On the evaluation of modular polynomials

Andrew V. Sutherland

Massachusetts Institute of Technology

ANTS X — July 10, 2012

http://math.mit.edu/~drew
Let $\ell$ be a prime and let $\mathbb{F}_q$ be a finite field (assume $q$ is prime).

**Problem:** Given an elliptic curve $E/\mathbb{F}_q$, identify any and all curves $E'/\mathbb{F}_q$ that are $\ell$-isogenous to $E$.

This problem arises in many applications, and it is often the computationally dominant step.
Let $\ell$ be a prime and let $\mathbb{F}_q$ be a finite field (assume $q$ is prime).

**Problem:** Given an elliptic curve $E/\mathbb{F}_q$, identify any and all curves $E'/\mathbb{F}_q$ that are $\ell$-isogenous to $E$.

This problem arises in many applications, and it is often the computationally dominant step.

**Solution:** Compute the polynomial

$$\phi_\ell(Y) = \Phi_\ell(j(E), Y),$$

and find its roots in $\mathbb{F}_q$. Here $\Phi_\ell \in \mathbb{Z}[X, Y]$ is the classical modular polynomial that parameterizes pairs of $\ell$-isogenous elliptic curves.

If $\ell$ is at all large, say $\ell = \Omega(\log q)$, the hard part is computing $\phi_\ell$. Finding its roots is easy by comparison.
The Modular Polynomial $\Phi_\ell(X, Y)$

$\Phi_\ell \in \mathbb{Z}[X, Y]$ is symmetric, with degree $\ell + 1$ in both $X$ and $Y$. Its total size is $O(\ell^3 \log \ell)$ bits.

<table>
<thead>
<tr>
<th>$\ell$</th>
<th>coefficients</th>
<th>largest</th>
<th>average</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>127</td>
<td>8258</td>
<td>7.5kb</td>
<td>5.3kb</td>
<td>5.5MB</td>
</tr>
<tr>
<td>251</td>
<td>31880</td>
<td>16kb</td>
<td>12kb</td>
<td>48MB</td>
</tr>
<tr>
<td>503</td>
<td>127262</td>
<td>36kb</td>
<td>27kb</td>
<td>431MB</td>
</tr>
<tr>
<td>1009</td>
<td>510557</td>
<td>78kb</td>
<td>60kb</td>
<td>3.9GB</td>
</tr>
<tr>
<td>2003</td>
<td>2009012</td>
<td>166kb</td>
<td>132kb</td>
<td>33GB</td>
</tr>
<tr>
<td>3001</td>
<td>4507505</td>
<td>259kb</td>
<td>208kb</td>
<td>117GB</td>
</tr>
<tr>
<td>4001</td>
<td>8010005</td>
<td>356kb</td>
<td>287kb</td>
<td>287GB</td>
</tr>
<tr>
<td>5003</td>
<td>12522512</td>
<td>454kb</td>
<td>369kb</td>
<td>577GB</td>
</tr>
<tr>
<td>10009</td>
<td>50085038</td>
<td>968kb</td>
<td>774kb</td>
<td>4.8TB</td>
</tr>
</tbody>
</table>

Size of $\Phi_\ell(X, Y)$
The Modular Polynomial $\Phi_\ell(X, Y)$

$\Phi_\ell \in \mathbb{Z}[X, Y]$ is symmetric, with degree $\ell + 1$ in both $X$ and $Y$. Its total size is $O(\ell^3 \log \ell)$ bits.

<table>
<thead>
<tr>
<th>$\ell$</th>
<th>coefficients</th>
<th>largest</th>
<th>average</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>127</td>
<td>8258</td>
<td>7.5kb</td>
<td>5.3kb</td>
<td>5.5MB</td>
</tr>
<tr>
<td>251</td>
<td>31880</td>
<td>16kb</td>
<td>12kb</td>
<td>48MB</td>
</tr>
<tr>
<td>503</td>
<td>127262</td>
<td>36kb</td>
<td>27kb</td>
<td>431MB</td>
</tr>
<tr>
<td>1009</td>
<td>510557</td>
<td>78kb</td>
<td>60kb</td>
<td>3.9GB</td>
</tr>
<tr>
<td>2003</td>
<td>2009012</td>
<td>166kb</td>
<td>132kb</td>
<td>33GB</td>
</tr>
<tr>
<td>3001</td>
<td>4507505</td>
<td>259kb</td>
<td>208kb</td>
<td>117GB</td>
</tr>
<tr>
<td>4001</td>
<td>8010005</td>
<td>356kb</td>
<td>287kb</td>
<td>287GB</td>
</tr>
<tr>
<td>5003</td>
<td>12522512</td>
<td>454kb</td>
<td>369kb</td>
<td>577GB</td>
</tr>
<tr>
<td>10009</td>
<td>50085038</td>
<td>968kb</td>
<td>774kb</td>
<td>4.8TB</td>
</tr>
</tbody>
</table>

Size of $\Phi_\ell(X, Y)$

But for $\ell = 10009$ and $q \approx 2^{256}$, the size of $\phi_\ell$ is just 320KB! Even with $\log q \approx \ell = 10009$ it is under 20MB.
"... Despite this progress, computing modular polynomials remains the stumbling block for new point counting records. Clearly, to circumvent the memory problems, one would need an algorithm that directly obtains the polynomial specialised in one variable."

INRIA Project-Team TANC Report

This record was set in December 2006.
Computing $\Phi_\ell$ with the CRT

Strategy: compute $\Phi_\ell \mod p$ for sufficiently many primes $p$ and use the CRT to compute $\Phi_\ell$ (or $\Phi_\ell \mod q$).

- For suitable primes $p$ we can compute $\Phi_\ell \mod p$ in time $O(\ell^2 \log^3 p \log \log p)$ using isogeny volcanoes [BLS 2011].
- Assuming the GRH, we can efficiently find sufficiently many such primes with $\log p = O(\log \ell)$.
- "Sufficiently many" is $O(\ell)$.

Henceforth we assume the GRH.
Computing $\Phi_\ell$ with the CRT

Strategy: compute $\Phi_\ell \mod p$ for sufficiently many primes $p$ and use the CRT to compute $\Phi_\ell$ (or $\Phi_\ell \mod q$).

- For suitable primes $p$ we can compute $\Phi_\ell \mod p$ in time $O(\ell^2 \log^3 p \log \ell)$ using isogeny volcanoes [BLS 2011].
- Assuming the GRH, we can efficiently find sufficiently many such primes with $\log p = O(\log \ell)$.
- "Sufficiently many" is $O(\ell)$.

Uses $O(\ell^3 \log^3 \ell \log \ell)$ expected time and $O(\ell^3 \log \ell)$ space.

Using the explicit CRT, we can directly compute $\Phi_\ell \mod q$ using $O(\ell^2 (n + \log \ell))$ space, where $n = \log q$.

But the size of $\phi_\ell$ is $O(\ell n)$.

---

Henceforth we assume the GRH.
Computing $\phi_\ell(Y)$ with the CRT (naïve approach)

**Strategy:** Lift $j(E)$ from $\mathbb{F}_q$ to $\mathbb{Z}$, compute $\Phi_\ell(X, Y) \mod p$ and evaluate

$$\phi_\ell(Y) = \Phi_\ell(j(E), Y) \mod p$$

for sufficiently many primes $p$. Obtain $\phi_\ell$ mod $q$ via the explicit CRT.

Uses $O(\ell^2 \log^{3+\epsilon} p)$ expected time for each $p$, and $O(\ell^2 \log p)$ space.
Computing $\phi_\ell(Y)$ with the CRT (naïve approach)

**Strategy:** lift $j(E)$ from $\mathbb{F}_q$ to $\mathbb{Z}$, compute $\Phi_\ell(X, Y) \mod p$ and evaluate

$$\phi_\ell(Y) = \Phi_\ell(j(E), Y) \mod p$$

for sufficiently many primes $p$. Obtain $\phi_\ell$ mod $q$ via the explicit CRT.

Uses $O(\ell^2 \log^{3+\epsilon} p)$ expected time for each $p$, and $O(\ell^2 \log p)$ space.

However, “sufficiently many” is now $O(\ell n)$, where $n = \log q$.
Total expected time is $O(\ell^3 n \log^{3+\epsilon} \ell)$, using $O(\ell n + \ell^2 \log \ell)$ space.

This approach is **not very useful**:

- If $n$ is large (e.g. $n \approx \ell$), it takes way too long (quartic in $\ell$).
- If $n$ is small (e.g. $n \approx \log \ell$), it doesn’t save any space.
Computing $\phi_\ell(Y)$ with the CRT (Algorithm 1)

Strategy: lift $j, j^2, j^3, \ldots, j^{\ell+1}$ from $\mathbb{F}_q$ to $\mathbb{Z}$ and then compute

$$\phi_\ell(Y) = \sum c_{ik} j^i Y^k \mod p$$

for sufficiently many primes $p$, where $\Phi_\ell = \sum c_{ik} X^i Y^k$. Obtain $\phi_\ell \mod q$ via the explicit CRT.
Computing $\phi_\ell(Y)$ with the CRT (Algorithm 1)

Strategy: lift $j, j^2, j^3, \ldots, j^{\ell+1}$ from $\mathbb{F}_q$ to $\mathbb{Z}$ and then compute

$$\phi_\ell(Y) = \sum c_{ik} j^i Y^k \mod p$$

for sufficiently many primes $p$, where $\Phi_\ell = \sum c_{ik} X^i Y^k$. Obtain $\phi_\ell \mod q$ via the explicit CRT.

Now “sufficiently many” is $O(\ell + n)$.

For $n = O(\ell \log \ell)$, uses $O(\ell^3 \log^3 + \epsilon \ell)$ expected time and $O(\ell^2 \log \ell)$ space (under GRH). For $n = \Omega(\ell \log \ell)$, the space bound is optimal.

This algorithm can also evaluate the partial derivatives of $\Phi_\ell$ needed to construct normalized equations for $\tilde{E}$ (important for SEA).
Computing $\phi_\ell(Y)$ with the CRT (Algorithm 2)

**Strategy:** lift $j(E)$ from $\mathbb{F}_q$ to $\mathbb{Z}$ and for sufficiently many primes $p$ compute $\phi_\ell \mod p$ as follows:

1. For each of $\ell + 2$ $j$-invariants $y_i$, compute $z_i = \prod_{k}(j(E) - j_k)$, where the $j_k$ range over $\ell + 1$ neighbors of $y_i$ in $G_\ell(\mathbb{F}_p)$.

2. Interpolate $\phi_\ell(Y) \in \mathbb{F}_p$ as the unique polynomial of degree $\ell + 1$ for which $\phi_\ell(y_i) = z_i$.

Obtain $\phi_\ell \mod q$ via the explicit CRT.
Computing $\phi_\ell(Y)$ with the CRT (Algorithm 2)

**Strategy:** lift $j(E)$ from $\mathbb{F}_q$ to $\mathbb{Z}$ and for sufficiently many primes $p$ compute $\phi_\ell \bmod p$ as follows:

1. For each of $\ell + 2$ $j$-invariants $y_i$, compute $z_i = \prod_k (j(E) - j_k)$, where the $j_k$ range over $\ell + 1$ neighbors of $y_i$ in $G_\ell(\mathbb{F}_p)$.

2. Interpolate $\phi_\ell(Y) \in \mathbb{F}_p$ as the unique polynomial of degree $\ell + 1$ for which $\phi_\ell(y_i) = z_i$.

Obtain $\phi_\ell \bmod q$ via the explicit CRT.

For $n = O(\ell^c)$, uses $O(\ell^3(n + \log \ell) \log^{1+\epsilon} \ell)$ expected time and $O(\ell n + \ell \log \ell)$ space (under GRH).

For $n = O(\log^{2-\epsilon} q)$ the algorithm is faster than computing $\Phi_\ell$.

For $n = \Omega(\log \ell)$ the space bound is optimal.
# Genus 1 point counting in large characteristic

## Algorithms to compute \( \#E(\mathbb{F}_q) = q + 1 - t \).  

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Totally naive</td>
<td>( O(e^{2n+\epsilon}) )</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>Slightly less naive</td>
<td>( O(e^{n+\epsilon}) )</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>Baby-step giant-step</td>
<td>( O(e^{n/4+\epsilon}) )</td>
<td>( O(e^{n/4+\epsilon}) )</td>
</tr>
<tr>
<td>Pollard kangaroo</td>
<td>( O(e^{n/4+\epsilon}) )</td>
<td>( O(n^2) )</td>
</tr>
<tr>
<td>Schoof</td>
<td>( O(n^5 \ l\log n) )</td>
<td>( O(n^3) )</td>
</tr>
<tr>
<td>SEA*</td>
<td>( O(n^4 \log^3 n \ l\log n) )</td>
<td>( O(n^3 \log n) )</td>
</tr>
<tr>
<td>SEA (( \Phi_\ell ) precomputed)</td>
<td>( O(n^4 \ l\log n) )</td>
<td>( O(n^4) )</td>
</tr>
</tbody>
</table>

*Complexity estimates for SEA-based algorithms are heuristic expected times.*
Genus 1 point counting in large characteristic

Algorithms to compute $\#E(\mathbb{F}_q) = q + 1 - t$.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Totally naive</td>
<td>$O(e^{2n+\epsilon})$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Slightly less naive</td>
<td>$O(e^{n+\epsilon})$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Baby-step giant-step</td>
<td>$O(e^{n/4+\epsilon})$</td>
<td>$O(e^{n/4+\epsilon})$</td>
</tr>
<tr>
<td>Pollard kangaroo</td>
<td>$O(e^{n/4+\epsilon})$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Schoof</td>
<td>$O(n^5 \ llog n)$</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>SEA*</td>
<td>$O(n^4 \ log^3 n \ llog n)$</td>
<td>$O(n^3 \ log n)$</td>
</tr>
<tr>
<td>SEA ($\Phi_\ell$ precomputed)</td>
<td>$O(n^4 \ llog n)$</td>
<td>$O(n^4)$</td>
</tr>
<tr>
<td>SEA with Algorithm 1</td>
<td>$O(n^4 \ log^2 n \ llog n)$</td>
<td>$O(n^2 \ log n)$</td>
</tr>
<tr>
<td>Amortized</td>
<td>$O(n^4 \ llog n)$</td>
<td>$O(n^2 \ log n)$</td>
</tr>
</tbody>
</table>

*Complexity estimates for SEA-based algorithms are heuristic expected times.
Alternative modular polynomials

In practice, the modular polynomials $\Phi_\ell$ are not used in SEA. There are alternatives (due to Atkin, Müller, and others) that are smaller by a large constant factor (100x to 1000x is typical).

The isogeny-volcano approach of [BLS 2010] can compute many types of (symmetric) modular polynomials derived from modular functions other than $j(z)$, but these do not include the modular polynomials commonly used with SEA.
Alternative modular polynomials

In practice, the modular polynomials $\Phi_\ell$ are not used in SEA. There are alternatives (due to Atkin, Müller, and others) that are smaller by a large constant factor (100x to 1000x is typical).

The isogeny-volcano approach of [BLS 2010] can compute many types of (symmetric) modular polynomials derived from modular functions other than $\tilde{j}(z)$, but these do not include the modular polynomials commonly used with SEA.

They do include modular polynomials $\Phi_\ell \tilde{f}$ derived from the Weber function $\tilde{f}(z)$, but these have never (?) been used with SEA before.

Provided $\text{End}(E)$ has discriminant $D \equiv 1 \mod 8$ with $3 \nmid D$, the polynomial $\phi_\ell \tilde{f}(Y) = \Phi_\ell \tilde{f}(\tilde{f}(E), Y)$ parameterizes $\ell$-isogenies from $E$.

This condition is easily checked (without knowing $D$). If it fails, powers of $\tilde{f}$, or other modular functions may be used.
The Weber function

The Weber \( f \)-function is defined by

\[
    f(\tau) = \frac{\eta((\tau + 1)/2)}{\zeta_{48}\eta(\tau)},
\]

and satisfies \( j(\tau) = (f(\tau)^{24} - 16)^3/f(\tau)^{24} \).

The coefficients of \( \Phi_{f\ell} \) are roughly 72 times smaller. This means we need 72 times fewer primes.

The polynomial \( \Phi_{f\ell} \) is roughly 24 times sparser. This means we need 24 times fewer interpolation points.

Overall, we get nearly a **1728-fold speedup** using \( \Phi_{f\ell} \).
Modular polynomials for $\ell = 11$

Classical:

\[
X^{12} + Y^{12} - X^{11}Y + 8184X^{11}Y^{10} - 28278756X^{11}Y^9 + 53686822816X^{11}Y^8 \\
- 61058988656490X^{11}Y^7 + 42570393135641712X^{11}Y^6 - 1789952672883039048X^{11}Y^5 \\
+ 429783238774928467520X^{11}Y^4 - 529134841844639613861795X^{11}Y^3 + 2720981165805664581552600X^{11}Y^2 \\
- 374642006356701393515817612X^{11}Y + 2964709023552405240575283200000X^{11} \\
+ 39242334509452765490869646240872004909952472337067462708993642064267017406194168673924546560000000000
\]

Atkin:

\[
X^{12} - X^{11}Y + 744X^{11} + 196680X^{10} + 187X^9Y + 21354080X^9 + 506X^8Y + 830467440X^8 \\
- 11440X^7Y + 16875327744X^7 - 57442X^6Y + 208564958976X^6 + 184184X^5Y + 1678582287360X^5 \\
+ 1675784X^4Y + 9031525113600X^4 + 1867712X^3Y + 3234997990400X^3 - 8252640X^2Y + 74246810880000X^2 \\
- 19849600XY + 9899773440000X + Y^2 - 8720000Y + 5841107200000
\]

Weber:

\[
X^{12} + Y^{12} - X^{11}Y + 11X^9Y^9 - 44X^7Y^7 + 88X^5Y^5 - 88X^3Y^3 + 32XY
\]
The number of points on the elliptic curve $E$ defined by
\[ y^2 = x^3 + 2718281828x + 3141592653, \]
modulo the 5011 digit prime $q = 16219299585 \cdot 2^{116} - 1$ is
Elliptic curve point counting record

<table>
<thead>
<tr>
<th>Task</th>
<th>Total CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute $\phi_\ell^f$</td>
<td>32 days</td>
</tr>
<tr>
<td>Find a root $\tilde{j}$</td>
<td>995 days</td>
</tr>
<tr>
<td>Compute $g_\ell$</td>
<td>3 days</td>
</tr>
<tr>
<td>Compute $\pi \mod g_\ell, E$</td>
<td>326 days</td>
</tr>
<tr>
<td>Find $\lambda_\ell$</td>
<td>22 days</td>
</tr>
</tbody>
</table>

$\phi_\ell^f(Y) = \Phi_\ell^f(j(E), Y)$ was computed for $\ell$ from 5 to 11681. Exactly 700 of 1400 were found to be Elkies primes. Atkin primes were not used.

The largest $\phi_\ell^f$ was under 20MB in size and took about two hours to compute using 1 core.
Modular polynomial evaluation record

For \(\ell = 100019\) and \(q = 2^{86243} - 1\) we computed \(\phi_{\ell}(Y) = \Phi_{\ell}(j(E), Y)\).

This is much larger than one would need to set a 25,000 digit point-counting record.

The size of \(\phi_{\ell}\) is about 1 GB.
For $\ell = 100019$ and $q = 2^{86243} - 1$ we computed $\phi_{\ell}(Y) = \Phi_{\ell}(j(E), Y)$.

This is much larger than one would need to set a 25,000 digit point-counting record.

The size of $\phi_{\ell}$ is about 1 GB.

For comparison:

- The size of $\Phi_{\ell} \mod q$ is about 2 TB.
For \( \ell = 100019 \) and \( q = 2^{86243} - 1 \) we computed \( \phi \ell(Y) = \Phi \ell(j(E), Y) \).

This is much larger than one would need to set a 25,000 digit point-counting record.

The size of \( \phi \ell \) is about 1 GB.

For comparison:

- The size of \( \Phi \ell \) mod \( q \) is about 2 TB.
- The size of \( \Phi \ell \) mod \( q \) is about 50 TB.
For $\ell = 100019$ and $q = 2^{86243} - 1$ we computed $\phi^\hat{\ell}(Y) = \Phi^\hat{\ell}(j(E), Y)$.

This is much larger than one would need to set a 25,000 digit point-counting record.

The size of $\phi^\hat{\ell}$ is about 1 GB.

For comparison:

- The size of $\Phi^\hat{\ell}$ mod $q$ is about 2 TB.
- The size of $\Phi^\ell$ mod $q$ is about 50 TB.
- The size of $\Phi^\ell$ is more than 10 PB.
Improved space complexity of computing horizontal isogenies

The algorithm of [Bisson-S 2011] for computing the endomorphism ring of an elliptic curve $E/\mathbb{F}_q$ runs in $L[1/2, \sqrt{3}/2]$ expected time and uses $L[1/2, 1/\sqrt{3}]$ space (under GRH).

The space complexity can now be improved to $L[1/2, 1/\sqrt{12}]$.

A similar improvement applies to algorithms for computing horizontal isogenies of large degree [Jao-Soukharev ANTS IX].