Notes on Polynomials, Interpolation, and Codes (Draft)

October 9, 2012

You've probably all seen polynomials before: e.g., $3x^2 - 5x + 17$, or $2x - 3$, or $-x^5 + 38x^3 - 1$, or x, or even a constant 3. These are all polynomials over a single variable (here, x). The degree of a polynomial is the highest exponent of x that appears: hence the degrees of the above polynomials are $2, 1, 5, 1, 0$ respectively.

In general, a polynomial over the variable x of degree at most d looks like:

$$P(x) = c_d x^d + c_{d-1}x^{d-1} + .. + c_1 x + c_0$$

Note that the sequence of $d + 1$ coefficients $\langle c_d, c_{d-1}, \ldots, c_0\rangle$ completely describes $P(x)$.

Hence, if the coefficients were all drawn from the field $\mathbb{F}_p$ ($\{0, 1, 2, \ldots, p - 1\}$ with addition and multipication mod p), then we have exactly p^{d+1} possible different polynomials. This includes the zero polynomial $0 = 0x^d + 0x^{d-1} + .. + 0x + 0$.

In this lecture, we will study some properties of polynomials, relate the ideas we use to stuff we've seen in Concepts (namely, Chinese remaindering), and then use the properties of polynomials to construct error correcting codes.

1 Operations on Polynomials

Before we study properties of polynomials, recall the following simple operations on polynomials:

- Given two polynomials $P(x)$ and $Q(x)$, we can add them to get another polynomial $R(x) = P(x) + Q(x)$. Note that the degree of $R(x)$ is at most the maximum of the degrees of P and Q. (Q: Why is it not equal to the maximum?)

$$(x^2 + 2x - 1) + (3x^3 + 7x) = 3x^3 + x^2 + 9x - 1$$

  The same holds for the difference of two polynomials $P(x) - Q(x)$, which is the same as $P(x) + (-Q(x))$.

- Given two polynomials $P(x)$ and $Q(x)$, we can multiply them to get another polynomial $S(x) = P(x) \times Q(x)$.

$$(x^2 + 2x - 1) \times (3x^3 + 7x) = 3x^5 + 4x^3 + 6x^4 + 14x^2 - 7x$$

  The degree of $S(x)$ is equal to the sum of the degrees of P and Q.

- Note that $P(x)/Q(x)$ may not be a polynomial.

- We can also *evaluate* polynomials. Given a polynomial $P(x)$ and a value a, $P(a) := c_d \cdot a^d + c_{d-1} \cdot a^{d-1} + .. + c_1 \cdot a + c_0$. For example, if $P(x) = 3x^5 + 4x^3 + 6x^4 + 14x^2 - 7x$, then

$$P(2) = 3 \cdot 2^5 + 4 \cdot 2^3 + 6 \cdot 2^4 + 14 \cdot 2^2 - 7 \cdot 2 = 266$$

Of course, the multiplication between the c_i's and a must be well-defined, as should the meaning of a^i for all i. E.g., if the c_i's and a are reals, this is immediate.

But also, if c_i's and a all belonged to $\mathbb{F}_p$ for prime p, evaluation would still be well-defined. For instance, if we were working in $\mathbb{Z}_{17}$, then

$$P(2) = 266 \pmod{17} = 11$$

- A *root* of a polynomial $P(x)$ is a value r such that $P(r) = 0$. For example, $P(x)$ above has three real roots $0, -1+\sqrt{2}, -1-\sqrt{2}$, and two complex roots.

2 How Many Roots?

Let's start with the following super-important theorem.

Theorem 1 (Few-Roots Theorem) *Any non-zero polynomial of degree at most d over a field has at most d distinct roots.*[1]

This holds true, regardless of what field we are working over. When we are working over the reals (i.e., the coefficients are reals, and we are allowed to plus in arbitrary reals for x), this theorem is a corollary of the fundamental theorem of Algebra. But it holds even if we are working over some other field (say $\mathbb{F}_p$ for prime p).

Let's relate this to what we know. Consider polynomials of degree 1, also known as linear polynomials. Say they have real coefficients, this gives a straight line when we plot it. Such a polynomial has at most one root: it crosses the x-axis at most once. And in fact, any degree-1 polynomial looks like $c_1x + c_0$, and hence setting $x = -c_0/c_1$ gives us a root. So, in fact, a polynomial of degree exactly 1 has exactly one root.

What about degree 2, the quadratics? Things get a little more tricky now, as you probably remember from high school. E.g., the polynomial $x^2 + 1$ has no real roots, but it has two complex roots. However, you might remember that if it has one real root, then both roots are real. But anyways, a quadratic crosses the x-axis at most twice. At most two roots.

And in general, the theorem says, any polynomial of degree at most d has at most d roots.

The proof of this theorem is not hard — it is done via induction. Here is a sketch of the details. The case of $d = 1$ follows since $ax + b$ has a single root $-b/a$ when $a \neq 0$ and no roots when $a = 0, b \neq 0$. Note that we already used the fact that the coefficients come from a field, because we assumed we could divide by any nonzero a.

For $d > 1$, let P be a nonzero polynomial. Suppose α is a root of P (if P has no roots, we are already done). Now let us divide $P(x)$ by $(x - \alpha)$ to get:

$$P(x) = Q(x)(x-\alpha) + R(x)\ ,$$

where $\text{degree}(R) < 1$. Therefore $R(x)$ must be a constant polynomial, say $R(x) = \beta$. Now note that

$$\beta = R(\alpha) = P(\alpha) - Q(\alpha)(\alpha - \alpha) = 0$$

[1]In fact, the polynomial will have at most d roots counting multiplicities, but for most applications the bound on number of distinct roots itself suffices.

implying R must be the zero polynomial. Hence $P(x) = Q(x)(x - \alpha)$. This implies that any root of P other than α must be a root of Q. (Important: We are again using the fact that we are working over a field here. Can you see why?) By induction Q has at most $d-1$ distinct roots, which together with α can give at most d distinct roots for P.

3 A New Representation for degree-d Polynomials

Let's prove a simple corollary of the theorem. It says that if we plot two polynomials of degree at most d, then they can intersect in at most d points—*unless they are the same polynomial*! Remember, two distinct lines intersect at most once, two distinct quadratics intersect at most twice, etc. Same principle.

Corollary 2 *Given $d+1$ pairs $(a_0, b_0), (a_1, b_1), \ldots, (a_d, b_d)$, there is* at most *one polynomial $P(x)$ of degree at most d, such that $P(a_i) = b_i$ for all $i = 0, 1, \ldots, d$.*

Proof: First, note that if $a_i = a_j$, then b_i better equal b_j—else no polynomial can equal both b_i and b_j when evaluated at $a_i = a_j$.

For a contradiction, suppose there are two distinct polynomials $P(x)$ and $Q(x)$ of degree at most d such that for all i,

$$P(a_i) = Q(a_i) = b_i.$$

Then consider the polynomial $R(x) = P(x) - Q(x)$. It has degree at most d, since it is the difference of two polynomials of degree at most d. Moreover,

$$R(a_i) = P(a_i) - Q(a_i) = 0$$

for all the $d+1$ settings of $i = 0, 1, \ldots, d$. Once again, R is a polynomial of degree at most d, with $d+1$ roots. By the contrapositive of Theorem 1, $R(x)$ must be the zero polynomial. And hence $P(x) = Q(x)$, which gives us the contradiction. ■

To paraphrase the theorem differently, given two (i.e., $1+1$) points there is at most one linear (i.e., degree-1) polynomial that passes through them, given three (i.e., $2+1$) points there is at most one quadratic (i.e., degree-2) polynomial that passes through them, etc.

Can it be the case that for some $d+1$ pairs $(a_0, b_0), (a_1, b_1), \ldots, (a_d, b_d)$, there is *no* polynomial of degree at most d that passes through them? Well, clearly if $a_i = a_j$ but $b_i \neq b_j$. But what if all the a_i's are distinct?

Theorem 3 (Unique Reconstruction Theorem) *Given $d+1$ pairs $(a_0, b_0), (a_1, b_1), \ldots, (a_d, b_d)$ with $a_i \neq a_j$ for all $i \neq j$, there always exists a polynomial $P(x)$ of degree at most d, such that $P(a_i) = b_i$ for all $i = 0, 1, \ldots, d$.*

We will prove this theorem soon, but before that let us note some implcations. Given $d+1$ pairs $(a_0, b_0), (a_1, b_1), \ldots, (a_d, b_d)$ with distinct a's, this means there is a *unique* polynomial of degree at most d that passes through these points. Exactly one.

In fact, given $d+1$ numbers $b_0, b_1, \ldots, b_d$, there is a unique polynomial $P(x)$ of degree at most d such that $P(i) = b_i$. (We're just using the theorem with $a_i = i$.) Earlier we saw how to represent any polynomial of degree at most d by $d+1$ numbers, the coefficients. Now we are saying that we can represent the polynomial of degree at most d by a different sequence of $d+1$ numbers: its values at $0, 1, \dots d$.

Two different representations for the same thing, cool! Surely there must be a use for this new representation. We will give at least two uses for this, but first let's see the proof of Theorem 3. (If you are impatient, you can skip over the proof, but do come back and read it—it is very elegant.)

4 The Proof of Theorem 3

OK, now the proof. We are given $d+1$ pairs (a_i, b_i), and the a's are all distinct. The proof will actually give an algorithm to find this polynomial $P(x)$ with degree at most d, and where $P(a_i) = b_i$.

Let's start easy: suppose all the $d+1$ values b_i's were zero. Then $P(x)$ has $d+1$ roots, and now Theorem 1 tells us that $P(x) = 0$, the zero polynomial!

OK, next step. Suppose $b_0 = 1$, but all the d other b_i's are zero. Do we know a degree-d polynomial which has roots at d places $a_1, a_2, \dots, a_d$. Sure, we do—it is just

$$Q_0(x) = (x-a_1)(x-a_2)\cdots(x-a_d).$$

So are we done? Not necessarily: $Q_0(a_0)$ might not equal $b_0 = 1$. But that is easy to fix! Just scale the polynomial by $1/Q_0(a_0)$. I.e., what we wanted was

$$\begin{aligned} R_0(x) &= (x-a_1)(x-a_2)\cdots(x-a_d)\cdot\frac{1}{Q_0(a_0)} \\ &= \frac{(x-a_1)(x-a_2)\cdots(x-a_d)}{(a_0-a_1)(a_0-a_2)\cdots(a_0-a_d)} \ . \end{aligned}$$

Again, $R_0(x)$ has degree d by construction, and satisfies what we wanted! (We'll call $R_0(x)$ the 0^{th} "switch" polynomial.)

Next, what if b_0 was not 1 but some other value. Easy again: just take $b_0 \times R_0(x)$. This has value $b_0 \times 1$ at a_0, and $b_0 \times 0 = 0$ at all other a_i's.

Similarly, one can define switch polynomials $R_i(x)$ of degree d that have $R_i(a_i) = 1$ and $R_i(a_j) = 0$ for all $i \neq j$. Indeed, this is

$$R_i(x) = \frac{(x-a_0)\cdots(x-a_{i-1})\cdot(x-a_{i-1})\cdots(x-a_d)}{(a_i-a_0)\cdots(a_i-a_{i-1})\cdot(a_i-a_{i-1})\cdots(a_i-a_d)} \ .$$

So the polynomial we wanted after all is just a linear combination of these switch polynomials:

$$P(x) = b_0R_0(x) + b_1R_1(x) + \ldots + b_dR_d(x)$$

Since it is a sum of degree-d polynomials, $P(x)$ has degree at most d. And what is $P(a_i)$? Since $R_j(a_i) = 0$ for all $j \neq i$, we get $P(a_i) = b_iR_i(a_i)$. Now $R_i(a_i) = 1$, so this is b_i. All done.

4.1 An example

Consider the tuples $(5,1),(6,2),(7,9)$: we want the unique degree-2 polynomial that passes through these points. So first we find $R_0(x)$, which evaluates to 1 at $x=5$, and has roots at 6 and 7. This is

$$R_0(x) = \frac{(x-6)(x-7)}{(5-6)(5-7)} = \frac{1}{2}(x-6)(x-7)$$

Similarly

$$R_1(x) = \frac{(x-5)(x-7)}{(6-5)(6-7)} = -(x-5)(x-7)$$

and

$$R_2(x) = \frac{(x-5)(x-6)}{(7-5)(7-6)} = \frac{1}{2}(x-5)(x-6)$$

Hence, the polynomial we want is

$$P(x) = 1 \cdot R_0(x) + 2 \cdot R_1(x) + 9 \cdot R_2(x) = 3x^2 - 32x + 86$$

Let's check our answer:

$$P(5) = 1, P(6) = 2, P(7) = 9.$$

Note that constructing the polynomial $P(x)$ takes $O(d^2)$ time. (Can you find the simplified version in this time as well?)

This algorithm is called Lagrange interpolation, after Joseph-Louis Lagrange, or Giuseppe Lodovico Lagrangia, depending on whether you ask the French or the Italians. (He was born in Turin, and both countries claim him for their own.) And to muddy things even further, much of his work was done at Berlin. He later moved to Paris, where he survived the French revolution—though Lavoisier was sent to the guillotine because he intervened on behalf of Lagrange. Among much other great research, he also gave the first known proof of Wilson's theorem, that n is a prime if and only if $n|(n-1)!+1$—apparently Wilson only conjectured Wilson's theorem.

4.2 Remember that Chinese Remainder Theorem?

While you have Lagrange interpolation fresh in your mind, let's recall Chinese remaindering and relate the two. The ideas are almost exactly the same—so if you remember one, you can remember both.

Theorem 4 (Chinese Remainder Theorem) *Consider integers $p_0, p_1, \ldots, p_d$ such that $\gcd(p_i, p_j) = 1$, define $M = p_0 p_1 \ldots p_d$. For any values $a_i \in \mathbb{Z}_{p_i}$, there is exactly one $x \in \mathbb{Z}_M$ such that*

$$x \equiv a_i \pmod{p_i}.$$

Proof: Before we show there exists at least one such x, note that if there are two solutions $x, y \in \mathbb{Z}_N$, then $z = x - y \equiv 0 \pmod{p_i}$. Hence $p_i|z$ for all i, and since all the p's are co-prime, we get that $N|z$. This means $z = 0$ and hence $x = y$.

To construct a solution x, let us consider a similar proof strategy to the one for Lagrange interpolation. Suppose we could we find numbers $r_i \in \mathbb{Z}_N$ such

$$r_i = 1 \pmod{p_i}$$
$$r_i = 0 \pmod{p_j} \quad \forall j \neq i$$

Then we could just set

$$x := a_0 r_0 + a_1 r_1 + \ldots + a_d r_d \pmod{N}$$

Note that $x \pmod{p_i} = a_i r_i \pmod{p_i}$ (since $r_j \pmod{p_i}$ for all $j \neq i$), which equals a_i (since $r_i \equiv 1 \pmod{p_i}$).

Now to construct these r_i's. We'll show the idea for r_0. Again, let us first get the zeros in place — let's make sure $r_0 \equiv 0 \pmod{p_i}$ for $i > 0$. This is easy: just take $q = p_1 p_2 \ldots p_d$.

Can we use this value q as r_0? Maybe not — possibly $q \pmod{p_0} = b \neq 1$. So let's fix that: multiply q by the multiplicative inverse of $b \in \mathbb{Z}_{p_0}$ (i.e., element b^{-1} such that $b^{-1} \cdot b \equiv_{p_0} 1$):

$$r_0 = q \times b^{-1} \pmod{p_0}.$$

(Hang on — why does this multiplicative inverse of b exist? Well, from what we know from the number theory lecture, such an inverse exists if $\gcd(b, p_0) = 1$. But hey, $b = p_1 p_2 \ldots p_d \pmod{p_0}$, so $\gcd(b, p_0) = \gcd(p_1 p_2 \ldots p_d, p_0) = 1$. It's all good.) ■

This procedure for reconstructing the large integer in Chinese remaindering is very similar to Lagrange interpolation, isn't it?

An Example for CRT

Ler us see an example of this — we'll use the Wikipedia example for CRT. We want a number x that is 2 modulo 3, 3 modulo 4 and 1 modulo 5. So let's first create those numbers r_i as in the previous section.

Start with $q_0 = p_1 \cdot p_2 = 4 \cdot 5 = 20 \equiv 2 \pmod{3}$. So $r_0 = q_0 \cdot (2^{-1} \bmod 3) = q_0 \cdot 2 = 40$.

Similarly, take $q_1 = p_0 \cdot p_2 = 3 \cdot 5 = 15 \equiv 3 \pmod{4}$. So $r_1 = q_1 \cdot (3^{-1} \bmod 4) = q_1 \cdot 3 = 45$.

And finally, $q_2 = p_0 \cdot p_1 = 3 \cdot 4 = 12 \equiv 2 \pmod{5}$. So $r_2 = q_2 \cdot (2^{-1} \bmod 5) = q_1 \cdot 3 = 36$.

Finally, $x = 2r_0 + 3r_1 + 1r_2 \pmod{2 \cdot 3 \cdot 5} = 2 \cdot 40 + 3 \cdot 45 + 1 \cdot 36 \pmod{30} = 251 \pmod{30} = 11$.

Let's check our answer: 11 is indeed 2 modulo 3, 3 modulo 4 and 1 modulo 5.

4.3 Another Proof of Theorem 3

Hey, Theorem 3's so nice, we'll prove it twice. (Sorry, bad one.) Here's a proof that uses simple linear algebra, and gives us another way to interpolate a degree d polynomial with any desired values at $d + 1$ distinct points.

Proof: Suppose the polynomial $P(x) = c_d x^d + c_{d-1}x^{d-1} + .. + c_1 x + c_0$, where the c_i's are currently unknown. Since $P(a_i) = b_i$ for all i, we get $d+1$ different equalities of the form

$$\begin{aligned} c_d a_0^d + c_{d-1}a_0^{d-1} + .. + c_1 a_0 + c_0 &= b_0 \\ c_d a_1^d + c_{d-1}a_1^{d-1} + .. + c_1 a_1 + c_0 &= b_1 \\ &\vdots \\ c_d a_i^d + c_{d-1}a_i^{d-1} + .. + c_1 a_i + c_0 &= b_i \\ &\vdots \\ c_d a_d^d + c_{d-1}a_d^{d-1} + .. + c_1 a_d + c_0 &= b_d \end{aligned}$$

for $i = 0, 1, \ldots, d$. Note that we want values of the unknowns c_i's that satisfy all these constraints: we want to "solve for the c's".

This we can write more succinctly using linear algebra:

$$\begin{bmatrix} a_0^d & a_0^{d-1} & \cdots & a_0 & 1 \\ a_1^d & a_1^{d-1} & \cdots & a_1 & 1 \\ \vdots & & \cdots & & \\ a_d^d & a_d^{d-1} & \cdots & a_d & 1 \end{bmatrix} \begin{bmatrix} c_d \\ c_{d-1} \\ \vdots \\ c_0 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_d \end{bmatrix}$$

Let's denote this by $A\vec{c} = \vec{b}$. Note that A is a $(d+1) \times (d+1)$ matrix. So if A was invertible, then we could multiply by A^{-1} on both sides, and get $\vec{c} = A^{-1}\vec{b}$.

So, the first question: when is A invertible? This is precisely when the determinant of A is non-zero. Here, A is so well-structured that it even has a name (it is called a *Vandermonde* matrix), and we can write a closed form for its determinant:

$$det(A) = \prod_{i<j}(a_i - a_j)$$

(Exercise: prove this!) And since all our a's are distinct, this determinant is non-zero, and hence the matrix A is invertible. In fact, this proves the theorem as stated, since we know that a solution *exists*.

Of course, if you do want another algorithm to find the values for c_i's. We can do that in two (closely related) ways:

- We can explicitly compute A^{-1}, and then $\vec{c} = A^{-1}\vec{b}$. You remember how to compute matrix inverses, right?

- Or use the extremely useful *Cramer's rule*. If you want to solve $A\vec{c} = \vec{b}$ and A is invertible, then the solution is
$$c_{d-i} = \frac{det(A_i[\vec{b}])}{det(A)}$$
where $A_i[\vec{b}]$ is the matrix obtained by replacing the i^{th} column of A by the column vector $\vec{b}$.

A note on efficiency: Note that both these approaches require computing determinants (either explicitly for Cramer's rule, or hidden in the computation of the inverse). The naïve way to compute the determinant of a $m \times m$ matrix takes $m!$ time, but one can do it in time $O(m^3)$ using Gaussian elimination. However, an algorithm based on this approach still takes more time than the approach for the first proof. ∎

5 Application: Error Correcting Codes

Consider the situation: I want to send you a sequence of $d+1$ numbers $\langle c_d, c_{d-1}, \ldots, c_1, c_0 \rangle$ over a noisy channel. I can't just send you these numbers in a message, because I know that whatever message I send you, the channel will corrupt up to k of the numbers in that message. For the current example, assume that the corruption is very simple: whenver a number is corrupted, it is replaced by a $\star$ [2]. Hence, if I send the sequence

$$\langle 5, 19, 2, 3, 2 \rangle$$

and the channel decides to corrupt the third and fourth numbers, you would get

$$\langle 5, 19, \star, \star, 2 \rangle.$$

On the other hand, if I decided to delete the fourth and fifth elements, you would get

$$\langle 5, 19, 2, \star, \star \rangle.$$

Since the channel is "erasing" some of the entries and replacing them with $\star$'s, the codes we will develop will be called *erasure* codes. The question then is: how can we send $d+1$ numbers so that the receiver can get back these $d+1$ numbers even if up to k numbers in the message are erased (replaced by $\star$s)? (Assume that both you and the receiver know d and k.)

A simple case: if $d = 0$, then one number is sent. Since the channel can erase k numbers, the best we can do is to repeat this single number $k+1$ times, and send these $k+1$ copies across. At least one of these copies will survive, and the receiver will know the number.

This suggests a strategy: no matter how many numbers you want to send, repeat each number $k+1$ times. So to send the message $\langle 5, 19, 2, 3, 2 \rangle$ with $k = 2$, you would send

$$\langle 5, 5, 5, 19, 19, 19, 2, 2, 2, 3, 3, 3, 2, 2, 2 \rangle$$

This takes $(d+1)(k+1)$ numbers, approximately dk. Can we do better?

Indeed we can! We view our sequence $\langle c_d, c_{d-1}, \ldots, c_1, c_0 \rangle$ as the $d+1$ coefficients of a polynomial of degree at most d, namely $P(x) = c_d x^d + c_{d-1} x^{d-1} + \ldots + c_1 x + c_0$. Now we evaluate P at some $d+k+1$ points, say $0, 1, 2, \ldots, d+k$, and send these $d+k+1$ numbers $(P(0), P(1), \ldots, P(d+k))$ across. The receiver will get back at least $d+1$ of these numbers, which by Theorem 3 uniquely specifies $P(x)$. Moreover, the receiver can also reconstruct $P(x)$ using, say, Langrange interpolation.

Here is an example: Suppose we want to send $\langle 5, 19, 2, 3, 2 \rangle$ with $k = 2$. Hence $P(x) = 5x^4 + 19x^3 + 2x^2 + 3x + 2$. Now we'll evaluate $P(x)$ at $0, 1, 2, \ldots d+k = 6$. This gives

$$P(0) = 2, P(1) = 31, P(2) = 248, P(3) = 947, P(4) = 2542, P(5) = 5567, P(6) = 10676$$

[2] I am assuming that you are only sending numbers, and not $\star$s.

So we send across the "encoded message":

$$\langle 2, 31, 248, 947, 2542, 5567, 10676\rangle$$

Now suppose the third and fifth entries get erased. the receiver gets:

$$\langle 2, 31, \star, 947, \star, 5567, 10676\rangle$$

So she wants to reconstruct a polynomial $R(x)$ of degree at most 4 such that $R(0) = 2, R(1) = 31, R(3) = 947, R(5) = 5567, R(6) = 10676$. (That is, she wants to "decode" the message.) By Langrange interpolation, we get that

$$R(x) = \frac{1}{45}(x-1)(x-3)(x-5)(x-6) - \frac{31}{40}x(x-3)(x-5)(x-6) + \frac{947}{36}x(x-1)(x-5)(x-6)$$
$$-\frac{5567}{40}x(x-1)(x-3)(x-6) + \frac{5338}{45}x(x-1)(x-3)(x-5)$$

which simplifies to $P(x) = 5x^4 + 19x^3 + 2x^2 + 3x + 2$!

Note: The numbers get large, so you may want work modulo a prime. Since we want to send numbers as large as 19, let's work in $\mathbb{Z}_{23}$. Then you'd send the numbers modulo 23, which would be

$$\langle 2, 8, 18, 4, 12, 1, 4\rangle$$

Now suppose you get

$$\langle 2, 8, \star, 4, \star, 1, 4\rangle$$

Interpolate to get

$$R(x) = 45^{-1}(x-1)(x-3)(x-5)(x-6) - 5^{-1}x(x-3)(x-5)(x-6) + 9^{-1}x(x-1)(x-5)(x-6)$$
$$-40^{-1}x(x-1)(x-3)(x-6) + 2\cdot 45^{-1}x(x-1)(x-3)(x-5)$$

where the multiplicative inverses are modulo 23, of course. Simplifying, we get $P(x) = 5x^4 + 19x^3 + 2x^2 + 3x + 2$ again. (Of course, if you are working modulo a prime p, both the sender and the receiver must know the prime p.)

5.1 Error Correction

One can imagine that the channel is more malicious: it decides to *replace* some k of the numbers not by stars but by other numbers, so the same encoding/decoding strategy cannot be used! Indeed, the receiver now has no clue which numbers were altered, and which ones were part of the original message! In fact, even for the $d = 0$ case of a single number, we need to send $2k+1$ numbers across, so that the receiver knows that the majority number must be the correct one.

Conversely, if you evaluate $P(x)$ at $d + 2k + 1$ locations and send those values across, even if the channel alters k of those numbers, there is a unique polynomial that agrees with $d + k + 1$ of these numbers (and this must of course be equal to the originally transmitted polynomial$P(x)$). We leave the proof of this fact as an exercise (it is not hard and again just based on the fact that two distinct degree d polynomials can't agree on $d + 1$ points).

Thus, in principle, one can determine $P(x)$ even in the wake of k errors. The idea would be throw out every subset of k values, and check if there is a degree d polynomial consistent with the remaining values. If we ever find one, that *must* be the original polynomial $P(x)$ (that is no spurious polynomial can slip by this test); make sure you can argue why this is the case.

The trouble with this approach is that one must try all $\binom{d+2k+1}{k}$ subsets which quickly becomes a very large number as the number of errors increases (imagine, for example, $k = d/2$; the number of subsets we need to try would grow exponentially in d).

Fortunately, there are better algorithms that can recover $P(x)$ much more efficiently. The first such algorithm was discovered by Peterson in 1960; this was probably one of the earliest non-trivial polynomial time algorithms, and indeed predated the formulation of polynomial running time as the metric for efficient algorithms. Later, Berlekamp and Massey gave improvements to this algorithm, making it practical and leading to the widespread adoption of polynomial-based coding in real world applications. Incidentally, this polynomial-based coding has a name — it is called Reed-Solomon coding after its discoverers Reed and Solomon who proposed the code in 1960.

In the (optional) appendix, we describe an easier to describe algorithm, due to Welch and Berlekamp, for performing Reed-Solomon error correction efficiently.

6 Half an Application: Polynomial Multiplication

This will only be half an application: we will mention the main idea behind this application, but defer the actual details to 15-451.

We saw two representations of a degree-d polynomial $P(x)$: the *coefficient representation* where we write down the $d+1$ coefficients $\langle c_d, c_{d-1}, \ldots, c_0 \rangle$, and the *value representation* where we write down the value of P at some specific $d+1$ points, say $(P(0), P(1), \ldots, P(d))$.

Let us consider the time it takes to perform various polynomial operations in these two representations:

- Addition: takes $O(d)$ time in both representations.
- Multiplication: takes $O(d^2)$ time in the coefficient representation, but $O(d)$ time in the value representation. (We need to make sure that the product also has degree at most d, else we will need more values.)
- Evaluation at a single point: can be done in time $O(d)$ in the coefficient representation. (Exercise: How?)

  In the value representation, one way to do evaluation would be to reconstruct $P(x)$ using Lagrange interpolation, and then evaluate $P(x)$. But that would require $O(d^2)$ time.

So suppose we want to speed up polynomial multiplication. Here's an approach: Given two degree-d polynomials P, Q as a list of coefficients, we evaluate them at $2d+1$ points to convert them to the value representation. Then we multiply them together using $2d+1$ multiplies. Finally, we convert this solution back to the coefficient represntation using Lagrange.

The problem is: the $2d+1$ evaluations, and also Lagrange interpolation, both these things take $\Omega(d^2)$ time.

The solution: don't evaluate them at any old $2d+1$ points. Evaluate them at the $2d+1$ roots of unity. (whoa!) This gives rise to the famous *Fast Fourier Transform*, which multiplies two polynomials in time $O(d \log d)$. You'll see more of this in 15-451.

Final note: why do we care about multiplying polynomials together, anyways? Well, the same techniques also speed up integer multiplication as well. In this course we'll see how to multiply two n-bit numbers in time $O(n^{1.58})$. But the techniques based on the Fast Fourier Transform allows us to multiply two n-bit integers in time $O(n \log n \log\log n)$.

A Appendix: Correcting errors using Reed-Solomon codes

Suppose a polynomial $f \in \mathbb{F}_p[X]$ of degree d is encoded as the sequence of values $(f(\alpha_1), \ldots, f(\alpha_n))$ and transmitted where $n = d+2k+1$, but it is received as the noisy word $y = (y_1, \ldots, y_n)$ satisfying $y_i \neq f(\alpha_i)$ for at most k values of i. The goal is to recover the polynomial $f(X)$ from y.

Note that if we knew the location of the errors, i.e., the subset $\{i \mid y_i \neq f(\alpha_i)\}$, then the decoding is easy, as we can erase the erroneous and interpolate the polynomial on the rest of the locations.

To this end, let us define the so-called *error locator polynomia*l (which is unknown to the decoder):

$$E(X) \overset{\text{def}}{=} \prod_{f(\alpha_i) \neq y_i} (X - \alpha_i) \tag{1}$$

The degree of $E(X)$ is $\leq k$. Clearly $E(X)$ has the property that for $1 \leq i \leq n$,

$$E(\alpha_i)(y_i - f(\alpha_i)) = 0$$

since either $f(\alpha_i) = y_i$ or $E(\alpha_i) = 0$ for every i. Thus the polynomial in two variables

$$T(X,Y) = E(X)(Y - f(X)) \tag{2}$$

satisfies $T(\alpha_i, y_i) = 0, \forall i$. Defining the polynomial

$$N(X) = E(X)f(X) , \tag{3}$$

which has degree at most $d + k$, we see that $T(X,Y)$ is of the form

$$T(X,Y) = E(X)Y - N(X) \tag{4}$$

where $\text{degree}(E) \leq k$ and $\text{degree}(N) \leq d+k$. We will use the existence of such a T to find a similar bivariate polynomial from which we can find $f(X)$.

Formally, the algorithms proceeds in two steps.

Step 1 : Find a non-zero polynomial $Q(X,Y)$ such that,

1. $Q(X,Y) = E_1(X)Y - N_1(X)$
2. $\text{degree}(E_1) \leq k$ and $\text{degree}(N_1) \leq d + k$
3. $Q(\alpha_i, y_i) = 0, \forall i$

Step 2 : Output $\frac{N_1(X)}{E_1(X)}$ as $f(X)$

We first show that the first step of the algorithm will succeed in finding some non-zero Q.

Proposition 5 *A non-zero solution Q to Step 1 exists.*

Proof: Take $E_1 = E$, $N_1 = N$. ■

We next show that, assuming at most k errors have occurred, the second step outputs the correct polynomial $f(X)$.

Proposition 6 *Any solution* (E_1, N_1) *must satisfy* $\frac{N_1}{E_1} = f$.

Proof: Define the polynomial

$$R(X) = E_1(X)f(X) - N_1(X) \tag{5}$$

Fact 1: $\mathrm{degree}(R) \leq d + k$. This follows immediately from the conditions imposed on the degree of E_1 and N_1.
Fact 2: R has at least $n - k = d + k + 1$ roots. Indeed, for each locations i that is not in error, i.e., $f(\alpha_i) = y_i$, we have $R(\alpha_i) = Q(\alpha_i, y_i) = 0$.

Using the above two facts, we can conclude that R is identically 0 (Why?). This means that $f(X) = N_(X)/E_1(X)$, as claimed. ■

We now briefly comment on why the above algorithm admits an efficient implementation. Clearly the second step is easy – it is just polynomial division. For the first step to find Q, note that it can be solved by finding a non-zero solution to a homogeneous linear system with unknowns being the coefficients of the polynomials N_1, E_1, and the linear constraints being the n equations $Q(\alpha_i, y_i) = 0$. Since we guaranteed the existence of a nonzero solution, one can find some nonzero solution by Gaussian elimination in $O(n^3)$ field operations. Faster methods with near-linear running time are known for both the steps, and practically fast implementations of this algorithm are known and used millions of times everyday for error-correction.