exp(2i\pi{}^t am)\theta_{[a;b]}(z,\tau)\quad (\theta_{[0;b]} \textit{ is invariant by } z\to z+m),$$
$$\theta_{[a;b]}(z+\tau m,\tau) = \exp(-2i\pi{}^t bm)\exp(-i\pi{}^t m\tau m)\exp(-2i\pi{}^t mz)\theta_{[a;b]}(z,\tau).$$

2.2. *Fundamental domain*

Definition 2.4. $\mathrm{Sp}_{2g}(\mathbb{Z})$, the *symplectic group* of dimension g, is the set of matrices $\left(\begin{smallmatrix}A & B\\ C & D\end{smallmatrix}\right)\in\mathcal{M}_{2g}(\mathbb{Z})$, such that ${}^tAC = {}^tCA$, ${}^tBD = {}^tDB$ and ${}^tAD - {}^tCB = I_g$.

PROPOSITION 2.5 [14, Proposition I.1.1]. *$\mathrm{Sp}_{2g}(\mathbb{Z})$ acts on $\mathcal{H}_g$ as follows. For $M = \left(\begin{smallmatrix} A & B \\ C & D \end{smallmatrix}\right) \in \mathrm{Sp}_{2g}(\mathbb{Z})$ and $\tau \in \mathcal{H}_g$, $M \cdot \tau = (A\tau + B)(C\tau + D)^{-1} \in \mathcal{H}_g$. Furthermore, M defines an isomorphism of complex tori between $\Lambda_\tau = \mathbb{C}^g/\tau\mathbb{Z}^g + \mathbb{Z}^g$ and $\Lambda_{M\cdot\tau}$ by $z \mapsto M \cdot_\tau z = {}^t(C\tau + D)^{-1}z$; we use the shorthand $M \cdot z$ when the context allows.*

PROPOSITION 2.6 [14, Definition I.3.1]. *The fundamental domain of the action of $\mathrm{Sp}_{2g}(\mathbb{Z})$ on $\mathcal{H}_g$ is the set $\mathcal{F}_g \subset \mathcal{H}_g$, defined as the matrices satisfying the conditions:*
- *$\mathrm{Im}(\tau)$ is Minkowski-reduced, that is, ${}^t v\,\mathrm{Im}(\tau)v \geqslant \mathrm{Im}(\tau_{k,k})$ for all integral v with $(v_k, \dots, v_n) = 1$ and $\mathrm{Im}(\tau_{k,k+1}) \geqslant 0$ for all k;*
- *$|\mathrm{Re}(\tau_{k,l})| \leqslant \frac{1}{2}$ for all $k, l \in \{1, \dots, n\}, k \leqslant l$; and*
- *$|\det(C\tau + D)| \geqslant 1$ for all $\left(\begin{smallmatrix} A & B \\ C & D \end{smallmatrix}\right) \in \mathrm{Sp}_{2g}(\mathbb{Z})$.*

The last condition can be replaced by a finite set of inequalities. However, an explicit description of those inequalities is not known, in general, which means that reducing a matrix to the fundamental domain is technically not feasible. The case $g = 2$ has been solved in [11], which gives nineteen necessary and sufficient inequalities.

THEOREM 2.7. *Let $M = \left(\begin{smallmatrix} A & B \\ C & D \end{smallmatrix}\right) \in \mathrm{Sp}_{2g}(\mathbb{Z})$ and $(z, \tau) \in \mathbb{C}^g \times \mathcal{H}_g$.*

$$\theta_i(M \cdot z, M \cdot \tau) = \zeta_M \sqrt{\det(C\tau + D)}\, e^{i\pi\,{}^t(M\cdot z)(Cz)} \theta_{\sigma_M(i)}(z, \tau), \tag{2.1}$$

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

This theorem is proved in [16, § II.5], in a special case, and in [13, Chapter 5, Theorem 2]; an outline of the proof can also be found in [1, Proposition 3.1.24].

2.3. *Algorithms for theta*

A naive algorithm for computing the θ function for any genus with arbitrary precision simply involves computing the series until the remainder is small enough. This naive approach is studied, for instance, in [3, 4]. Results giving the number of terms to compute usually require some assumptions, such as $\tau \in \mathcal{F}_g$, or that the quasi-periodicity has been used to make z small. The complexity of this method is, in general, $O(\mathcal{M}(P)P^{g/2})$ for P bits of precision, as we prove in § 4.

Fast algorithms to compute theta constants in the cases $g = 1$ and $g = 2$ are known; these algorithms require $O(\mathcal{M}(P)\log P)$ operations. Here we give a brief outline of these algorithms; more details can be found in [4, 5, 9]. The idea of both algorithms is to construct a function $\mathfrak{F}$ such that (using Notation 1.1)

$$\mathfrak{F}\left(\frac{\theta_{1,\dots,2^g-1}(0,\tau)^2}{\theta_0(0,\tau)^2}\right) = \mathfrak{F}\left(\frac{\theta_1(0,\tau)^2}{\theta_0(0,\tau)^2}, \dots, \frac{\theta_{2^g-1}(0,\tau)^2}{\theta_0(0,\tau)^2}\right) = \frac{1}{\theta_0(0,\tau)^2}.$$

The function $\mathfrak{F}$ is the arithmetic–geometric mean in genus one and the Borchardt mean $\mathcal{B}_2$ (as defined and studied in [4]) in genus two. One then uses equation (2.1) to find other quotients of theta constants such that evaluating $\mathfrak{F}$ at those points allows one to compute $\tau_{i,j}$. For instance, in genus two, the following holds for τ within a large domain

$$\mathcal{B}_2\left(\frac{\theta_{8,4,12}(0,\tau)^2}{\theta_0(0,\tau)^2}\right) = \mathcal{B}_2\left(\frac{\theta_8(0,\tau)^2}{\theta_0(0,\tau)^2}, \frac{\theta_4(0,\tau)^2}{\theta_0(0,\tau)^2}, \frac{\theta_{12}(0,\tau)^2}{\theta_0(0,\tau)^2}\right) = \frac{1}{(\tau_{12}^2 - \tau_{11}\tau_{22})\theta_0(0,\tau)^2}.$$

This also means that one must be able to compute all the theta constants from the fundamental theta constants. In the end, we get a function computing τ from the quotients of fundamental theta constants; Newton's method is then applied to give an algorithm for computing the theta constants.

The algorithm's complexity relies on computing $\mathfrak{F}$ efficiently, since Newton's method, while doubling the working precision at each step, does not add any extra asymptotic complexity. The efficiency stems from a fast speed of convergence.

Definition 2.8. A sequence (a_n) is said to be *quadratically convergent* to ℓ if it is convergent to ℓ and there is a $C > 0$ such that, for n large enough,

$$|a_{n+1} - a_n| \leqslant C|a_n - a_{n-1}|^2.$$

It has been known since Gauss that the arithmetic–geometric mean converges quadratically (see, for example, [2]), provided one does not pick the 'wrong' sign for the square root an infinite number of times; in genus two, the Borchardt mean also converges quadratically provided similar conditions are met [4]. These technical requirements were shown to hold in genus one for τ within the fundamental domain; a similar property is conjectured to hold in genus two as well [9, Conjecture 5.7 and Remark 5.9]. In both cases, this gives a $O(\mathcal{M}(P)\log P)$ algorithm to evaluate $\mathfrak{F}$ to P bits; a modification of the algorithm can remove the dependency of the complexity in z, τ.

We successfully generalized this strategy to the computation of the genus one function $\theta(z,\tau)$ in [15]. The function $\mathfrak{F}$ is more complex, since the generalization of the AGM we consider does not converge quadratically. Instead, a related sequence, obtained after considering homogenization, does. We obtained a $O(\mathcal{M}(P)\log P)$ complexity, which does not depend on z, τ. We released an GNU MPC [7] implementation of the algorithm, which is faster than the naive algorithm for precisions larger than 100 000 bits.

3. *Computing the genus two theta function*

In this section, we outline an algorithm for computing $\theta(z,\tau)$, where $z \in \mathbb{C}^2$ and $\tau \in \mathcal{H}_2$, with precision P in $O(\mathcal{M}(P)\log P)$ operations.

3.1. *Argument reduction*

For our purposes, we will not require τ to belong to the fundamental domain; weaker conditions are sufficient. Let $z = (z_1, z_2)$ and $\tau = \left(\begin{smallmatrix}\tau_1 & \tau_3\\ \tau_3 & \tau_2\end{smallmatrix}\right)$. First, we require τ to belong to the domain $\mathcal{F}_2'$, defined by the inequalities

$$0 \leqslant 2\,\mathrm{Im}(\tau_3) \leqslant \mathrm{Im}(\tau_1) \leqslant \mathrm{Im}(\tau_2), \quad |\mathrm{Re}(\tau_i)| \leqslant \tfrac{1}{2}, \quad \mathrm{Im}(\tau_1) \geqslant \sqrt{3}/2. \tag{3.1}$$

The first inequality corresponds to $\mathrm{Im}(\tau)$ being Minkowski-reduced. This domain is called $\mathcal{B}$ in [17], but we choose to highlight the genus in the name. We also require that z satisfy

$$|\mathrm{Re}(z_i)| \leqslant \frac{1}{2}, \quad |\mathrm{Im}(z_1)| \leqslant \frac{\mathrm{Im}(\tau_1) + \mathrm{Im}(\tau_3)}{2}, \quad |\mathrm{Im}(z_2)| \leqslant \frac{\mathrm{Im}(\tau_2) + \mathrm{Im}(\tau_3)}{2}. \tag{3.2}$$

Given any τ, reducing z so as to satisfy the above condition follows easily from the quasi-periodicity of θ (Proposition 2.3). One can write $z = x + \tau y$ with $x, y \in \mathbb{R}^2$ explicitly by solving the system formed by $z = x + \tau y$ and $\overline{z} = x + \overline{\tau} y$; one can then subtract multiples of one and τ from the first argument so as to get $|x_i|, |y_i| \leqslant \frac{1}{2}$.

Reducing τ to the domain $\mathcal{F}_2'$ instead of $\mathcal{F}_2$ is a coarser notion, which has the advantage of being generalizable to arbitrary genus, unlike the reduction to the fundamental domain $\mathcal{F}_g$; we discuss this generalization in § 4.1. In genus two, the strategy for this reduction is described in [17, § 6.3]: this gives a quasi-linear time algorithm for reducing τ to $\mathcal{F}_2'$.

3.2. *Naive algorithm*

Let $q_j = e^{i\pi\tau_j}$ and $w_j = e^{i\pi z_j}$. We have $|\theta_{[a;b]}(z,\tau)| \leqslant \sum_{m,n\in\mathbb{Z}} |q_1^{m^2} q_2^{n^2} q_3^{2mn} w_1^{2m} w_2^{2n}|$, using the triangular inequality. Since $\tau \in \mathcal{F}_2'$ and z satisfies (3.2),

$$|w_1| + |w_1^{-1}| \leqslant 2e^{\pi(3/4)\operatorname{Im}(\tau_1)}, \quad |w_2| + |w_2^{-1}| \leqslant 2e^{\pi(3/4)\operatorname{Im}(\tau_2)}.$$

Hence $(|w_1^{2m}| + |w_1^{-2m}|)(|w_2^{2n}| + |w_2^{-2n}|) \leqslant 4e^{i\pi((3/2)m\operatorname{Im}(\tau_1)+(3/2)n\operatorname{Im}(\tau_2))}$. Also

$$|q_1^{m^2} q_2^{n^2} q_3^{2mn}| \leqslant |q_1^{m^2} q_2^{n^2} q_3^{-m^2-n^2}| \leqslant |q_1^{m^2/2} q_2^{n^2/2}|.$$

Hence, if S_B denotes the sum of the series defining θ with $m, n \in [-B, B]$, a calculation very similar to the one in [15, Proposition 2.6] proves that

$$\begin{aligned}
|\theta(z,\tau) - S_B| &\leqslant \sum_{m\geqslant B} |q_1^{m^2}|(|w_1^{2m}| + |w_1^{-2m}|) \\
&\quad + \sum_{n\geqslant B} |q_2^{n^2}|(|w_2^{2n}| + |w_2^{-2n}|) + 4\sum_{m,n\geqslant B-2} |q_1^{m^2/2} q_2^{n^2/2}| \\
&\leqslant \frac{4}{1-|q_1|}(|q_1|^{(B-2)^2/2-2} + |q_1|^{(B-2)^2-4}) \\
&\leqslant 5(|q_1|^{(B-2)^2/2-2} + |q_1|^{2(B-2)^2-4}).
\end{aligned}$$

This means that $B = O(\sqrt{P/\operatorname{Im}(\tau_1)})$ terms sum to an approximation that is accurate to 2^{-P}.

Following and extending the strategy in [15, § 2.2.2] or [9, § 5.1], we use induction relations to compute terms more efficiently. Let $Q(m,n) = q_1^{m^2} q_2^{n^2} q_3^{2mn}$ and $T(m,n) = Q(m,n)(w_1^{2m}w_2^{2n} + w_1^{-2m}w_2^{-2n})$. With minor rewriting,

$$\theta_i(z,\tau) = \sum_{m,n\in\mathbb{N}} s(i,m,n)(T(m,n) + T(m,-n)),$$

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.

$$(w_1^2 + w_1^{-2})T(m,n) = q_1^{-2m-1} q_3^{-2n} T(m+1,n) + q_1^{2m-1} q_3^{2n} T(m-1,n), \tag{3.3}$$

$$(w_2^2 + w_2^{-2})T(m,n) = q_2^{-2n-1} q_3^{-2m} T(m,n+1) + q_2^{2n-1} q_3^{2m} T(m,n-1). \tag{3.4}$$

We propose an algorithm, Algorithm A.1 (cf. Appendix A), that uses these induction relations in such a way that the memory needed is only $O(1)$: it consists of iteratively computing the terms $T(m,n)$ for $n \in \{-1,0,1\}$, using equations (3.3), and then using those values as soon as they are computed to initialize the other induction, using equations (3.4). We implemented it in Magma and will discuss timings in § 3.6.

Even if the complexity of the naive algorithm is worse asymptotically than the complexity of the quasi-linear algorithm that we outline in the next sections, it is still an important building block, used for two purposes: to provide an initial approximation of the result, which is needed to initialize Newton's method, and to determine the sign of $\operatorname{Re}(\theta)$ or $\operatorname{Im}(\theta)$, which allows us to choose correct square roots for computing the function F, which we now define.

3.3. *The function F*

PROPOSITION 3.1 (Duplication formula, [1, formula (3.13)]). *For all* $a, b \in \frac{1}{2}\mathbb{Z}^g/\mathbb{Z}^g$,

$$\theta_{[a;b]}(z,\tau)^2 = \frac{1}{2^g} \sum_{\beta\in(1/2)\mathbb{Z}^g/\mathbb{Z}^g} e^{-4i\pi^t a\beta} \theta_{[0;b+\beta]}\left(z, \frac{\tau}{2}\right) \theta_{[0;\beta]}\left(0, \frac{\tau}{2}\right). \tag{3.5}$$

The definition above prompts us to define the following function, crafted so that Proposition 3.2 holds, as a direct consequence of Proposition 3.1. The definition below is ambiguous (because of square roots), an issue we deal with in what follows.

$$\begin{aligned} F : \mathbb{C}^8 &\to \mathbb{C}^8 \\ a_{0\ldots3}, b_{0\ldots3} &\mapsto \Bigg(\frac{\sqrt{a_0}\sqrt{b_0}+\sqrt{a_1}\sqrt{b_1}+\sqrt{a_2}\sqrt{b_2}+\sqrt{a_3}\sqrt{b_3}}{4}, \frac{\sqrt{a_0}\sqrt{b_1}+\sqrt{a_1}\sqrt{b_0}+\sqrt{a_2}\sqrt{b_3}+\sqrt{a_3}\sqrt{b_2}}{4}, \\ &\frac{\sqrt{a_0}\sqrt{b_2}+\sqrt{a_1}\sqrt{b_3}+\sqrt{a_2}\sqrt{b_0}+\sqrt{a_3}\sqrt{b_1}}{4}, \frac{\sqrt{a_0}\sqrt{b_3}+\sqrt{a_1}\sqrt{b_2}+\sqrt{a_2}\sqrt{b_1}+\sqrt{a_3}\sqrt{b_0}}{4}, \\ &\frac{b_0+b_1+b_2+b_3}{4}, \frac{2\sqrt{b_0}\sqrt{b_1}+2\sqrt{b_2}\sqrt{b_3}}{4}, \frac{2\sqrt{b_0}\sqrt{b_2}+2\sqrt{b_1}\sqrt{b_3}}{4}, \frac{2\sqrt{b_0}\sqrt{b_3}+2\sqrt{b_1}\sqrt{b_2}}{4}\Bigg). \end{aligned}$$

PROPOSITION 3.2. *For a suitable choice of square roots,*

$$F(\theta_{0,1,2,3}(z,\tau)^2, \theta_{0,1,2,3}(0,\tau)^2) = (\theta_{0,1,2,3}(z,2\tau)^2, \theta_{0,1,2,3}(0,2\tau)^2).$$

Bad, good and correct choices of square roots. We discuss what we mean above by a suitable choice of square roots. Two different notions must be considered.

– 'Good choices' in the sense of [2, 4], that is, such that $\mathrm{Re}(\sqrt{a_i}/\sqrt{a_j}), \mathrm{Re}(\sqrt{b_i}/\sqrt{b_j}) \geqslant 0$. Note that not all tuples of complex numbers admit a set of 'good' square roots. In genus one, having an infinite number of bad choices means that 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 that 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, that is, given two quadruples that 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' choices, which need not be 'good' choices. 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 one, [2] proves that, for $z = 0$ and τ within a large domain that includes the fundamental domain, the correct choice is always good. In [15], we proved a similar result for arbitrary z. In genus two, we do not determine an explicit domain for which correct choices are good. Although one can try to improve on the approach of § 3.2 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 § 3.4), so that proofs are difficult to obtain.

Iterates of F. In any genus, $\lim_{k\to\infty}(\theta_{[0;b]}(z,2^k\tau)/\theta_{[0;b']}(z,2^k\tau)) = 1$, which easily implies that correct choices are good for large enough τ. Therefore, given an 8-uple X that 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 on z and τ; since we neglect the dependency in z, τ in the complexity of our algorithm[†], 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 3.3. *Let* $(a^{(0)}_{0,1,2,3}, b^{(0)}_{0,1,2,3}) \in \mathbb{C}^8$ *and let* $(a^{(n+1)}_{0,1,2,3}, b^{(n+1)}_{0,1,2,3}) = F(a^{(n)}_{0,1,2,3}, b^{(n)}_{0,1,2,3})$ *for any integer* $n \in \mathbb{N}$. *Assume that there exists* $\alpha, \beta \in \mathbb{C}^*$ *and* $n_0 \in \mathbb{N}$ *such that* $\mathrm{Re}(a_i^{(n_0)}/\alpha) > 0$

[†]To extend this work into an algorithm whose complexity is uniform in z, τ, one could follow the approach of [4, 15], since we have again a naive algorithm with complexity that decreases as $\mathrm{Im}(\tau_1)$ increases.

and $\mathrm{Re}(b_i^{(n_0)}/\beta) > 0$ *for all* $i \in \{0,1,2,3\}$. *Then there exists positive real constants* c, C *such that for all* $n \geqslant n_0$, *for all* $i \in \{0,1,2,3\}, c \leqslant |a_i^{(n)}|, |b_i^{(n)}| < C$, *assuming that all choices of square roots from iteration* n_0 *onwards are good.*

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 $c = \min(\mathrm{Re}(a_{0,1,2,3}^{(n_0)}/\alpha), \mathrm{Re}(b_{0,1,2,3}^{(n_0)}/\beta))$. For good choices of square roots and any i, j, $\mathrm{Re}\left(\sqrt{a_i^{(n_0)}/\alpha}\sqrt{b_j^{(n_0)}/\beta}\right) \geqslant \min(\mathrm{Re}(a_i^{(n_0)}/\alpha), \mathrm{Re}(b_j^{(n_0)}/\beta)) \geqslant c$ (for a proof, see, for example, [4, Lemme 7.3]). This implies, from the definition, that $|a_i^{(n_0+1)}| \geqslant \mathrm{Re}(a_i^{(n_0+1)}/\sqrt{\alpha\beta}) \geqslant c$ and, similarly, for $b_i^{(n_0+1)}$. The result follows by induction (with $\sqrt{\alpha\beta}$, of modulus one, 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 the conditions of Lemma 3.3 hold after a few steps.

A Karatsuba-like trick to compute F. Proposition 3.4 computes F in four products and four squares instead of the twenty-two products in its definition. Section 4 extends this to genus g.

PROPOSITION 3.4. *Put* $\mathcal{H} = \left(\begin{smallmatrix}1 & 1\\ 1 & -1\end{smallmatrix}\right)$ *and* $\mathcal{H}_2 = \mathcal{H} \otimes \mathcal{H}$. *Let* ${}^t(m_{0,1,2,3}) = \mathcal{H}_2 {}^t\left(\sqrt{a_{0,1,2,3}^{(n)}}\right)$ *and* ${}^t(s_{0,1,2,3}) = \mathcal{H}_2 {}^t\left(\sqrt{b_{0,1,2,3}^{(n)}}\right)$. *We have* ${}^t(a_{0,1,2,3}^{(n+1)}) = \frac{1}{16}\mathcal{H}_2 {}^t(m_{0,1,2,3} * s_{0,1,2,3})$ ($*$ *being the termwise product) and* ${}^t(b_{0,1,2,3}^{(n+1)}) = \frac{1}{16}\mathcal{H}_2 {}^t(s_{0,1,2,3}^2)$.

3.4. *Constructing and inverting the* $\mathfrak{F}$ *function*

The following homogeneity property of F is a trivial fact.

PROPOSITION 3.5. *Let*

$$({a_{0,1,2,3}}^{(n)}, {b_{0,1,2,3}}^{(n)}) = F^n(\theta_{0,1,2,3}(z,\tau)^2, \theta_{0,1,2,3}(0,\tau)^2),$$
$$({a'_{0,1,2,3}}^{(n)}, {b'_{0,1,2,3}}^{(n)}) = F^n(\lambda\theta_{0,1,2,3}(z,\tau)^2, \mu\theta_{0,1,2,3}(0,\tau)^2).$$

Then ${a'_0}^{(n)} = \epsilon_n \lambda^{1/2^n} \mu^{1-1/2^n} a_0^{(n)}$ (*where* $\epsilon_n^{2^n} = 1$) *and* ${b'_0}^{(n)} = \mu b_0^{(n)}$.

PROPOSITION 3.6. *Let* $(a_{0,1,2,3}^{(0)}, b_{0,1,2,3}^{(0)})$ *within a neighborhood of diameter* 2^{-P_0} *of a pair of two quadruples proportional to* $(\theta_{0,1,2,3}(z,\tau)^2, \theta_{0,1,2,3}(0,\tau)^2)$. *Define* $\mathfrak{G}$ *as*

$$\mathfrak{G}(a_{0,1,2,3}^{(0)}, b_{0,1,2,3}^{(0)}) = \left(\lim_{n\to\infty} ({a_0}^{(n)}/{b_0}^{(n)})^{2^n} \times b_0^{(n)}, \lim_{n\to\infty} b_0^{(n)}\right),$$

using the approximations at precision P_0 *of* θ *to choose the correct signs (assuming the neighborhood is narrow enough). Then* $\mathfrak{G}(\lambda\theta_{0,1,2,3}(z,\tau)^2, \mu\theta_{0,1,2,3}(0,\tau)^2) = (\lambda, \mu)$.

As discussed earlier, the precision P_0 that is needed above to define $\mathfrak{G}$ depends only on z, τ, and not on the precision P desired for $\mathfrak{G}$. We prove, in § 3.5, that $\mathfrak{G}$ can be computed in $O(\mathcal{M}(P)\log P)$ operations.

Next, we build $\mathfrak{F}$ from $\mathfrak{G}$. The idea is to evaluate $\mathfrak{G}$ at approximations of well-chosen theta functions, selected according to the action of $(\mathfrak{J}\mathfrak{M}_i)^2$. This gives values of λ and μ which are the given by the following proposition.

PROPOSITION 3.7. *Define* $\mathfrak{J} = \left(\begin{smallmatrix} 0 & -I_2 \\ I_2 & 0 \end{smallmatrix}\right)$ *and* $\mathfrak{M}_i = \left(\begin{smallmatrix} I_2 & m_i \\ 0 & I_2 \end{smallmatrix}\right)$, *with* $m_1 = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right)$, $m_2 = \left(\begin{smallmatrix} 0 & 0 \\ 0 & 1 \end{smallmatrix}\right)$, $m_3 = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$ *(as in* [4, Chapter 6]*). Then*

$$\theta_{0,1,2,3}((\mathfrak{J}\mathfrak{M}_1)^2 \cdot z, (\mathfrak{J}\mathfrak{M}_1)^2 \cdot \tau)^2 = -\tau_1 e^{2i\pi z_1^2/\tau_1} \theta_{8,9,0,1}(z,\tau)^2,$$
$$\theta_{0,1,2,3}((\mathfrak{J}\mathfrak{M}_2)^2 \cdot z, (\mathfrak{J}\mathfrak{M}_2)^2 \cdot \tau)^2 = -\tau_2 e^{2i\pi z_2^2/\tau_2} \theta_{4,0,6,2}(z,\tau)^2,$$
$$\theta_{0,1,2,3}((\mathfrak{J}\mathfrak{M}_3)^2 \cdot z, (\mathfrak{J}\mathfrak{M}_3)^2 \cdot \tau)^2 = (\tau_3^2 - \tau_1\tau_2) e^{2i\pi((z_1^2\tau_2 + z_2^2\tau_1 - 2z_1z_2\tau_3)/\det(\tau))} \theta_{0,8,4,12}(z,\tau)^2.$$

This proposition is a direct consequence of equation (2.1)[†]. Note that $(\mathfrak{J}\mathfrak{M}_i)^2 \cdot \tau \notin \mathcal{F}'_g$, which prevents us from generalizing proofs which worked for genus one to make 'good choices' and 'correct choices' coincide. However, it is still possible to determine the sign in the computation of $\mathfrak{G}$ using low-precision approximations.

In order to use Proposition 3.7, we need to compute $\theta^2_{[a;b]}(z,\tau)$ for $a \neq 0$. Following [9, Algorithm 10], we define $\theta_{[0;b]}(z,\tau/2)$ and $\theta_{[0;b]}(0,\tau/2)$ as our input values and obtain $\theta^2_{[a;b]}(z,\tau)$ and $\theta^2_{[a;b]}(0,\tau)$ with equation (3.5). Then evaluating $\mathfrak{G}$ at selected quadruples (for example, $\theta_{8,9,0,1}$) gives proportionality factors which are inverses of the factors in Proposition 3.7.

Defining $\mathfrak{F}$ (that is, defining which complex numbers we derive from the proportionality factors computed above) so that it is locally invertible and Newton's method can be used, requires some care. In genus one, 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 theta. In genus two, the function is from $\mathbb{C}^6$ to $\mathbb{C}^5$. There are two possible workarounds. A natural idea would be to add an extra equation: the equation of a variety on which the theta lie. In [10], the equation of the Kummer surface is given. Generalizing this approach to higher genus is, unfortunately, cumbersome. 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 choose to compute three pairs of (λ, μ), so that

$$\mathfrak{F}\left(\frac{\theta_{1,2,3}(z,\tau/2)}{\theta_0(z,\tau/2)}, \frac{\theta_{1,2,3}(0,\tau/2)}{\theta_0(0,\tau/2)}\right)$$
$$= \left(e^{2i\pi(z_1^2/\tau_1)}, e^{2i\pi(z_2^2/\tau_2)}, e^{2i\pi((z_1^2\tau_2+z_2^2\tau_1-2z_1z_2\tau_3)/\det(\tau))}, \tau_1, \tau_2, \tau_3^2 - \tau_1\tau_2\right).$$

Our function $\mathfrak{F}$ is defined in short form in Algorithm 1 through the action it has on the particular values that are quotients of theta functions. The calculations involved are valid for any 6-uple of complex numbers, and a complete description is easy to obtain. We do so in Algorithm B.1 (Appendix B).

We conjecture the following, which holds experimentally.

CONJECTURE 3.8. The Jacobian of $\mathfrak{F}$ is invertible.

To finish, we describe how we use Newton's method to get an approximation of θ with precision $p - \delta$, where δ is a small constant, from an approximation with precision $p/2$. We compute an approximation of $\partial\mathfrak{F}_i/\partial x_j$ with precision p using finite differences, that is, $(\mathfrak{F}_i(x + \epsilon_j) - \mathfrak{F}_i(x))/\|\epsilon_j\|$, where ϵ_j is a perturbation of 2^{-p} on the jth coordinate. 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 [4, 9, 15], we can compute an approximation of θ with precision P_0 using the naive algorithm and then use Newton's method to refine it into a value of θ with precision P; the total cost of this algorithm is $O(\mathcal{M}(P)\log P)$.

[†]Note that the result appears different from [4], only because the tables for $\mathfrak{M}_1$ and $\mathfrak{M}_2$ (p. 146) have been switched by mistake; and it differs from [9] because their $\mathfrak{M}_2$ is our $\mathfrak{M}_3$, and vice-versa.

Algorithm 1 Compute $\mathfrak{F}\left(\frac{\theta_{1,2,3}}{\theta_0}(z,\tau/2), \frac{\theta_{1,2,3}}{\theta_0}(0,\tau/2)\right)$ (see also Appendix B).

```
1: Compute $\frac{\theta_i(z,\tau)^2}{\theta_0(z,\tau/2)\theta_0(0,\tau/2)}, \frac{\theta_i(0,\tau)^2}{\theta_0(0,\tau/2)^2}$ using equation (3.5).
2: Compute $\mathfrak{G}\left(\frac{\theta_{0,1,2,3}(z,\tau)^2}{\theta_0(z,\tau/2)\theta_0(0,\tau/2)}, \frac{\theta_{0,1,2,3}(0,\tau)^2}{\theta_0(0,\tau/2)^2}\right) = \left(\frac{1}{\theta_0(z,\tau/2)\theta_0(0,\tau/2)}, \frac{1}{\theta_0(0,\tau/2)^2}\right)$.
3: Compute $\lambda_1, \mu_1 = \mathfrak{G}\ (\theta^2_{8,9,0,1}(z,\tau), \theta^2_{8,9,0,1}(0,\tau))$.
4: Compute $\lambda_2, \mu_2 = \mathfrak{G}\ (\theta^2_{4,0,6,2}(z,\tau), \theta^2_{4,0,6,2}(0,\tau))$.
5: Compute $\lambda_3, \mu_3 = \mathfrak{G}\ (\theta^2_{0,8,4,12}(z,\tau), \theta^2_{0,8,4,12}(0,\tau))$.
6: return $(\mu_1/\lambda_1, \mu_2/\lambda_2, \mu_3/\lambda_3, -1/\mu_1, -1/\mu_2, 1/\mu_3)$.
```

3.5. *Proof of quasi-linear time*

THEOREM 3.9. *One can compute $\mathfrak{G}(a^{(0)}_{0,1,2,3}, b^{(0)}_{0,1,2,3}) = (\lambda,\mu)$ with precision P in $O(\mathcal{M}(P)\log P)$ operations, assuming that the choice of signs is always good.*

The result if the arguments are $(\lambda\theta^2_{0,1,2,3}(z,\tau), \mu\theta^2_{0,1,2,3}(0,\tau))$ 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.

Proof. By Lemma 3.3, $0 < c \leqslant |a_i^{(n)}|, |b_i^{(n)}| \leqslant C$ for any i, for n large enough (independent of P). The sequence $d_n = \max_{i,j} |b_i^{(n)} - b_j^{(n)}|$ converges quadratically to zero [4, Proposition 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 $d_{n+1} \leqslant Ad_n^2$ for all $n \geqslant n_1$ and, additionally, so that $d_{n_1} < 1/2A$. This implies that $d_n \leqslant (1/A)2^{-2^{n-n_1}}$ for any $n \geqslant n_1$. The $|a_i^{(n)} - a_j^{(n)}|$ can be linked to d_n. For instance, for any $n \geqslant n_1$,

$$\begin{aligned}
\left|a_0^{(n+1)} - a_1^{(n+1)}\right| &= \frac{|m_1s_1 + m_3s_3|}{2} \\
&\leqslant 2C(|s_1| + |s_3|) \quad \text{(using the notation of Proposition 3.4)} \\
&\leqslant 4C(|\sqrt{b_0} - \sqrt{b_1}| + |\sqrt{b_2} - \sqrt{b_3}|) \\
&\leqslant 8C\sqrt{A}\sqrt{d_n} \quad \text{using } |\sqrt{b_i} - \sqrt{b_j}| \leqslant |\sqrt{b_i} + \sqrt{b_j}|.
\end{aligned}$$

Calculus also shows that

$$\begin{aligned}
\left|a_0^{(n+1)} - \sqrt{a_0^{(n)}b_0^{(n)}}\right| &= \frac{1}{8}\left|\sum_{i=1}^{3}(\sqrt{a_i} - \sqrt{a_0})(\sqrt{b_i} + \sqrt{b_0}) + (\sqrt{a_i} + \sqrt{a_0})(\sqrt{b_i} - \sqrt{b_0})\right| \\
&\leqslant K\sqrt{d_n} \quad \text{for some explicit constant } K.
\end{aligned}$$

Superscripts $^{(n)}$ have been omitted from the right-hand sides above for brevity. For both inequalities, we used the fact that the choices of roots are good.

We now show that $\lambda_n = (a_0^{(n)}/b_0^{(n)})^{2^n}$ converges quadratically. Let $q_n = (a_0^{(n+1)}/b_0^{(n+1)})^2/(a_0^{(n)}/b_0^{(n)})$, so that $\lambda_{n+1} = \lambda_n q_n^{2^n}$. Given the bounds established above, it is relatively easy to check that $|q_{n+1} - 1|$ is also bounded by $K'\sqrt{d_n}$ for an explicit constant K'. It follows, from an unsurprising calculation done in [15, § 3.4], 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. □

3.6. *Implementation results*

Our Magma implementation of Algorithm A.1 and our quasi-linear time algorithm is at http://www.hlabrande.fr/pubs/fastthetasgenus2.m

We compared this with Magma's general-purpose `Theta` function. Assuming that the latter computes each term by exponentiation, its complexity would be $O(\mathcal{M}(P)P\log P)$. However, practice reveals that it behaves much worse. Table 1 shows that, for precision above 1000 decimal digits, our algorithm, which outputs eight values, is faster than one call to Magma's `Theta` function, which only computes $\theta(z,\tau)$. Furthermore, it is also faster than Algorithm A.1 for precisions greater than 3000 digits. This cut-off is much lower than in genus one, which is expected since the complexity of the naive algorithm is $O(\mathcal{M}(P)\sqrt{P})$ in genus one and $O(\mathcal{M}(P)P)$ in genus two. Our results are consistent with the situation for theta constants, studied in [9][†]

TABLE 1. *Times (in s) of different methods.*

Prec (digits)	Magma	Algorithm A.1	This work
1 000	0.42	0.38	0.38
2 000	2.58	1.86	1.86
4 000	18.4	9.51	6.65
8 000	128	53.9	13.2
16 000	889	303	25
32 000	6 368	1535	50
64 000	46 566	8798	120

4. *Extending the algorithm to 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.

4.1. *Argument reduction*

We extend the domain $\mathcal{F}'_2$ to genus g as follows. $\mathcal{F}'_g$ is the set of τ such that

$$\mathrm{Re}(\tau_{i,j}) \leqslant \tfrac{1}{2}, \quad \mathrm{Im}(\tau) \text{ is Minkowski-reduced}, \quad \mathrm{Im}(\tau_{1,1}) \geqslant \sqrt{3}/2.$$

Note that $\mathcal{F}_g \subset \mathcal{F}'_g$, since $N_0 = \left(\begin{smallmatrix} I_g-\delta_{1,1} & -\delta_{1,1} \\ \delta_{1,1} & I_g-\delta_{1,1}\end{smallmatrix}\right)$, where $\delta_{1,1}$ is the $g\times g$ Kronecker matrix, which is symplectic and such that $|\det(C\tau+D)| = |\tau_{1,1}|$. The algorithm to reduce τ is similar to [17, Algorithm 6.8]; this is Algorithm 2.

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

```
1: $\tau' \leftarrow$ Minkowski reduction of $\tau$.
2: Subtract an integer matrix to $\tau'$ so that $|\mathrm{Re}(\tau'_{i,j})| \leqslant \frac{1}{2}$.
3: If $|\tau'_{1,1}| \leqslant 1$, do $\tau' \leftarrow N_0 \cdot \tau'$ and go back to Step 2.
4: **return** $\tau'$.
```

PROPOSITION 4.1. *Algorithm 2 terminates.*

[†]Compared with [9], we compute more, and we do it in Magma, not in C. Hence the slower timings.

Proof. We generalize the lemmas in [17, § 6.4]; the proof is rather technical. Lemma 6.9 holds in genus g; Lemma 6.14 becomes

$$(m_2 \dots m_g)(\mathrm{Im}(M(Z))) \leqslant c_1(g) \max\{m_1(Y)^{-1}, (m_2 \dots m_g)(Y)\}$$

where $c_1(g)$ is the constant in Minkowski's inequality [14, Proposition I.2.1, p. 13].

We generalize Lemma 6.12, as follows. Let R_g be the set of Minkowski-reduced matrices and let $Q'_g(t)$ be defined as in [14, Definition I.2.3]. Then the remark after [14, Definition I.2.2] and [14, Proposition I.2.2] proves that there is a $t' > 2$ such that $R_g \subset Q'_g(t')$. Consider the set of matrices τ' obtained during the execution of Algorithm 2 for which $y_1 \geqslant 1/t'$; this set injects in the set $L_g(t')$, defined in [14, Definition I.3.2]. Applying [14, Theorem I.3.1] yields the result that there are only a finite number of steps in which $y_1 > 1/t'$; note that c is the number of such steps (that is, the cardinality of the set in [14, Theorem I.3.1]).

Lemma 6.11, with the bound $y_1 \leqslant 1/t'$, holds in genus g, since $t' > 2$. Combining all the lemmas, as in [17, Propisition 6.13], proves termination since, after k iterations,

$$2^{k-c} \leqslant c_1(g)^3 \max\{m_1(Y_0)^{-3}(m_2 \dots m_g)(Y_0)^{-1}, (m_2 \dots m_g)(Y_0) m_1(Y_0)^{-1}\}. \qquad \square$$

Bounding the number of steps in the algorithm requires making a few theorems explicit, namely, [14, Proposition I.2.2] (making t' explicit) and [14, Theorem I.3.1] (determining c). We believe that the number of steps is exponential in g. Furthermore, each step requires computing the Minkowsi reduction of a $g \times g$ matrix, which has a cost of $O(2^{1.3g^3})$ arithmetic operations [12]. Hence the running time of this reduction is exponential in g.

The conditions on z, which can be met using Proposition 2.3, are

$$|\mathrm{Re}(z_i)| \leqslant \frac{1}{2}, \quad |\mathrm{Im}(z_i)| \leqslant \frac{\sum_{j \in [1..2g]} |\mathrm{Im}(\tau_{i,j})|}{2}.$$

We note that [3] uses another approach, the so-called Siegel reduction, with weaker conditions than the ones we impose here, for example using LLL reduction instead of Minkowski reduction. It is apparently enough to limit the number of terms in the naive algorithm.

4.2. *Naive algorithm*

In [3], the authors compute an ellipsoid containing the indices of the terms one needs to sum to get an approximation of $\theta(z, \tau)$ up to ϵ. Its size depends on R, defined as the solution to the equation $\epsilon = (g/2)(2^g/\rho^g)\Gamma(g/2, (R - \rho/2)^2)$, where Γ is the incomplete gamma function and ρ is the smallest vector after an orthogonal change of basis.

Neglecting the dependency in τ and z, we get the rather coarse bound of $O(R^g)$ terms needed. We complete the analysis in [3] by computing an explicit estimate on R.

PROPOSITION 4.2. *Treating z, τ (and hence ρ) as constants, we have $R = O(\sqrt{P})$, that is, 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

$$\begin{aligned}\Gamma(g/2, d) &= (g/2-1)! e^{-d} + \sum_{i=1}^{g/2} (g/2-1) \dots (g/2-i) d^{g/2-i} e^{-d} \\ &\leqslant \frac{g}{2}(g/2-1)! d^{g/2-1} e^{-d} \leqslant e^{-d + g/2(\log d + \log(g/2))}.\end{aligned}$$

Hence

$$\frac{g}{2}\frac{2^g}{\rho^g}\Gamma(g/2, d) \leqslant 2^{-d\log_2 e + g/2(\log d + \log(g/2)) + \log(g/2) + g\log(2/\rho)}.$$

Asymptotically, that is, 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 $R = O(\sqrt{P})$. $\square$

The terms can be computed using induction relations, which exist whatever the genus: if $g-1$ indices are fixed, there exists a relationship between three consecutive terms for the remaining index. However, exploiting those relationships gets increasingly complicated and harder to code efficiently as the genus grows. We assume that such induction relations are used in the naive algorithm: that is, that the cost of computing each term is only $O(\mathcal{M}(P))$ and that the memory needs are $O(1)$ or $O(g)$. Under this assumption, the cost of the naive algorithm is $O(\mathcal{M}(P)P^{g/2})$ operations, which agrees with our analyses in genus one and genus two.

4.3. *The function F*

We use, once again, the τ-duplication formula (equation (3.5)) with $a=0$

$$\theta_i(z,2\tau)^2 = \frac{1}{2^g}\sum_{k\in\{0,\dots,2^g-1\}} \theta_{i\oplus k}(z,\tau)\theta_k(0,\tau),$$

where $\oplus$ is the bitwise XOR. This gives us a function

$$F:\mathbb{C}^{2^{g+1}} \to \mathbb{C}^{2^{g+1}}$$
$$((\theta^2_{[0;b]}(z,\tau)),(\theta^2_{[0;b]}(0,\tau))) \mapsto ((\theta^2_{[0;b]}(z,2\tau)),(\theta^2_{[0;b]}(0,2\tau))).$$

Just as in genus two, we solve the problem of computing the correct square root by evaluating θ at low precision; this requires a number of operations independent of P.

The trick used in Proposition 3.4 can be generalized, using $\mathcal{H}_g = \mathcal{H}\otimes\ldots\otimes\mathcal{H}$ (g times). Hence F can be computed with 2^{g+1} multiplications, which is better than 2^{2g+1}.

4.4. *Extending the quasi-linear time algorithm*

4.4.1. *Defining $\mathfrak{F}$.* We can study the homogeneity of the formulas defining F; equation (3.5) gives, for instance, $\theta(z,2\tau)^2 = \sum_{i=0}^{2^g-1}\sqrt{\theta_i(z,\tau)^2}\sqrt{\theta_i(0,\tau)^2}$. Combined with the expression of the Borchardt mean, it is then easy to generalize $\mathfrak{G}$ in the same way as it is defined in Proposition 3.6. We conjecture that Theorem 3.9 generalizes.

Conjecture 4.3. If all the choices of sign are good, $\mathfrak{G}$ can be evaluated to P bits of precision in $O(\log P)$ steps.

Finally, defining $\mathfrak{F}$ so that Newton's method is applicable requires finding 2^g-1 symplectic matrices M such that

$$\theta_j(M\cdot z, M\cdot\tau)^2 = f_M(\tau)e^{2i\pi g_M(z,\tau)}\theta_{\sigma_M(j)}(z,\tau)^2,$$

where f_M, g_M are rational functions and σ_M is a permutation. We then define the function $\mathfrak{F}$ with Algorithm 3.

Algorithm 3 Compute $\mathfrak{F}\ (\theta_{1,\dots,2^g-1}(z,\tau/2)/\theta_0(z,\tau/2), \theta_{1,\dots,2^g-1}(0,\tau/2)/\theta_0(0,\tau/2))$.

```
1: Compute $\frac{\theta_i(z,\tau)^2}{\theta_0(z,\tau/2)\theta_0(0,\tau/2)}, \frac{\theta_i(0,\tau)^2}{\theta_0(0,\tau/2)^2}$ using equation (3.5).
2: Compute $\mathfrak{G}\ \left(\frac{\theta_{0,\dots,2^g-1}(z,\tau)^2}{\theta_0(z,\tau/2)\theta_0(0,\tau/2)}, \frac{\theta_{0,\dots,2^g-1}(0,\tau)^2}{\theta_0(0,\tau/2)^2}\right) = \left(\frac{1}{\theta_0(z,\tau/2)\theta_0(0,\tau/2)}, \frac{1}{\theta_0(0,\tau/2)^2}\right)$.
3: for $M$ within a set of $2^g-1$ well-chosen symplectic matrices do
4:     Set $\sigma_M$ as in Theorem 2.7 (for example, [13, Chapter 5, Theorem 2]).
5:     Compute $\lambda_M, \mu_M = \mathfrak{G}\ (\theta_{\sigma_M(0),\dots,\sigma_M(2^g-1)}(z,\tau)^2, \theta_{\sigma_M(0),\dots,\sigma_M(2^g-1)}(0,\tau)^2)$.
6: return $(\lambda_M, \mu_M)$.
```

4.4.2. *The final algorithm.* We can compute λ_i, μ_i from z, τ using equation (2.1) in $O(2^g \mathcal{M}(P) \log P)$ operations. We can then apply Newton's method to $\mathfrak{F}$ to compute the quotients of theta functions and theta constants, but only if the Jacobian of the system is invertible. We conjecture that one can choose matrices M to ensure this, as in genus one and genus two.

The total complexity of this method is the same as the complexity of evaluating $\mathfrak{F}$, 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 was the case between genus one and genus two, we expect the precision for which our algorithm is better than a naive approach to be smaller as the genus grows.

Appendix A. *Naive algorithm in genus two*

Algorithm A.1 computes $\theta(z, \tau)$ in genus two using the partial evaluation of the series and induction relations (3.3) and (3.4) to speed up the computation of each term.

Algorithm A.1 Naive algorithm for $\theta(z, \tau)$ in genus two.
Input: z, τ and B a summation bound. **Output**: $\theta_i(z, \tau), \theta_i(0, \tau)$ for $i \in \{0..3\}$.

```
1:  $a \leftarrow (1,1,1,1)$; $b \leftarrow (1,1,1,1)$        ▷ These arrays will store $\theta_i(z,\tau)$ and $\theta_i(0,\tau)$.
2:  Compute $R(m,n) = Q(m,n) + Q(-m,n)$ for $m,n \in \{0,1\}$.
3:  Compute $T(m,n)$ for $m \in \{0,1\}$ and $n \in \{-1,0,1\}$.
4:  Add contributions for $(0,1)$ to $a$ and $b$, with the correct sign.
5:  $u \leftarrow w_1^2 + w_1^{-2}$, $v \leftarrow w_2^2 + w_2^{-2}$
6:  for $n = 1$ to $B-1$ do
7:      $R(0,n+1) \leftarrow q_2^{2n+1} R(0,n)$
8:      Add contribution for $(0,n+1)$ to $b$, with the correct sign.
9:      $T(0,n+1) \leftarrow q_2^{2n+1} v T(0,n) - q_2^{4n} T(0,n-1)$
10:     Add contributions for $(0,n+1)$ to $a$, with the correct sign.
11: for $m = 1$ to $B$ do
12:     $\rho_m \leftarrow q_3^{2m} + q_3^{-2m}$        ▷ can be computed inductively
13:     Add contributions for $(m,0)$ and $(m,1)$ to $a$ and $b$, with the correct sign.
14:     for $n = 1$ to $B-1$ do        ▷ One may refine the bound depending on $m$
15:         $R(m,n+1) \leftarrow q_2^{2n+1} \rho_m R(m,n) - q_2^{4n} R(m,n-1)$
16:         Add contribution for $(m,n+1)$ to $b$, with the correct sign.
17:         $T(m,n+1) \leftarrow q_2^{2n+1} q_3^{2m} v T(m,n) - q_2^{4n} q_3^{4m} T(m,n-1)$
18:         $T(m,-(n+1)) \leftarrow q_2^{2n+1} q_3^{-2m} v T(m,-n) - q_2^{4n} q_3^{-4m} T(m,-(n-1))$
19:         Add contributions for $(m,n+1)$ to $a$, with the correct sign.
20:     $R(m+1,0) \leftarrow q_1^{2m+1} R(m,0)$
21:     $R(m+1,1) \leftarrow q_1^{2m+1} (q_3^2 + q_3^{-2}) R(m,1) - q_1^{4m} R(m-1,1)$
22:     $T(m+1,0) \leftarrow q_1^{2m+1} u T(m,0) - q_1^{4m} T(m-1,0)$
23:     $T(m+1,1) \leftarrow q_1^{2m+1} q_3^2 u T(m,1) - q_1^{4m} q_3^4 T(m-1,1)$
24:     $T(m+1,-1) \leftarrow q_1^{2m+1} q_3^{-2} u T(m,-1) - q_1^{4m} q_3^{-4} T(m-1,-1)$
25: return $a$, $b$.
```

In this algorithm, terms of the form q_i^k as well as products thereof must also be computed by induction. We did so in our implementation, but this is deliberately omitted here for brevity. Also, despite the use of notation $T(m, n)$, 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.

Appendix B. *Generic implementation of $\mathfrak{F}$*

Algorithm 1 defines $\mathfrak{F}$ in terms of quotients of theta functions for clarity. Algorithm B.1 below shows how the function $\mathfrak{F}$ is implemented generically, that is, how it operates on general arguments.

Algorithm B.1 Computation of $\mathfrak{F}(a_{1,2,3}, b_{1,2,3})$.
Input: $a_{1,2,3}, b_{1,2,3} \in \mathbb{C}^6$ and a pair $z, \tau \in \mathbb{C}^2 \times \mathcal{H}_2$.
We assume that $a_{1,2,3}, b_{1,2,3}$ belong to a narrow enough neighborhood of $\left(\frac{\theta_{1,2,3}}{\theta_0}(z, \tau/2), \frac{\theta_{1,2,3}}{\theta_0}(0, \tau/2)\right)$, to some constant base precision P_0. These coarse estimates serve as a guide to choose the correct signs of square roots when computing $\mathfrak{G}$.

```
1: $(a_0, b_0) \leftarrow (1, 1)$.
2: **for** i = 0 to 15 **do**
3:     $v \leftarrow i \pmod 4$, $u = \frac{i-v}{4}$.
4:     $x_i \leftarrow \frac{1}{4}\sum_{j=0}^{3}(-1)^{u\cdot j} a_{v\oplus j} b_j$.
5:     $y_i \leftarrow \frac{1}{4}\sum_{j=0}^{3}(-1)^{a\cdot j} b_{b\oplus j} b_j$.
6: $(\lambda_0, \mu_0) \leftarrow \mathfrak{G}(x_{0,1,2,3}, y_{0,1,2,3})$,        ▷ guide with low-precision approximations
7: $(x_{0,\ldots,15}, y_{0,\ldots,15}) \leftarrow (\frac{1}{\lambda_0} x_{0,\ldots,15}, \frac{1}{\mu_0} y_{0,\ldots,15})$.
8: $(\lambda_1, \mu_1) \leftarrow \mathfrak{G}(x_{8,9,0,1}, y_{8,9,0,1})$,        ▷ guide with low-precision approximations
9: $(\lambda_2, \mu_2) \leftarrow \mathfrak{G}(x_{4,0,6,2}, y_{4,0,6,2})$,        ▷ guide with low-precision approximations
10: $(\lambda_3, \mu_3) \leftarrow \mathfrak{G}(x_{0,8,4,12}, y_{0,8,4,12})$,     ▷ guide with low-precision approximations
11: **return** $(\mu_1/\lambda_1, \mu_2/\lambda_2, \mu_3/\lambda_3, -1/\mu_1, -1/\mu_2, 1/\mu_3)$.
```

Acknowledgement. We thank the anonymous reviewers for the high quality of their feedback.

References

1. R. Cosset, 'Applications des fonctions thêta à la cryptographie sur courbes hyperelliptiques', PhD Thesis, Université Henri Poincaré-Nancy I, 2011.

2. D. A. Cox, 'The arithmetic-geometric mean of Gauss', *Enseign. Math.* 30 (1984) no. 2, 275–330.

3. B. Deconinck, M. Heil, A. Bobenko, M. Van Hoeij and M. Schmies, 'Computing Riemann theta functions', *Math. Comp.* 73 (2004) no. 247, 1417–1442.

4. R. Dupont, 'Moyenne arithmético-géométrique, suites de Borchardt et applications', PhD Thesis, École polytechnique, Palaiseau, 2006, http://www.lix.polytechnique.fr/Labo/Regis.Dupont/these_soutenance.pdf.

5. R. Dupont, 'Fast evaluation of modular functions using Newton iterations and the AGM', *Math. Comp.* 80 (2011) no. 275, 1823–1847.

6. A. Enge, 'The complexity of class polynomial computation via floating point approximations', *Math. Comp.* 78 (2009) no. 266, 1089–1107.

7. A. Enge, M. Gastineau, P. Théveny and P. Zimmerman, 'GNU MPC. INRIA, September 2012. Release 1.0.1', http://mpc.multiprecision.org/.
8. A. Enge and E. Thomé, 'CMH — Computation of Igusa Class Polynomials, Version 1.0', 2014, http://cmh.gforge.inria.fr/.
9. A. Enge and E. Thomé, 'Computing class polynomials for abelian surfaces', *Exp. Math.* 23 (2014) no. 2, 129–145.
10. P. Gaudry, 'Fast genus 2 arithmetic based on theta functions', *J. Math. Cryptol.* 1 (2007) no. 3, 243–265.
11. E. Gottschling, 'Explizite bestimmung der randflächen des fundamentalbereiches der modulgruppe zweiten grades', *Math. Ann.* 138 (1959) no. 2, 103–124.
12. B. Helfrich, 'Algorithms to construct Minkowski reduced and Hermite reduced lattice bases', *Theoret. Comput. Sci.* 41 (1985) 125–139.
13. J.-I. Igusa, *Theta functions* (Springer, Berlin, Heidelberg, 1972).
14. H. Klingen, *Introductory lectures on Siegel modular forms* (Cambridge University Press, Cambridge, 1990).
15. H. Labrande, 'Computing Jacobi's θ in quasi-linear time', Preprint, 2015, arXiv:1511.04248 [math.NT].
16. D. Mumford, *Tata lectures on theta*, vol. I (Birkhäuser, Boston, 1983).
17. M. Streng, 'Computing Igusa class polynomials', *Math. Comp.* 83 (2014) no. 285.
18. P. Van Wamelen, 'Equations for the Jacobian of a hyperelliptic curve', *Trans. Amer. Math. Soc.* 350 (1998) no. 8, 3083–3106.

Hugo Labrande
Université de Lorraine
LORIA (UMR CNRS 7503)
INRIA Nancy
615 rue du jardin botanique
54602 Villers-lès-Nancy Cedex
France

and
University of Calgary
Department of Computer Science
2500 University Dr NW
Calgary, Alberta
Canada T2N 1N4

hugo.labrande@inria.fr

Emmanuel Thomé
Université de Lorraine
LORIA (UMR CNRS 7503)
INRIA Nancy
615 rue du jardin botanique
54602 Villers-lès-Nancy Cedex
France

emmanuel.thome@inria.fr