Contemporary Mathematics
Volume **663**, 2016
http://dx.doi.org/10.1090/conm/663/13352

Computing Hasse–Witt matrices of hyperelliptic curves in average polynomial time, II

David Harvey and Andrew V. Sutherland

Abstract. We present an algorithm that computes the Hasse–Witt matrix of a given hyperelliptic curve over $\mathbb{Q}$ at all primes of good reduction up to a given bound N. It is simpler and faster than the previous algorithm developed by the authors.

1. Introduction

Let $C/\mathbb{Q}$ be a (smooth projective) hyperelliptic curve of genus g defined by an affine equation of the form $y^2 = f(x)$, with $f = \sum_i f_i x^i \in \mathbb{Z}[x]$ squarefree. Primes p for which the reduced equation $y^2 = f(x) \bmod p$ defines a hyperelliptic curve $C_p/\mathbb{F}_p$ of genus g are *primes of good reduction* (for the equation $y^2 = f(x)$). For each such prime p, the *Hasse–Witt matrix* (or *Cartier–Manin matrix*) of C_p is the $g \times g$ matrix $W_p = [w_{ij}]$ over $\mathbb{Z}/p\mathbb{Z}$ with entries

$$w_{ij} = f_{pi-j}^{(p-1)/2} \bmod p \qquad (1 \le i, j \le g),$$

where f_k^n denotes the coefficient of x^k in $f(x)^n$; see [**14, 23**] for details. The matrix W_p depends on the equation $y^2 = f(x) \bmod p$ for the curve C_p, but its conjugacy class, and in particular, its characteristic polynomial, is an invariant of the function field of C_p.

The Hasse–Witt matrix W_p is closely related to the *zeta function*

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

Indeed, the numerator $L_p \in \mathbb{Z}[T]$ satisfies

$$L_p(T) \equiv \det(I - TW_p) \pmod p,$$

and we also have

$$\chi_p(T) \equiv (-1)^g T^g \det(W_p - TI) \pmod p,$$

where $\chi_p(T)$ denotes the characteristic polynomial of the Frobenius endomorphism of the Jacobian of C_p. In particular, the trace of W_p is equal to the trace of

---

2010 *Mathematics Subject Classification.* Primary 11G20; Secondary 11Y16, 11M38, 14G10.
The first author was supported by the Australian Research Council (DE120101293).
The second author was supported by NSF grant DMS-1115455.

Frobenius modulo p, and for $p > 16g^2$ the Riemann Hypothesis for curves implies that this relationship uniquely determines the trace of Frobenius.

In this paper we present an algorithm that takes as input the polynomial $f(x)$ and an integer $N > 1$, and simultaneously computes W_p for all primes $p \leq N$ of good reduction. Our main result is the complexity bound given in Theorem 1.1 below; the details of the algorithm are given in §4.

All time complexity bounds refer to bit complexity. We denote by $\mathsf{M}(s)$ the time needed to multiply s-bit integers; we may take $\mathsf{M}(s) = O(s\,(\log s)^{1+o(1)})$ (see [**4**, **17**], or [**9**] for recent improvements). We assume throughout that $\mathsf{M}(s)/(s \log s)$ is increasing, and that the space complexity of s-bit integer multiplication is $O(s)$.

THEOREM 1.1. *Assume that* $g \log g = O(\log N)$ *and* $\log \max_i |f_i| = O(\log N)$. *The algorithm* COMPUTEHASSEWITTMATRICES *computes* W_p *for all primes* $p \leq N$ *of good reduction in* $O(g^3\,\mathsf{M}(N \log N) \log N)$ *time and* $O(g^2 N)$ *space.*

The average running time of COMPUTEHASSEWITTMATRICES per $p \leq N$ is

$$O(g^3 (\log p)^{4+o(1)}),$$

which is polynomial in $g \log p$, the bit-size of the equation defining C_p. While it is known that one can compute the characteristic polynomial of W_p for any particular prime p in time polynomial in $\log p$ (the Schoof–Pila algorithm [**16**, **18**]), or polynomial in g (Kedlaya's algorithm [**12**], for example), we are aware of no algorithm that can compute even the trace of W_p in time that is polynomial in $g \log p$.

The algorithm presented here improves the previous algorithm given by the authors in [**10**], which was in turn based on the approach introduced in [**8**]. The new algorithm is easier to describe and implement, and it is significantly faster. Asymptotically we gain a factor of g^2 in the running time: one factor of g arises from genuine algorithmic improvements in the present paper, and another factor of g follows from unrelated recent work on the theoretical complexity of integer matrix multiplication [**11**]. The new algorithm also uses less memory by a significant constant factor. Tables 1 and 2 in §7 give performance comparisons in genus 2 and 3, where the the new algorithm is already up to 8 times faster. Compared to previous methods for solving this problem (i.e., prior to [**8**]), the new algorithm is dramatically faster, with more than a 300-fold speed advantage for $N = 2^{30}$; see Table 3. Performance results for hyperelliptic curves of genus $g \leq 10$ can be found in Table 4.

In addition to the average polynomial-time algorithm, we give an algorithm to compute W_p for a single prime $p \geq g$ that runs in $O(g^2 p\,\mathsf{M}(\log p))$ time. While the dependence on p is not asymptotically competitive with existing algorithms, it is easy to implement, and uses very little memory. The small constant factors in its complexity make it a good choice for small to medium values of p; see Table 5.

We also introduce techniques that may be of interest beyond the scope of our algorithmic applications. In particular, we show that the matrix W_p for a given curve may be derived from knowledge of just the *first* row of the matrices W_p corresponding to g isomorphic curves. Algorithmically, this has the advantage that we never need to go beyond the coefficient of x^{p-1} in the expansion of $f(x)^{((p-1)/2)}$ in order to compute W_p.

2. Recurrence relations

As above, let $C/\mathbb{Q}$ be a hyperelliptic curve of genus g. We may assume without loss of generality that C is defined by an equation of the form

$$y^2 = f(x) = \sum_{i=c}^{d} f_i x^i \qquad (f_i \in \mathbb{Z})$$

with $c \in \{0,1\}$, $d \in \{2g+1, 2g+2\}$ and $f_c f_d \neq 0$ (we have $c \leq 1$ because f is squarefree). It is convenient to normalize by taking $h(x) := f(x)/x^c$ and $r := d - c$. Then

$$h(x) = \sum_{i=0}^{r} h_i x^i$$

where $h_i = f_{i+c}$ for $i = 0, \ldots, r$, and $h_0 h_r \neq 0$.

We now derive a recurrence for the coefficients h_k^n of h^n, following the strategy of Bostan–Gaudry–Schost [**1**, §8]. The identities

$$h^{n+1} = h \cdot h^n \qquad \text{and} \qquad (h^{n+1})' = (n+1)h' \cdot h^n$$

yield the relations

$$h_k^{n+1} = \sum_{j=0}^{r} h_j h_{k-j}^n \qquad \text{and} \qquad k h_k^{n+1} = (n+1) \sum_{j=0}^{r} j h_j h_{k-j}^n.$$

Subtracting k times the first relation from the second and solving for h_k^n yields the recurrence

$$h_k^n = \frac{1}{k h_0} \sum_{j=1}^{r} ((n+1)j - k) h_j h_{k-j}^n, \tag{2}$$

which expresses h_k^n in terms of the previous r coefficients $h_{k-1}^n, \ldots, h_{k-r}^n$, for all $k > 0$.

The recurrence may be written in matrix form as follows. Define the integer row vector

$$v_k^n := [h_{k-r+1}^n, \ldots, h_k^n] \in \mathbb{Z}^r.$$

Then

$$v_k^n = \frac{1}{k h_0} v_{k-1}^n M_k^n,$$

where

$$M_k^n := \begin{bmatrix} 0 & \cdots & 0 & ((n+1)r - k)h_r \\ k h_0 & \cdots & 0 & ((n+1)(r-1) - k)h_{r-1} \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & k h_0 & (n+1-k)h_1 \end{bmatrix}. \tag{3}$$

Iterating the recurrence, we obtain an explicit formula for the mth term, for any $m \geq 0$:

$$v_m^n = \frac{1}{(h_0)^m m!} v_0^n M_1^n \cdots M_m^n.$$

Since the initial vector is simply $v_0^n = [0, \ldots, 0, (h_0)^n]$, we may rewrite this as

$$v_m^n = \frac{1}{(h_0)^{m-n} m!} V_0 M_1^n \cdots M_m^n, \tag{4}$$

where

$$V_0 := [0, \ldots, 0, 1] \in \mathbb{Z}^r.$$

Everything we have discussed so far holds over $\mathbb{Z}$. Now consider a prime p of good reduction that does not divide h_0, and let W_p be the Hasse–Witt matrix of $y^2 = f(x) \bmod p$, as in §1. Write W_p^i for the ith row of W_p. Specializing the above discussion to $n := (p-1)/2$, the entries of W_p^i are $w_{ij} = f_{pi-j}^n = h_{pi-j-cn}^n \bmod p$ for $j = 1, \ldots, g$. These are the last g entries, in reversed order, of the vector $v_{pi-cn-1}^n \bmod p$, which by (4) is equal to

$$\frac{1}{(h_0)^{(pi-cn-1)-n}(pi-cn-1)!} V_0 M_1^n \cdots M_{pi-cn-1}^n \pmod{p}. \tag{5}$$

Remark 2.1. Bostan, Gaudry and Schost [**1**] used a formula essentially equivalent to (5) to compute W_p, for a single prime p. Their innovation was to evaluate the matrix product using an improvement of the Chudnovsky–Chudnovsky method, leading to an overall complexity bound of $g^{O(1)}p^{1/2+o(1)}$ for computing W_p.

Remark 2.2. The power of p dividing the denominator $(pi-cn-1)!$ in (5) is at least p^{i-1}. In particular, if $i \geq 2$, the denominator is divisible by p. Thus, to compute the second and subsequent rows of W_p, the algorithm in [**1**] artificially lifts the input polynomial $f \in \mathbb{F}_p[x]$ to $(\mathbb{Z}/p^\lambda\mathbb{Z})[x]$ for a suitable $\lambda \geq 2$, and then works modulo p^λ throughout the computation, reducing modulo p at the end. In this paper we compute W_p by computing the first row of g conjugate Hasse–Witt matrices (as explained in §5), so we can work modulo p everywhere.

We now focus on W_p^1, the first row of W_p. Taking $i = 1$ in (5) and putting $e := 2 - c$, we find that W_p^1 is given by the last g entries of

$$v_{en}^n \equiv \frac{1}{(h_0)^{n(e-1)}(en)!} V_0 M_1^n \cdots M_{en}^n \pmod{p}. \tag{6}$$

To compute W_p^1, it suffices to evaluate the vector-matrix product $V_0 M_1^n \cdots M_{en}^n$ modulo p, since, having assumed $p \nmid h_0$, the denominator $(h_0)^{n(e-1)}(en)!$ is not divisible by p (note that $e \in \{1, 2\}$, so en is at most $p-1$).

Remark 2.3. The quantity $(h_0)^n = (h_0)^{(p-1)/2}$ is just the Legendre symbol $(h_0/p) = \pm 1$, and the denominator $(h_0/p)^{e-1}(en)!$ is always a fourth root of unity modulo p. Indeed, if $e = 2$ then $(en)! = (p-1)! \equiv -1 \pmod{p}$, by Wilson's theorem; if $e = 1$ then it is well known that $(en)! = ((p-1)/2)!$ is a fourth root of unity modulo p. However, we know of no easy way to determine *which* fourth root of unity occurs. For example, if $p > 3$ and $p \equiv 3 \pmod 4$, then $((p-1)/2)! \equiv \pm 1 \pmod{p}$, where the sign depends on the class number of $\mathbb{Q}(\sqrt{-p})$ modulo 4; see [**15**].

To evaluate (6) simultaneously for many primes, a crucial observation is that the matrix M_k^n becomes "independent of n" after reduction modulo $p = 2n+1$. More precisely, we have $2(n+1) = 1 \pmod{p}$, so $2M_k^n \equiv M_k \pmod{p}$ where

$$M_k := \begin{bmatrix} 0 & \cdots & 0 & (r-2k)h_r \\ 2kh_0 & \cdots & 0 & (r-1-2k)h_{r-1} \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 2kh_0 & (1-2k)h_1 \end{bmatrix}. \tag{7}$$

Note that M_k is defined over $\mathbb{Z}$, and, unlike M_k^n, it is independent of p. Multiplying (6) by 2^{en} yields the following lemma.

LEMMA 2.4. *The first row W_p^1 of W_p consists of the last g entries of the vector*

$$\frac{\left(\frac{2}{p}\right)^e}{\left(\frac{h_0}{p}\right)^{e-1}(en)!} V_0 M_1 \cdots M_{en} \pmod{p}. \tag{8}$$

In order to unify the indexing for the cases $e = 1$ and $e = 2$ we now define

$$M_k' := \begin{cases} M_k & e = 1, \\ M_{2k-1}M_{2k} & e = 2. \end{cases}$$

Then (8) becomes

$$\frac{\left(\frac{2}{p}\right)^e}{\left(\frac{h_0}{p}\right)^{e-1}(en)!} V_0 M_1' \cdots M_n' \pmod{p}. \tag{9}$$

In the next section we will recall how to use the *accumulating remainder tree* algorithm to evaluate the product $V_0 M_1' \cdots M_n' \pmod{p}$ for many primes p simultaneously.

REMARK 2.5. The key difference between this approach and that of [**10**] is that here we express v_k^n in terms of v_{k-1}^n, instead of expressing v_{2k}^k in terms of v_{2k-2}^{k-1}. In the terminology of [**8**], we have replaced "reduction towards zero" by "horizontal reduction". In both cases, the aim is to obtain recurrences whose defining matrices are independent of p, so that the machinery of accumulating remainder trees can be applied. In the original "reduction towards zero" method of [**10**], the desired independence follows from the choice of indexing, i.e., because the superscript and subscript in v_{2k}^k do not depend on p. In the new method, the superscript in v_k^n does depend on p, but *after reduction modulo p* the recurrence matrices turn out to be independent of p anyway. The new method may also be viewed as a special case of (and was inspired by) the "generic prime" device introduced in [**7**]; see for example the proof of [**7**, Prop. 4.6].

Computationally, the new approach has two main advantages. First, the matrices M_k are simpler, sparser, and have smaller coefficients than those in [**10**]. In fact, the formula for M_k in [**10**] was so complicated that even for genus 2 we did not write it out explicitly in the paper. Second, as pointed out earlier, none of the denominators kh_0 used to compute the first row of W_p are divisible by p, so we can work modulo p throughout; in [**10**] we needed to work modulo a large power of p (at least p^g) in order to handle powers of p appearing in the denominators.

3. Accumulating remainder trees

In this section we recall how the accumulating remainder tree algorithm works and sharpen some of the complexity bounds given in [**10**].

We follow the notation and framework of [**10**, §4]. Let $b \geq 2$ be an integer, and let $m_1, \ldots, m_{b-1}$ be a sequence of positive integer moduli. Let $A_0, \ldots, A_{b-2}$ be a sequence of $r \times r$ integer matrices, and let V be an r-dimensional integer row vector. We wish to compute the sequence of reduced row vectors $C_1, \ldots, C_{b-1}$ defined by

$$C_n := V A_0 \cdots A_{n-1} \bmod m_n.$$

For convenience, we define $m_0 := 1$, so C_0 is the zero vector, and we let A_{b-1} be the identity matrix. (To apply this to the recurrences in §2, we let $V = V_0$ and $A_k = M'_{k+1}$; note that the index k is shifted by one place.)

In [**10**, §4] we gave an algorithm RemainderTree to efficiently compute the vectors $C_1, \ldots, C_{b-1}$ simultaneously. For the reader's convenience we now recall some details of this algorithm. As in [**10**], for simplicity we assume that $b = 2^\ell$ is a power of 2, though this is not strictly necessary.

We work with complete binary trees of depth ℓ with nodes indexed by pairs (i, j) with $0 \le i \le \ell$ and $0 \le j < 2^i$. For each node we define the intermediate quantities

$$\begin{aligned} m_{i,j} &:= m_{j2^{\ell-i}} m_{j2^{\ell-i}+1} \cdots m_{(j+1)2^{\ell-i}-1}, \\ A_{i,j} &:= A_{j2^{\ell-i}} A_{j2^{\ell-i}+1} \cdots A_{(j+1)2^{\ell-i}-1}, \\ C_{i,j} &:= V A_{i,0} \cdots A_{i,j-1} \bmod m_{i,j}. \end{aligned} \tag{10}$$

The values $m_{i,j}$ and $A_{i,j}$ may be viewed as nodes in a *product tree*, in which each node is the product of its children, with leaves $m_j = m_{\ell,j}$ and $A_j = A_{\ell,j}$, for $0 \le j < b$. Each vector $C_{i,j}$ is the product of V and all the matrices $A_{i,j'}$ that are nodes on the same level and to the left of $A_{i,j}$, reduced modulo $m_{i,j}$. The following algorithm, copied verbatim from [**10**], computes the target values $C_j = C_{\ell,j}$.

Algorithm RemainderTree

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 $0 \le j < b$.
2. For i from $\ell - 1$ down to 0:
   For $0 \le j < 2^i$, 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 ℓ:
   For $0 \le j < 2^i$ set $C_{i,j} = \begin{cases} C_{i-1,\lfloor j/2 \rfloor} \bmod m_{i,j} & \text{if } j \text{ is even,} \\ C_{i-1,\lfloor j/2 \rfloor} A_{i,j-1} \bmod m_{i,j} & \text{if } j \text{ is odd.} \end{cases}$

A complexity analysis of this algorithm was given in [**10**, Theorem 4.1], closely following the argument of [**8**, Prop. 4]. The analysis assumed that classical matrix multiplication was used to compute the products $A_{i+1,2j} A_{i+1,2j+1}$ in step 2. More precisely, defining $\mathsf{M}_r(s)$ to be the cost (bit complexity) of multiplying $r \times r$ matrices with s-bit integer entries, it was assumed that $\mathsf{M}_r(s) = O(r^3\, \mathsf{M}(s))$. Of course, this can be improved to $\mathsf{M}_r(s) = O(r^\omega\, \mathsf{M}(s))$, where $\omega \le 3$ is any feasible exponent for matrix multiplication [**22**, Ch. 12]. For example, we can take $\omega = \log 7 / \log 2 \approx 2.807$ using Strassen's algorithm.

However, it was pointed out in [**10**] and [**8**] that one can do even better in practice, by reusing Fourier transforms of the matrix entries; heuristically this reduces the complexity to only $O(r^2\, \mathsf{M}(s))$ when s is large compared to r. Recently, a rigorous statement along these lines was established by van der Hoeven and the first author [**11**]. The following slightly weaker claim is enough for our purposes.

Lemma 3.1. *We have*

$$\mathsf{M}_r(s) = O(r^2\, \mathsf{M}(s) + r^\omega s (\log\log s)^2),$$

uniformly for r and s in the region $r = O(s)$.

PROOF. According to [**11**, Prop. 4] (which depends on our running hypothesis that $\mathsf{M}(s)/(s\log s)$ is increasing), in this region we have

$$\mathsf{M}_r(s) = O(r^2\,\mathsf{M}(s) + r^\omega (s/\log s)\,\mathsf{M}(\log s)).$$

The desired bound follows, since certainly $\mathsf{M}(n) = O(n\log^2 n)$. □

We can now prove the following strengthening of [**10**, Theorem 4.1].

THEOREM 3.2. *Let B be an upper bound on the bit-size of $\prod_{j=0}^{b-1} m_j$, let B' be an upper bound on the bit-size of any entry of V, and let H be an upper bound on the bit-size of any $m_0, \dots, m_{b-1}$ and any entry of $A_0, \dots, A_{b-1}$. Assume that $\log r = O(H)$ and that $r = O(\log b)$. The running time of the* REMAINDERTREE *algorithm is*

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

and its space complexity is

$$O(r^2(B + bH)\log b + rB').$$

This statement differs from [**10**, Theorem 4.1] in two ways: we have imposed the additional requirement that $r = O(\log b)$, and the $r^3\,\mathsf{M}(B + bH)\log b$ term in the time complexity is improved to $r^2\,\mathsf{M}(B + bH)\log b$ (we have also changed h to H to avoid a collision of notation).

PROOF. Let us first estimate the complexity of computing the $A_{i,j}$ tree. The entries of any product $A_{j_1} \cdots A_{j_2-1}$ have bit-size $O((j_2 - j_1)(H + \log r))$, which is $O((j_2 - j_1)H)$.[1] Thus at level $\ell - i$ of the tree, each matrix product has cost

$$O(\mathsf{M}_r(2^i H)) = O(r^2\,\mathsf{M}(2^i H) + r^\omega (2^i H)(\log\log 2^i H)^2),$$

by Lemma 3.1. There are $2^{\ell - i}$ such products at this level, whose total cost is

$$O(r^2\,\mathsf{M}(bH) + r^\omega (bH)(\log\log bH)^2).$$

Since we assumed that $r = O(\log b) = O(\log bH)$, this is bounded by

$$O(r^2\,\mathsf{M}(bH) + r^2 (bh)(\log bH)^{\omega-2}(\log\log bH)^2).$$

We may take $\omega < 3$, and then certainly $(bH)(\log bH)^{\omega-2}(\log\log bH)^2 = O(\mathsf{M}(bH))$, again by the assumption that $\mathsf{M}(s)/(s\log s)$ is increasing. The first term dominates, and we are left with the bound $O(r^2\,\mathsf{M}(bH))$ for this level of the tree, and hence

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

for the whole tree.

The rest of the argument is exactly the same as in [**10**] and [**8**]; we omit the details. □

In [**10**] we also gave an algorithm REMAINDERFOREST, which has the same input and output specifications as REMAINDERTREE, but saves space by splitting the work into 2^κ subtrees, where $\kappa \in [0, \ell]$ is a parameter. (In [**10**] the parameter κ was called k.) This is crucial for practical computations, as REMAINDERTREE is extremely memory-intensive.

Theorem 3.2 leads to the following complexity bound for REMAINDERFOREST, improving Theorem 4.2 of [**10**]. We omit the proof, which is exactly the same as in [**10**], with obvious modifications to take account of Theorem 3.2.

[1] There are some missing parentheses in the corresponding estimate in the proof of [**10**, Theorem 4.1].

THEOREM 3.3. *Let B be an upper bound on the bit-size of $\prod_{j=0}^{b-1} m_j$ such that $B/2^\kappa$ is an upper bound on the bit-size of $\prod_{j=st}^{st+t-1} m_j$ for all s, where $t := 2^{\ell-\kappa}$. Let B' be an upper bound on the bit-size of any entry of V, and let H be an upper bound on the bit-size of any $m_0, \ldots, m_{b-1}$ and any entry in $A_0, \ldots, A_{b-1}$. Assume that $\log r = O(H)$ and that $r = O(\log b)$. The running time of the* REMAINDERFOREST *algorithm is*

$$O(r^2\,\mathsf{M}(B + bH)(\ell - \kappa) + 2^\kappa r^2\,\mathsf{M}(B) + r\,\mathsf{M}(B')),$$

and its space complexity is

$$O(2^{-\kappa} r^2 (B + bH)(\ell - \kappa) + r(B + B')).$$

4. Computing the first row

As above we work with a hyperelliptic curve $C/\mathbb{Q}$ of genus g defined by an equation $y^2 = f(x)$, where $f(x) = \sum_{i=c}^{d} f_i x^i \in \mathbb{Z}[x]$ is squarefree and $f_c f_d \neq 0$. We define $h(x) := f(x)/x^c = \sum_{i=0}^{r} h_i x^i$ with $r := d - c$ and put $e := 2 - c \in \{1, 2\}$, as in §2. We call a prime p *admissible* if it is a prime of good reduction that does not divide h_0. The following algorithm uses (9) to compute W_p^1, the first row of the Hasse–Witt matrix W_p, simultaneously for admissible primes $p \leq N$.

Algorithm COMPUTEHASSEWITTFIRSTROWS

Given an integer $N > 1$ and a hyperelliptic curve $y^2 = f(x)$ with $f(x), h(x)$, r, and e as above, compute W_p^1 for all admissible primes $p \leq N$ as follows:

1. Let $\ell = \lceil \log_2 N \rceil - 1$ and initialize a sequence of moduli m_n by
$$m_n := \begin{cases} p & \text{if } p = 2n+1 \text{ is an admissible prime in } [1, N], \\ 1 & \text{otherwise}, \end{cases}$$
for all integers $n \in [1, 2^\ell)$.
2. Run REMAINDERFOREST with $b := 2^\ell$ and $\kappa := \lceil 2 \log_2 \ell \rceil$ on inputs $V := V_0$, $A_k := M'_{k+1}$, and m_k with $k \in [0, b)$, where V_0 and M'_k are as defined in §2, to compute
$$u_n := V_0 M'_1 \cdots M'_n \bmod m_n,$$
for all integers $n \in [1, N/2)$. (One may set A_k to the zero matrix for $k \geq N/2$).
3. Similarly, use REMAINDERFOREST to compute $\delta_n := (en)! \bmod m_n$ for all $n \in [1, N/2)$.
(For $e = 2$ one can skip this step and simply set $\delta_n := -1$; see Remark 2.3.)
4. For each admissible prime $p = 2n + 1 \in [1, N]$ output the last g entries of the vector
$$\frac{\left(\frac{2}{p}\right)^e}{\left(\frac{h_0}{p}\right)^{e-1} \delta_n} u_n \bmod m_n$$
in reverse order (this is the vector W_p^1).

REMARK 4.1. The bulk of the work in COMPUTEHASSEWITTFIRSTROWS occurs in step 2. In the $e = 2$ case, this step is about twice as expensive as the $e = 1$ case, because there are twice as many matrices M_k; equivalently, the entries of M'_k are about twice as big. This suggests that it is advantageous to change variables, if possible, to achieve $e = 1$. (Step 3 is more expensive when $e = 1$, but the extra cost is negligible compared to the savings achieved in step 2.) This can be done if the curve has a rational Weierstrass point, by moving the Weierstrass point to $x = 0$. In §6.1 we discuss further optimizations along these lines that may utilize up to $g + 1$ rational Weierstrass points.

THEOREM 4.2. *Assume that* $g = O(\log N)$ *and that* $\log \max_i |f_i| = O(\log N)$. *The algorithm* COMPUTEHASSEWITTFIRSTROWS *runs in* $O(g^2\, \mathsf{M}(N \log N) \log N)$ *time and* $O(g^2 N)$ *space.*

PROOF. Let us analyze the complexity of step 2, using Theorem 3.3. The prime number theorem implies that we may take $B = O(N)$, and the hypothesis on $B/2^\kappa$ follows easily (see the proof of [**10**, Theorem 1.1]). We also have $B' = O(1)$ and

$$H = O(\log N + \log g + \log \max_i |h_i|) = O(\log N),$$

by the definition of the matrices M'_k. The remaining hypotheses are satisfied since we have $r = O(g) = O(\log N)$. By Theorem 3.3 and our choice of κ, step 2 runs in time

$$O(g^2\, \mathsf{M}(N \log N) \log N + (\log N)^2 g^2\, \mathsf{M}(N));$$

since $\mathsf{M}(s)/(s \log s)$ is increasing, this simplifies to $O(g^2\, \mathsf{M}(N \log N) \log N)$. The space complexity of step 2 is

$$O((\log N)^{-2} g^2 (N \log N) \log N + gN) = O(g^2 N).$$

The second invocation of REMAINDERFOREST (step 3) is certainly no more expensive than the first. The remaining steps such as enumerating primes and computing quadratic residue symbols take negligible time (see the proof Theorem 1.1 in [**10**]). □

REMARK 4.3. In practice, the parameter κ is chosen based on empirical performance considerations, rather than strictly according to the formula given above.

REMARK 4.4. The space complexity of COMPUTEHASSEWITTFIRSTROWS is $O(g^2 N)$, which is larger than the $O(gN)$ size of its output. Most of this space can be reused in subsequent calls to COMPUTEHASSEWITTFIRSTROWS, so we only need $O(g^2 N)$ space to handle g calls, as in algorithm COMPUTEHASSEWITTMATRICES below; this allows us to obtain the $O(g^2 N)$ space bound of Theorem 1.1.

For inadmissible primes p, COMPUTEHASSEWITTFIRSTROWS does not yield any information about W_p^1. When $f_0 = 0$ every good prime is admissible (if $f_0 = 0$ and p divides $h_0 = f_1$ then $f(x)$ is not squarefree modulo p), but when $f_0 \neq 0$ (the case $e = 2 - c = 2$), there may be primes of good reduction that divide f_0 and are therefore inadmissible.

To compute W_p^1 at such primes, we must use an alternative algorithm. There are many possible choices, but we present here an algorithm that uses the framework developed in §2. The idea is to evaluate the product in (8) in the most naive way possible. To our knowledge, this simple algorithm has not been mentioned previously in the literature.

We first set some notation. Let $\bar{f} \in \mathbb{F}_p[x]$ denote a polynomial for which $y^2 = \bar{f}(x)$ defines a hyperelliptic curve $C_p/\mathbb{F}_p$ of genus g (so $\bar{f}$ is squarefree of degree $\bar{d} = 2g + 1$ or $\bar{d} = 2g + 2$), let $\bar{c}$ be the least integer for which $\bar{f}_{\bar{c}} \neq 0$, and set $\bar{r} := \bar{d} - \bar{c}$. (Note: in the case of interest, $\bar{f} = f \bmod p$, but p divides $f_0 \neq 0$, so $\bar{c} \neq c$). Now define $\bar{h}(x) := \sum_{i=0}^{\bar{r}} \bar{h}_i x^i = \bar{f}(x)/x^{\bar{c}}$ and $\bar{e} := 2 - \bar{c}$. Let $n := (p-1)/2$

as usual, and let $\bar{M}_k$ be the matrix over $\mathbb{F}_p$ defined in (7), with h_i replaced by $\bar{h}_i$. The following algorithm uses Lemma 2.4 to compute the first row W_p^1 of the Hasse–Witt matrix of C_p.

Algorithm COMPUTEHASSEWITTFIRSTROW
Let $y^2 = \bar{f}(x)$ be a hyperelliptic curve $C_p/\mathbb{F}_p$, with notation as above. Compute the first row W_p^1 of the Hasse–Witt matrix of C_p as follows:

1. Initialize $u_0 := [0, \dots, 0, 1] \in (\mathbb{F}_p)^r$ and $\delta_0 := 1 \in \mathbb{F}_p$.
2. For k from 1 to $\bar{e}n$, compute $u_k := u_{k-1}\bar{M}_k$ and $\delta_k := \delta_{k-1}k$.
3. Output the last g entries of the vector

$$\frac{\left(\frac{2}{p}\right)^{\bar{e}}}{\left(\frac{\bar{h}_0}{p}\right)^{\bar{e}-1}\delta_{\bar{e}n}} u_{\bar{e}n}.$$

THEOREM 4.5. COMPUTEHASSEWITTFIRSTROW *runs in* $O(gp\,\mathsf{M}(\log p))$ *time and uses* $O(g \log p)$ *space.*

PROOF. Each matrix $\bar{M}_k$ has at most $2r - 1 = O(g)$ nonzero entries, each of which can be computed using $O(1)$ ring operations. Each iteration of step 2 uses $O(g\,\mathsf{M}(\log p))$ bit operations, and the number of iterations is $O(p)$, yielding a total cost of $O(gp\,\mathsf{M}(\log p))$ for step 2. The Legendre symbol and division in step 3 require at most $O(\log^2 p)$ bit operations, which is negligible.

For the space bound, each u_k and δ_k may overwrite u_{k-1} and δ_{k-1}, respectively. Each $\bar{M}_k$ requires just $O(g \log p)$ space and can be computed as needed and then discarded. ☐

The space bound in Theorem 4.5 is optimal; in fact, it matches the size of both the input and the output. In practice, this algorithm performs quite well for small to moderate p, primarily due its extremely small memory footprint; see §7 for performance details.

5. Hasse–Witt matrices of translated curves

In this section we fix a prime p of good reduction for our hyperelliptic equation $y^2 = f(x)$. For each integer a, let $W_p(a) = [w_{ij}(a)]$ denote the Hasse–Witt matrix of the translated curve $y^2 = f(x + a)$ at p, and let $W_p^1(a)$ denote its first row. In this section we show that if we know $W_p^1(a_i)$ for integers $a_1, \dots, a_g$ that are distinct modulo p, then we can deduce the entire Hasse–Witt matrix $W_p = [w_{ij}]$ of our original curve at p.

We first study how W_p transforms under the translation $x \mapsto x + a$.

THEOREM 5.1. *With notation as above we have*

$$W_p(a) = T(a)W_pT(-a),$$

where $T(a) := [t_{ij}(a)]$ *is the* $g \times g$ *upper triangular matrix with entries*

$$t_{ij}(a) := \binom{j-1}{i-1} a^{j-i} \qquad (1 \le i, j \le g).$$

PROOF. Let F be the function field of the curve $y^2 = f(x)$ over $\mathbb{F}_p$ (the fraction field of $\mathbb{F}_p[x,y]/(y^2 - f(x))$). The space $\Omega_F(0)$ of regular differentials on F (as defined in [**20**, §1.5], for example) is a g-dimensional $\mathbb{F}_p$-vector space with basis $(\omega_1, \dots, \omega_g)$, where

$$\omega_i := \frac{x^{i-1}dx}{y} \qquad (1 \le i \le g);$$

see [**20**, Ex. 4.6] and [**23**, Eq. 3]. It follows from [**23**, Prop. 2.2] that the Hasse–Witt matrix $W_p(a)$ has the form SW_pS^{-1}, where $S = [s_{ij}]$ is the change of basis matrix from the basis $(\omega_1, \dots, \omega_g)$ to the basis $(\theta_1, \dots, \theta_g)$, with

$$\theta_j := \frac{(x+a)^{j-1}dx}{y} \qquad (1 \le j \le g).$$

(Note that we have replaced the matrix $S^{(p)} = [s_{ij}^p]$ that appears in [**23**, Prop. 2.2] with $S = [s_{ij}]$ because we are working over $\mathbb{F}_p$ and therefore have $s_{ij}^p = s_{ij}$.) We then have

$$\theta_j = \frac{(x+a)^{j-1}dx}{y} = \sum_{i=1}^{j} \binom{j-1}{i-1} a^{j-i} \frac{x^{i-1}dx}{y} = \sum_{i=1}^{g} \binom{j-1}{i-1} a^{j-i}\omega_i = \sum_{i=1}^{g} t_{ij}(a)\omega_i,$$

so $S = T(a)$, and this implies $S^{-1} = T(a)^{-1} = T(-a)$. It then follows that $W_p(a) = SW_pS^{-1} = T(a)W_pT(-a)$, as claimed. □

Now suppose $p \ge g$ and that we have computed $W_p^1(a_i)$ for integers $a_1, \dots, a_g$ that are distinct modulo p. Writing out the equation for the first row of the matrix $W_p(a_i) = T(a_i)W_pT(-a_i)$ explicitly, we have

$$w_{1j}(a_i) = \sum_{k=1}^{g}\sum_{\ell=1}^{g} t_{1k}(a_i)w_{k\ell}t_{\ell j}(-a_i) = \sum_{k=1}^{g}\sum_{\ell=1}^{j} \binom{j-1}{\ell-1}(-1)^{j-\ell}a_i^{k-1+j-\ell}w_{k\ell}.$$

As i and j range over $\{1, \dots, g\}$, this may be regarded as a system of g^2 linear equations in the g^2 unknowns $w_{k\ell}$ over $\mathbb{F}_p$.

We claim that this system has a unique solution. Indeed, separating out the terms with $\ell = j$, we may write

$$w_{1j}(a_i) = \left(w_{1j} + a_iw_{2j} + a_i^2w_{3j} + \dots + a_i^{g-1}w_{gj}\right) + w_j(a_i),$$

where the last term

$$w_j(a) := \sum_{k=1}^{g}\sum_{\ell=1}^{j-1} \binom{j-1}{\ell-1}(-1)^{j-\ell}a^{k-1+j-\ell}w_{k\ell} \tag{11}$$

depends only on a and the first $j-1$ columns of W_p. Thus for each j we have a system

$$\begin{bmatrix} 1 & a_1 & a_1^2 & \cdots & a_1^{g-1} \\ 1 & a_2 & a_2^2 & \cdots & a_2^{g-1} \\ \vdots & \vdots & \vdots & \vdots & \\ 1 & a_g & a_g^2 & \cdots & a_g^{g-1} \end{bmatrix} \begin{bmatrix} w_{1j} \\ w_{2j} \\ \vdots \\ w_{gj} \end{bmatrix} = \begin{bmatrix} w_{1j}(a_1) - w_j(a_1) \\ w_{1j}(a_2) - w_j(a_2) \\ \vdots \\ w_{1j}(a_g) - w_j(a_g) \end{bmatrix}. \tag{12}$$

The matrix on the left is a Vandermonde matrix $V(a_1, \dots, a_g)$; it is non-singular because the a_i are distinct in $\mathbb{F}_p$. Therefore, for each $j = 1, \dots, g$ the system (12) determines the jth column of W_p uniquely in terms of the $w_{1j}(a_i)$ and the first $j-1$

columns of W_p. Given as input $w_{1j}(a_i)$ for all i and j, we may solve this system successively for $j = 1, \dots, g$ to determine all g columns of W_p.

EXAMPLE 5.2. Consider the hyperelliptic curve

$$y^2 = f(x) = 2x^8 + 3x^7 + 5x^6 + 7x^5 + 11x^4 + 13x^3 + 17x^2 + 19x + 23,$$

over the finite field $\mathbb{F}_{97}$, and let $a_1 = 0, a_2 = 1, a_3 = 2$. Computing the first rows of the Hasse–Witt matrices $W_p(0), W_p(1), W_p(2)$ yields

$$\begin{aligned} w_{11}(0) &= 9, & w_{12}(0) &= 37, & w_{13}(0) &= 54,\\ w_{11}(1) &= 43, & w_{12}(1) &= 60, & w_{13}(1) &= 30,\\ w_{11}(2) &= 5, & w_{12}(2) &= 70, & w_{13}(2) &= 84. \end{aligned}$$

Solving the system

$$\begin{bmatrix} 1 & 0 & 0\\ 1 & 1 & 1\\ 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} w_{11}\\ w_{21}\\ w_{31} \end{bmatrix} = \begin{bmatrix} w_{11}(0)\\ w_{11}(1)\\ w_{11}(2) \end{bmatrix} = \begin{bmatrix} 9\\ 43\\ 5 \end{bmatrix}$$

gives $w_{11} = 9, w_{21} = 70, w_{31} = 61$, the first column of W_p. Using (11) to compute $w_2(0) = 0$, $w_2(1) = 54$, $w_2(2) = 87$, we then solve

$$\begin{bmatrix} 1 & 0 & 0\\ 1 & 1 & 1\\ 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} w_{12}\\ w_{22}\\ w_{32} \end{bmatrix} = \begin{bmatrix} w_{12}(0) - w_2(0)\\ w_{12}(1) - w_2(1)\\ w_{12}(2) - w_2(2) \end{bmatrix} = \begin{bmatrix} 37\\ 6\\ 80 \end{bmatrix}$$

to get the second column $w_{12} = 37$, $w_{22} = 62$, $w_{32} = 4$. Finally, using (11) to compute $w_3(0) = 0$, $w_3(1) = 31$, $w_3(2) = 88$ we solve

$$\begin{bmatrix} 1 & 0 & 0\\ 1 & 1 & 1\\ 1 & 2 & 4 \end{bmatrix} \begin{bmatrix} w_{13}\\ w_{23}\\ w_{33} \end{bmatrix} = \begin{bmatrix} w_{13}(0) - w_3(0)\\ w_{13}(1) - w_3(1)\\ w_{13}(2) - w_3(2) \end{bmatrix} = \begin{bmatrix} 54\\ 96\\ 93 \end{bmatrix}$$

to get the third column $w_{13} = 54$, $w_{23} = 16$, $w_{33} = 26$, and we have

$$W_p = \begin{bmatrix} 9 & 37 & 54\\ 70 & 62 & 16\\ 61 & 4 & 26 \end{bmatrix},$$

which is the Hasse–Witt matrix of $y^2 = f(x)$.

We now bound the complexity of this procedure. The bound given in the lemma below is likely not the best possible, but it suffices for our purposes here.

LEMMA 5.3. *Given $W_p^1(a_1), \dots, W_p^1(a_g)$, we may compute W_p in*

$$O((g^3 + g^2 \log\log p)\,\mathsf{M}(\log p))$$

time and $O(g^2 \log p)$ space.

PROOF. We first invert the Vandermonde matrix $V(a_1, \dots, a_g)$. This requires $O(g^2)$ field operations in $\mathbb{F}_p$, by [**2**], or $O(g^2\,\mathsf{M}(\log p)\log\log p)$ bit operations.

To compute W_p, we use the algorithm sketched above. More precisely, let us define $\beta_j(a) := \sum_{k=1}^{g} a^{k-1} w_{kj}$, so that

$$w_j(a_i) = \sum_{\ell=1}^{j-1} \binom{j-1}{\ell-1} (-a_i)^{j-\ell} \beta_\ell(a_i).$$

Having computed column $j-1$ of W_p, we may compute $\beta_{j-1}(a_i)$ for all i, at a cost of $O(g^2)$ ring operations in $\mathbb{F}_p$. We then use the formula above to compute $w_j(a_i)$ for all i, again at a cost of $O(g^2)$ ring operations. Finally we solve (12) for the jth column of W_p, using the known inverse of $V(a_1,\dots,a_g)$, with $O(g^2)$ ring operations. This procedure is repeated for $j = 1,\dots,g$, for a total of $O(g^3)$ ring operations, or $O(g^3\,\mathsf{M}(\log p))$ bit operations.

The space bound is clear; we only need space for $O(g^2)$ elements of $\mathbb{F}_p$. □

6. Computing the whole matrix

In this section we assemble the various components we have developed to obtain an algorithm for computing the whole Hasse–Matrix matrix W_p. We begin with an algorithm that handles a single prime $p \geq g$.

Algorithm ComputeHasseWittMatrix
Given a hyperelliptic curve $y^2 = f(x)$ over $\mathbb{F}_p$ of genus $g \leq p$, compute the Hasse–Witt matrix W_p as follows:

1. For g distinct $a_i \in \mathbb{F}_p$, compute $W_p^1(a_i)$ by applying ComputeHasseWittFirstRow to the equation $y^2 = f(x + a_i)$.
2. Deduce W_p from $W_p^1(a_1),\dots,W_p^1(a_g)$ using Lemma 5.3.

Theorem 6.1. ComputeHasseWittMatrix *runs in* $O(g^2 p\,\mathsf{M}(\log p))$ *time and* $O(g^2 \log p)$ *space.*

Proof. For each i, computing the polynomial $f(x+a_i)$ costs $O(g^2)$ ring operations in $\mathbb{F}_p$, and this is dominated by the $O(gp\,\mathsf{M}(\log p))$ bit complexity of computing $W_p^1(a_i)$ (Theorem 4.5), since $g \leq p$. Thus we can compute $W_p^1(a_1),\dots,W_p^1(a_g)$ in $O(g^2 p\,\mathsf{M}(\log p))$ time. By Lemma 5.3 we may deduce W_p in time

$$O((g^3 + g^2 \log\log p)\,\mathsf{M}(\log p)) = O(g^2 p\,\mathsf{M}(\log p)).$$

The space bound follows immediately from Theorem 4.5 and Lemma 5.3. □

Finally, we present ComputeHasseWittMatrices, which computes W_p for all good primes $p \leq N$ by computing $W_p^1(a_i)$ for g chosen integers a_i and all suitable primes $p \leq N$. We rely on ComputeHasseWittMatrix to fill in the values of W_p at any good primes p that are inadmissible for one of the translated curves $y^2 = f(x + a_i)$ or for which the a_i are not distinct modulo p.

Algorithm ComputeHasseWittMatrices
Given a hyperelliptic curve $y^2 = f(x)$ with $f \in \mathbb{Z}[x]$, and an integer $N > 1$, compute the Hasse–Witt matrices W_p for all primes $p \leq N$ of good reduction as follows:

1. For odd primes $p < g$ of good reduction compute W_p directly from its definition by expanding $f(x)^{(p-1)/2} \bmod p$ and selecting the appropriate coefficients.
2. Choose g distinct integers $a_1,\dots,a_g$ that are either roots of $f(x)$ or in the interval $[0, g)$. Let $\mathcal{S}$ be the set of primes $p \in [g, N]$ of good reduction that divide some $a_i - a_j$ or some nonzero $f(a_i)$.
3. For primes $p \in \mathcal{S}$, use ComputeHasseWittMatrix to compute W_p.

4. Use ComputeHasseWittFirstRows to compute $W_p^1(a_1), \ldots, W_p^1(a_g)$ for all primes $p \in [g, N]$ of good reduction that do not lie in $\mathcal{S}$. Then deduce W_p for each such prime using Lemma 5.3.

Remark 6.2. As in [**10**], the computations performed in the calls ComputeHasseWittFirstRows in step 4 may be interleaved so that $W_p^1(a_1), \ldots, W_p^1(a_g)$ are computed for batches of primes p corresponding to subtrees of the remainder forest, and the computation of the matrices W_p for these primes can then be completed batch by batch.

We now prove the main theorem announced in §1, which states that ComputeHasseWittMatrices runs in $O(g^3\,\mathsf{M}(N \log N) \log N)$ time and $O(g^2 N)$ space, under the hypotheses $g \log g = O(\log N)$ and $\log \max_i |f_i| = O(\log N)$.

In order to simplify the analysis, *we assume that we always choose* $a_i = i - 1$ in step 2. (The complexity bounds of the theorem hold without this assumption, i.e., if we allow a_i to be any root of $f(x)$, but we do not prove this.)

Proof of Theorem 1.1. The time complexity of step 1 may be bounded by $O(g\,\mathsf{M}(\log N) + \mathsf{M}(g^2 \log g))$ for each prime; the first term covers the cost of reducing $f(x)$ modulo p, and the second covers the cost of computing $f(x)^{(p-1)/2}$ in the ring $\mathbb{F}_p[x]$. There are at most g primes, so the overall cost is bounded by $O(g^2\,\mathsf{M}(\log N) + g\,\mathsf{M}(g^2 \log g))$. The space complexity is $O(g^2 \log g)$ for each prime, and also $O(g^2 \log g)$ overall. Both bounds are dominated by the bounds given in the theorem.

The coefficient of x^j in the translated polynomial $f^{(i)}(x) := f(x + a_i)$ is $\sum_k \binom{k}{j} a^{k-j} f_k$, thus

$$\max_j |(f^{(i)})_j| \le (2g+2)\binom{2g+2}{g+1} g^{2g+2} \max_k |f_k|,$$

and therefore $\log \max_j |(f^{(i)})_j| = O(g \log g + \log \max_k |f_k|) = O(\log N)$. In particular, the bit-size of $f(a_i)$ is $O(\log N)$, so the number of primes that divide any particular $f(a_i)$ is $O(\log N)$. Consequently, $|\mathcal{S}| = O(g \log N)$.

By Theorem 6.1, the total time spent in step 3 is $O((g \log N)(g^2 N\,\mathsf{M}(\log N)))$, which is dominated by $O(g^3\,\mathsf{M}(N \log N) \log N)$. The space used in step 3 is negligible.

Finally, computing $W_p^1(a_1), \ldots, W_p^1(a_g)$ for suitable primes $p \le N$ in step 4 requires time $O(g^3\,\mathsf{M}(N \log N) \log N)$ and space $O(g^2 N)$, by Theorem 4.5. This dominates the contribution from Lemma 5.3, which gives a total time bound of

$$O((N/\log N)(g^3 + g^2 \log\log N)\,\mathsf{M}(\log N))$$

over all primes p, and has a negligible space bound. $\square$

6.1. Optimizations for curves with rational Weierstrass points. For hyperelliptic curves with one or more rational Weierstrass points the complexity of ComputeHasseWittMatrices can be improved by a significant constant factor. For curves with a rational Weierstrass point P, we can ensure that $d = 2g + 1$ by putting P at infinity. This also ensures that every translated curve $y^2 = f(x + a_i)$ also has $d = 2g + 1$, and we get an overall speedup by a factor of at least

$$\left(\frac{2g+2}{2g+1}\right)^2$$

compared to the case where C has no rational Weierstrass points, since we work with vectors and matrices of dimension $2g+1$ rather than $2g+2$. For example, the speedup is approximately $(6/5)^2 = 1.44$ for $g = 2$ and $(8/7)^2 \approx 1.31$ for $g = 3$.

Alternatively, putting P at zero and choosing $a_1 = 0$ speeds up the computation of $W_p^1(a_1)$ by a factor of two (because we have half as many matrices M_k to deal with), but it does not necessarily speed up the computation of $W_p^1(a_2), \ldots, W_p^1(a_g)$. If we have just a single rational Weierstrass point we should put it at zero when $g \leq 2$, but otherwise we should put it at infinity.

When C has more than one rational Weierstrass point we can get a further performance improvement by putting rational Weierstrass points at both zero and infinity and choosing $a_1 = 0$. If there are any other rational Weierstrass points, we should then choose $a_2, \ldots, a_g$ to be the negations of the x-coordinates of any other rational Weierstrass points. Without loss of generality we may assume that these coordinates are all integral, since once we have $y^2 = f(x)$ with a rational Weierstrass point at infinity, the polynomial $f(x)$ has odd degree and can be made monic (and integral) by scaling x and y appropriately. These changes may impact the size of the coefficients of f, but such changes are typically small, and may even be beneficial (in any case, for sufficiently large N the benefit outweighs the cost).

For each a_i for which $y^2 = f(x + a_i)$ has rational Weierstrass points at zero and infinity we get a speedup by a factor of

$$2\left(\frac{g+1}{g}\right)^2 \tag{13}$$

in the time to compute $W_p^1(a_i)$, relative to the case where $y^2 = f(x)$ has no rational Weierstrass points; we get a factor of 2 because the number of matrices M_k is halved, and then a factor of $((2g+2)/(2g))^2$ from the reduction in dimension of matrices and vectors. When C has $g+1$ rational Weierstrass points we get a total speedup by the factor given in (13), relative to the case where there are no rational Weierstrass points. The speedup observed in practice is a bit better than this, as may be seen in Tables 1 and 2. This can be explained by the fact that the cost of matrix multiplication is actually super-quadratic at the lower levels of the accumulating remainder trees.

Remark 6.3. The same speedup can be achieved when there are just g rational Weierstrass points, by also computing the *last* row of W_p and only using $g - 1$ translated curves.

7. Performance results

We implemented our algorithms using the GNU C compiler (`gcc` version 4.8.2) and the GNU multiple precision arithmetic library (GMP version 6.0.0). The timings listed in the tables that follow were all obtained on a single core of an Intel Xeon E5-2697v2 CPU running at a fixed clock rate of 2.70GHz with 256 GB of RAM.

In our tests we used curves with small coefficients; we generally set $f_{d-i} = p_{i+1}$, where p_i is the ith prime, implying that $\log \max |f_i| = O(g \log g)$. For curves with $w > 1$ rational Weierstrass points we chose $f(x)$ monic with integer roots at $0, \ldots, w-2$. For genus 3 curves with 3 rational Weierstrass points we applied Remark 6.3.

	$w=0$		$w=1$		$w=2$	
N	hw1	hw2	hw1	hw2	hw1	hw2
2^{14}	0.8	0.2	0.5	0.1	0.3	0.1
2^{15}	2.6	0.6	1.2	0.3	0.6	0.2
2^{16}	5.8	1.6	3.2	0.8	1.6	0.4
2^{17}	14.0	4.1	8.1	2.2	4.1	1.0
2^{18}	33.1	9.5	20.4	5.1	9.7	2.3
2^{19}	81.3	21.8	49.6	12.1	23.5	5.3
2^{20}	192	51.3	116	28.2	56.4	12.6
2^{21}	470	122	274	66.7	142	29.0
2^{22}	1,183	280	638	155	335	67.6
2^{23}	2,830	654	1,510	353	789	160
2^{24}	6,500	1,520	3,500	845	1,820	347
2^{25}	15,000	3,460	8,190	1,890	4,240	834
2^{26}	34,100	7,480	18,700	4,280	9,620	1,870

TABLE 1. Comparison of old (hw1) versus new (hw2) average polynomial-time algorithms for genus 2 curves over $\mathbb{Q}$ with w rational Weierstrass points (times in CPU seconds).

	$w=0$		$w=1$		$w=2$		$w=3$	
N	hw1	hw2	hw1	hw2	hw1	hw2	hw1	hw2
2^{14}	3.3	0.5	2.3	0.4	1.5	0.3	1.4	0.2
2^{15}	10.8	1.5	6.1	1.0	5.1	0.7	3.7	0.5
2^{16}	25.9	4.6	16.8	2.9	10.0	2.1	9.9	1.2
2^{16}	62.1	12.6	40.4	7.8	23.2	5.5	23.6	3.3
2^{18}	147	28.9	96.1	17.3	57.1	12.6	56.7	7.7
2^{19}	347	68.1	230	42.7	141	30.2	139	18.5
2^{20}	878	156	544	99.4	326	68.2	329	42.6
2^{21}	1,950	363	1,280	231	792	161	782	97.1
2^{22}	4,500	841	3,130	528	1,840	370	1,820	225
2^{23}	10,700	1,920	7,370	1,260	4,380	859	4,330	533
2^{24}	24,300	4,360	16,800	2,830	10,200	2,010	9,960	1,200
2^{25}	60,400	9,910	39,000	6,220	23,800	4,430	2,320	2,710
2^{26}	128,000	21,000	83,900	13,700	53,400	9,930	53,100	5,980

TABLE 2. Comparison of old (hw1) versus new (hw2) average polynomial-time algorithms for genus 3 hyperelliptic curves over $\mathbb{Q}$ with w rational Weierstrass points (times in CPU seconds).

Tables 1–4 compare the performance of the new average polynomial-time algorithm COMPUTEHASSEWITTMATRICES to the average polynomial-time algorithm of [**10**], and also to the smalljac library [**21**] based on [**13**], which was previously the fastest available package for performing these computations in genus $g \leq 2$ (within the feasible range of N), and the hypellfrob library [**5**] based on [**6**], which

	genus 2		genus 3	
N	sj	hw2	hf	hw2
2^{14}	0.2	0.1	7.2	0.4
2^{15}	0.6	0.3	16.3	1.0
2^{16}	1.7	0.9	39.1	2.9
2^{17}	5.5	2.2	98.3	7.8
2^{18}	19.2	5.3	255	18.3
2^{19}	78.4	12.5	695	43.2
2^{20}	271	27.8	1,950	98.8
2^{21}	1,120	64.5	5,600	229
2^{22}	2,820	155	16,700	537
2^{23}	9,840	357	51,200	1,240
2^{24}	31,900	823	158,000	2,800
2^{25}	105,000	1,890	501,000	6,280
2^{26}	349,000	4,250	1,480,000	13,900
2^{27}	1,210,000	9,590	4,360,000	31,100
2^{28}	4,010,000	21,200	12,500,000	69,700
2^{29}	13,200,000	48,300	39,500,000	155,000
2^{30}	45,500,000	108,000	120,000,000	344,000

TABLE 3. Comparison of the new average polynomial-time algorithm (`hw2`) to `smalljac` (`sj`) in genus 2 and `hypellfrob` (`hf`) in genus 3 for hyperelliptic curves over $\mathbb{Q}$ with one rational Weierstrass point (times in CPU seconds). For $N > 2^{26}$ the `sj` and `hf` timings were estimated by sampling $p \leq N$.

	$N = 2^{16}$		$N = 2^{18}$		$N = 2^{20}$		$N = 2^{22}$		$N = 2^{24}$	
g	hf	hw2	hf	hw2	hf	hw2	hf	hw2	hf	hw2
3	39	3	255	18	1,950	99	16,700	537	158,000	2,800
4	77	9	479	60	3,550	322	30,000	1,680	277,000	8,640
5	140	18	836	136	5,990	694	48,900	3,590	440,000	18,000
6	239	28	1,360	278	9,330	1,400	74,200	7,340	661,000	35,200
7	375	44	2,070	492	13,800	2,460	106,000	12,500	949,000	60,600
8	570	63	3,060	825	19,800	4,310	147,000	21,400	1,330,000	103,000
9	835	89	4,410	1,400	27,500	7,120	200,000	34,200	1,780,000	166,000
10	1,189	122	6,060	2,230	37,400	10,900	273,000	53,700	2,340,000	259,000

TABLE 4. Comparison of the new average polynomial-time algorithm (`hw2`) to `hypellfrob` (`hf`) for hyperelliptic curves over $\mathbb{Q}$ of genus 3 to 10 with one rational Weierstrass point (times in CPU seconds).

was previously the fastest available package for performing these computations in genus $g \geq 3$.

	$p=2^{16}+1$		$p=2^{17}+29$		$p=2^{18}+3$		$p=2^{19}+21$		$p=2^{20}+7$	
g	hf	hwp	hf	hwp	hf	hwp	hf	hwp	hf	hwp
3	8	4	11	8	16	16	23	33	36	66
4	16	8	20	15	29	31	40	61	61	123
5	27	11	34	22	47	43	65	86	98	172
6	44	19	54	39	74	78	100	155	149	310
7	68	26	82	52	110	104	144	207	213	414
8	102	33	120	67	159	134	203	267	295	534
9	148	42	171	84	222	168	279	335	401	670
10	207	42	239	102	303	205	377	409	539	819
11	285	62	325	123	407	246	497	492	721	983
12	381	73	430	146	533	292	642	582	965	1160
13	494	86	552	171	685	341	828	681	1220	1370
14	633	99	714	197	863	393	1070	786	1530	1570
15	803	113	884	225	1070	450	1330	899	1910	1800
16	1180	128	1120	256	1340	511	1650	1020	2330	2040
17	1260	145	1370	289	1660	575	1990	1150	2820	2300
18	1530	162	1690	322	2010	643	2600	1280	3330	2570
19	1880	180	2050	359	2410	715	2880	1430	3930	2860
20	2270	200	2480	397	2870	791	3400	1580	4630	3160

TABLE 5. Comparison of new algorithm to compute a single Hasse–Witt matrix (`hwp`) to `hypellfrob` (`hf`) for hyperelliptic curves over $\mathbb{F}_p$ of genus 3 to 20 with a rational Weierstrass point (times in CPU milliseconds).

Table 5 compares the performance of algorithm COMPUTEHASSEWITTMATRIX with complexity $O(g^2 p(\log p)^{1+o(1)})$ to the $O(g^3 p^{1/2}(\log p)^{2+o(1)})$ algorithm implemented by `hypellfrob` for computing a single Hasse–Witt matrix W_p for a hyperelliptic curve of genus g.

While the performance data listed here focuses on running times, we should note that the new algorithm is also more space efficient than the average polynomial-time algorithm given in [**10**]. The improvement in space is not as dramatic as the improvement in time, but we typically gain a a small constant factor. For example, the most memory intensive computation in Table 2 (genus 3 curves) occurs when $N = 2^{26}$ and $w = 0$ (no rational Weierstrass points); in this case the new algorithm (`hw2`) uses 11.4 GB of memory, versus 22.4 GB for the old algorithm (`hw1`). In both cases the memory footprint can be reduced by increasing the number of subtrees used in the REMAINDERFOREST algorithm, as determined by the parameter κ that appears in §4.2 (the parameter k in [**10**, Table 3]). Here we chose parameters that optimize the running time.

8. Computing Sato–Tate distributions

A notable application of our algorithm is the computation of Sato–Tate statistics. Associated to each smooth projective curve $C/\mathbb{Q}$ of genus g is the sequence of integer polynomials $L_p(T)$ at primes p of good reduction that appear in the numerator of the zeta function in (1). It follows from the Weil conjectures that each normalized L-polynomial

$$\overline{L}_p(T) = L_p(T/\sqrt{p}) = \sum_{i=0}^{2g} a_i T^i$$

is a real monic polynomial of degree $2g$ whose roots lie on the unit circle, with coefficients $a_i = a_{2g-i}$ that satisfy $|a_i| \le \binom{2g}{i}$. We may then consider the distribution of the a_i (jointly or individually) as p varies over primes of good reduction up to a bound N, as $N \to \infty$.

In order to compute these Sato–Tate statistics we need to know the integer values of the coefficients of $L_p(T)$, not just their reductions modulo p. As explained in [**7**, **8**], the integer polynomial $L_p(T)$ can be computed in average polynomial time using a generalization of the method presented here. However, for $g \le 3$ this can be more efficiently accomplished (for the feasible range of N) using group computations in the Jacobian of C_p and its quadratic twist, as explained in [**13**]. For $g \le 2$ there are at most 5 possible values for $L_p(T) \in \mathbb{Z}[T]$ given its reduction modulo $p > 13$, and the correct value can be determined in $O((\log p)^{2+o(1)})$ time, which is negligible. For $g \le 3$ there are $O(p^{1/2})$ possible values, and the correct value can be determined in $O(p^{1/4}\,\mathsf{M}(\log p))$ time using a baby-steps giant-steps approach. This time complexity is exponential in $\log p$ and asymptotically dominates the $O((\log p)^{4+o(1)})$ average time to compute $L_p(T) \bmod p$, but within the practical range of N this is not a problem. For example, when $N = 2^{30}$ it takes approximately 344,000 CPU seconds to compute $L_p(T) \bmod p$ for all good $p \le N$ for a hyperelliptic curve of genus 3, while the time to lift $L_p(T) \bmod p$ to $\mathbb{Z}[T]$ for all good $p \le N$ using the algorithm of [**13**] is just 55,370 CPU seconds, far less than it would take to compute $L_p(T)$ via [**7**, **8**].

Figure 1 shows the distributions of the normalized L-polynomial coefficients a_1, a_2, and a_3 over good primes $p \le 2^{30}$ for the curve

$$y^2 = x^7 - x + 1.$$

It follows from a result of Zarhin [**24**] that hyperelliptic curves of the form $y^2 = x^{2g+1} - x + 1$ over $\mathbb{Q}$ have large Galois image. As a consequence, the Sato–Tate group of this curve, as defined in [**3**] or [**19**], is the unitary symplectic group USp(6). Under the generalized Sato–Tate conjecture the distribution of normalized L-polynomials should match the distribution of characteristic polynomials of a random matrix in USp(6), under the Haar measure, and this indeed appears to be the case.

We also used our algorithm to compute Sato–Tate statistics for several other hyperelliptic curves of genus 3, including the curve

$$y^2 = x^7 + 3x^6 + 2x^5 + 6x^4 + 4x^3 + 12x^2 + 8x,$$

which has an unusual Sato–Tate distribution as can be seen in Figure 2. This curve was found in a large search of genus 3 hyperelliptic curves with small coefficients. This curve has a non-hyperelliptic involution $[x : y : z] \mapsto [z : y/4 : x/2]$, which implies that its Jacobian has extra endomorphisms and its Sato–Tate group must be a proper subgroup of USp(6) (the exact group has yet to be determined).

More examples can be found at `http://math.mit.edu/~drew`.

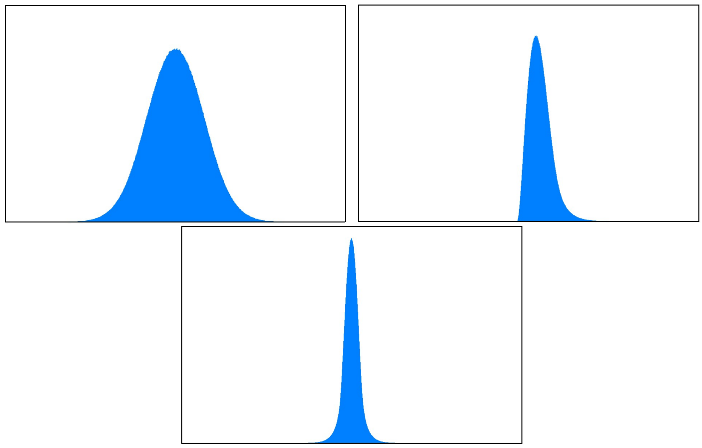

FIGURE 1. Distributions of normalized L-polynomial coefficients a_1, a_2, a_3 for $y^2 = x^7 - x + 1$ over primes $p \leq 2^{30}$.

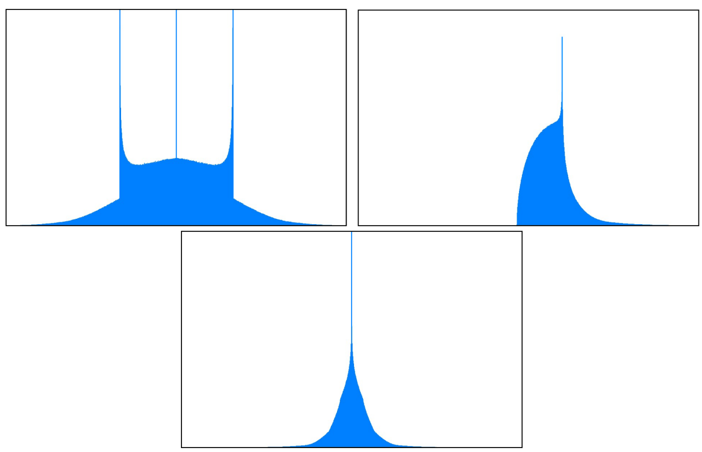

FIGURE 2. Distributions of normalized L-polynomial coefficients a_1, a_2, a_3 for $y^2 = x^7 + 3x^6 + 2x^5 + 6x^4 + 4x^3 + 12x^2 + 8x$ over good primes $p \leq 2^{30}$.

References

[1] Alin Bostan, Pierrick Gaudry, and Éric Schost, *Linear recurrences with polynomial coefficients and application to integer factorization and Cartier-Manin operator*, SIAM J. Comput. **36** (2007), no. 6, 1777–1806, DOI 10.1137/S0097539704443793. MR2299425 (2008a:11156)

[2] A. Eisinberg and G. Fedele, *On the inversion of the Vandermonde matrix*, Appl. Math. Comput. **174** (2006), no. 2, 1384–1397, DOI 10.1016/j.amc.2005.06.014. MR2220623

[3] Francesc Fité, Kiran S. Kedlaya, Víctor Rotger, and Andrew V. Sutherland, *Sato-Tate distributions and Galois endomorphism modules in genus 2*, Compos. Math. **148** (2012), no. 5, 1390–1442, DOI 10.1112/S0010437X12000279. MR2982436

[4] Martin Fürer, *Faster integer multiplication*, SIAM J. Comput. **39** (2009), no. 3, 979–1005, DOI 10.1137/070711761. MR2538847 (2011b:68296)

[5] David Harvey, `hypellfrob` *software library*, version 2.1.1 available at `http://web.maths.unsw.edu.au/~davidharvey/code/hypellfrob/hypellfrob-2.1.1.tar.gz`, 2008.

[6] ———, *Kedlaya's algorithm in larger characteristic*, Int. Math. Res. Not. IMRN (2007), no. 22, Art. ID rnm095, 29. MR2376210 (2009d:11096)

[7] ———, *Computing zeta functions of arithmetic schemes*, Proc. Lond. Math. Soc. **111** (2015), 1379–1401.

[8] David Harvey, *Counting points on hyperelliptic curves in average polynomial time*, Ann. of Math. (2) **179** (2014), no. 2, 783–803, DOI 10.4007/annals.2014.179.2.7. MR3152945

[9] David Harvey, Grégoire Lecerf, and Joris van der Hoeven, *Even faster integer multiplication*, J. Complexity, to appear, preprint `http://arxiv.org/abs/1407.3360`, 2014.

[10] David Harvey and Andrew V. Sutherland, *Computing Hasse-Witt matrices of hyperelliptic curves in average polynomial time*, LMS J. Comput. Math. **17** (2014), no. suppl. A, 257–273, DOI 10.1112/S1461157014000187. MR3240808

[11] David Harvey and Joris van der Hoeven, *On the complexity of integer matrix multiplication*, preprint `http://hal.archives-ouvertes.fr/hal-01071191`, 2014.

[12] Kiran S. Kedlaya, *Counting points on hyperelliptic curves using Monsky-Washnitzer cohomology*, J. Ramanujan Math. Soc. **16** (2001), no. 4, 323–338. MR1877805 (2002m:14019)

[13] Kiran S. Kedlaya and Andrew V. Sutherland, *Computing L-series of hyperelliptic curves*, Algorithmic number theory, Lecture Notes in Comput. Sci., vol. 5011, Springer, Berlin, 2008, pp. 312–326, DOI 10.1007/978-3-540-79456-1_21. MR2467855 (2010d:11070)

[14] Ju. I. Manin, *The Hasse-Witt matrix of an algebraic curve* (Russian), Izv. Akad. Nauk SSSR Ser. Mat. **25** (1961), 153–172. MR0124324 (23 #A1638)

[15] L. J. Mordell, *The congruence* $(p-1/2)! \equiv \pm 1 \pmod p$, Amer. Math. Monthly **68** (1961), 145–146. MR0123512 (23 #A837)

[16] J. Pila, *Frobenius maps of abelian varieties and finding roots of unity in finite fields*, Math. Comp. **55** (1990), no. 192, 745–763, DOI 10.2307/2008445. MR1035941 (91a:11071)

[17] A. Schönhage and V. Strassen, *Schnelle Multiplikation grosser Zahlen* (German, with English summary), Computing (Arch. Elektron. Rechnen) **7** (1971), 281–292. MR0292344 (45 #1431)

[18] René Schoof, *Elliptic curves over finite fields and the computation of square roots mod p*, Math. Comp. **44** (1985), no. 170, 483–494, DOI 10.2307/2007968. MR777280 (86e:11122)

[19] Jean-Pierre Serre, *Lectures on* $N_X(p)$, Chapman & Hall/CRC Research Notes in Mathematics, vol. 11, CRC Press, Boca Raton, FL, 2012. MR2920749

[20] Henning Stichtenoth, *Algebraic function fields and codes*, Universitext, Springer-Verlag, Berlin, 1993. MR1251961 (94k:14016)

[21] Andrew V. Sutherland, `smalljac` *software library*, version 4.0.23 available at `http://math.mit.edu/~drew/smalljac_v4.0.23.tar`, 2013.

[22] Joachim von zur Gathen and Jürgen Gerhard, *Modern computer algebra*, 3rd ed., Cambridge University Press, Cambridge, 2013. MR3087522

[23] Noriko Yui, *On the Jacobian varieties of hyperelliptic curves over fields of characteristic* $p > 2$, J. Algebra **52** (1978), no. 2, 378–410. MR0491717 (58 #10920)

[24] Yuri G. Zarhin, *Hyperelliptic Jacobians without complex multiplication*, Math. Res. Lett. **7** (2000), no. 1, 123–132, DOI 10.4310/MRL.2000.v7.n1.a11. MR1748293 (2001a:11097)

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

E-mail address: `d.harvey@unsw.edu.au`

URL: `http://web.maths.unsw.edu.au/~davidharvey/`

Department of Mathematics, Massachusetts Institute of Technology, 77 Mass. Ave., Cambridge, Massachusetts 02139

E-mail address: `drew@math.mit.edu`

URL: `http://math.mit.edu/~drew`