LMS J. Comput. Math. 19 (Special issue A) (2016) 220–234 © 2016 Authors
doi:10.1112/S1461157016000383

Computing L-series of geometrically hyperelliptic curves of genus three

David Harvey, Maike Massierer and Andrew V. Sutherland

Abstract

Let $C/\mathbf{Q}$ be a curve of genus three, given as a double cover of a plane conic. Such a curve is hyperelliptic over the algebraic closure of $\mathbf{Q}$, but may not have a hyperelliptic model of the usual form over $\mathbf{Q}$. We describe an algorithm that computes the local zeta functions of C at all odd primes of good reduction up to a prescribed bound N. The algorithm relies on an adaptation of the 'accumulating remainder tree' to matrices with entries in a quadratic field. We report on an implementation and compare its performance to previous algorithms for the ordinary hyperelliptic case.

1. *Introduction*

Let $C/\mathbf{Q}$ be a curve of genus three. For an odd prime p of good reduction for C, let C_p denote the reduction of C modulo p. The zeta function of C_p is defined by

$$Z_p(T) := \exp\left(\sum_{k=1}^{\infty} \frac{\#C_p(\mathbf{F}_{p^k})}{k} T^k\right) = \frac{L_p(T)}{(1-T)(1-pT)}. \tag{1.1}$$

By the Weil Conjectures for curves, the numerator is of the form

$$L_p(T) = 1 + a_1T + a_2T^2 + a_3T^3 + pa_2T^4 + p^2a_1T^5 + p^3T^6 \in \mathbf{Z}[T],$$

and has reciprocal roots of complex absolute value $p^{1/2}$. In this paper, we are interested in algorithms for computing $Z_p(T)$ for all good primes p up to a prescribed bound N.

A closely related problem is to compute the first N terms of the L-series associated to C. This is defined by formally expanding the Euler product

$$L(C, s) = \prod_p L_p(p^{-s})^{-1} = \sum_{n \geqslant 1} c_n n^{-s}.$$

To compute c_n for all $n < N$, one must compute a_1 for all $p < N$, but one needs a_2 only for $p < N^{1/2}$ and a_3 only for $p < N^{1/3}$. Note that, for the primes of bad reduction, $L_p(T)$ is not necessarily given by (1.1) and must be computed by other means; this problem is not addressed in this paper.

Curves of genus three over $\mathbf{Q}$ come in two flavors, depending on the behavior of the canonical embedding $\phi : C \to \mathbf{P}^2$. The first possibility is that ϕ is a two-to-one cover of a plane conic Q, in which case C is geometrically hyperelliptic. Otherwise, ϕ is an isomorphism from C to a smooth plane quartic defined over $\mathbf{Q}$, and we are in the non-hyperelliptic case.

Received 22 February 2016.

2010 Mathematics Subject Classification 11G20 (primary), 11Y16, 11M38, 14G10 (secondary).

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

The first and second authors were supported by the Australian Research Council (DP150101689). The third author was supported by NSF grants DMS-1115455 and DMS-1522526.

https://doi.org/10.1112/S1461157016000383 Published online by Cambridge University Press

In the hyperelliptic case, if Q possesses a $\mathbf{Q}$-rational point, then Q is isomorphic to $\mathbf{P}^1$ over $\mathbf{Q}$ and this yields a model for C of the form $y^2 = h(x)$, with $h \in \mathbf{Z}[x]$. For such curves, the first author proposed an algorithm that computes $L_p(T)$ for all odd $p < N$ using a total of $N(\log N)^{3+o(1)}$ bit operations [19]. The average time per prime is thus $(\log N)^{4+o(1)}$. Although that algorithm has not been implemented in full generality, the first and third authors [22, 23] have developed a simpler and closely related algorithm for computing the Hasse–Witt matrices W_p of the reductions modulo primes $p < N$ of a fixed hyperelliptic curve $y^2 = h(x)$ in average time $(\log N)^{4+o(1)}$ per prime; in practice, for curves of genus two and three this yields enough information to quickly deduce the local zeta functions. This implementation outperforms existing packages, based on older algorithms, by several orders of magnitude. An analogous algorithm for the non-hyperelliptic genus three case is currently under development and will be presented in a forthcoming paper.

The main contribution of this paper is an analogue of the algorithm of [23] for the geometrically hyperelliptic case, without the assumption that $Q(\mathbf{Q}) \neq \emptyset$. The new algorithm takes as input an integer N and homogeneous polynomials $f, g \in \mathbf{Z}[X, Y, Z]$, with $\deg g = 2$ and $\deg f = 4$, specifying the curve

$$g(X, Y, Z) = 0, \quad w^2 = f(X, Y, Z). \tag{1.2}$$

The equation $g = 0$ defines the conic Q and $w^2 = f$ describes the two-to-one cover. The output of the algorithm is the sequence of polynomials $L_p(T)$ for all good primes $p < N$.

The new algorithm is mainly intended for use in the case where $Q(\mathbf{Q}) = \emptyset$. However, the algorithm works perfectly well if $Q(\mathbf{Q}) \neq \emptyset$; this may be useful, for example, if a $\mathbf{Q}$-rational point on Q exists but cannot be determined efficiently due to the difficulty of factoring the discriminant of g. Of course, if a $\mathbf{Q}$-rational point is known, then it may be profitable to switch to a standard hyperelliptic model $y^2 = h(x)$ and apply the algorithm of [23] instead.

Our focus is on designing a *practical* algorithm: we want to actually compute local zeta functions on real hardware for values of N that are as large as is practical. From a theoretical point of view, the existence of a complexity bound analogous to [19] for curves of the type (1.2) was essentially demonstrated by the first author in [20]. The result may be stated as follows. For any polynomial F with integer coefficients, we denote by $\|F\|$ the maximum of the absolute values of the coefficients of F.

THEOREM 1.1. *There exists an explicit deterministic algorithm with the following properties. The input consists of an integer $N \geqslant 2$ and polynomials f and g describing a genus three curve C as in (1.2). The output is the collection of $L_p(T)$ associated to C for all good primes $p < N$. The algorithm runs in $N \log^2 N \log^{1+o(1)}(N\|f\|\|g\|)$ bit operations.*

We omit the details of the proof. Ignoring the dependence on $\|f\|\|g\|$, the complexity bound is a special case of [20, Theorem 1.1], which applies to *any* fixed variety over $\mathbf{Z}$. To get the right dependence on $\|f\|\|g\|$, one may invoke [20, Theorem 1.4] and apply the 'inclusion–exclusion trick' of [29, § 3] (see the proof of [20, Theorem 1.1] for a similar argument). The difficulty with the algorithm just sketched is that the implied big-O constant is enormous, essentially because the algorithms of [20] are designed for maximum possible generality. To obtain a practical algorithm, we must exploit the geometry of the situation at hand.

Our strategy is motivated by the following observation. If we only want to compute $L_p(T)$ for a *single* prime p, we may start by finding some $\mathbf{F}_p$-rational point on the conic (such a point exists for all odd p). This leads to a rational parametrization for the conic over $\mathbf{F}_p$ and hence a model for C_p of the form $y^2 = h(x)$ over $\mathbf{F}_p$. We may then apply any of the known point-counting algorithms for hyperelliptic curves over finite fields. To mount a global attack along these lines, we must somehow choose these $\mathbf{F}_p$-rational points 'coherently' as p varies. This cannot be done over $\mathbf{Q}$, because we are expressly avoiding any assumptions about

https://doi.org/10.1112/S1461157016000383 Published online by Cambridge University Press

$\mathbf{Q}$-rational points on the conic. On the other hand, it is easy to construct a quadratic extension $K = \mathbf{Q}(\sqrt{D})$ for which Q has K-rational points. We may then parametrize Q over K to obtain a model $y^2 = h(x)$ of a hyperelliptic curve C'/K that is isomorphic to the base change of C to K, where $h \in \mathcal{O}_K[x]$. Of course, the curve C' is not isomorphic to the original curve over $\mathbf{Q}$ (it is not even defined over $\mathbf{Q}$), but it nevertheless retains much arithmetic information about the original curve.

This is exactly the approach we take in this paper. We start in § 2 by explaining how to construct an appropriate field K and a model $y^2 = h(x)$ for C' over K. In § 3, we set up recurrences for computing coefficients of powers of $h(x)$, analogous to [23], and, in § 4, we show how to solve these recurrences efficiently by means of an 'accumulating remainder tree' for matrices defined over a quadratic field. Section 5 applies these techniques to the problem of computing the Hasse–Witt matrices associated to C', which, in turn, leads in § 6 to information about $L_p(T) \pmod{p}$. Finally, to pin down $L_p(T) \in \mathbf{Z}[T]$, we perform a baby-steps giant-steps search in the Jacobian of the curve; this is discussed in § 7. Section 8 presents a complete statement of the algorithm and the last section reports on an implementation and gives some performance data.

We will not give a formal complexity analysis of the algorithm; instead, we will discuss complexity issues as they arise, with an eye towards practical computations. From an asymptotic perspective, our algorithm to compute $L_p(T) \in \mathbf{Z}[T]$ does *not* run in average polynomial time, because the lifting step (see § 7) uses $p^{1/4+o(1)}$ bit operations per prime. Nevertheless, as demonstrated by the timings in § 9, the cost of the lifting step is negligible over the range of our experiments and, by extrapolation, over the range of all currently feasible computations. Moreover, the lifting step is trivially parallelizable (the rest of the algorithm is not), so this is unlikely to ever be a problem in practice.

There are two main applications of this new class of 'average polynomial time' algorithms. The first is the investigation of higher-genus variants of the Sato–Tate conjecture. The original Sato–Tate conjecture proposed that, for a fixed elliptic curve over $\mathbf{Q}$, the distribution of the polynomials $L_p(T)$ (suitably normalized) obeys a particular statistical law when sampled over increasing values of p. This is now a theorem thanks to work of Richard Taylor and collaborators [**4**, **16**, **39**], but analogues for curves of higher genus remain open. The last few years have seen significant progress on understanding the details of the genus two case [**9**, **14**, **24**, **25**, **27**], and attention is now shifting to genus three [**10**, **28**]. Briefly, the role of these algorithms is to assist in identifying potential candidate curves possessing certain Sato–Tate groups, by computing corresponding Sato–Tate statistics (moments of the sequence of normalized L-polynomials) for each of a large set of candidates. The algorithm described in the present paper will be used to investigate the possibility that certain Sato–Tate distributions in genus three are encountered only for curves of the form (1.2).

The second application is computing zeros and special values of L-functions to high precision; this played an important role in the recent addition of genus two curves to the L-functions and modular forms database (LMFDB) [**6**], as described in [**1**] (as noted above, this application also requires the Euler factors at primes of bad reduction).

Notation. We denote by $\mathsf{M}(s)$ the number of bit operations required to multiply s-bit integers. We may take $\mathsf{M}(s) = s(\log s)^{1+o(1)}$ [**12**, **21**, **33**]. As in [**23**], we assume that $\mathsf{M}(s)/(s \log s)$ is increasing and that the space complexity of s-bit integer multiplication is $O(s)$.

2. *Constructing a suitable quadratic field and hyperelliptic model*

Let C be a genus three curve over $\mathbf{Q}$, as in (1.2). The goal of this section is to construct an integer D, not a square and not divisible by four, and a squarefree polynomial $h \in O_K[x]$, where

https://doi.org/10.1112/S1461157016000383 Published online by Cambridge University Press

O_K is the ring of integers of $K = \mathbf{Q}(\sqrt{D})$, such that $y^2 = h(x)$ is a model for $C' = C \times_{\mathbf{Q}} K$ (the base change of C to K). Moreover, we require that $\deg h = 8$ and that $h(0) \neq 0$.

We assume that elements of $O_K = \mathbf{Z}[\alpha]$ are represented by pairs of integers corresponding to the coefficients of 1 and α, where $\alpha = \sqrt{D}$ if $D \equiv 2, 3 \pmod 4$, or $\alpha = \frac{1}{2}(1 + \sqrt{D})$ if $D \equiv 1 \pmod 4$. In our applications, we take D to be squarefree, but this is not strictly necessary.

Choose any line L in $\mathbf{P}^2$ defined over $\mathbf{Q}$, say, $X = 0$. The points of intersection of L and Q are defined over an extension $K = \mathbf{Q}(\sqrt{D})$ for some $D \in \mathbf{Z}$. Note that D is obtained as the discriminant (possibly adjusted by some square factor) of a quadratic equation obtained by solving $g = 0$ simultaneously with the equation of L. Let $P_0 \in Q(K)$ be one of the intersection points (of which there are at most two). Now take a second line L' in $\mathbf{P}^2$, also defined over $\mathbf{Q}$, which does not contain P_0. By projection from P_0, we obtain a K-rational parametrization of $Q(K)$ by the points of $L'(K)$. Taking x to be a coordinate for some affine piece of L', we may write the parametrization as $(\psi_1(x), \psi_2(x), \psi_3(x)) \in \mathbf{P}^2$, where the $\psi_i \in O_K[x]$ are polynomials of degree at most two. Our preliminary model for C' is then $y^2 = h(x)$, where $h(x) = f(\psi_1(x), \psi_2(x), \psi_3(x))$.

If D is a square, then $K = \mathbf{Q}$ and we have actually found a $\mathbf{Q}$-rational point on Q. In this case, we could now simply apply the algorithm of [23] to the equation $y^2 = h(x)$. For the remainder of the paper, we assume that D is not a square, so that $K/\mathbf{Q}$ is a quadratic extension (although, in fact, the algorithm still works, *mutatis mutandis*, for square D).

Clearly $\deg h \leqslant 8$. Note that C' is isomorphic to C over K so it must have genus three; hence $\deg h \geqslant 7$ and h is squarefree. It remains to enforce the conditions that $\deg h = 8$ and that $h(0) \neq 0$. If $h(0) = 0$, we may replace $h(x)$ by $h(x-c)$, where c is a small integer with $h(c) \neq 0$. If $\deg h = 7$, we can replace $h(x)$ by $x^8 h(1/x)$ and translate again. These transformations all correspond to birational maps. In this way, we obtain a model for C' with $\deg h = 8$ and $h(0) \neq 0$.

Note that the conditions $\deg h = 8$ and $h(0) \neq 0$ are imposed only to simplify the presentation later. From a complexity point of view, it is actually *better* to have $\deg h = 7$ or $h(0) = 0$, or both. These occur when the curve has Weierstrass points defined over K and, in these cases, we can work with smaller recurrence matrices in § 3 (see [23, § 6.2] for details). Our current implementation always assumes that $\deg h = 8$ and that $h(0) \neq 0$.

The running time of the main algorithm is quite sensitive to the bit size of the coefficients of h and, to some extent, the bit size of $|D|$. In the procedure described above, we have made no attempt to minimize these quantities. If this became a bottleneck, one could try changing variables to obtain a conic with smaller coefficients [8], and one can also attempt to reparametrize L' to minimize the coefficients of the resulting $h(x)$ [35]. We do not know if these methods would lead to optimal running times; this seems to be a difficult problem, because of the dependence of the hyperelliptic model on the choice of D. We suspect that, to obtain a truly optimal model, one would need to optimize D and $h(x)$ simultaneously. In any case, if we restrict our attention to certain very simple conics, then we can often write down parametrizations for which the bit sizes remain under control (see § 9 for an example). We expect that this will be sufficient for the application to the Sato–Tate conjecture.

3. *Recurrences for the hyperelliptic model*

Let $y^2 = h(x)$ be a model for C' over $K = \mathbf{Q}(\sqrt{D})$, as in § 2. For each odd prime p, we define a row vector $U_p \in (O_K/p)^3$ by

$$(U_p)_j := h_{p-j}^{(p-1)/2} \bmod p, \quad j \in \{1, 2, 3\}.$$

Here $h_{p-j}^{(p-1)/2}$ denotes the coefficient of x^{p-j} in $h^{(p-1)/2}$. Note that O_K/p is not necessarily a field, because p may split in K.

https://doi.org/10.1112/S1461157016000383 Published online by Cambridge University Press

These vectors are closely related to the Hasse–Witt matrices for C, which are, in turn, related to the local zeta functions. The exact relationship is discussed in §§ 5 and 6. In this section, we concentrate on the following problem: given a bound N, compute U_p for all odd $p < N$ (except for a small number of 'exceptional' primes, as indicated below).

Write $h(x) = h_0 + h_1 x + \ldots + h_8 x^8$, where $h_i \in O_K$, $h_0 \neq 0$. For each integer $k \geqslant 1$, define an 8×8 matrix M_k with entries in O_K by

$$M_k := \begin{bmatrix} 0 & \ldots & 0 & (8-2k)h_8 \\ 2kh_0 & \ldots & 0 & (7-2k)h_7 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \ldots & 2kh_0 & (1-2k)h_1 \end{bmatrix}.$$

Also, define the vector $V_0 = [0,0,0,0,0,0,0,1] \in (O_K)^8$.

Proposition 3.1. *Let p be an odd prime with $(h_0, p) = 1$. Then U_p is equal to the vector consisting of the last three entries (in reversed order) of the vector*

$$\frac{-1}{h_0^{(p-1)/2}} V_0 M_1 \ldots M_{p-1} \pmod{p}.$$

Proof. For $1 \leqslant k \leqslant p-1$, let $v_k := [h_{k-7}^{(p-1)/2}, \ldots, h_k^{(p-1)/2}] \in (O_K/p)^8$. Using exactly the same argument as in [23, § 2], one may show that v_k satisfies the recurrence

$$v_k = \frac{1}{2kh_0} v_{k-1} M_k \pmod{p}. \tag{3.1}$$

Iterating this recurrence yields

$$v_{p-1} = \frac{1}{(p-1)!(2h_0)^{p-1}} v_0 M_1 \ldots M_{p-1} \pmod{p}.$$

Since $2^{p-1}(p-1)! = -1 \pmod{p}$ and $v_0 = [0, \ldots, 0, (h_0)^{(p-1)/2}] \pmod{p}$, we have

$$v_{p-1} = \frac{-1}{h_0^{(p-1)/2}} V_0 M_1 \ldots M_{p-1} \pmod{p}.$$

The last three entries of v_{p-1} are precisely the entries of U_p. □

According to the proposition, the problem of computing U_p for all odd primes $p < N$, except those for which $(h_0, p) \neq 1$, reduces to the problem of computing $V_0 M_1 \ldots M_{p-1} \pmod{p}$ for all $p < N$. In § 4, we will explain how to efficiently compute products of this type; this step constitutes the bulk of the running time of the main algorithm.

4. *The accumulating remainder tree over a quadratic field*

The accumulating remainder tree is a computational technique that lies at the heart of all of the recent average polynomial time point-counting algorithms. The basic scalar version was introduced in [7] and it was generalized to integer matrices in [19]. In this section, we present a variant that works over the ring of integers of a quadratic field K.

We will use the same notation as in [23, § 3]. Let $b \geqslant 2$ and $r \geqslant 1$. Let $m_1, \ldots, m_{b-1}$ be a sequence of positive integers. Let $A_0, \ldots, A_{b-2}$ be a sequence of $r \times r$ matrices with entries

https://doi.org/10.1112/S1461157016000383 Published online by Cambridge University Press

in O_K and let V be an r-dimensional row vector with entries in O_K. The aim is to compute the sequence of reduced row vectors $C_1, \ldots, C_{b-1}$ defined by

$$C_n := VA_0 \ldots A_{n-1} \bmod m_n. \tag{4.1}$$

So far, this set-up is identical to [23], except that, in that paper, A_j and C_j had entries in $\mathbf{Z}$ rather than O_K.

To apply this to the situation in § 3, we set $r = 8$, $b = \lfloor N/2 \rfloor$, $A_j = M_{2j+1}M_{2j+2}$, $V = V_0$ and $m_n = 2n+1$ if $2n+1$ is prime, or 1 if not. Then, for any odd $p < N$, $C_{(p-1)/2} = V_0 M_1 \ldots M_{p-1} \pmod p$, from which we can read off the entries of U_p by Proposition 3.1 (provided that $(h_0, p) = 1$).

The naive algorithm for computing C_n, which separately computes each product $VA_0 \ldots A_{n-1}$ modulo m_n, leads to a running time bound that is quasi-quadratic in b. The accumulating remainder tree improves this to a quasi-linear bound. Pseudocode is given in Algorithm QuadraticRemainderTree below. For simplicity, we assume that $b = 2^\ell$ is a power of two, although this is not strictly necessary. The algorithm actually computes various intermediate quantities $m_{i,j}$, $A_{i,j}$ and $C_{i,j}$, where $0 \leqslant i \leqslant \ell$ and $0 \leqslant j < 2^i$ (see [23] for precise definitions); the output is obtained as $C_j = C_{\ell,j}$. For convenience we set $m_0 = 1$ and let A_{b-1} be the identity matrix.

Algorithm. QuadraticRemainderTree

```
Given $V, A_0, \ldots, A_{b-1}$, $m_0, \ldots, m_{b-1}$, with $b = 2^\ell$, compute $m_{i,j}$, $A_{i,j}$, $C_{i,j}$:
1. Set $m_{\ell,j} = m_j$ and $A_{\ell,j} = A_j$, for $0 \leqslant j < b$.
2. For $i$ from $\ell - 1$ down to 0:
       For $0 \leqslant j < 2^i$, set $m_{i,j} = m_{i+1,2j} m_{i+1,2j+1}$ and $A_{i,j} = A_{i+1,2j} A_{i+1,2j+1}$.
3. Set $C_{0,0} = V \bmod m_{0,0}$ and then for $i$ from 1 to $\ell$:
       For $0 \leqslant j < 2^i$ set $C_{i,j} = \begin{cases} C_{i-1,\lfloor j/2 \rfloor} \bmod m_{i,j} & \text{if } j \text{ is even,} \\ C_{i-1,\lfloor j/2 \rfloor} A_{i,j-1} \bmod m_{i,j} & \text{if } j \text{ is odd.} \end{cases}$
```

In fact, this pseudocode is copied verbatim from algorithm RemainderTree in [23]. The only difference between RemainderTree and QuadraticRemainderTree is the underlying data type; in RemainderTree, the objects $A_{i,j}$ and $C_{i,j}$ are defined over $\mathbf{Z}$, whereas, in QuadraticRemainderTree, they are defined over O_K. In all other respects, including the proof of correctness, the algorithms are identical.

The following theorem summarizes the performance characteristics of QuadraticRemainderTree. The bit size of an element of $O_K = \mathbf{Z}[\alpha]$ is defined to be the maximum of the bit sizes of the coefficients of 1 and α.

Theorem 4.1. *Let B be an upper bound for the bit size of $\prod_{j=0}^{b-1} m_j$, let B' be an upper bound for the bit size of any entry of V and let H be an upper bound for the bit size of any $m_0, \ldots, m_{b-1}$ and any entry of $A_0, \ldots, A_{b-1}$. Assume that $\log r = O(H)$ and that $r = O(\log b)$. The running time of the* QuadraticRemainderTree *algorithm is*

$$O(r^2 \mathsf{M}(B + bH) \log b + r\mathsf{M}(B')),$$

and its space complexity is $O(r^2(B + bH) \log b + rB')$.

Proof. The statement is identical to [23, Theorem 3.2]. The only difference in the analysis is that we must bound the cost of all operations over O_K instead of over $\mathbf{Z}$. We assume, for this discussion, that D is *fixed*; in our applications we arrange for D to be small, say, -1 or 2.

https://doi.org/10.1112/S1461157016000383 Published online by Cambridge University Press

The main operation to consider is computing the product of two $r \times r$ matrices, say, R and S, with entries in O_K. Write them as $R = R_0 + R_1\alpha$ and $S = S_0 + S_1\alpha$, where the R_i and S_i are integer matrices. In the $D \not\equiv 1 \pmod 4$ case, $RS = (R_0S_0 + DR_1S_1) + (R_0S_1 + R_0S_1)\alpha$. This clearly reduces to four matrix multiplications over $\mathbf{Z}$, plus several much cheaper operations (matrix additions and scalar multiplications by D). A similar formula holds for the $D \equiv 1 \pmod 4$ case and similar remarks apply to the matrix-vector multiplications in step 3.

Overall, we clearly lose only a constant factor compared with the analysis in [23]. □

Remark 1. One can greatly improve the space consumption (and to a lesser extent, the running time) of the accumulating remainder tree algorithm by utilizing the *remainder forest* technique, introduced in [22] (see also [23, Theorem 3.3]). The idea is to split the work into 2^κ subtrees, where $\kappa \in [0, \ell]$ is a parameter. This is important for practical computations, because QuadraticRemainderTree is extremely memory intensive.

4.1. *Practical considerations*

In practice, the running time of the main algorithm is dominated by the matrix-matrix and matrix-vector multiplications over $\mathbf{Z}[\alpha]$, so it is important to optimize this step.

Let us first recall the discussion in [23] for the case of matrices over $\mathbf{Z}$. For multiplying $r \times r$ integer matrices, the classical matrix multiplication algorithm requires r^3 integer multiplications. This is exactly what we do near the bottom of the tree, where the matrix entries are relatively small.

Further up the tree, when the matrix entries become sufficiently large, it becomes profitable to use integer multiplication algorithms based on fast Fourier transforms (FFTs). For example, the well-known GNU Multiple Precision (GMP) arithmetic library [15] will automatically switch to a variant of the Schönhage–Strassen algorithm for large enough multiplicands. However, this is inefficient because each matrix entry will be transformed r times. This redundancy can be eliminated by means of the following alternative algorithm: (1) transform each of the $2r^2$ matrix entries, then (2) multiply the matrices of Fourier coefficients and finally (3) perform an inverse transform on each of the r^2 entries of the target matrix. This strategy reduces the number of transforms from $3r^3$ to $3r^2$.

Unfortunately, in our implementation, we cannot carry out this plan using GMP, because GMP currently does not provide an interface to access the internals of its FFT representation. Moreover, the Schönhage–Strassen framework is not well suited to the matrix case, because the Fourier coefficients are relatively large. Instead, we implemented our own FFT based on number-theoretic transforms modulo word-sized primes (see [22, § 5.1]).

Asymptotically, for large matrix entries, we expect the running time to be dominated by the Fourier transforms, and so we expect a speed-up of a factor of about r compared to the classical algorithm. The measured speed-up is somewhat less than this, because of the contribution of step (2). For example, taking $r = 8$ and matrices with entries of 500 million bits, we observe a speed-up of around 6.4, rather than 8.

Turning now to $\mathbf{Z}[\alpha]$, the same principle applies. Suppose that $D \not\equiv 1 \pmod 4$ and that $R = R_0 + R_1\alpha$ and $S = S_0 + S_1\alpha$, where the R_i and S_i are integer matrices. We may write the product as $RS = (R_0S_0 + (DR_1)S_1) + (R_0S_1 + R_1S_0)\alpha$. We compute this as follows.

(1) Transform the entries of R_0, S_0, R_1, S_1 and also DR_1. There are $5r^2$ transforms here. Denote these by $T(R_0), \ldots, T(DR_1)$.

(2) Multiply the matrices of Fourier coefficients, to obtain $T(R_0)T(S_0)$, $T(DR_1)T(S_1)$, $T(R_0)T(S_1)$ and $T(R_1)T(S_0)$.

(3) Add the matrices of Fourier coefficients, to obtain $T(R_0)T(S_0) + T(DR_1)T(S_1)$ and $T(R_0)T(S_1) + T(R_1)T(S_0)$.

(4) Perform $2r^2$ inverse transforms to obtain the components of each entry of RS.

A similar discussion applies to the $D \equiv 1 \pmod 4$ case.

https://doi.org/10.1112/S1461157016000383 Published online by Cambridge University Press

Altogether we count $7r^2$ transforms, compared with $3r^2$ for the plain integer case. We thus expect the ratio of the cost of multiplying two matrices over $\mathbf{Z}[\alpha]$ to the cost of multiplying two matrices over $\mathbf{Z}$ to be about $7/3 \approx 2.33$, assuming inputs of the same bit size. The measured ratio is somewhat worse than this, mainly because of the non-negligible contribution of step (2). For example, with $r = 8$ and entries of 500 million bits, we observe a ratio of around 2.67.

One further optimization, which we did not pursue in our implementation, is to absorb the factor D directly into the transforms themselves. For example, if $D = -1$, the transform of DR_1 is just the negative of the transform of R_1, which we have already computed. This would reduce the number of transforms from $7r^2$ to $6r^2$. Unfortunately, this leads to technical complications for larger values of $|D|$, because the size of the Fourier coefficients needs to be increased to accommodate the extra factor of D. In the context of our 'small prime' FFTs, this optimization might be reasonable for very small $|D|$, but, in the interests of maintaining generality and simplicity of our code, we did not implement it.

5. *Computing the Hasse–Witt matrices*

We now return to the hyperelliptic model $y^2 = h(x)$ for C' over $K = \mathbf{Q}(\sqrt{D})$ that was constructed in § 2.

For each odd prime $p < N$, we select a prime ideal $\mathfrak{p}$ of K above p that we assume is unramified (we ignore the primes p that ramify in K). Now assume that $\mathfrak{p}$ does not divide the discriminant of $h(x)$, so that C' has good reduction at $\mathfrak{p}$. The *Hasse–Witt matrix* of C' at $\mathfrak{p}$ is the 3×3 matrix $W_\mathfrak{p}$ over $O_K/\mathfrak{p}$ with entries

$$(W_\mathfrak{p})_{i,j} = h_{pi-j}^{(p-1)/2} \bmod \mathfrak{p}, \quad i, j \in \{1, 2, 3\}.$$

There is a close relationship between $W_\mathfrak{p}$ and the local zeta function of the original curve C, which is discussed in § 6. In the remainder of this section, we explain how to compute the $W_\mathfrak{p}$.

Let $W_\mathfrak{p}^1$ denote the *first row* of $W_\mathfrak{p}$. By definition, $W_\mathfrak{p}^1$ is simply $U_p \pmod{\mathfrak{p}}$, where U_p is the vector defined in § 3. We may, therefore, compute $W_\mathfrak{p}^1$ for all $p < N$ by using QuadraticRemainderTree (§ 4) to compute U_p for all $p < N$ and then reduce each U_p modulo our chosen prime ideal $\mathfrak{p}$ for each prime.

To obtain the remaining rows of $W_\mathfrak{p}$, the most obvious approach is to continue iterating the recurrence of § 3 to reach v_{2p-1} and v_{3p-1} (in the notation of the proof of Proposition 3.1). This can be made to work, but there are technical difficulties: the factor of k in the denominator of (3.1) leads to divisions by p. Bostan, Gaudry and Schost deal with this by artificially introducing extra p-adic digits [**3**]. Instead, we will use the following trick, which was suggested in [**23**, § 5].

For each integer β, let $W_\mathfrak{p}(\beta)$ denote the Hasse–Witt matrix of the *translated* curve $y^2 = h(x + \beta)$ and let $W_\mathfrak{p}^1(\beta)$ denote its first row. The relation between $W_\mathfrak{p}$ and $W_\mathfrak{p}(\beta)$ is given by

$$W_\mathfrak{p}(\beta) = T(\beta) W_\mathfrak{p} T(-\beta), \tag{5.1}$$

where

$$T(\beta) = \begin{bmatrix} 1 & \beta & \beta^2 \\ 0 & 1 & 2\beta \\ 0 & 0 & 1 \end{bmatrix}.$$

For a proof, see [**23**, Theorem 5.1]. (Note that, in [**23**], we work over $\mathbf{F}_p$, whereas here we are possibly working over an extension, but this does not change the resulting formula, because β is a rational integer.)

https://doi.org/10.1112/S1461157016000383 Published online by Cambridge University Press

Now suppose that we have computed $W^1_{\mathfrak{p}}(\beta_i)$ for three integers $\beta_1, \beta_2, \beta_3$ and that we wish to deduce $W_{\mathfrak{p}}$. For each i, the equation $W_{\mathfrak{p}}(\beta_i) = T(\beta_i) W_{\mathfrak{p}} T(-\beta_i)$ yields a system of three linear equations in the nine unknown entries of $W_{\mathfrak{p}}$. We therefore have nine equations in nine unknowns, and the same argument as in [23, § 5] shows that this system has a unique solution, provided that β_1, β_2 and β_3 are distinct modulo p.

6. *Computing the L-polynomials modulo p*

At this stage, for each prime $p < N$ (except for various exceptional primes), we have computed $W_{\mathfrak{p}}$ for our chosen $\mathfrak{p}$ above p. In this section, we explain how this determines $L_p(T) \pmod p$ in the split case, and $L_p(T)L_p(-T) \pmod p$ in the inert case.

Consider the zeta function of C' at $\mathfrak{p}$. This is defined by

$$Z'_{\mathfrak{p}}(T) := \exp\left(\sum_{k=1}^{\infty} \frac{N_k}{k} T^k\right) = \frac{L'_{\mathfrak{p}}(T)}{(1-T)(1-qT)},$$

where N_k is the number of points on $C'_{\mathfrak{p}}$ (the reduction of C' modulo $\mathfrak{p}$) defined over the extension of $O_K/\mathfrak{p}$ of degree k. As before, $L'_{\mathfrak{p}}(T) \in \mathbf{Z}[T]$ has degree six.

In the split case, we simply have $L'_{\mathfrak{p}}(T) = \det(I - TW_{\mathfrak{p}}) \pmod{\mathfrak{p}}$. Thus $W_{\mathfrak{p}}$ determines $L'_{\mathfrak{p}}(T) \pmod p$. Moreover, since $C'_{\mathfrak{p}}$ is isomorphic to C_p over $O_K/\mathfrak{p} \cong \mathbf{Z}/p\mathbf{Z}$, they have the same zeta functions, so $L_p(T) = L'_{\mathfrak{p}}(T)$ (in $\mathbf{Z}[T]$). Hence $W_{\mathfrak{p}}$ determines $L_p(T) \pmod p$.

In the inert case, $L'_{\mathfrak{p}}(T) = \det(I - TW_{\mathfrak{p}}W^{(p)}_{\mathfrak{p}}) \pmod{\mathfrak{p}}$, where $W^{(p)}_{\mathfrak{p}}$ denotes the matrix obtained by applying the absolute Frobenius map to each entry of $W_{\mathfrak{p}}$, which raises each entry to the pth power. So again, in this case, $W_{\mathfrak{p}}$ determines $L_{\mathfrak{p}}(T) \pmod p$. Unfortunately, because of the base change from $\mathbf{Q}$ to K, we lose information when passing from C to C'; in effect, we have computed the zeta function of C_p over $\mathbf{F}_{p^2}$. All we can conclude is that $L_p(T)L_p(-T) = L'_{\mathfrak{p}}(T^2)$ (see [31, Chapter VIII, Lemma 5.12]), so $W_{\mathfrak{p}}$ determines only $L_p(T)L_p(-T) \pmod p$.

7. *Lifting the L-polynomials*

We now turn to the problem of determining $L_p(T) \in \mathbf{Z}[T]$, given as input either (1) $L_p(T) \pmod p$ (for p split in K) or (2) $L_p(T)L_p(-T) \pmod p$ (for p inert in K). Our approach to this problem utilizes generic group algorithms operating in $\mathrm{Jac}(C_p)(\mathbf{F}_p)$, the group of $\mathbf{F}_p$-rational points on the Jacobian variety of the reduction of C modulo p. It is a finite abelian group of order $p^3 + O(p^{5/2})$.

We first need a model for the curve that supports efficient arithmetic in $\mathrm{Jac}(C_p)(\mathbf{F}_p)$. We start with the reduction modulo p of the model given in (1.2). Although the conic $g = 0$ has no $\mathbf{Q}$-rational points, its reduction modulo p does have $\mathbf{F}_p$-rational points and therefore admits a rational parametrization that can be used to construct a hyperelliptic model $y^2 = h(x)$ with $h \in \mathbf{F}_p[x]$, as in § 2. The cost of constructing this model is negligible. Now, if C_p has a rational Weierstrass point, we move it to infinity and thus make $h(x)$ monic of degree seven; in this case, fast explicit formulas for arithmetic in $\mathrm{Jac}(C_p)(\mathbf{F}_p)$ are well known [5, § 14.6]. If C_p does not have a rational Weierstrass point, then, provided $p \geqslant 37$ (which we assume), it has a rational non-Weierstrass point P; moving this point to infinity, we obtain a model with $h(x)$ monic of degree eight. Fast explicit formulas for arithmetic in $\mathrm{Jac}(C_p)(\mathbf{F}_p)$ for such models have recently been developed [38], using the balanced divisor approach of [13, 32].

Case (1) is considered in [26], where it is noted that the problem of determining $L_p(T) \in \mathbf{Z}[T]$ given $L_p(T) \pmod p$ can be solved in $p^{1/4+o(1)}$ time (for a curve of genus three). Let us briefly recall how this is done.

https://doi.org/10.1112/S1461157016000383 Published online by Cambridge University Press

If $p \geqslant 149$, then $L_p(T) \pmod p$ uniquely determines the coefficient a_1 of $L_p(T)$. Indeed, from the Weil bounds, $|a_i| \leqslant \binom{6}{i} p^{i/2}$ for $i = 1, 2, 3$. This inequality constrains a_2 to at most $2\binom{6}{2} = 30$ values compatible with $a_2 \pmod p$. In fact, once a_1 is known, there are at most six possibilities for a_2; this follows from [26, Proposition 4]. For each of these six values of a_2, the pair (a_1, a_2) determines a set of at most $40p^{1/2}$ possible values of a_3, corresponding to an arithmetic progression modulo p. The pair (a_1, a_2) also determines corresponding arithmetic progressions modulo p in which the integers

$$\begin{aligned} \#\mathrm{Jac}(C_p)(\mathbf{F}_p) &= L_p(1) = (p^3+1) + (p^2+1)a_1 + (p+1)a_2 + a_3, \\ \#\mathrm{Jac}(\tilde{C}_p)(\mathbf{F}_p) &= L_p(-1) = (p^3+1) - (p^2+1)a_1 + (p+1)a_2 - a_3 \end{aligned} \tag{7.1}$$

must lie; here $\tilde{C}_p$ denotes a (non-trivial) quadratic twist of C_p.

Now, given any $\alpha \in \mathrm{Jac}(C_p)(\mathbf{F}_p)$ (or $\mathrm{Jac}(\tilde{C}_p)(\mathbf{F}_p)$), we may compute its order $|\alpha|$ as follows. First, apply a baby-steps giant-steps search to the appropriate arithmetic progression to obtain a multiple m of $|\alpha|$. Then factor m and use a polynomial-time fast order algorithm (see [36, Chapter 7]) to compute $|\alpha|$. The time to factor $m = O(p^3)$ is negligible compared with the cost of the baby-steps giant-steps search, both in theory [30] and in practice. Note that if our candidate value of a_2 is incorrect, we may not find such an m, in which case we discard this value of a_2 and proceed to the next of our (at most six) candidates. One of the candidates must work, and hence we can determine the order of α in $p^{1/4+o(1)}$ time. This applies more generally to any situation where we have $O(1)$ possible pairs (a_1, a_2) and we know the value of a_3 modulo p; this includes case (2), as we explain below (and also the case $p \leqslant 149$).

With the ability to compute the orders of arbitrary group elements, we obtain a Monte Carlo algorithm to compute the group exponent λ of $\mathrm{Jac}(C_p)(\mathbf{F}_p)$ in $p^{1/4+o(1)}$ time, via [36, Algorithm 8.1] (and, similarly, for $\mathrm{Jac}(\tilde{C}_p)(\mathbf{F}_p)$). The positive integer n output by this algorithm is guaranteed to divide λ and the probability that $n \neq \lambda$ can be made arbitrarily small, at an exponential rate. Note that this algorithm needs access to random elements of $\mathrm{Jac}(C_p)(\mathbf{F}_p)$; such elements may be found by picking random polynomials $u \in \mathbf{F}_p[x]$ with $\deg u \leqslant g = 3$ and attempting to construct the Mumford representation $[u(x), v(x)]$ of the affine part of a representative for a divisor class in $\mathrm{Jac}(C_p)(\mathbf{F}_p)$. This can be viewed as a generalization of the decompression technique described in [5, § 14.2].

As shown in [26, Proposition 4], given the group exponent λ, we can compute $\#\mathrm{Jac}(C_p)(\mathbf{F}_p)$ using the generic group algorithm in [36, Algorithm 9.1] in $p^{1/4+o(1)}$ time. The same applies to $\#\mathrm{Jac}(\tilde{C}_p)(\mathbf{F}_p)$; we can thus determine the values of both $L_p(1)$ and $L_p(-1)$, which is enough to determine $L_p(T)$. Indeed, adding the equations in (7.1) yields the value of a_2 and subtracting them and substituting a_1 yields a_3 (see [37, Lemma 4] for a more general result that applies whenever $p \geqslant 1600$).

The fact that we used a Monte Carlo algorithm to compute λ means that there is some (exponentially small) probability of error. We can eliminate this possibility by considering the set S of candidate values for $\#\mathrm{Jac}(C_p)$ that are both multiples of our divisor n of λ and are compatible with the constraints imposed by (7.1), the set of candidate pairs (a_1, a_2) and the value of a_3 modulo p. Typically, $|S| = 1$ and we immediately obtain a verified result. If not, any two candidates N_1 and N_2 for $\#\mathrm{Jac}(C_p)$ must differ in their ℓ-adic valuations for at least two primes ℓ (for $p > 30$, we cannot have N_1 divisible by N_2 or *vice versa*). By computing the group structure of the ℓ-Sylow subgroup H of the smaller of these two primes ℓ via [36, Algorithm 9.1] (a Monte Carlo algorithm that always outputs a subgroup of H), we may be able to provably rule out one of the candidates by obtaining a lower bound on the ℓ-adic valuation of $\#\mathrm{Jac}(C_p)(\mathbf{F}_p)$ that exceeds the ℓ-adic valuation of one of them. Provided $\ell = O(p^{1/2})$, this takes $p^{1/4+o(1)}$ time; we can also use $\#\mathrm{Jac}(\tilde{C}_p)$. In the computations described in § 9 this method was used to verify $L_p(T)$ in every case; we expect that one can prove that the

https://doi.org/10.1112/S1461157016000383 Published online by Cambridge University Press

complexity of this computation is bounded by $p^{1/4+o(1)}$ (at least on average), but we do not attempt this here. The timings listed in Table 1 include the (negligible) cost of this verification in the 'lift' columns.

In case (2), where we are given $L_p(T)L_p(-T) \pmod p$, there are at most eight possible values of $L_p(T) \pmod p$. To see this, let $\sum_{i=0}^{6} b_i T^{2i} = L_p(T)L_p(-T)$. One obtains the relations

$$b_1 \equiv 2a_2 - a_1^2 \pmod p, \quad b_2 \equiv a_2^2 - 2a_1a_3 \pmod p, \quad b_3 \equiv -a_3^2 \pmod p.$$

Given $b_1, b_2, b_3 \pmod p$, there are two possibilities for $a_3 \pmod p$, each of which determines a pair of quadratic equations in a_1 and a_2, which, in turn, has at most four solutions modulo p. Even though the value of a_1 is not uniquely determined in this case (no matter how big p is), we can apply the procedure described above to compute the orders of arbitrary elements of $\mathrm{Jac}(C_p)(\mathbf{F}_p)$ or $\mathrm{Jac}(\tilde{C}_p)(\mathbf{F}_p)$ in $p^{1/4+o(1)}$ time, and the rest of the discussion follows; the key point is that we have $O(1)$ arithmetic progressions of length $O(p^{1/2})$ in which $\#\mathrm{Jac}(C_p)(\mathbf{F}_p)$ and $\#\mathrm{Jac}(\tilde{C}_p)(\mathbf{F}_p)$ are known to lie.

8. *Summary of the algorithm*

We now describe the complete algorithm. The input consists of the polynomials f and g defining the curve C according to (1.2), a bound N and a parameter κ (see Remark 1). Our goal is to compute $L_p(T) \in \mathbf{Z}[T]$ for all odd primes $p < N$, except for a small number of exceptional primes, as documented below.

(1) Find a quadratic field $K = \mathbf{Q}(\sqrt{D})$ and a suitable model $y^2 = h(x)$ for C' over K, using (for example) the method of § 2.
Choose small integers $\beta_1, \beta_2, \beta_3$ so that $h(x + \beta_i) \neq 0$ for each i.

(2) Make a list of all odd primes $p < N$. Perform the following steps for each p:
  - If p satisfies any of the following conditions, declare p exceptional:
    * p divides D (ramified prime);
    * p divides some $\beta_i - \beta_j$;
    * p is not relatively prime to the discriminant of $h(x)$ (and hence of $h(x + \beta_i)$ for all i);
    * p is not relatively prime to the constant term of some $h(x + \beta_i)$.
  - Otherwise:
    * if $(D/p) = 1$ (split prime), pick a solution of $\gamma^2 = D \pmod p$ and let $\mathfrak{p} = (p, \gamma - \sqrt{D})$ be the corresponding prime ideal above p;
    * if $(D/p) = -1$ (inert prime), let $\mathfrak{p} = (p)$.

(3) Let $U_p(\beta_i)$ be the vector U_p (defined in § 3) corresponding to the translated curve $y^2 = h(x + \beta_i)$. Call QuadraticRemainderTree (or the 'forest' variant with parameter κ) three times, once for each translated curve, with parameters as specified in § 4, to compute $U_p(\beta_i)$ for all non-exceptional $p < N$.

(4) For each non-exceptional prime $p < N$:
  - reduce $U_p(\beta_i)$ modulo $\mathfrak{p}$ to obtain $W_{\mathfrak{p}}^1(\beta_i)$ for $i = 1, 2, 3$;
  - solve the system described at the end of § 5, using (5.1) to deduce $W_{\mathfrak{p}}$;
  - compute $\det(I - TW_{\mathfrak{p}})$ (split case) or $\det(I - TW_{\mathfrak{p}}W_{\mathfrak{p}}^{(p)})$ (inert case), to determine $L_p(T) \pmod p$ (split case) or $L_p(T)L_p(-T) \pmod p$ (inert case), according to § 6; and
  - apply the lifting procedure of § 7 to finally obtain $L_p(T) \in \mathbf{Z}[T]$.

As pointed out earlier, in practice, the running time is dominated by the calls to QuadraticRemainderTree. The exceptional primes (of good reduction) can be handled by any other suitable method; for example, naive point counting for the small exceptional

https://doi.org/10.1112/S1461157016000383 Published online by Cambridge University Press

TABLE 1. *Comparison of algorithms for computing $L_p(T)$ for $p < N$. See text for column explanations. Time in CPU seconds, space in gigabytes and all values rounded to three significant figures.*

	C_1			C_2			`hypellfrob`	
N	Time	Space	Lift	Time	Space	Lift	mod p	mod p^2
2^{16}	4	0.05	2	14	0.06	3	36	127
2^{17}	9	0.06	4	33	0.08	6	92	326
2^{18}	22	0.08	8	75	0.11	13	234	849
2^{19}	53	0.10	16	178	0.17	25	600	2 680
2^{20}	129	0.17	32	418	0.30	48	1 770	7 500
2^{21}	310	0.30	66	992	0.57	99	4 830	25 300
2^{22}	753	0.58	136	2 390	1.18	201	14 900	189 000
2^{23}	1 780	1.13	278	5 520	2.47	413	42 700	653 000
2^{24}	4 090	2.41	574	12 600	5.33	850	125 000	1 680 000
2^{25}	9 410	4.98	1 190	29 000	11.8	1 760	395 000	5 030 000
2^{26}	22 100	10.5	2 470	66 300	24.5	3 650	1 230 000	16 000 000
2^{27}	50 900	23.5	5 160	151 000	52.0	7 610	3 730 000	44 100 000
2^{28}	118 000	54.0	10 800	344 000	112	15 900	10 000 000	113 000 000
2^{29}	276 000	124	22 800	783 000	241	33 600	35 600 000	368 000 000
2^{30}	681 000	288	48 200	1980 000	480	71 100	97 100 000	948 000 000

primes and, for the larger ones, parametrizing the conic over $\mathbf{F}_p$ and then applying [18]. One can easily prove that the number of exceptional primes is small, and one can also prove that these primes make negligible overall contribution to the complexity. We omit the details.

9. *Implementation and performance*

We implemented most of the steps of the main algorithm in the C programming language, building on the implementation for the ordinary hyperelliptic case described in [23]. It uses the GMP library [15] for basic integer arithmetic and a customized FFT library for matrix arithmetic over O_K when the entries have large coefficients (see § 4.1 and [22, § 5.1]).

The program takes as input the original model for the curve C over $\mathbf{Q}$ and also the data describing the model C' over K, namely, the integer D and the polynomial $h \in O_K[x]$. The construction of C' itself is not yet fully automated; for this we use *ad hoc* methods, including Magma [2] and Sage [34] scripts. The output is the sequence of polynomials $L_p(T)$ for all $p < N$, except for a small number of exceptional primes (listed below). As pointed out earlier, it is not difficult to handle the missing primes of good reduction, but our implementation does not yet do this. For the application to the Sato–Tate problem, it is safe to ignore a small number of primes, because they have a negligible effect on the statistical data being collected, but, for the application to computing L-series, one would need to address the missing primes (including those of bad reduction, which is a problem we do not address here).

We give one numerical example to illustrate the performance of our implementation and compare it to the implementation for the ordinary hyperelliptic case from [23]. For the hyperelliptic case we take the curve C_1 defined by

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

For the new algorithm, we take C_2 to be the curve given by $w^2 = f(X, Y, Z)$, where

$$\begin{aligned} f(X,Y,Z) = X^4 - 2X^2Y^2 - 2Y^4 - X^3Z - 2X^2YZ - XY^2Z \\ - Y^3Z - X^2Z^2 - XYZ^2 - Y^2Z^2 + XZ^3 + Z^4, \end{aligned}$$

over the pointless conic $X^2 + Y^2 + Z^2 = 0$. We base extend to $K = \mathbf{Q}(i)$ with $i^2 = -1$ (so $D = -1$) and we parametrize the conic by $(\psi_1(x), \psi_2(x), \psi_3(x)) = (x^2 - 1, 2u, i(x^2 + 1))$. This leads to the curve C_2' over K given by the hyperelliptic equation $y^2 = h(x)$, where

$$\begin{aligned} h(x) = (3-2i)x^8 + (2-4i)x^7 + (-4-4i)x^6 + (2-4i)x^5 \\ + 2x^4 + (-2-4i)x^3 + (-4+4i)x^2 + (-2-4i)x + (3+2i). \end{aligned}$$

Note that the polynomial $f(X, Y, Z)$ was chosen carefully (by a random search) to ensure that the coefficients of $h(x)$ would not be too large.

We ran both programs to determine the zeta functions for C_1 and C_2 at all $p < N$ for various values of N. In both cases, we used the translates $\beta_i = i$ for $i = 0, 1, 2$. For C_1 the exceptional primes were 3, 5, 7, 19, 181, 931 781; for C_2, they were 3, 5, 7, 13, 31, 269, 10 169, 22 229. The computations were run on a single core of an otherwise idle 64-core 2.5 GHz Intel Xeon (E7-8867W v3) server with 1088 GB RAM, running Ubuntu Linux version 14.04. We used the GCC compiler, version 4.8.4 [11], with optimization flags `-O3 -funroll-loops`.

Performance figures are given in Table 1. We set the parameter κ (see Remark 1) to 7 in all our tests, a choice that optimized (or very nearly optimized) the running time in every case. The 'time' columns show the total running time, excluding the lifting phase, and the 'space' columns show the peak memory usage. The running time of the lifting phase is given in the 'lift' column; the memory usage is negligible for this phase.

The last two columns give estimates for the time to run `hypellfrob`, an implementation of the algorithm described in [18]. For a hyperelliptic curve of genus three, and for a given p-adic precision parameter $\alpha \geqslant 1$, it computes $L_p(T) \pmod{p^\alpha}$ in time $\alpha^{O(1)} p^{1/2+o(1)}$ for each p separately; prior to [22, 23], it was the fastest available software for this problem. The 'mod p^α' column, for $\alpha = 1, 2$, gives an estimate for the total time to compute $L_p(T) \pmod{p^\alpha}$ for all $p < N$. The estimates were obtained by sampling for several $p < N$ and extrapolating based on the number of primes in each interval. For $\alpha = 1$, this is enough to determine a_1 (provided $p \geqslant 149$) but not all of $L_p(T)$; one would still need to run a lifting step to obtain $L_p(T)$. For $\alpha = 2$, it determines $L_p(T)$ completely.

Note that `hypellfrob` is limited to curves with a rational Weierstrass point, so we used the curve $y^2 = x^7 + 3x^6 + 5x^5 + 7x^4 + 11x^3 + 13x^2 + 17x + 19$. For this reason the timings are not directly comparable with the columns for C_1 and C_2, but they still provide a reasonable indication of what should be expected. No implementation for the general case $y^2 = h(x)$ with $\deg h = 8$ is currently available; one could presumably be developed by adapting [17].

It is clear from Table 1 that, broadly speaking, the new algorithm performs similarly to its hyperelliptic antecedent [23]. In particular, the running time is close to linear in N. For the largest N in the table, we observe a slowdown from C_1 to C_2 by a factor of about 3. This is only slightly worse than the factor 2.33 that one expects asymptotically (see § 4.1). For $N = 2^{30}$, we see that the new algorithm is nearly 50 times faster than `hypellfrob`. As promised, the lifting phase makes a negligible overall contribution to the running time.

The memory footprint for C_2 is about twice that for C_1. This is exactly as expected, since the input coefficient sizes are roughly equal and, for C_2, we carry around twice as much information in each matrix (the coefficients of 1 and α).

An obvious disadvantage of the new algorithm is that it is more difficult to parallelize than `hypellfrob`. The latter is trivially parallelizable, by distributing primes among threads. In fact, there is some scope for parallelization in the new algorithm, but this is a rather involved question that will be deferred to a subsequent paper.

Acknowledgements. The authors thank Jesse Kass, Kiran Kedlaya, Christophe Ritzenthaler and John Voight for helpful conversations, and the referees for their comments that led to improvements in the presentation of these results.

https://doi.org/10.1112/S1461157016000383 Published online by Cambridge University Press

References

1. A. R. Booker, J. Sijsling, A. V. Sutherland, J. Voight and D. Yasaki, 'A database of genus-2 curves over the rational numbers', Algorithmic Number Theory 12th International Symposium (ANTS XII), *LMS J. Comput. Math.*, 19 (2016) special issue A, 235–254.
2. W. Bosma, J. Cannon and C. Playoust, 'The Magma algebra system. I. The user language', *J. Symbolic Comput.* 24 (1997) no. 3–4, 235–265; *Computational algebra and number theory* (London, 1993); MR 1484478.
3. A. Bostan, P. Gaudry and É. Schost, 'Linear recurrences with polynomial coefficients and application to integer factorization and Cartier–Manin operator', *SIAM J. Comput.* 36 (2007) no. 6, 1777–1806; MR 2299425 (2008a:11156).
4. L. Clozel, M. Harris and R. Taylor, 'Automorphy for some l-adic lifts of automorphic mod l Galois representations', *Publ. Math. Inst. Hautes Études Sci.* (2008) no. 108, 1–181; With Appendix A, summarizing unpublished work of Russ Mann, and Appendix B by Marie-France Vignéras; MR 2470687.
5. H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen and F. Vercauteren (eds), 'Handbook of elliptic and hyperelliptic curve cryptography', Discrete Mathematics and its Applications (Chapman & Hall/CRC, Boca Raton, FL, 2006); MR 2162716.
6. The LMFDB Collaboration, 'The L-functions and modular forms database', website http://www.lmfdb.org.
7. E. Costa, R. Gerbicz and D. Harvey, 'A search for Wilson primes', *Math. Comp.* 83 (2014) no. 290, 3071–3091; MR 3246824.
8. J. E. Cremona and D. Rusin, 'Efficient solution of rational conics', *Math. Comp.* 72 (2003) no. 243, 1417–1441; (electronic); MR 1972744.
9. F. Fité, K. S. Kedlaya, V. Rotger and A. V. Sutherland, 'Sato–Tate distributions and Galois endomorphism modules in genus 2', *Compositio Math.* 148 (2012) no. 5, 1390–1442; MR 2982436.
10. F. Fité and A. V. Sutherland, 'Sato–Tate groups of $y^2 = x^8 + c$ and $y^2 = x^7 - cx$', *Frobenius distributions: Lang–Trotter and Sato–Tate conjectures*, Contemporary Mathematics 663 (American Mathematical Society, Providence, RI, 2016) 103–126.
11. Free Software Foundation, 'GNU compiler collection, version 4.8.4', 2013, available at http://gcc.gnu.org/.
12. M. Fürer, 'Faster integer multiplication', *SIAM J. Comput.* 39 (2009) no. 3, 979–1005.
13. S. D. Galbraith, M. Harrison and D. J. Mireles Morales, 'Efficient hyperelliptic arithmetic using balanced representation for divisors', *Algorithmic Number Theory 8th International Symposium (ANTS VIII)*, Lecture Notes in Computational Science 5011 (Springer, Berlin, 2008) 342–356; MR 2467851.
14. J. González, 'The Frobenius traces distribution for modular Abelian surfaces', *Ramanujan J.* 33 (2014) no. 2, 247–261; MR 3165538.
15. T. Granlund and the GMP development team, 'GNU Multiple Precision Arithmetic Library, version 6.0', 2015, available at http://gmplib.org/.
16. M. Harris, N. Shepherd-Barron and R. Taylor, 'A family of Calabi–Yau varieties and potential automorphy', *Ann. of Math.* (2) 171 (2010) no. 2, 779–813; MR 2630056.
17. M. C. Harrison, 'An extension of Kedlaya's algorithm for hyperelliptic curves', *J. Symbolic Comput.* 47 (2012) no. 1, 89–101; MR 2854849.
18. D. Harvey, 'Kedlaya's algorithm in larger characteristic', *Int. Math. Res. Not. IMRN* (2007) no. 22, Art. ID rnm095, 29; MR 2376210 (2009d:11096).
19. D. Harvey, 'Counting points on hyperelliptic curves in average polynomial time', *Ann. of Math.* (2) 179 (2014) no. 2, 783–803; MR 3152945.
20. D. Harvey, 'Computing zeta functions of arithmetic schemes', *Proc. Lond. Math. Soc.* (3) 111 (2015) no. 6, 1379–1401; MR 3447797.
21. D. Harvey, G. Lecerf and J. van der Hoeven, 'Even faster integer multiplication', *J. Complexity* 36 (2016) 1–30, doi:10.1016/j.jco.2016.03.001.
22. D. Harvey and A. V. Sutherland, 'Computing Hasse–Witt matrices of hyperelliptic curves in average polynomial time', *Algorithmic Number Theory Eleventh International Symposium (ANTS XI)*, Vol. 17, London Mathematical Society Journal of Computation and Mathematics (2014) 257–273; MR 3240808.
23. D. Harvey and A. V. Sutherland, 'Computing Hasse–Witt matrices of hyperelliptic curves in average polynomial time, II', *Frobenius distributions: Lang–Trotter and Sato–Tate conjectures*, Contemporary Mathematics 663 (American Mathematical Society, Providence, RI, 2016) 127–148.
24. C. Johansson, 'On the Sato–Tate conjecture for non-generic abelian surfaces, with an appendix by Francesc Fité', *Trans. Amer. Math. Soc.* (2016) to appear, doi:10.1090/tran/6847.
25. N. M. Katz and P. Sarnak, *Random matrices, Frobenius eigenvalues, and monodromy*, American Mathematical Society Colloquium Publications 45 (American Mathematical Society, Providence, RI, 1999); MR 1659828.

https://doi.org/10.1112/S1461157016000383 Published online by Cambridge University Press

26. K. S. Kedlaya and A. V. Sutherland, 'Computing L-series of hyperelliptic curves', *Algorithmic Number Theory 8th International Symposium (ANTS VIII)*, Lecture Notes in Computational Science 5011 (Springer, Berlin, 2008) 312–326; MR 2467855.

27. K. S. Kedlaya and A. V. Sutherland, 'Hyperelliptic curves, L-polynomials, and random matrices', *Arithmetic, geometry, cryptography and coding theory*, Contemporary Mathematics 487 (American Mathematical Society, Providence, RI, 2009) 119–162; MR 2555991 (2011d:11154).

28. J.-C. Lario and A. Somoza, 'The Sato–Tate conjecture for a Picard curve with complex multiplication', Preprint, 2014, arXiv:1409.6020.

29. A. G. B. Lauder and D. Wan, 'Counting points on varieties over finite fields of small characteristic', *Algorithmic number theory: lattices, number fields, curves and cryptography*, Mathematical Sciences Research Institute Publications 44 (Cambridge University Press, Cambridge, 2008) 579–612; MR 2467558 (2009j:14029).

30. H. W. Lenstra Jr and C. Pomerance, 'A rigorous time bound for factoring integers', *J. Amer. Math. Soc.* 5 (1992) no. 3, 483–516; MR 1137100.

31. D. Lorenzini, *An invitation to arithmetic geometry*, Graduate Studies in Mathematics 9 (American Mathematical Society, Providence, RI, 1996); MR 1376367 (97e:14035).

32. D. J. M. Morales, 'Efficient arithmetic on hyperelliptic curves with real representation', PhD Thesis, Royal Holloway and Bedford New College, University of London, 2008, available at https://www.math.auckland.ac.nz/∼sgal018/Dave-Mireles-Full.pdf.

33. A. Schönhage and V. Strassen, 'Schnelle Multiplikation grosser Zahlen', *Computing (Arch. Elektron. Rechnen)* 7 (1971) 281–292; MR 0292344 (45 #1431).

34. W. A. Stein *et al.*, 'Sage Mathematics Software (Version 6.8)', The Sage Development Team, 2015, http://www.sagemath.org.

35. M. Stoll and J. E. Cremona, 'On the reduction theory of binary forms', *J. reine angew. Math.* 565 (2003) 79–99; MR 2024647.

36. A. V. Sutherland, 'Order computations in generic groups', PhD Thesis, Massachusetts Institute of Technology, 2007, available at http://groups.csail.mit.edu/cis/theses/sutherland-phd.pdf, MR 2717420.

37. A. V. Sutherland, 'A generic approach to searching for Jacobians', *Math. Comp.* 78 (2009) no. 265, 485–507; MR 2448717.

38. A. V. Sutherland, 'Fast Jacobian arithmetic for hyperelliptic curves of genus 3', Preprint, 2016, arXiv:1607.08602.

39. R. Taylor, 'Automorphy for some l-adic lifts of automorphic mod l Galois representations. II', *Publ. Math. Inst. Hautes Études Sci.* (2008) no. 108, 183–239; MR 2470688.

David Harvey
School of Mathematics and Statistics
University of New South Wales
Sydney NSW 2052
Australia

d.harvey@unsw.edu.au

Maike Massierer
School of Mathematics and Statistics
University of New South Wales
Sydney NSW 2052
Australia

maike@unsw.edu.au

Andrew V. Sutherland
Department of Mathematics
Massachusetts Institute of Technology
Cambridge MA 02139
USA

drew@math.mit.edu

https://doi.org/10.1112/S1461157016000383 Published online by Cambridge University Press