18.783 Elliptic Curves **Fall 2025**

Problem Set #2 **Due: 09/19/2025**

These problems are primarily related to Lecture 3. Note that the lecture notes include material we did not cover in class that you are expected to have read (this is particularly relevant to Problem 3). Your solutions should be written in LaTeX and submitted as a PDF file to Gradescope by midnight on the date due.

Instructions: Solve any combination of problems that sums to 100 points. Collaboration is permitted/encouraged, but you must identify your collaborators (including any LLMs you discussed the problem set with), as well as any references you consulted outside the syllabus or lecture notes. Include this information after the **Collaborators/Sources** prompt at the end of the problem set (if there are none, you should enter "none", do not leave it blank). Each student is expected to write their own solutions; it is fine to problems with others, but your writing must be your own.

Note: Several problems require a portion of your MIT ID as input. If you would prefer not to use your MIT ID, let me know and I will choose a random 9-digit number that you can use in place of your MIT ID for the purpose of solving 18.783 problem sets.

Problem 1. Cornacchia's algorithm (33 points)

Cornacchia's algorithm computes primitive solutions (x, y) to the Diophantine equation

$$x^2 + dy^2 = m, \tag{1}$$

where d and m are positive integers. A *primitive* solution has x and y relatively prime. Typically $m = p$ or $m = 4p$, where p is a prime, but the algorithm works for any m, provided we are given an appropriate square root r of $-d$ mod m (if there are only two square roots, as when m is prime p or $4p$ with p odd, it does not matter which we use, but in general one needs to check one of $\pm r$ for each square root r of $-d$ modulo m).[1]

The algorithm uses a partial Euclidean algorithm that terminates as soon as the sequence of remainders r_i drops below the square root of $r_0 = m$.

1. Let $r_0 = m$ and $r_1 = r$, where $r^2 \equiv -d \bmod m$ and $0 \le r \le m/2$.
2. Compute $r_{i+2} = r_i \bmod r_{i+1}$ until $r_k^2 < m$ is reached.
3. If $(m - r_k^2)/d$ is the square of an integer s, return the solution (r_k, s).
   Otherwise, return `null`.

It is clear that if the algorithm returns (r_k, s), then it is a solution to (1). It is not so clear that the algorithm always finds a primitive solution if one exists, but this is true; see [1] for a short elementary proof.[2] If m is square-free (as when m is prime), every solution to (1) is primitive, but this is not true in general (this is relevant to part (e)).

In this problem let $N := n \cdot 10^{100}$, where n is the last 4 digits of your student ID.

[1] Using the probabilistic root-finding algorithm described in Lecture 3 one can efficiently compute square roots modulo p, and one can use this to compute square roots modulo $4p$, or modulo any integer whose prime factorization is known. But in general, the problem of computing square roots modulo m is believed to be as hard as factoring m (for which no polynomial-time probabilistic algorithm is known).

[2] There is an obvious typo in step 3 of the algorithm given in [1], which is corrected above.

(a) Implement this algorithm in Sage. Use `mod(-d,m).is_square()` to test if $-d$ has a square root mod m, and use `int(mod(-d,m).sqrt())` to get a square root.

(b) You may recall Fermat's "Christmas theorem", which states that an odd prime p is the sum of two squares if and only if $p \equiv 1 \bmod 4$. You may also recall that -1 is a square modulo an odd prime p if and only if $p \equiv 1 \bmod 4$.

Let n be the integer corresponding to the last 4 digits of your student ID. For the least prime $p > N$ congruent to 1 mod 4, write p as the sum of two squares.

(c) Fermat also proved that a prime p can be written in the form $p = x^2 + 3y^2$ if and only if $p \equiv 1 \bmod 3$, which is equivalent to the condition that -3 is a square mod p. For the least prime $p > N$ congruent to 1 mod 3, write p in the form $p = x^2 + 3y^2$.

(d) Show that this does not work for $d = 5$ by finding a prime p for which -5 is a square modulo p but p cannot be written in the form $x^2 + 5y^2$. Empirically determine a stronger congruence condition on p that guarantees not only that -5 a square mod p, but also that p can be written in the form x^2+5y^2. Then find the least prime $p > N$ that satisfies your condition and write p in the form $p = x^2 + 5y^2$.

(e) Let E be the elliptic curve $y^2 = x^3 - 35x - 98$ and let $p \neq 2, 7$ be a prime. Like the elliptic curves considered in Problem 5 of Problem Set 1, the elliptic curve E has *complex multiplication*, and the integer $a_p = p + 1 - \#E(\mathbb{F}_p)$ is zero if and only if -7 is not a square modulo p (you can take this as given, we will prove it later in the course). When -7 is a square modulo p, the integer a_p satisfies the equation $4p = a_p^2 + 7y^2$, for some $y \in \mathbb{Z}$ (also take this as given, we will prove it later in the course). Prove that this equation has a solution (a_p, y) if and only if the equation $p = u^2 + 7v^2$ has a solution (u, v), and that any such solution is unique up to signs.

Use Cornacchia's algorithm to find a solution to $p = u^2 + 7v^2$ for the least prime $p > N$ for which -7 is a square modulo p, and use this to deduce the absolute value of a_p. Determine the sign of a_p, by finding a random point $P \in E(\mathbb{F}_p)$ for which exactly one of $(p + 1 - a_p)P$ and $(p + 1 + a_p)P$ is zero.

(f) Let E be the elliptic curve $y^2 = x^3 - 2818048320x + 57579881513616$ and let $p \neq 2, 3, 163$ be prime. The elliptic curve E has complex multiplication, the integer $a_p = p+1-\#E(\mathbb{F}_p)$ is zero if and only if -163 is not a square modulo p, and when -163 is a square modulo p, the integer a_p satisfies the equation $4p = a_p^2 + 163y^2$, for some $y \in \mathbb{Z}$ (take these facts as given).

Show that the equation $p = a_p^2 + 163y^2$ need not have a solution when $4 * p = u^2 + 163v^2$ has a solution. Where does your proof in (e) break down when you replace 7 with 163?

Use Cornacchia's algorithm to find a solution to $4p = u^2 + 163v^2$ for the least prime $p > N$ for which -163 is a square modulo p, and use this to deduce the absolute value of a_p. Determine the sign of a_p, by finding a random point $P \in E(\mathbb{F}_p)$ for which exactly one of $(p + 1 - a_p)P$ and $(p + 1 + a_p)P$ is zero.

Problem 2. Computing rth roots in cyclic groups (33 points)

In Lecture 3 we saw how to compute rth roots in a finite field $\mathbb{F}_q$ using a probabilistic root-finding algorithm. In this problem you will implement an entirely different approach

for computing rth roots that works in any finite cyclic group G, including $G = \mathbb{F}_q^\times$. In addition to being more general, this method is typically faster than using probabilistic root-finding to compute rth roots in $\mathbb{F}_q^\times$ (but this depends on the values of r and q).

We assume without loss of generality that r is prime (to compute nth roots, successively compute rth roots for the primes r dividing n, with multiplicity). To simplify notation we write G additively, so an *rth root* of $\gamma \in G$ is an element $\rho \in G$ for which $r\rho = \gamma$. Let $|\gamma|$ denote the *order* of γ, the least positive integer m for which $m\gamma = 0$.

Prove the following statements:

(a) For all $\gamma \in G$ and $n \in \mathbb{Z}$ we have $|n\gamma| = |\gamma| / \gcd(n, |\gamma|)$.

(b) If r does not divide $|G|$, then there is an integer s such that for all $\gamma \in G$ the element $\rho = s\gamma$ is the unique rth root of γ.

(c) If r does divide $|G|$, then the number of rth roots of each $\gamma \in G$ is either 0 or r. In the latter case, the rth roots of γ do not necessarily lie in $\langle\gamma\rangle$ (give an example).

(d) Suppose r divides $|G|$. Let $|G| = ar^k$, where $r \nmid a$. Let $\delta \in G$ be an element of order r^k, let γ be any element of G, and let $\alpha = a\gamma$ and $\beta = r^k\gamma$.

(i) $\alpha = x\delta$ for some integer $x \in [1, r^k]$.

(ii) If r does not divide x then there is no $\rho \in G$ for which $r\rho = \gamma$.

(iii) If r divides x, and s and t are integers satisfying $sa + tr^{k+1} = 1$, then the element $\rho = s(x/r)\delta + t\beta$ satisfies $r\rho = \gamma$.

The element δ is a generator for the r-Sylow subgroup of G. If $G = \langle\sigma\rangle$, we can use $\delta = a\sigma$. The integer x is the *discrete logarithm* of α with respect to δ.

Implement the following algorithm for computing an rth root of γ in a cyclic group G of order ar^k, where r is a prime that does not divide a, given $\delta \in G$ of order r^k:

1. If $k = 0$ then compute $s = 1/r \bmod a$ and return $\rho = s\gamma$.
2. Compute $\alpha = a\gamma$ and $\beta = r^k\gamma$.
3. Compute the discrete logarithm x of α with respect to δ by brute force: check whether $\alpha = x\delta$ for each x from 1 to r^k (this holds for some x, by part (i) of **(d)**).[3]
4. If r does not divide x then return `null`.
5. Compute s and t such that $sa + tr^{k+1} = 1$ using the extended Euclidean algorithm.
6. Return $\rho = s(x/r)\delta + t\beta$.

The return value `null` is used to indicate that γ does not have any rth roots in G. To compute $s = 1/r \bmod a$ in Sage, use: `s=1/mod(r,a)`. To compute s and t such that $sa + tr^{k+1} = 1$, use: `d,s,t=xgcd(a,r**(k+1))` (here $d = \gcd(a, r^{k+1})$ is 1).

The Python language used by Sage is untyped, so your algorithm can be used to compute rth roots in any cyclic group that Sage knows how to represent; it will automatically perform operations in whatever group the inputs δ and γ happen to lie in. To test your algorithm, you may find it useful to work in the additive group of the ring

[3] We will learn much better ways to compute this discrete logarithm later in the course. For the moment, assume r^k is small (for finite fields $\mathbb{F}_q$, this is usually the case, even when q is very large).

$\mathbb{Z}/n\mathbb{Z}$, where $n = ar^k$, which you can create in Sage using `Zn=Integers(n)`. You can then use `delta=Zn(a)` to create an element of $\mathbb{Z}/n\mathbb{Z}$ with additive order r^k.

Let E be the elliptic curve $y^2 = x^3 + 31415926x + 27182818$ over $\mathbb{F}_p$ with $p = 2^{255} - 19$. The group $E(\mathbb{F}_p)$ is cyclic, of order $n = 2 \cdot 3 \cdot 31 \cdot m$, where

$$m = 31126905708955966511712630378679545121741846343686298568983577739593446 6489,$$

and the point $P = (x_0 : y_0 : 1)$, where $x_0 = 99$ and

$$y_0 = 364605163313528648890204612945807701472550180139601517676013737542774864 2285,$$

is a generator for $E(\mathbb{F}_p)$. Thus for $r = 2, 3, 31$ you can use $\delta = (n/r)P$ as a generator of the r-Sylow subgroup (which in each case has order r).

Let c be the least prime greater than the integer formed by the last four digits of your student ID. Let $Q = (x_1 : y_1 : 1)$, where

$$x_1 = 43125933575059134974422288266359854378815207690220011740187158431378585841262,$$
$$y_1 = 30438392960540783858586956956150489842875282144799753811252714114065692010946$$

(e) Use your algorithm to find an rth root R of $\gamma = cQ$, for $r = 2, 3, 31$. Note that you can easily check your result by testing whether `r*R==c*Q` holds using Sage (please be sure to do this). In your answer you only need to list the point R for each value of r, you don't need to include your code. Be sure to format your answer so that the coordinates of R all fit on the page.

Problem 3. Exponentiating with addition chains (33 points)

An *addition chain* for a positive integer n is an increasing sequence of integers $(c_0, \ldots, c_m)$ with $c_0 = 1$ and $c_m = n$ such that each entry other than c_0 is the sum of two (not necessarily distinct) preceding entries. The *length* of an addition chain is the index m of the last entry. When computing a^n with a generic algorithm, the exponents k of the powers a^k computed by the algorithm define an addition chain whose length is the number of multiplications performed. For example, using left-to-right binary exponentiation to compute a^{47} yields the addition chain $(1, 2, 4, 5, 10, 11, 22, 23, 46, 47)$, and right-to-left binary exponentiation yields the addition chain $(1, 2, 3, 4, 7, 8, 15, 16, 32, 47)$, both of which have length 9.

(a) For $n = 715$, determine the addition chains given by: (i) left-to-right binary, (ii) right-to-left binary, (iii) fixed-window, and (iv) sliding-window exponentiation, using a window of size 2 for (iii) and (iv).

(b) Find an addition chain for $n = 715$ that is shorter than any you found in part (a).

(c) Repeat part (a) for the integer N obtained by adding 990,000 to the last 4 digits of your student ID, using a window of size 3 for the fixed and sliding window cases.

(d) Find the shortest addition chain for N that you can. There is a good chance you can do better than any of the chains you found in part (c).

In groups where inversions are cheap (such as the group of points on an elliptic curve), it is advantageous to use *signed* binary representations of exponents, where we write the

exponent n in the form $n = \sum n_i 2^i$ with $n_i \in \{-1, 0, 1\}$. Such a representation is generally not unique, but there is a unique signed representation with the property that no two adjacent digits are both nonzero. This is known as *non-adjacent form* (NAF). The NAF representation of 47, for example, is $10\bar{1}000\bar{1}$, where $\bar{1}$ denotes -1.

To construct the NAF representation one begins by writing n in binary with a leading 0, and then successively replaces the least significant block of the form $01\cdots1$ with $10\cdots0\bar{1}$ until there are no adjacent nonzero digits. For example, the computation for 47 proceeds as $0101111, 11000\bar{1}, 10\bar{1}000\bar{1}$, which reduces the number of nonzero digits from 5 to 3. Even though the length is increased by 1, the total cost of exponentiation may be reduced.

An *addition-subtraction* chain extends the definition of an addition chain by allowing $c_k = c_i \pm c_j$. Exponentiation using the NAF representation defines an addition-subtraction chain. For example, using left-to-right binary exponentiation, the NAF representation of 47 yields the chain $(1, 2, 4, 3, 6, 12, 24, 48, 47)$, which is shorter than the addition chain $(1, 2, 4, 5, 10, 11, 22, 23, 46, 47)$ given by standard left-to-right binary exponentiation.

(e) Compute addition-subtraction chains for $n = 715$ and the integer N defined in part (c) using left-to-right binary exponentiation with the NAF representation.

(f) Find the shortest addition-subtraction chains for n and N that you can.

Problem 4. Root-finding over $\mathbb{Z}$ (66 points)

In this problem you will develop an algorithm to find integer roots of polynomials in $\mathbb{Z}[x]$ using a p-adic version of Newton's method (also known as Hensel lifting). As an application, this gives us an efficient way to factor perfect powers (a special case that we will need to handle when we come to the elliptic curve factorization method), and it will be used as a black box in a later problem set to find integer roots of division polynomials.

In the questions below, p can be any integer greater than 1, but you may assume it is a prime power if you wish.

(a) Let $x_0 \in \mathbb{Z}$ and $f \in \mathbb{Z}[x]$. Prove that the following equivalence holds in $\mathbb{Z}[x]$:

$$f(x) \equiv f(x_0) + f'(x_0)(x - x_0) \bmod (x - x_0)^2.$$

(b) Let $x_0, z_0 \in \mathbb{Z}$ and $f \in \mathbb{Z}[x]$ satisfy $f(x_0) \equiv 0 \bmod p$ and $f'(x_0)z_0 \equiv 1 \bmod p$. Let

$$x_1 \equiv x_0 - f(x_0)z_0 \bmod p^2,$$
$$z_1 \equiv 2z_0 - f'(x_1)z_0^2 \bmod p^2.$$

Prove that the following three equivalences hold:

$$x_1 \equiv x_0 \bmod p, \tag{i}$$
$$f(x_1) \equiv 0 \bmod p^2, \tag{ii}$$
$$f'(x_1)z_1 \equiv 1 \bmod p^2. \tag{iii}$$

Show that (i) and (ii) characterize $x_1 \bmod p^2$ uniquely by proving that if $x_2 \in \mathbb{Z}$ also satisfies $x_2 \equiv x_0 \bmod p$ and $f(x_2) \equiv 0 \bmod p^2$, then $x_1 \equiv x_2 \bmod p^2$.

Iteratively applying **(b)** yields an algorithm that, given an integer k and x_0, z_0, and f satisfying the hypothesis **(b)**, outputs an integer x_k that satisfies $f(x_k) \equiv 0 \bmod p^{2^k}$.

(c) Prove that if f has an integer root r for which $f'(r)$ is invertible modulo p, then given $x_0 \equiv r \bmod p$, $z_0 \equiv 1/f'(x_0) \bmod p$, and k such that $|r| < p^{2^k}/2$, this algorithm outputs x_k such that r is the unique integer $r \equiv x_k \bmod p^{2^k}$ satisfying $|r| < p^{2^k}/2$.

To apply the result in (c), we need to know a suitable starting value (or values) for x_0. For the two applications we have in mind, this will be straightforward, so let us proceed on the assumption that we are given a suitable x_0 and z_0 such that $f'(x_0)z_0 \equiv 1 \bmod p$.

Let B be the maximum of the absolute values of the coefficients of f, and let B_0 be an upper bound on the absolute value of its largest integer root. It suffices to choose the least k such that $p^{2^k} > 2B_0$, and since any integer root of f must divide its constant coefficient, we can assume that $B_0 \leq B$. We can also assume $p < 2B_0$, since otherwise the problem is trivial ($k = 0$ and $x_k = x_0$).

(d) Prove that with this choice of k the algorithm can be implemented to run in time $O(d\,\mathsf{M}(\log B))$, where d is the degree of f (be careful here, the most obvious implementation will not achieve this time bound). Prove that if f has $O(1)$ terms, then the algorithm can be implemented to run in time $O(\mathsf{M}(\log B) + \mathsf{M}(\log B_0)\log d)$.

(e) Using the primes $p = 2$ and $p = 3$, describe an efficient algorithm that, given an integer N relatively prime to 6, either outputs an integer a and a prime q such that $a^q = N$, or proves that N is not a perfect power. Prove that your algorithm runs in quasi-quadratic time (meaning $O(n^2(\log n)^c)$ for some constant c, where $n = \log N$).

(f) Implement your algorithm and report the result and running time on the following inputs: $2^{1000} + 297$, 5^{503}, $(2^{500} + 55)^2$, $(2^{333} + 285)^3$, $(2^{32} + 15)^{31}$, $500!/(2^{494}3^{247})$. To time your code in SageMath, use the `timeit` function (e.g. `timeit("1+1")`), or in a CoCalc Jupyter notebook, use the `%timeit` line magic.

(g) Prove that the algorithm you gave in (e) can be implemented to run in sub-quadratic time (meaning $o(n^2)$ where $n = \log N$). You may need to modify your algorithm slightly in order to achieve this.[4]

Problem 5. Survey (1 point)

Complete the following survey by rating each of the problems you attempted on a scale of 1 to 10 according to how interesting you found the problem (1 = "mind-numbing," 10 = "mind-blowing"), and how difficult you found it (1 = "trivial," 10 = "brutal"). Also estimate the amount of time you spent on each problem to the nearest half hour.

	Interest	Difficulty	Time Spent
Problem 1			
Problem 2			
Problem 3			
Problem 4			

[4] In fact, this problem can be solved in quasi-linear time, see [2].

Feel free to record any additional comments you have on the problem sets or lectures, and in particular, ways in which you think they could be improved.

Collaborators/sources:

References

[1] Julius Magalona Basilla, *On the solution of $x^2 + dy^2 = m$*, Proc. Japan Acad. Ser. A Math Sci. **80** (2004), 40–41.

[2] D. J. Bernstein, *Detecting perfect powers in essentially linear time*, Math. Comp. **67** (1998), 1252–1283.