Computing *L*-series of hyperelliptic curves

Kiran S. Kedlaya[*] and Andrew V. Sutherland

Department of Mathematics
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139
(kedlaya|drew)@math.mit.edu

Abstract. We discuss the computation of coefficients of the *L*-series associated to a hyperelliptic curve over $\mathbb{Q}$ of genus at most 3, using point counting, generic group algorithms, and *p*-adic methods.

1 Introduction

For C a smooth projective curve of genus g defined over $\mathbb{Q}$, the *L*-function $L(C, s)$ is conjecturally (and provably for $g = 1$) an entire function containing much arithmetic information about C. Most notably, according to the conjecture of Birch and Swinnerton-Dyer, the order of vanishing of $L(C, s)$ at $s = 1$ equals the rank of the group $J(C/\mathbb{Q})$ of rational points on the Jacobian of C.

It is thus natural to ask to what extent we are able to compute with the *L*-function. This splits into two subproblems:

1. For appropriate N, compute the first N coefficients of the Dirichlet series expansion $L(C, s) = \prod_p L_p(p^{-s})^{-1} = \sum_{n=1}^{\infty} c_n n^{-s}$.
2. From the Dirichlet series, compute $L(C, s)$ at various values of s to suitable numerical accuracy. (The Dirichlet series converges for $\text{Real}(s) > 3/2$.)

In this paper, we address problem 1 for hyperelliptic curves of genus $g \leq 3$ with a distinguished rational Weierstrass point. This includes in particular the case of elliptic curves, and indeed we have something new to say in this case; we can handle significantly larger coefficient ranges than other existing implementations. We say nothing about problem 2; we refer instead to [5].

Our methods combine efficient point enumeration with generic group algorithms as discussed in the second author's PhD thesis [23]. For $g > 2$, we also apply *p*-adic cohomological methods, as introduced by the first author [12] and refined by Harvey [9]. Since what we need is adequately described in these papers, we focus our presentation on the point counting and generic group techniques and use an existing *p*-adic cohomological implementation provided by Harvey. (The asymptotically superior Schoof-Pila method [16, 15] only becomes practically better far beyond the ranges we can hope to handle.)

As a sample application, we compare statistics for Frobenius eigenvalues of particular curves to theoretical predictions. These include the Sato-Tate conjecture for $g = 1$, and appropriate analogues in the Katz-Sarnak framework for $g > 1$; for the latter, we find little prior numerical evidence in the literature.

2　The Problem

Let C be a smooth projective curve over $\mathbb{Q}$ of genus g . We wish to determine the polynomial $L_p(T)$ appearing in $L(C, s) = \prod L_p(p^{-s})^{-1}$, for $p \leq N$. We consider only p for which C is defined and nonsingular over $\mathbb{F}_p$ (almost all of them), referring to [17, 4] in the case of bad reduction. The polynomial $L_q(T)$ appears as the numerator of the local zeta function

$$Z(C/\mathbb{F}_q; T) = \exp\left(\sum_{k=1}^{\infty} N_k T^k / k\right) = \frac{L_q(T)}{(1-T)(1-qT)}, \tag{1}$$

where N_k counts the points on C over $\mathbb{F}_{q^k}$. Here q is any prime power, however we are primarily concerned with $q = p$ an odd prime. The rationality of $Z(C/\mathbb{F}_q; T)$ is part of the well known theorem of Weil [25], which also requires

$$L_q(T) = \sum_{i=0}^{2g} a_j T^j \tag{2}$$

to have integer coefficients satisfying $a_0 = 1$ and $a_{2g-j} = p^{g-j}a_j$, for $0 \leq j < g$. To determine $L_q(T)$, it suffices to compute $a_1, \ldots, a_g$.

For reasons of computational efficiency we restrict ourselves to curves which may be described by an affine equation of the form $y^2 = f(x)$, where $f(x)$ is a monic polynomial of degree $d = 2g+1$ (hyperelliptic curves with a distinguished rational Weierstrass point). We denote by $J(C/\mathbb{F}_q)$ the group of $\mathbb{F}_q$-rational points on the Jacobian variety of C over $\mathbb{F}_q$ (the *Jacobian* of C over $\mathbb{F}_q$), and use $J(\tilde{C}/\mathbb{F}_q)$ to denote the Jacobian of the quadratic twist of C over $\mathbb{F}_q$.

We consider three approaches to determining $L_p(T)$ for $g \leq 3$:

1. **Point counting**: Compute $N_1, \ldots, N_g$ of (1) by enumerating the points on C over $\mathbb{F}_p, \mathbb{F}_{p^2}, \ldots, \mathbb{F}_{p^g}$. The coefficients $a_1, \ldots, a_g$ can then be readily derived from (1) [3, p. 135]. This requires $O(p^g)$ field operations.

2. **Group computation**: Use generic algorithms to compute $L_p(1) = \#J(C/\mathbb{F}_p)$, and, for $g > 1$, compute $L_p(-1) = \#J(\tilde{C}/\mathbb{F}_p)$. Then use $L_p(1)$ and $L_p(-1)$ to determine $L_p(T)$ [22, Lemma 4]. This involves a total of $O(p^{(2g-1)/4})$ group operations.

3. **p-adic methods**: Apply extensions of Kedlaya's algorithm [12, 9] to compute (modulo p) the characteristic polynomial $\chi(T) = T^{-2g}L_p(T)$ of the Frobenius endomorphism on $J(C/\mathbb{F}_p)$, then use generic algorithms to compute the exact coefficients of $L_p(T)$. The asymptotic complexity is $\tilde{O}(p^{1/2})$.[1]

---

[1] For fixed $g \geq 4$, one works modulo $p^{\lfloor g/2-1 \rfloor}$ to obtain the same complexity.

Computing the coefficients of $L_p(T)$ for all $p \leq N$ necessarily requires time and space exponential in $\lg N$, since the output contains $\Theta(N)$ bits. In practice, we are limited to N of moderate size: on the order of 2^{40} in genus 1, 2^{28} in genus 2, and 2^{26} in genus 3 (larger in parallel computations). We expect to compute $L_p(T)$ for a large number of relatively small values of p. Constant factors will have considerable impact, however we first consider the asymptotic situation.

The $O(p^g)$ complexity of point counting makes it an impractical method to compute a_1, ..., a_g unless p is very small. However, point counting over $\mathbb{F}_p$ is an efficient way to compute $a_1 = N_1 - p - 1$ for a reasonably large range of p when $g > 1$, requiring only $O(p)$ field operations. Knowledge of a_1 aids the computation of $\#J(C/\mathbb{F}_p)$, reducing the complexity of the baby-steps giant-steps search to $O(p^{1/4})$ in genus 2 and $O(p)$ in genus 3. The optimal strategy then varies (c.f. [6, pp. 32-33]), according to genus and range of p:

Genus 1 The $O(p^{1/4})$ complexity of generic group computation makes it the compelling choice, easily outperforming point counting for $p > 2^{10}$.

Genus 2 There are three alternatives: (i) $O(p)$ field operations followed by $O(p^{1/2})$ group operations, (ii) $O(p^{3/4})$ group operations, or (iii) an $\tilde{O}(p^{1/2})$ p-adic computation. We find the range in which (iii) becomes optimal to be past the feasible values of N.

Genus 3 The choice is between (i) $O(p)$ field operations followed by $O(p)$ group operations and (ii) an $\tilde{O}(p^{1/2})$ p-adic computation followed by $O(p^{1/4})$ group operations. Here the p-adic algorithm plays the major role once $p > 2^{15}$.

3 Point Counting

Counting points on C over $\mathbb{F}_p$ plays a key role in our strategy for genus 2 and 3 curves. Moreover, it is a useful tool in its own right. If one wishes to study the distribution of $\#J(C/\mathbb{F}_p) = L_p(1)$, or to simply estimate $L_p(p^{-s})$, the value a_1 may be all that is required.

Given C in the form $y^2 = f(x)$, the simplest approach is to build a table of the quadratic residues in $\mathbb{F}_p$ (typically stored as a bit-vector), then evaluate $f(x)$ for all $x \in \mathbb{F}_p$. If $f(x) = 0$, there is a single point on the curve, and otherwise either two points (if $f(x)$ is a residue) or none. Additionally, we add a single point at infinity (recall that f has odd degree). A not-too-naïve implementation computes the table of quadratic residues by squaring half the field elements, then uses d field multiplications and d field additions for each evaluation of $f(x)$, where d is the degree of f. A better approach uses finite differences, requiring only d field additions (subtractions) to compute each $f(x)$.

Let $f(x) = \sum f_j x^j$ be a degree d polynomial over a commutative ring R. Fix a nonzero $\delta \in R$ and define the linear operator $\boldsymbol{\Delta}$ on $R[x]$ by

$$(\boldsymbol{\Delta}f)(x) = f(x + \delta) - f(x). \tag{3}$$

For any $x_0 \in R[x]$, given $f(x_0)$, we may enumerate the values $f(x_0 + n\delta)$ via

$$f(x_0 + (n + 1)\delta) = f(x_0 + n\delta) + \boldsymbol{\Delta}f(x_0 + n\delta). \tag{4}$$

To enumerate $f(x_0 + n\delta)$ it suffices to enumerate $\boldsymbol{\Delta} f(x_0 + n\delta)$, which we also do via (4), replacing f with $\boldsymbol{\Delta} f$. Since $\boldsymbol{\Delta}^{d+1} f = 0$, each step requires only d additions in R, starting from the initial values $\boldsymbol{\Delta}^k f(x_0)$ for $0 \le k \le d$.

When $R = \mathbb{F}_p$, this process enumerates $f(x)$ over the entire field and we simply set $\delta = 1$ and $x_0 = 0$. As subtraction modulo p is typically faster than addition, instead of (4) we use

$$f(x_0 + (n+1)\delta) = f(x_0 + n\delta) - (-\boldsymbol{\Delta} f)(x_0 + n\delta). \tag{5}$$

The necessary initial values are then $(-1)^k \boldsymbol{\Delta} f(0)$.

Algorithm 1 (Point Counting over $\mathbb{F}_p$) *Given a polynomial $f(x)$ over $\mathbb{F}_p$ of odd degree d and a vector M identifying nonzero quadratic residues in $\mathbb{F}_p$:*

1. Set $t_k \leftarrow (-1)^k \boldsymbol{\Delta}^k f(0)$, for $0 \le k \le d$, and set $N \leftarrow 1$.

2. For i from 1 to p:
   (a) If $t_0 = 0$, set $N \leftarrow N + 1$, and if $M[t_0]$, set $N \leftarrow N + 2$.
   (b) Set $t_0 \leftarrow t_0 - t_1$, $t_1 \leftarrow t_1 - t_0$, ..., and $t_{d-1} \leftarrow t_{d-1} - t_d$.

Output N.

The computation $t_k = t_k - t_{k+1}$ is performed using integer subtraction, adding p if the result is negative. The map M is computed by enumerating the polynomial $f(x) = x^2$ for x from 1 to $(p-1)/2$ and setting $M[f(x)] = 1$, using a total of p subtractions (and no multiplications).

The size of M may be cut in half by only storing residues less than $p/2$. One then uses $M[\min(t_0, p - t_0)]$, inverting $M[p - t_0]$ when $p \equiv 3 \bmod 4$. This slows down the algorithm, but is worth doing if M exceeds the size of cache memory.

It remains only to compute $\boldsymbol{\Delta}^k f(0)$. We find that

$$\boldsymbol{\Delta}^k f(0) = \sum_j k! \begin{Bmatrix} j \\ k \end{Bmatrix} f_j = \sum_j T_{j,k} f_j, \tag{6}$$

where the bracketed coefficient denotes a Stirling number of the second kind. The triangle of values $T_{j,k}$ is represented by sequence A019538 in the OEIS [18]. Since (6) does not depend on p, it is computed just once for each $k \le d$.

In the process of enumerating $f(x)$, we can also enumerate $f(x) + g(x)$ with $e + 1$ additional field subtractions, where e is the degree of $g(x)$. The case where $g(x)$ is a small constant is particularly efficient, since nearby entries in M are used. The last two columns in Table 1 show the amortized cost per point of applying this approach to the curves $y^2 = f(x)$, $f(x) + 1$, ..., $f(x) + 31$.

4 Group Computations

The performance of generic group algorithms is typically determined by two quantities: the time required to perform a group operation, and the number of operations performed. We briefly mention two techniques that reduce the former, then consider the latter in more detail.

	Polynomial Evaluation		Finite Differences		Finite Differences $\times 32$	
$p \approx$	Genus 2	Genus 3	Genus 2	Genus 3	Genus 2	Genus 3
2^{16}	195.1	257.2	6.1	7.8	1.1	1.1
2^{17}	196.3	262.6	6.0	6.9	1.1	1.1
2^{18}	192.4	259.8	6.0	6.8	1.1	1.1
2^{19}	186.3	251.1	6.0	6.8	1.1	1.1
2^{20}	187.3	244.1	7.2	8.0	1.1	1.3
2^{21}	172.3	240.8	8.8	9.4	1.2	1.3
2^{22}	197.9	233.9	12.1	13.4	1.2	1.3
2^{23}	229.2	285.8	12.8	14.6	2.6	2.7
2^{24}	258.1	331.8	41.2	44.0	3.5	4.7
2^{25}	304.8	350.4	53.6	55.7	4.8	4.9
2^{26}	308.0	366.9	65.4	67.8	4.8	4.6
2^{27}	318.4	376.8	70.5	73.1	4.9	5.0
2^{28}	332.2	387.8	74.6	76.5	5.1	5.2

Table 1. Point counting $y^2 = f(x)$ over $\mathbb{F}_p$ (CPU nanoseconds/point)

The middle rows of Table 1 show the transition of M from $L2$ cache to general memory. The top section of the table is the most relevant for the algorithms considered here, as asymptotically superior methods are used for larger p.

4.1 Faster Black Boxes

The performance of the underlying finite field operations used to implement the group law on the Jacobian can be substantially improved using a Montgomery representation to perform arithmetic modulo p [14]. Another optimization due to Montgomery that is especially useful for the algorithms considered here is the simultaneous inversion of field elements (see [3, Alg. 11.15]).[2] With an affine representation of the Jacobian each group operation requires a field inversion, but uses fewer multiplications than alternative representations. To ameliorate the high cost of field inversions, we then modify our algorithms to perform group operations "in parallel".

In the baby-steps giant-steps algorithm, for example, we fix a small constant n, compute n "babies" β, β^2, ..., β^n, then march them in parallel using steps of size n (the giant steps are handled similarly). In each parallel step we execute n group operations to the point where a field inverse is required, perform all the field inversions together for a cost of $3n - 3$ multiplications and one inversion, then use the results to complete the group operations. Exponentiation can also benefit from parallelization, albeit to a lesser extent.

These two optimizations are most effective when applied in combination, as may be seen in Table 2.

---

[2] This algorithm can be applied to any group.

| | | Standard | | | Montgomery | | |
g	p	×1	×10	×100	×1	×10	×100
1	$2^{20}+7$	501	245	215	239	89	69
1	$2^{25}+35$	592	255	216	286	93	69
1	$2^{30}+3$	683	264	217	333	98	69
2	$2^{20}+7$	1178	933	902	362	216	196
2	$2^{25}+35$	1269	942	900	409	220	197
2	$2^{30}+3$	1357	949	902	455	225	196
3	$2^{20}+7$	2804	2556	2526	642	498	478
3	$2^{25}+35$	2896	2562	2528	690	502	476
3	$2^{30}+3$	2986	2574	2526	736	506	478

Table 2. Black box performance (CPU nanoseconds/group operation)

The heading $\times n$ indicates n group operations performed "in parallel". All times are for a single thread of execution.

4.2 Generic Order Computations

Our approach to computing $\#J(C/\mathbb{F}_q) = L_q(1)$ is based on a generic algorithm to compute the structure of an arbitrary abelian group [23]. We are aided both by absolute bounds on $L_q(1)$ derived from the Weil conjectures (theorems), as well as predictions regarding its distribution within these bounds based on a generalized form of the Sato-Tate conjecture (proven for most genus 1 curves over $\mathbb{Q}$ in [7]). We first consider the general algorithm.

We assume we have a black box for an abelian group G (written multiplicatively) that can generate uniformly random group elements. For Jacobians, these can be obtained via decompression techniques [3, 14.1-2].[3] We also suppose we are given bounds M_0 and M_1 such that $M_0 \leq |G| \leq M_1$.

The first (typically only) step is to compute the group exponent, $\lambda(G)$, the least common multiple of the orders of all the elements of G. This is accomplished by initially setting $E = 1$, and for a random $\alpha \in G$, computing the order of $\beta = \alpha^E$ using a baby-steps giant-steps search on the interval $[M_0/E, M_1/E]$. We then update $E \leftarrow |\beta|E$ and repeat the process until either (1) there is only one multiple of E in the interval $[M_0, M_1]$, or (2) we have generated c random elements, where c is a confidence parameter. In the former case we must have $|G| = E$, and in the latter case $E = \lambda(G)$, with probability greater than $1 - 2^{2-c}$ [23, Proposition 8.3]. For large Jacobians, (1) almost always applies, however for the relatively small groups considered here, (2) arises more often, particularly when $g > 1$. Fortunately, this does not present undue difficulty.

Proposition 2. *Given* $\lambda(G)$ *and* M_0 *such that* $M_0 \leq |G| < 2M_0$, *the value of* $|G|$ *can be computed using* $O(|G|^{1/4})$ *group operations.*

---

[3] This becomes costly when $g > 2$, where we use the simpler approach of [3, p. 307].

Proof (sketch). The bounds on $|G|$ imply that it is enough to know the order of all but one of the p-Sylow subgroups of G (the p dividing $|G|$ are obtained from $\lambda(G)$). Following Algorithm 9.1 of [23], we use $\lambda(G)$ to compute the order of each p-Sylow subgroup $H \subseteq G$ using $O(|H|^{1/2})$ group operations; however, we abandon the computation for any p-Sylow subgroup that proves to be larger than $\sqrt{|G|}$. This can happen at most once, and the remaining successful computations uniquely determine $|G|$ within the interval $[M_0, 2M_0)$. $\square$

From the Weil interval (see (8) in section 4.4) we find that $M_1 < 2M_0$ for all $q > 300$ and $g \le 3$. Proposition 2 implies that group structure computations will not impact the complexity of our task. Indeed, computing $\#J(C/\mathbb{F}_q)$ is almost always dominated by the first computation of $|\beta|$.

Given $\beta \in G$ and the knowledge that the interval $[M_0, M_1]$ contains an integer M for which $\beta^M = 1_G$, a baby-steps giant-steps search may be used to find such an M. This is not necessarily the order of β, it is a multiple of it. We can then factor M and compute $|\beta|$ using $\tilde{O}(\lg M)$ group operations [23, Ch. 7]. The time to factor M is negligible in genus 2 and 3 (compared to the group computations), and in genus 1 we note that if a sieve is used to enumerate the primes up to N, the factorization of every M in the interval $[M_0, M_1]$ can be obtained at essentially no additional cost, using $O(\sqrt{N})$ bytes of memory.

An alternative approach avoids the computation of $|\beta|$ from M by attempting to prove that M is the only multiple of $|\beta|$ in the interval. Write $[M_0, M_1]$ as $[C - R, C + R]$, and suppose the search to find $M = C \pm r$ has shown $\beta^n \ne 1_G$ for all n in $(C - r, C + r)$. If M is not the only multiple of $|\beta|$ in $[C - R, C + R]$, then $|\beta|$ is a divisor of M satisfying $2r \le |\beta| \le R + r$. In particular, if P is the largest prime factor of M and $P > R + r$ and $M/P < 2r$, then M must be unique. When $R = O(M^{1/2})$ this happens fairly often (about half the time). When it does not happen, one can avoid an $\tilde{O}(\lg M)$ order computation at the cost of $O(R^{1/2})$ group operations by searching the remainder of the interval *on the opposite side of M*. This is only worthwhile when R is quite small, but can be helpful in genus 1.[4]

4.3   Optimized Baby-Steps Giant-Steps in the Jacobian - Part I

The Mumford representation of $J(C/\mathbb{F}_q)$ uniquely represents a reduced divisor of the curve $y^2 = f(x)$ by a pair of polynomials (u, v). The polynomial u is monic, with degree at most g, and divides $v^2 - f$ [3, p. 307]. The inverse of (u, v) is simply $(u, -v)$, which makes two facts immediate:

1. The cost of group inversions is effectively zero.
2. The element (u, v) has order 2 if and only if $v = 0$ and u divides f.

Fact 1 allows us to apply the usual optimization for fast inverses [2, p. 250], reducing the number of group operations by a factor of $\sqrt{2}$ (we no longer count inversions). Fact 2 gives us a bijection between the 2-torsion subgroup of $J(C/\mathbb{F}_q)$

---

[4] These ideas were sparked by a conversation with Mark Watkins, who also credits Geoff Bailey.

and polynomials dividing f of degree at most g (exactly half the polynomials dividing f). If k counts the irreducible polynomials in the unique factorization of f, then the 2-rank of $J(C/\mathbb{F}_q)$ is $k-1$ and 2^{k-1} divides $\#J(C/\mathbb{F}_q)$.[5]

When $k > 1$, we start with $E = 2^{k-1}$ in our computation of $\lambda(G)$ above, reducing the number of group operations by a factor of $2^{(k-1)/2}$. Otherwise, we know $\#J(C/\mathbb{F}_q)$ is odd and can reduce the number of group operations by a factor of $\sqrt{2}$. The total expected benefit of fast inversions and knowledge of 2-rank is at least a factor of 2.10 in genus 1, 2.31 in genus 2, and 2.48 in genus 3.

4.4 Optimized Baby-Steps Giant-Steps in the Jacobian - Part II

We come now to the most interesting class of optimizations, those based on the distribution of $\#J(C/\mathbb{F}_q)$. The Riemann hypothesis for curves (proven by Weil) states that $L_q(T)$ has roots lying on a circle of radius $q^{-1/2}$ about the origin of the complex plane. As $L_q(T)$ is a real polynomial of even degree with $L_q(0) = 1$, these roots may be grouped into conjugate pairs.

Definition 3. A unitary symplectic polynomial $p(z)$ is a real polynomial of even degree with roots $\alpha_1, ... \alpha_g, \bar{\alpha}_1, ... \bar{\alpha}_g$ all on the unit circle.

The unitary symplectic polynomials are precisely those arising as the characteristic polynomial of a unitary symplectic matrix. The Riemann hypothesis for curves implies that $p(z) = L_q(zq^{-1/2})$ is a unitary symplectic polynomial. The coefficients of $p(z) = \sum a_j z^j$ may be bounded by

$$|a_j| \leq \binom{2g}{j}. \tag{7}$$

The corresponding bounds on the coefficients of $L_q(T)$ constrain the value of $L_q(1) = \#J(C/\mathbb{F}_q)$, yielding the Weil interval

$$(\sqrt{q} - 1)^{2g} \leq \#J(C/\mathbb{F}_q) \leq (\sqrt{q} + 1)^{2g}. \tag{8}$$

For the a_j with j odd, the well known bounds in (7) are tight, however for even j they are not. We are particularly interested in the coefficient a_2.

Proposition 4. Let $p(z) = \sum a_j z^j$ be a unitary symplectic polynomial of degree $2g$. For fixed a_1, a_2 is bounded by an interval of radius at most g. In fact

$$a_2 \leq g + \left(\frac{g-1}{2g}\right) a_1^2; \tag{9}$$

$$a_2 \geq -g + 2 + \left(a_1^2 - \delta^2\right)/2. \tag{10}$$

The value $\delta \leq 2$ is the distance from a_1 to the nearest integer congruent to 0 mod 4 (when g is odd), or 2 mod 4 (when g is even).

---

[5] Computing k requires only a distinct-degree factorization of f, see [2, Alg. 3.4.3].

Proof. Define $\beta_j = \alpha_j + \bar{\alpha}_j$ for $1 \le j \le g$, where the α_j are the roots of $p(z)$. Then $a_1 = \sum \beta_j$ and $a_2 = g + (a_1^2 - t_2)/2$, where $t_2 = \sum \beta_j^2$. For fixed a_1, t_2 is minimized by $\beta_j = a_1/g$, yielding (9), and t_2 is maximized by $\beta_j = \pm 2$ for $j < g$ and $\beta_g = \delta$, yielding (10) (note that $|\beta_j| \le 2$). The proposition follows. $\square$

We have as a corollary, independent of a_1, the bound $a_2 \ge -g$, and for g odd, $a_2 \ge 2 - g$. In genus 2, the proposition reduces to Lemma 1 of [13], however we are especially interested in the genus 3 case, where our estimate of a_2 will determine the leading constant factor in the time to compute $\#J(C/\mathbb{F}_q)$. In genus 3, Proposition 4 constrains a_2 to an interval of radius 3 once a_1 is known, whereas (7) would give a radius of 15.

Having bounded the interval as tightly as possible, we consider the search within. We suppose we are seeking the value of a random variable X with some distribution over $[M_0, M_1]$. We assume that we start from an initial estimate M and search outward in both directions using a standard baby-steps giant-steps search with all baby steps taken first (see [20] for a more general analysis). Ignoring the boundaries, the cost of the search is

$$c = s + 2|X - M|/s \tag{11}$$

group operations. As our cost function is linear in $|X - M|$, we minimize the mean absolute error in our estimate by setting M to the median value of X and $s = \sqrt{2E}$, where E is the expectation of $|X - M|$. This holds for any distribution on X, we simply need the median value of X and its expected distance from it.

If we consider $p(z) = L_q(zq^{-1/2})$ as a "random" unitary symplectic polynomial, a natural distribution for $p(z)$ can be derived from the Haar measure on the compact Lie group $USp(2g)$ (the group of $2g \times 2g$ matrices over $\mathbb{C}$ that are both unitary and symplectic). Each $p(z)$ corresponds to a conjugacy class of matrices with $p(z)$ as their characteristic polynomial. Let the eigenvalues of a random matrix in $USp(2g)$ be $e^{\pm i\theta_1}, \ldots, e^{\pm i\theta_g}$, with $\theta_j \in [0, \pi)$. The joint probability density function on the θ_j given by the Haar measure on $USp(2g)$ is

$$\mu(USp(2g)) = \frac{1}{g!} \left(\prod_{j<k} (2\cos\theta_j - 2\cos\theta_k) \right)^2 \prod_j \frac{2}{\pi} \sin^2\theta_j d\theta_j. \tag{12}$$

This distribution is derived from the Weyl integration formula [26, p. 218] and can be found in [11, p. 107]. For $g = 1$, this simplifies to $(2/\pi)\sin^2\theta d\theta$, which corresponds to the Sato-Tate distribution. We may apply (12) to compute various statistical properties of random unitary symplectic polynomials. The coefficient a_1 is simply the negative sum of the eigenvalues,

$$a_1 = - \sum_{j=1}^{g} 2\cos\theta_j, \tag{13}$$

and we find that the median (and expectation) of a_1 is 0. In genus 1, the expected distance of a_1 from its median is

$$\mathbf{E}\left[|a_1|\right] = \frac{2}{\pi} \int_0^{\pi} |2\cos\theta| \sin^2\theta d\theta = \frac{8}{3\pi}. \tag{14}$$

The value $8/(3\pi) \approx 0.8488$ is not much smaller than 1, which corresponds to a uniform distribution, so the potential benefit is small in genus 1. In genus 2, however, the expected distance of a_1 from its median is $4096/(625\pi^2) \approx 0.7905$, versus an expected distance of 2 for the uniform distribution. The corresponding values for genus 3 are ≈ 0.7985 and 3.

Given the value of a_1 we can take this approach further, computing the median and expected distance for a_2 conditioned on a_1. Applying (12), we pre-compute a table of median and expected distance values for a_2 for various ranges of a_1. In genus 3, we find that the largest expected distance for a_2 given a_1 is about 0.66, much smaller than the value 7.5 for a uniform distribution of a_2 over the interval given by (7).

Of course such optimizations are effective only when the polynomials $L_p(T)$ for a particular curve and relatively small values of p actually correspond to (apparently) random unitary symplectic polynomials. For $g > 1$, it is not known whether this occurs at all, even as $p \to \infty$.[6] In genus 1, while the Sato-Tate conjecture is now largely proven over $\mathbb{Q}$ [7], the convergence rate remains the subject of conjecture. Indeed, the investigation of such questions was one motivation for undertaking these computations. It is only natural to ask whether our assumptions are met.

| Histogram of actual a_2 values | Distribution of a_2 given by (12) |
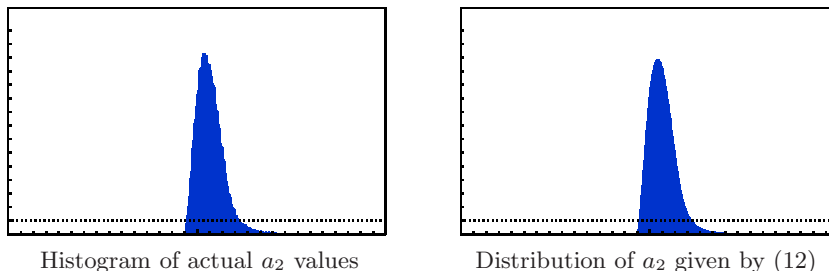
The figure on the left is a histogram of a_2 coefficient values obtained by computing $L_p(T)$ for $p \le 2^{24}$ for an arbitrarily chosen genus 3 curve (see Table 6). The figure on the right is the distribution of a_2 predicted by the Haar measure on $USp(2g)$, obtained by numerically integrating

$$a_2 = g + \prod_{j<k} 4\cos\theta_j \cos\theta_k \qquad (15)$$

over the distribution in (12). The dotted lines show the height of the uniform distribution. Similarly matching graphs are found for the other coefficients.

This remarkable degree of convergence is typical for a randomly chosen curve. We should note, however, that the generalized form of the Sato-Tate conjecture considered here applies only to curves whose Jacobian over $\mathbb{Q}$ has a trivial endomorphism ring (isomorphic to $\mathbb{Z}$), so there are exceptional cases. In genus 1 these are curves with complex multiplication. In higher genera, other exceptional cases occur, such as the genus 2 QM-curves considered in [10].

---

[6] Results are known for certain universal families of curves, e.g. [11, Thm. 10.8.2].

5 Results

To compare different methods for computing $L_p(T)$ and to assess the feasible range of L-series computations, we conducted extensive performance tests. Our test platform consisted of eight networked PCs, each equipped with a 2.5GHz AMD Athlon processor running a 64-bit Linux operating system. The point-counting and generic group algorithms were implemented using the techniques described in this paper, and we incorporated David Harvey's source code for the p-adic computations (the algorithm of [9], including recent improvements described in [8]). All code was compiled with the GNU C/C++ compiler using the options "-O2 -m64 -mtune=k8" [19].

In genus 1 there are several existing implementations of the computation contemplated here: given an elliptic curve defined over $\mathbb{Q}$, determine the coefficient a_1 of $L_p(T) = pT^2 + a_1T + 1$ for all $p \leq N$. We were able to compare our implementation with two software packages specifically optimized for this purpose: Magma [1], and the PARI library [24] as incorporated in SAGE [21]. The range of N we could use in this comparison was necessarily limited; results for larger N may be found in Table 5.

N	PARI	Magma	smalljac
2^{16}	0.26	0.29	0.07
2^{17}	0.55	0.59	0.15
2^{18}	1.17	1.24	0.30
2^{19}	2.51	2.53	0.62
2^{20}	5.46	5.26	1.29
2^{21}	11.67	11.09	2.65
2^{22}	25.46	23.31	5.53
2^{23}	55.50	49.22	11.56
2^{24}	123.02	104.50	24.31
2^{25}	266.40	222.56	51.60
2^{26}	598.16	476.74	110.29
2^{27}	1367.46	1017.55	233.94
2^{28}	3152.91	2159.87	498.46
2^{29}	7317.01	4646.24	1065.28
2^{30}	17167.29	10141.28	2292.74

Table 3. L-series computations in genus 1 (CPU seconds)

Each row lists CPU times for a single thread of execution to compute the coefficient a_1 of $L_p(T)$ for all $p \leq N$, using the elliptic curve $y^2 = x^3 + 314159x + 271828$. In SAGE, the function aplist(N) performs this computation via the PARI function *ellap*(N). The corresponding function in Magma is *TracesOfFrobenius*(N). The column labeled "smalljac" list times for our implementation.

Before undertaking similar computations in genus 2 and 3, we first determined the appropriate algorithm to use for various ranges of p using Table 4.

Each row gives timings for the algorithms considered here, averaged over a small sample of primes of similar size.

$p \approx 2^k$	Genus 2 – $L_p(T)$			Genus 3 – $L_p(T)$		Genus 3 – a_1
	pts/grp	group	p-adic	pts/grp	p-adic/grp	points
2^{14}	**0.22**	0.55	4	**10**	15	**0.12**
2^{15}	**0.34**	0.88	6	**21**	23	**0.23**
2^{16}	**0.56**	1.33	8	43	**31**	**0.45**
2^{17}	**0.98**	2.21	11	82	**40**	**0.89**
2^{18}	**1.82**	3.42	17		**51**	**1.78**
2^{19}	**3.44**	5.87	27		**67**	**3.57**
2^{20}	**7.98**	10.1	40		**97**	**8.48**
2^{21}	18.9	**17.9**	66		**148**	**19.7**
2^{22}	52	**35**	104		**212**	**56**
2^{23}		**54**	176		**355**	**123**
2^{24}		**104**	288		**577**	738
2^{25}		**173**	494		**995**	1870
2^{26}		**306**	871		**1753**	4550
2^{27}		**505**	1532		**3070**	9800

Table 4. $L_p(T)$ computations (CPU milliseconds)

Random curves of the appropriate genus were generated with coefficients uniformly distributed over $[1, 2^k)$. The polynomial $L_p(T)$ was then computed for 100 primes $\approx 2^k$, with the average CPU time listed. Columns labeled "pts/grp" compute a_1 by point counting over $\mathbb{F}_p$, followed by a group computation to obtain $L_p(T)$. The column "p-adic/grp" computes $L_p(T)$ mod p, then applies a group computation to get $L_p(T)$. The rightmost column computes just the coefficient a_1, via point counting over $\mathbb{F}_p$.

The task of computing L-series coefficients is well-suited to parallel computation. We implemented a simple distributed program which partitions the range $[1, N]$ into subintervals I_1, I_2, ..., I_m, distributes the task of computing $L_p(T)$ for $p \in I_m$ to n CPUs on a network, then collects and collates the results. This is useful even on a single computer whose microprocessor may have two or more cores. On our 8 node test platform we had 16 CPUs available for computation. Tables 5 and 6 lists elapsed times for L-series computations in single and 8-node configurations.

For practical reasons, we limited the duration of any single test. Larger computations could be undertaken with additional time and/or computing resources, without requiring software modifications. As they stand, the results extend to values of N substantially larger than any we could find in the literature.

Source code for the software can be freely obtained under a GNU General Public License (GPL) and is expected to be incorporated into SAGE. It is a pleasure to thank William Stein for access to the SAGE computational resources at the University of Washington, and especially David Harvey for providing the code used for the p-adic computations.

	Genus 1			Genus 1	
N	$\times 1$	$\times 8$	N	$\times 1$	$\times 8$
2^{21}	1.5	0.5	2^{30}	20:43	2:41
2^{22}	3.1	0.7	2^{31}	45:13	5:52
2^{23}	6.3	1.1	2^{32}	1:45:45	13:12
2^{24}	13.3	2.0	2^{33}	4:24:50	32:51
2^{25}	28.2	4.2	2^{34}	10:16:11	1:16:18
2^{26}	59.2	8.1	2^{35}	23:15:58	2:52:47
2^{27}	126.2	16.6	2^{36}		6:29:46
2^{28}	271.3	35.1	2^{37}		14:44:33
2^{29}	578.0	74.5	2^{38}		33:11:08

Table 5. L-series computations in genus 1 (elapsed times)

For the elliptic curve $y^2 = x^3 + 314159x + 271828$, the coefficients of $L_p(T)$ were computed for all $p \leq N$. Columns labeled $\times n$ list total elapsed times (seconds or hh:mm:ss) for a computation performed on n nodes (two cores per node), including communication overhead and time spent collating responses.

	Genus 2		Genus 3		Genus 3 - a_1 only	
N	$\times 1$	$\times 8$	$\times 1$	$\times 8$	$\times 1$	$\times 8$
2^{16}	1	< 1	43	13	1	< 1
2^{17}	4	2	1:49	18	5	1
2^{18}	12	3	4:42	41	11	2
2^{19}	40	7	12:43	1:47	41	6
2^{20}	2:32	24	36:14	4:52	2:41	21
2^{21}	10:46	1:38	1:45:36	13:40	11:33	1:27
2^{22}	40:20	5:38	5:23:31	41:07	53:26	6:38
2^{23}	2:23:56	19:04	16:38:11	2:05:40	4:33:26	33:00
2^{24}	8:00:09	1:16:47		6:28:25	38:51:07	4:42:43
2^{25}	26:51:27	3:24:40		20:35:16		20:35:16
2^{26}		11:07:28				
2^{27}		36:48:52				

Table 6. L-series computations in genus 2 and 3 (elapsed times)

The coefficients of $L_p(T)$ were computed for the genus 2 and 3 hyperelliptic curves

$$y^2 = x^5 + 31419x^3 + 271828x^2 + 1644934x + 57721566;$$
$$y^2 = x^7 + 314159x^5 + 271828x^4 + 1644934x^3 + 57721566x^2 + 1618034x + 141421,$$

for all $p \leq N$ where the curves had good reduction. Columns labeled $\times n$ list total elapsed wall times (hh:mm:ss) for a computation performed on n nodes, including all overhead. The last two columns give times to compute just the coefficient a_1.

References

[1] J.J. Cannon and W. Bosma (Eds.), *Handbook of Magma functions*, 2.14 ed., 2007, available at `http://magma.maths.usyd.edu.au/magma/htmlhelp/MAGMA.htm`.

[2] Henri Cohen, *A course in computational algebraic number theory*, Springer, 1996.

[3] Henri Cohen (Ed.) et al., *Handbook of elliptic and hyperelliptic curve cryptography*, Chapman and Hall, 2006.

[4] Christopher Deninger and Anthony J. Scholl, *The Beilinson conjectures*, L-functions and Arithmetic (Durham 1989), London Math. Soc. Lecture Note Series, vol. 153, Cambridge University Press, 1991, pp. 173–209.

[5] Tim Dokchitser, *Computing special values of motivic L-functions*, Experimental Mathematics **13** (2004), no. 2, 137–149.

[6] Noam D. Elkies, *Elliptic and modular curves over finite fields and related computational issues*, Computational perspectives in number theory: Proceedings of a conference in honor of A.O.L. Atkin (D.A. Buell and J.T. Teitelbaum, eds.), AMS, 1998, pp. 21–76.

[7] Michael Harris, Nick Shepherd-Barron, and Richard Taylor, *A family of Calabi-Yau varieties and potential automorphy*, May 2006, preprint.

[8] David Harvey, *Faster polynomial multiplication via multipoint Kronecker substitution*, 2007, preprint, `http://arxiv.org/abs/0712.4046`.

[9] ———, *Kedlaya's algorithm in larger characteristic*, Int Math Res Notices (2007).

[10] Ki Ichiro Hashimoto and Hiroshi Tsunogai, *On the Sato-Tate conjecture for QM-curves of genus two*, Mathematics of Computation **68** (1999), no. 228, 1649–1662.

[11] Nicholas M. Katz and Peter Sarnak, *Random matrices, Frobenius eigenvalues, and monodromy*, American Mathematical Society, 1999.

[12] Kiran Kedlaya, *Counting points on hyperelliptic curves using Monsky-Washnitzer cohomology*, Journal of the Ramanujan Mathematical Society **16** (2001), 332–338.

[13] Kazuto Matsuo, Jinhui Chao, and Shigeo Tsujii, *An improved baby step giant step algorithm for point counting of hyperelliptic curves over finite fields*, Algorithmic Number Theory Symposium–ANTS V, LNCS, vol. 2369, 2002, pp. 461–474.

[14] Peter L. Montgomery, *Modular multiplication without trial division*, Mathematics of Computation **44** (1985), no. 170, 519–521.

[15] J. Pila, *Frobenius maps of abelian varieties and finding roots of unity in finite fields*, Mathematics of Computation **55** (1990), no. 102, 745–763.

[16] René Schoof, *Counting points on elliptic curves over finite fields*, Jounral de Théorie des Nombres de Bordeaux **7** (1995), 219–254.

[17] J.Silverman, *Advanced topics in the arithmetic of elliptic curves*, Springer, 1999.

[18] N. J. A. Sloane, *The on-line encyclopedia of integer sequences*, 2007, `www.research.att.com/~njas/sequences/`.

[19] Richard Stallman et al., *GNU compiler collection 4.1.2*, February 2007, available at `http://gcc.gnu.org/index.html`.

[20] Andreas Stein and Edlyn Teske, *Optimized baby step-giant step methods*, Journal of the Ramanujan Mathematical Society **20** (2005), no. 1, 1–32.

[21] William Stein and David Joyner, *SAGE: System for Algebra and Geometry Experimentation*, Communications in Computer Algebra (SIGSAM Bulletin) (2005), version 2.8.5 (September 2007), available at `http://sage.sourceforge.net/`.

[22] Andrew V. Sutherland, *A generic approach to searching for Jacobians*, Math. Comp. (to appear), `http://arxiv.org/abs/0708.3168`.

[23] ———, *Order computations in generic groups*, PhD thesis, M.I.T., 2007, `http://groups.csail.mit.edu/cis/theses/sutherland-phd.pdf`.

[24] The PARI Group, Bordeaux, *PARI/GP, version* `2.3.2`, 2007, available from `http://pari.math.u-bordeaux.fr/`.

[25] André Weil, *Numbers of solutions of equations in finite fields*, Bulletin of the American Mathematical Society **55** (1949), 497–508.

[26] Hermann Weyl, *Classical groups*, second ed., Princeton University Press, 1946.