15 Number Theory

15.6 Computing Roots Modulo p (7 units)

Background material for this project is contained in the Part II course Number Theory.

1 Introduction

Throughout, p will be an odd prime number. An integer a coprime to p is called a *quadratic residue* mod p if the congruence $x^2 \equiv a \pmod{p}$ is soluble, otherwise a is termed a *non-residue* mod p. The *Legendre symbol*, (a/p), is defined (for a any integer and p an odd prime as above) by

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{if } p|a\,; \\ 1 & \text{if } a \text{ is a quadratic residue mod } p\,; \\ -1 & \text{if } a \text{ is a non-residue mod } p. \end{cases}$$

In Section 2 we consider the problem of distinguishing quadratic residues from non-residues. The remainder of the project is concerned with computing square roots mod p, or more generally finding the roots of a polynomial mod p.

2 Computing Legendre symbols

The Legendre symbol (a/p) can be computed using Euler's criterion:

$$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \pmod{p}.$$

Question 1 Write a program to compute (a/p) for p an odd prime and a any integer, using Euler's criterion. (You should use the repeated squaring method for modular exponentiation – see [1] if this is not familiar.) Test your program with $p = 10708729$ and

(i) 100 random values for a between 1 and p;
(ii) all a between 1 and 100.

For each of (i) and (ii), keep a tally of the number of values of a for which $(a/p) = 1$.

The Jacobi symbol is a generalisation of the Legendre symbol. For n odd and positive it is defined by

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right) \dots \left(\frac{a}{p_r}\right)$$

where $n = p_1 \dots p_r$ is a product of (not necessarily distinct) primes, and the symbols on the right are Legendre symbols. The Jacobi symbol satisfies the properties

$$\left(\frac{a}{n}\right) = \left(\frac{a \bmod n}{n}\right),$$

$$\left(\frac{ab}{n}\right) = \left(\frac{a}{n}\right)\left(\frac{b}{n}\right),$$

and if m and n are odd positive and coprime,

$$\left(\frac{m}{n}\right)\left(\frac{n}{m}\right) = (-1)^{(m-1)(n-1)/4}.$$

 © University of Cambridge

Question 2 Write a program to compute (a/p) for p an odd prime and a any integer, by manipulating Jacobi symbols. You should try to make your algorithm reasonably efficient. Make a note in your report of any properties of the Jacobi symbol you needed in addition to those listed above.

Estimate the complexity of your algorithm and compare with Question 1.

[You should estimate the number of basic operations required, where a basic operation could be addition or multiplication of two numbers. A more sophisticated analysis might also take into account the time required to add and multiply large numbers on a finite machine, but it isn't necessary to go into such details here.]

3 Computing square roots mod p

Suppose that p is an odd prime and a is a quadratic residue mod p. How can we find x such that $x^2 \equiv a \pmod{p}$? We could simply search through all congruence classes mod p, but if p is large then we need a better method.

Question 3 Show that if $p \equiv 3 \pmod{4}$ then the congruence $x^2 \equiv a \pmod{p}$ has solution $x \equiv a^{(p+1)/4} \pmod{p}$. Further show that if $p \equiv 5 \pmod{8}$ then the congruence $x^2 \equiv a \pmod{p}$ has solution $x \equiv 2^{k(p-1)/4} a^{(p+3)/8} \pmod{p}$ for some $k \in \{0, 1\}$.

Now suppose that $p-1$ is a power of 2 and let g be a primitive root mod p, *i.e.* a generator for the multiplicative group of non-zero residues mod p. To solve the congruence $x^2 \equiv a \pmod{p}$ we substitute $x \equiv g^r \pmod{p}$ where $r = \sum_{j \geqslant 0} r_j 2^j$ with $r_j \in \{0, 1\}$. Raising each side of the original congruence to suitable powers it is possible to solve for the binary digits $r_0, r_1, r_2, \ldots$ in turn. For example the first step is

$$r_0 = \begin{cases} 0 & \text{if } a^{(p-1)/4} \equiv 1 \pmod{p} \\ 1 & \text{if } a^{(p-1)/4} \equiv -1 \pmod{p}. \end{cases}$$

Question 4 Use this method to solve the congruence $x^2 \equiv 58256 \pmod{65537}$.

A general algorithm for computing square roots mod p is obtained by combining the methods of Questions 3 and 4. We begin by writing $p - 1 = 2^\alpha s$ with s odd. Then we find a non-residue n, and compute $b \equiv n^s \pmod{p}$. Since s is odd it suffices to solve the congruence $y^2 \equiv a^s \pmod{p}$. We do this by substituting $y \equiv b^r \pmod{p}$ and solving for the binary digits of r.

Question 5 Write a program for computing square roots mod p, based on the method described above. Test your program for $p = 10708729$ and all quadratic residues mod p between 1 and 20, and for a few other values of p and a. Estimate the complexity of your algorithm.

4 Computing roots of polynomials mod p

In this section we work with polynomials whose coefficients are integers mod p.

 © University of Cambridge

Question 6 Write procedures to compute the quotient and remainder when we divide one polynomial by another. Use them to write a procedure to find the greatest common divisor of two polynomials. Illustrate by computing

$$\begin{aligned}
&\gcd(x^3 + 8x^2 + 12x + 4, x^3 + 6x^2 + 2x + 10) \quad \text{with } p = 109,\\
&\gcd(x^3 + 2x^2 + 6x + 8, x^3 + 11x^2 + x + 2) \quad \text{with } p = 131,\\
&\gcd(x^3 + 3x^2 + 7x + 1, x^3 + 3x^2 + 4x + 12) \quad \text{with } p = 157.
\end{aligned}$$

To compute the roots of a polynomial $f(x)$ mod p we first compute $\gcd(f(x), x^p - x)$. This reduces us to the case where $f(x)$ is a product of distinct linear factors. We then pick a small integer v at random and attempt to factor $f(x)$ by computing $\gcd(f(x), g(x))$ where $g(x) = (x + v)^{(p-1)/2} - 1$. This will be successful unless the numbers $\alpha + v$ for α running over the roots of $f(x)$ are either all quadratic residues, or all non-residues. If unsuccessful we try another value of v.

Question 7 Write a program for computing square roots mod p, based on the method described above. Explain how your program avoids working with polynomials of excessively large degree. Investigate how many values of v we expect to use (on average). Compare this method with that of Question 5 and comment on the theoretical behaviour for large p.

Question 8 Modify your program to compute the roots of any polynomial mod p, and run it on the polynomials

$$\begin{aligned}
f_1(x) &= x^4 + 9x^3 + 13x^2 + 2x - 9,\\
f_2(x) &= x^4 + x^3 + x^2 + x + 25,\\
f_3(x) &= x^4 + x^3 - 10x^2 - 749379x - 120288
\end{aligned}$$

with $p = 10708729$.

5 Programming

If you use MATLAB then you may wish to use the `DocPolynom` class that is included as an example in the help browser. To use this you should create a directory `@DocPolynom` and place `DocPolynom.m` into it. This will enable you to define and display (non-zero) polynomials and to carry out standard algebraic manipulations with them. There is no need to include the class file in your program listings (assuming you do not modify it). *[The latest version requires MATLAB 2022b or later to run.]*

If you use a computer algebra package (such as MAPLE), then you may find that some of the routines asked for in this project are included in the package. In such cases, no credit will be given for using the packaged routines — you are expected to write your own programs.

References

[1] Koblitz, N. *A Course in Number Theory and Cryptography*, Graduate Texts in Mathematics 114, Springer, 1987.

 © University of Cambridge