EECS 70 Discrete Mathematics and Probability Theory
Fall 2014 Anant Sahai

Homework 8

This homework is due October 27, 2014, at 12:00 noon.

1. Section Rollcall!

In your self-grading for this question, give yourself a 10, and write down what you wrote for parts (a) and (b) below as a comment. You can optionally put the answers in your written homework as well.

(a) What discussion did you attend on Monday last week? If you did not attend section on that day, please tell us why.

(b) What discussion did you attend on Wednesday last week? If you did not attend section on that day, please tell us why.

2. Intro to Randomness Lab

We now have all the tools to start diving into the world of randomness. The following questions are designed to give you some intuition about concepts in probability. Some of the powerful ideas behind the plots will be further explored later in the course.

Starting from this Virtual Lab, please leave your answers in the empty Markdown cells in the skeleton. The converted pdf will be your entire answer to this question – there's no need to individually save figures or write down anything for the lab.

Please download the IPython starter code from Piazza or the course webpage, and answer the following questions.

As the problem statement above says, the main purpose of this virtual lab is to help you develop intuition about concepts in probability. For instance, intuitively, you can tell that if you do a large number of fair coin tosses, roughly half those tosses will result in a heads outcome, and the other (roughly) half will result in a tails outcome – since the coin has no inherent preference between heads and tails.

But what does "roughly half" mean? The idea is that, if you conduct an experiment where you toss a large number of coins many many times, and plot a histogram of the number of heads N that you get in each of these times, that should give you some intuition. Similarly, what does "large number of coin tosses" mean? Well, if you plot histograms for different numbers of coin tosses, and shift and scale these histograms in different ways, you will get some feel for the kinds of numbers that practically qualify as a "large number of coin tosses" for this purpose.

Another related question is: suppose I toss a coin k times; what are the chances that the fraction of heads that I get (which is supposed to be roughly half, remember) is less than or equal to half? In fact, we can be even more general and ask what are the chances that the fraction of heads is less than or equal to q, for every $0 \leq q \leq 1$? This will be explored more in Homework 9.

Please take a look at the solution and play around with the code, since the next few labs will be very similar in style to this one.

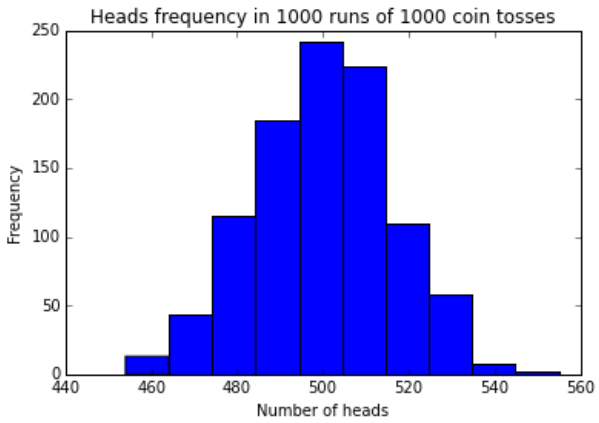
(a) Last week, we did 1000 coin tosses and plotted a bar chart of how many heads we got v.s. how many tails. This week, we will do the same thing again 1000 times.

Plot a histogram of how many times you got N heads, where $0 \leq N \leq 1000$. What do you observe about N and its frequency?

Hint: Implement the function `count_heads_in_runs(k, n)` in the skeleton, which returns a list of length n, where each element is the number of heads in k flips. In this case, $k = n = 1000$.

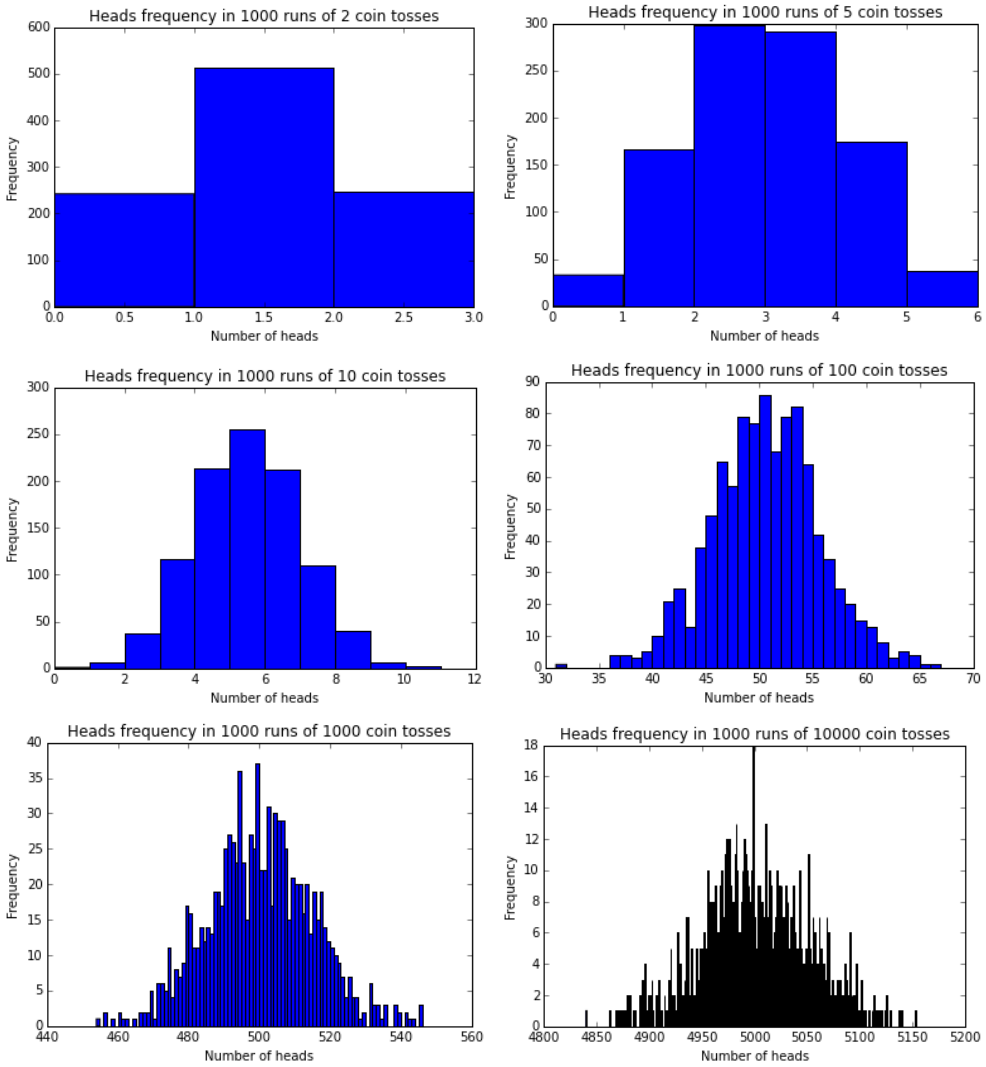
(In general, always try to parameterize the values instead of hard-coding it. This will make your code reusable for later parts.)

The results are tightly clustered around $N = 500$ (highest frequency). As expected, most of the time, half of the tosses are heads, and the other half are tails.



(b) Let k be a parameter that tells how many coins you toss in one experiment. Do part (a) again for the following sequence of ks: 2, 5, 10, 100, 1000, 10000. What do you observe as k gets larger?

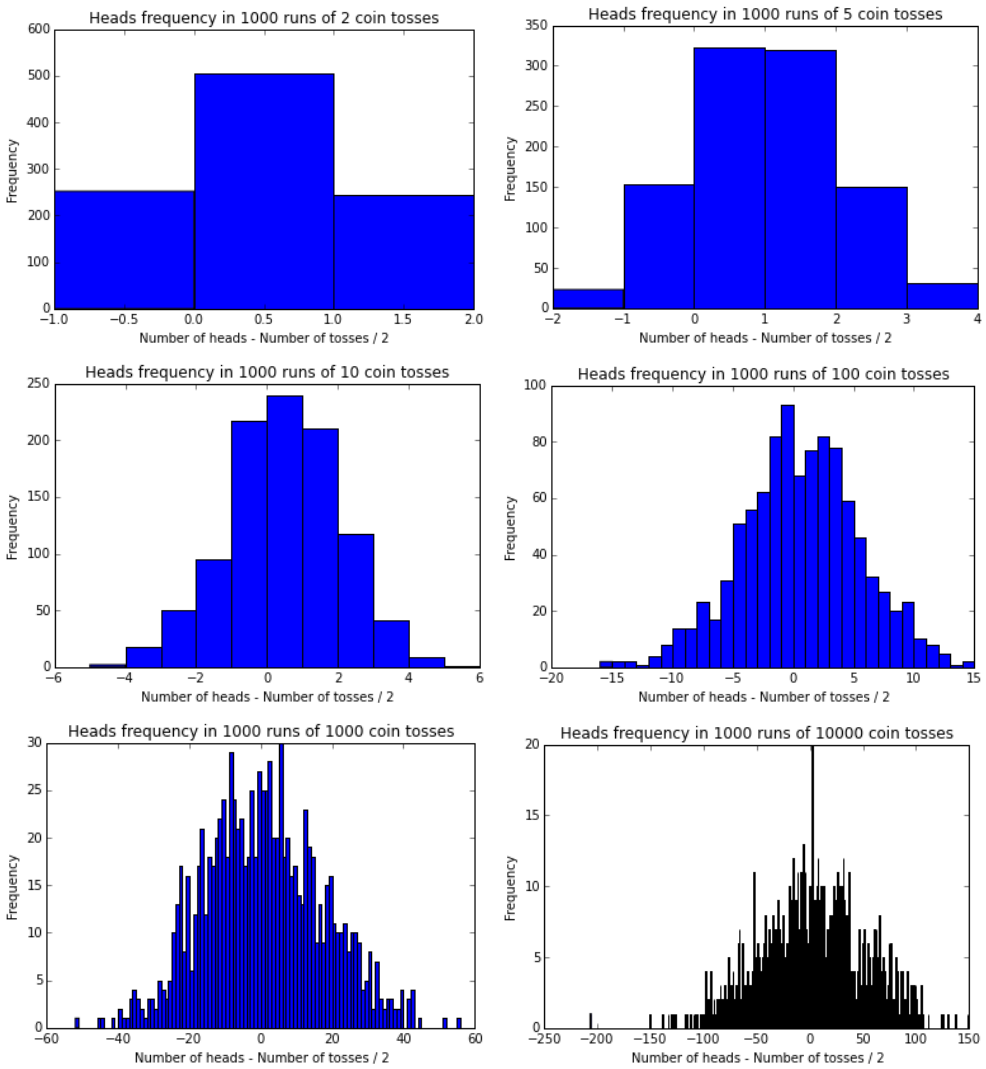
We observe that at lower values of k, the graph is somewhat random-looking, but it gets smoother and smoother as k increases.



(c) Notice that the horizontal axis has different scales as k varies. Suppose you wanted to "center" these histogram plots. How should you change your code for the above part to center the plots around the origin (0) as k varies?

Plot the resulting histogram of $N - f(k)$ where N is the number of heads in a particular run of fair coin tosses and $f(k)$ is the shift that you have chosen.

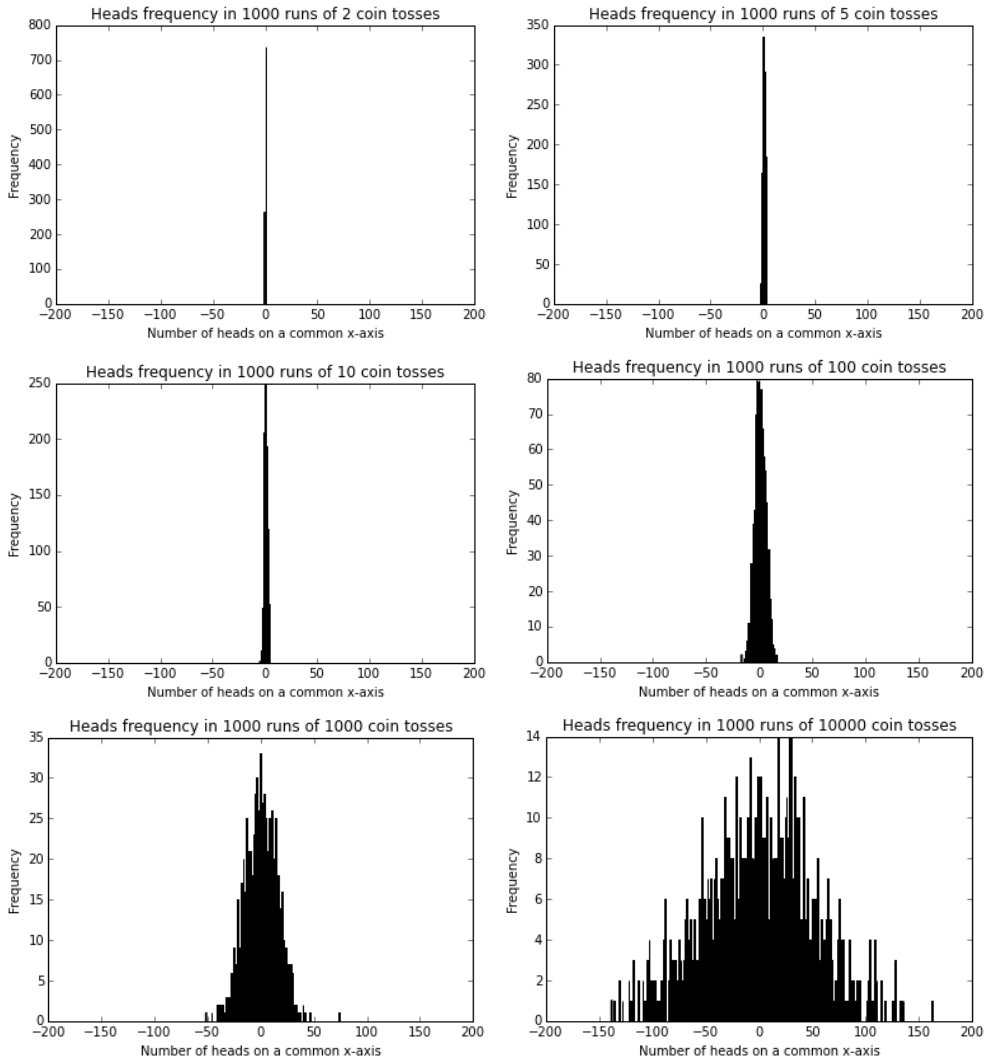
Since the results were clustered around half heads and half tails, we want to shift our plot left by half the number of flips, so that all the plots are centered at the origin.



(d) Repeat the plots of the previous part except this time, choose a common set of units for the x-axes (so the x-axes in all plots will have the same range). What range did you choose, and why?

Hint: You can use `plt.xlim()` to set the x-axis's range.

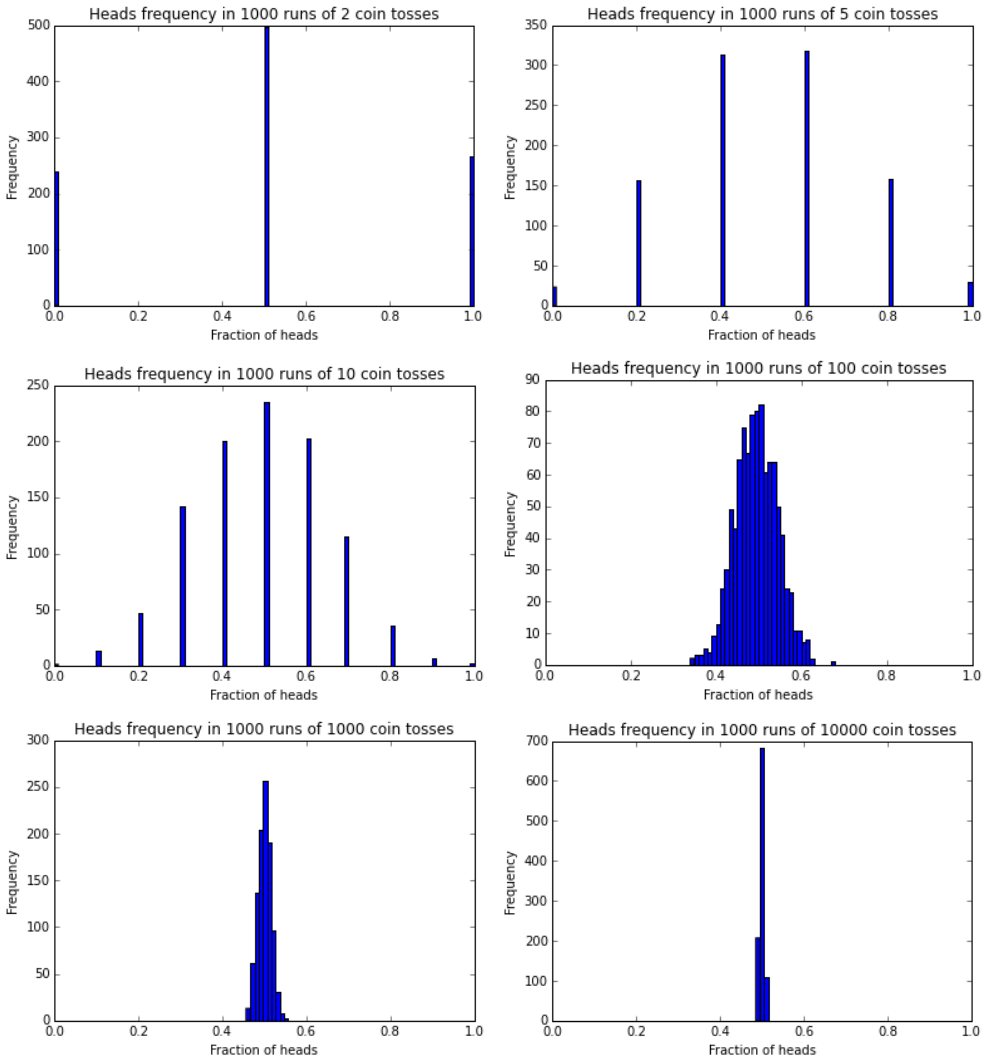
We selected the range to be $[-200, 200]$. By inspection, we rarely ever see a deviation more than that, so all the graphs will fit in this range.



(e) Repeat the plots of the previous part except this time, choose a normalized set of units, corresponding to the fraction of heads appearing (from 0 to 1).

The left-most point should correspond to the case of tossing all tails. And the right-most point should correspond to the case of tossing all heads. How is this set of plots different from the previous ones?

In contrast to our previous scale, the histograms are getting tighter and tighter as k increases.



(f) Comment on what you observed in the three sets of plots above. Five or less sentences should be sufficient.

- Part (c): As the number of tosses increases, the distribution moves closer to a bell-shaped curve at a coarse scale, but the fine features become quite jagged.
- Part (d): Since we are working on the same scale, the histogram gets wider as the number of tosses increases, since the number of heads also increases
- Part (e): The curve becomes tighter and tends toward the middle as the number of tosses increases. This makes sense because as we toss more coins, the chance of getting all heads or all tails decreases significantly comparing to say, when we toss 2 or 5 coins.

Reminder: When you finish, don't forget to convert the notebook to pdf and merge it with your written homework. Please also zip the `ipynb` file and submit it as `hw8.zip`.

3. Chinese Remainder Theorem for Polynomials

(a) Prove that the remainder of polynomial $p(x)$ divided by $(x-c)$ is $p(c)$.

By the polynomial division algorithm, we can find some polynomials $q(x), r(x)$ such that $p(x) = (x-c)q(x) + r(x)$ and $\deg r(x) < \deg(x-c) = 1$. Therefore $r(x)$ is just a constant (degree 0). Let the remainder $r(x) = e$. So we have $p(x) = (x-c)q(x) + e$

Now, evaluate both sides at $x = c$: $p(c) = (c-c)q(x) + e = e$.

(b) Consider extending the notion of "modding" to polynomials. Similar to how x mod 5 is the remainder when x is divided by 5 (or the equivalence class of $\{x + 5k : k \in Z\}$), let $p(x)$ mod $(x-1)$ be the remainder when $p(x)$ is divided by $(x-1)$ (or the equivalence class of... *what?*).

Solve the following system for all polynomials $p(x)$ over $GF(5)$ which satisfy:

$$\begin{aligned} p(x) &\equiv 3 && \mod (x-1) \\ p(x) &\equiv 3 && \mod (x+2) \\ p(x) &\equiv 1 && \mod (x) \end{aligned}$$

(Hint: Interpret the system using the result of part (a))

By part (a), we can interpret each congruence as specifying the value of $p(x)$ at a particular point. That is:

$$\begin{aligned} p(x) \equiv 3 \mod (x-1) &\iff p(1) = 3 \\ p(x) \equiv 3 \mod (x+2) &\iff p(-2) = 3 \\ p(x) \equiv 1 \mod (x) &\iff p(0) = 1 \end{aligned}$$

Now, we can use Lagrange Interpolation to find some polynomial $\widetilde{p}(x)$ which satisfies the above:

$$\begin{aligned} \widetilde{p}(x) &\equiv x(x+2) + x(x-1)(3) + (x-1)(x+2)(-2)^{-1} \\ &\equiv x^2 + x + 1 \end{aligned}$$

Notice Lagrange Interpolation only gives us the polynomial $\widetilde{p}(x)$ **of lowest degree** which matches the given points. To find all such polynomials, notice that all polynomials of the form:

$$p(x) = \widetilde{p}(x) + q(x)(x-1)(x+2)(x) \tag{1}$$

also match the given points, for any polynomial $q(x)$.

Further, any polynomial that matches the given points must be of the above form: Say the polynomial $f(x)$ satisfies $f(x) = \widetilde{p}(x)$ for $x = 1, -2, 0$ (that is, it matches the given points).

Consider dividing $f(x)$ by $(x-1)(x+2)(x)$, to get

$$f(x) = r(x) + g(x)(x-1)(x+2)(x)$$

where $\deg r(x) < 3$. Notice $f(x) = r(x) \forall x \in \{1, -2, 0\}$. Therefore, $r(x) = f(x) = \widetilde{p}(x) \forall x \in \{1, -2, 0\}$. But r and $\widetilde{p}$ are both polynomials of degree < 3 which agree on 3 points, therefore we must have $r = \widetilde{p}$, and $f(x)$ is of the form in (1).

By analogy to the CRT, notice that the given congruences only specify a unique polynomial $p(x)$ "mod $(x-1)(x+2)(x)$". To further help understanding, let us answer the hint posed in the problem statement: In the "mod 5" universe over integers, a number x mod 5 corresponds to the equivalence class $x + \mathbb{Z} = \{x + 5k : k \in Z\}$. Now in the "mod $(x-1)$" universe over polynomials with coefficients in $GF(5)$, a polynomial $p(x)$ corresponds to the equivalence class $\{p(x) + (x-1)q(x) : q(x) \in GF(5)[x]\}$. (That is, $q(x)$ is any polynomial with coefficients in $GF(5)$).

So for example, just like $3 \equiv 3 - (2)(5) \mod 5$, we have $(2x-1) \equiv (2x-1) - (2)(x-1) \equiv 1 \mod (x-1)$.

(c) From the previous HW, we know $gcd((x-1),(x+2)) = 1$. Still considering polynomials over $GF(5)$, does $(x+2)$ have a multiplicative inverse mod $(x-1)$?

Yes, the multiplicative inverse is 2, because:

$$(x+2)(2) \equiv 2x-1 \equiv 1+2(x-1) \equiv 1 \bmod (x-1)$$

We could compute this by running $egcd(x+2, x-1)$, to find $3 = (1)(x+2)+(-1)(x-1)$. Then we multiply both sides by $3^{-1} = 2$ (since coefficients are from $GF(5)$), to find $1 = (2)(x+2)+(-2)(x-1)$.

Alternatively, we could notice: $x+2 \equiv x+2-(x-1) \equiv 3 \bmod (x-1)$ So $(x+2)^{-1} \equiv 3^{-1} = 2 \bmod (x-1)$. This is the same as using the result of part (a), but written more explicitly.

It may seem like we are simultaneously working "mod 5" and "mod $(x-1)$". This is due to our notation. We are considering polynomials with coefficients in $GF(5)$, a field of 5 elements. Instead of arbitrarily calling these elements $\{0,1,a,b,c\}$, and remembering rules like $a*a = c$ and $a+b = 0$, we suggestively denote elements by integers (instead of "a", we write "2" or "7" or "-3", etc). We know that $GF(5)$ can be thought of as equivalence classes of integers $\bmod 5$, which justifies this notation.

(d) Show how to solve the system of congruences in part (b) using the explicit form of the Chinese Remainder Theorem (by analogy to the usual CRT). Comment on the similarities/differences to how you solved (b) previously.

By analogy to the usual CRT, we can construct:

$$\begin{aligned} p(x) \equiv\ & (3)x(x+2)[x(x+2)]^{-1}_{x-1} \\ & + (3)x(x-1)[x(x-1)]^{-1}_{x+2} \\ & + (1)(x-1)(x+2)[(x-1)(x+2)]^{-1}_{x} \end{aligned}$$

Where the notation $[f(x)]^{-1}_{x-c}$ means "inverse of $f(x)$ mod $(x-c)$". Finally, notice that by part (a): $f(x) \equiv f(c) \bmod (x-c)$, so

$$[f(x)]^{-1}_{x-c} = f(c)^{-1}$$

(If you didn't notice this, you could still write, for example: $[x(x+2)]^{-1}_{x-1}$ as $[x]^{-1}_{x-1}[x+2]^{-1}_{x-1}$, then compute each factor individually as in part (b)).

Using this simplification, we find the **CRT gives us exactly the same form as Lagrange Interpolation!** That is, the delta polynomials are exactly the same, and even constructed in exactly the same way.

In this case:

$$\begin{aligned} p(x) &\equiv (3)x(x+2)(3)^{-1} \\ &\quad + (3)x(x-1)(1)^{-1} \\ &\quad + (x-1)(x+2)(-2)^{-1} \\ &\equiv x^2+x+1 \end{aligned}$$

(e) *(optional)* Now that we can take the inverse of certain polynomials in the "mod polynomial" universe (as in part (c)), let's see how far this takes us (unrelated to the CRT). Consider the set of all polynomials over $GF(3)$, modulo (x^2+1). How many distinct elements are there? How many of them have multiplicative inverses? Does this construction form a field?

There are 9 elements: $S = \{ax+b : a,b \in GF(3)\}$. (Notice we can define addition and multiplication over this set in the natural way.)

All non-zero elements have multiplicative inverses. This could be confirmed manually, or by noticing that (x^2+1) has no roots in $GF(3)$, so it cannot be factored further. Therefore for any two non-zero elements $u,v \in S$, we must have $uv \neq 0$. Since S is finite and closed under multiplication, we may conclude that every non-zero element u has an inverse: uS must be a permutation of S, because if $\exists a \neq b$ such that $ua = ub$, then $u(a-b) = 0$ but $u \neq 0, (a-b) \neq 0$, contradiction. (If you look closely, this is similar to how we proved the FLT).

For example, $(2x+1)^{-1} = 2x+2$, because $(2x+1)(2x+2) = 4x^2+6x+2 \equiv x^2+2 = 1+(x^2+1) \equiv 1$. Similarly, $x^{-1} = -x$, because $x(-x) = -x^2 \equiv -x^2+(x^2+1) = 1$.

The other field axioms follow naturally by construction (we can add, subtract, multiply, addition commutes, and multiplication distributes). Therefore we have a finite-field with 9 elements! This field has different structure than fields we've encountered before... for example, we cannot get all elements by simply adding 1s: $0, 1, 1+1, 1+1+1$, etc. However, we can get all non-zero elements by considering powers of $(x+1)$: $1, (x+1), 2x, 2x+1, 2, 2x+2, x, x+2$. In fact, any finite field with 9 elements must behave exactly like this one, so we are justified in calling it $GF(9)$.

Polynomials have a very deep algebraic structure, as you've seen here. Those interested are encouraged to take abstract algebra, Math (H)113.

4. Guardians of the Galaxy Rendez-Vous

The Guardians of the Galaxy are back, and this time they need your help! The band has gone their separate ways, and Peter Quill needs to organize an urgent meeting to alert his friends about the latest threat to the galaxy. Unfortunately, communication across the galaxy still isn't perfect – stray radiation can erase parts of messages!

Rocket the Raccoon has told Peter about a strategy to get his message across:

1. There are only 105 possible safe planets Peter would use to meet his friends, and he labels them 0 to 104. His friends also know which planet each number ID refers to.
2. He takes the unique ID and finds the remainder mod 3, 5, 7, 11 and 13.
3. This is the message he sends across, as a five-number tuple.
4. The receiver makes clever use of the CRT to find the unique message.

For example, if Peter wants to go planet 51, he sends the message $(0,1,2,7,12)$. If the first symbol is erased in transit, the received message will be $(X,1,2,7,12)$.

(a) Gamora gets the message $(X,4,X,6,3)$! What planet does she thinks Peter is at?

Gamora gets the message (X, 4, X, 6, 3), which corresponds to the following table:

X	4	X	6	3
3	5	7	11	13

She can then set up the following congruences:

$$x \equiv 4 \bmod 5$$
$$x \equiv 6 \bmod 11$$
$$x \equiv 3 \bmod 13$$

Through a straightforward application of the Chinese Remainder Theorem, we find:

$$\begin{aligned} x &\equiv (4)(143)[143^{-1}]_5 + (6)(65)[65^{-1}]_{11} + (3)(55)[55^{-1}]_{13} \bmod (5*11*13) \\ &\equiv (4)(143)[3^{-1}]_5 + (6)(65)[10^{-1}]_{11} + (3)(55)[3^{-1}]_{13} \bmod 715 \\ &\equiv 4(143)(2) + (6)(65)(10) + (3)(55)(9) \bmod 715 \\ &\equiv 6529 \bmod 715 \\ &\equiv \boxed{94} \end{aligned}$$

So Gamora thinks Peter wants to rendez-vous at the planet $\boxed{94}$.

(b) Although Peter wants to use this scheme, he's still a little paranoid everyone will make the rendez-vous. Describe the "clever use" of the CRT more precisely. How many radiation-erasures can this scheme tolerate, in the worst case? Prove it.

Assume the received message has ≤ 2 erasures.

Select some three pairs of received (prime, residue) as $(p_1, r_1), (p_2, r_2), (p_3, r_3)$. For example, for part (a), Gamora would have chosen $(5,4), (11,6), (13,3)$. The decoding algorithm is:

1. Use the CRT to find the message x with residue $r_i \pmod{p_i}$ for $i = 1,2,3$. This x will be unique mod $p_1 p_2 p_3$.
2. Since the original message x was chosen with $0 \leq x < 105 = (3)(5)(7) \leq p_1 p_2 p_3$, knowing the message uniquely $\bmod p_1 p_2 p_3$ is sufficient to decode the exact message. Notice in this step, it was important the message was chosen to be less than the product of the three smallest primes – so any other set of 3 primes (corresponding to the non-erased locations) has a larger product.

So we can correctly decode with ≤ 2 erasures. Assume for contradiction that this CRT code can tolerate any 3 erasures. Then we could erase the 3 largest primes, and only receive the message x mod 3 and mod 5. This only uniquely identifies the message mod 15, not mod 105, so we cannot decode uniquely. (In particular, $x = 0$ and $x = 15$ both have the same residues mod 3 and mod 5).

Notice that if the first three locations were erased, we would in fact have enough information to uniquely decode, since $11 * 13 \geq 105$. But we are considering the worst-case pattern of erasures.

(c) Even with Ronan gone, Thanos and the Ravagers are still out to get Peter! They will do their best to corrupt his message. Groot gets the message $(2,1,2,2,10)$, which may have at most one corruption. Can he/it determine where Peter wants to meet?

We are given that at most one location was corrupted. So we try the brute-force decoding strategy:

1. Assume location 1 was corrupted. Treat it as an erasure, and decode the remaining message to some x'.
2. Re-encode x', and see if the encoding matches differs from the received message in at most 1 location.
3. If not, repeat for locations $2,3,4,5$.

We find that only one message $x' = 101$ matches the received codeword in enough locations. This encodes as $101 \mapsto (2,1,3,2,10)$. So Peter wants to meet at Planet 101.

In fact, for such a small message-space, we can execute an even bruter-force decoding strategy (with help from a computer): Simply try all possible messages $x \in 0...105$, and see which one differs from the received codeword in at most 1 location.

But how do we know we will decode correctly – that is, what if there are two possible codewords which both differ in at most 1 location from the received message? This is impossible: Fix some received-tuple Z. For contradiction, assume there were some two messages, $x \neq y$ such that $d_H(E(x), Z) \leq 1$

and $d_H(E(y),Z) \leq 1$. (Where d_H is the hamming-distance). But then $d_H(E(x),E(y)) \leq 2$ (that is, they differ in at most two locations). Now consider sending the message x, encoded as $E(x)$, then erasing the locations of these two differences. Then a receiver can't determine if we encoded the message x or y! But we have already proven that this code can tolerate 2 erasures, contradiction.

Note: In general, this reflects how any $2t$-erasure-correcting code can be used as a t-error-correcting code.

5. Magic!

In this problem we will investigate what happens when in error-correcting codes there are fewer errors than the decoding algorithm is able to handle. For the entire problem we are working in $GF(7)$.

Assume that we wish to transfer a message of length 2, which we denote by (m_1,m_2). Each m_i is a member of $GF(7)$. We also wish to be able to correct up to $k = 1$ error. Using the error-correcting codes we learned in class, we have to first find a polynomial $P(x)$ of degree at most 1 such that $P(1) = m_1$ and $P(2) = m_2$. Then we have to extend the message we send by $2k$ symbols, i.e., we will send $(P(1),P(2),P(3),P(4))$ to the recipient.

(a) Consider an example where $(m_1,m_2) = (4,2)$. What are the four symbols that are transmitted?

We need $P(1) = 4$ and $P(2) = 2$. Suppose the polynomial is of the form $ax+b$. Then, solving the linear equations

$$\begin{aligned} a+b &= 4 \\ 2a+b &= 2, \end{aligned}$$

leads to $P(x) = 5x+6$. Therefore, the message will be $(P(1),P(2),P(3),P(4)) = (4,2,0,5)$. □

(b) Now let's work with a different message of length 2. Assume that you have received these numbers: $(5,0,2,4)$, i.e., if there were no errors then we would have $P(1) = 5, P(2) = 0, P(3) = 2, P(4) = 4$. Write down the linear equations that help decode error-correcting codes: $Q(i) = P(i)E(i) = r_iE(i)$, for $1 \leq i \leq 4$.

We are looking for

$$\begin{aligned} P(x) &= ax+b, \\ E(x) &= x+e_0, \\ Q(x) = q_2x^2+q_1x+q_0 &= P(x)E(x) = r_xE(x). \end{aligned}$$

The four equations are

$$\begin{aligned} Q(1) &= q_2+q_1+q_0-5e_0 = 5 \\ &= q_2+q_1+q_0+2e_0 = 5 && (2) \\ Q(2) &= 4q_2+2q_1+q_0 = 0 && (3) \\ Q(3) &= 9q_2+3q_1+q_0-2e_0 = 6 \\ &= 2q_2+3q_1+q_0+5e_0 = 6 && (4) \\ Q(4) &= 16q_2+4q_1+q_0-4e_0 = 16 \\ &= 2q_2+4q_1+q_0+3e_0 = 2 && (5) \end{aligned}$$

□

(c) Try to solve the linear equations you got in the previous section. You should observe that there are multiple solutions to these equations. Pick two different solutions and for each one write down the error-locating polynomial $E(x)$ and the polynomial $Q(x)$. In each of the two solutions, divide $Q(x)$ by $E(x)$ to get the original polynomial. Do you get the same polynomial in both cases?

One common mistake we found in the HW Party is solving the equations with solvers for real numbers. Equations that are linearly independent in the reals can be linearly dependent in modular arithmetic and vice versa. This particular system of equations happens to be just such a case. Although it has a unique solution over the reals, it does not in mod 7.

We will solve the system using Gaussian Elimination.

$$\left[\begin{array}{cccc|c} 1 & 1 & 1 & 2 & 5 \\ 4 & 2 & 1 & 0 & 0 \\ 2 & 3 & 1 & 5 & 6 \\ 2 & 4 & 1 & 3 & 2 \end{array}\right] \rightarrow \left[\begin{array}{cccc|c} 1 & 1 & 1 & 2 & 5 \\ 0 & 5 & 4 & 6 & 1 \\ 0 & 1 & 6 & 1 & 3 \\ 0 & 2 & 6 & 6 & 6 \end{array}\right] \begin{array}{l} R_1 \\ R_2 - 4R_1 \\ R_3 - 2R_1 \\ R_4 - 2R_1 \end{array}$$

$$\rightarrow \left[\begin{array}{cccc|c} 1 & 0 & 3 & 5 & 2 \\ 0 & 1 & 5 & 4 & 3 \\ 0 & 0 & 1 & 4 & 0 \\ 0 & 0 & 3 & 5 & 0 \end{array}\right] \begin{array}{l} R_1 - 3R_2 \\ 3R_2 \\ R_3 - 3R_2 \\ R_4 - 2 \times 3R_2 \end{array}$$

$$\rightarrow \left[\begin{array}{cccc|c} 1 & 0 & 0 & 0 & 2 \\ 0 & 1 & 0 & 5 & 3 \\ 0 & 0 & 1 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right] \begin{array}{l} R_1 - 3R_3 \\ R_2 - 5R_3 \\ R_3 \\ R_4 - 3R_3 \end{array}$$

Observe that we have lost one equation along the way, meaning the system has multiple solutions. Writing the matrix back to equations,

$$q_2 = 2 \tag{6}$$
$$q_1 + 5e_0 = 3 \tag{7}$$
$$q_0 + 4e_0 = 0 \tag{8}$$

Picking $e_0 = 1$ gives us,

$$E(x) = x + 1$$
$$Q(x) = 2x^2 + 5x + 3$$
$$P(x) = \frac{Q(x)}{E(x)} = 2x + 3$$

Picking $e_0 = -1$ gives us,

$$E(x) = x - 1$$
$$Q(x) = 2x^2 + x + 4$$
$$P(x) = \frac{Q(x)}{E(x)} = 2x + 3$$

Both give the same polynomial $P(x)$ with $P(1) = 5, P(2) = 0, P(3) = 2, P(4) = 4$. □

(d) Do both $E(x)$'s have the same root? What does that tell you about the position of error in the transmitted message?

No, they don't have a common root. In fact, anything can be the root because we get to pick e_0. This means the error in the transmitted message can be anywhere, but this contradicts the fact that the error can be at only one place, so there must be no error.

If you're not convinced, think of the meaning behind the Berlekamp-Welch algorithm. We are finding k points that, when ignored, allow us to link through the rest $n+k$ points with a polynomial of degree just $n-1$, which only happens when that polynomial is $P(x)$. Back to our example, the fact that the root of $E(x)$ can be anywhere means any subset of 3 received packets are correct, implying that all 4 points are correct. □

(e) Consider accounting for $k=2$ general errors instead of $k=1$. If we want to send a message of length 2, we have to send $2+2k=6$ packets. Suppose only one packet is corrupted, then you will see multiple possible $Q(x)$'s and $E(x)$'s again. What do you expect in common between these $E(x)$'s? Give a brief justification. *(Hint: Think about the positions of errors they point to again.)*

They share one common root which is the position of that one error. The other root is arbitrary.

$$Q(x) = E(x)P(x) = (x - anything)(x - common\_root)P(x)$$

□

6. Po(l)lynomial Pranks

Alex and Barb talk to each other via Polly. Knowing her tendency to prank, they use polynomials to ensure they can recover their original messages in case she decides to erase (i.e., replace with a blank) or change some of the packets.

Throughout this question, let x_i and y_i denote the i^{th} x and y values the sender sends, and x'_i and y'_i denote the i^{th} x and y values the receiver receives. We use one-based indexing. Although we only show the solutions for interpolation-encoding Reed-Solomon scheme, similar techniques can be applied to coefficient-encoding codewords as well.

(a) Never the one to run out of ideas, Polly adds an integer offset c to the x values of the packets instead, i.e., each packet (x,y) becomes $(x+c,y)$. Will Alex and Barb be able to get their original messages back without knowing c beforehand? Do they need to modify their scheme to handle this prank? If so, describe the method briefly.

They can get their original message back without having to change anything.

When c is added, each packet (x_i,y_i) becomes (x_i+c,y_i). The y_i values are unaffected, and the ordering of the packets doesn't change, i.e., if $x_i < x_j$, then $x'_i = x_i + c < x_j + c = x'_j$. The receiver can just read the received values as normal. □

(b) Realizing what you just showed, Polly adds the integer offset c to the y values of the packets instead, i.e., each packet (x,y) becomes $(x,y+c)$. Can Alex and Barb get their original messages back using their current scheme? If not, propose a modified scheme that will work.

No, they can't. There are at least a few possible schemes.

Method 1: Send one additional packet at the end with $y=0$ to figure out the offset c to subtract back.

- **Sender:** Interpolate n data points to find $P(x)$. Send $(x_1,P(x_1)),(x_2,P(x_2)),\ldots,(x_n,P(x_n)),(x_{n+1},0)$.
- **Receiver:** Let $c = y'_{n+1}$. Get $y'_1 - c$, $y'_2 - c$, ..., $y'_n - c$ as the original message.

Method 2: Shift $P(x)$'s degree by one place and let Polly's offset occupy the coefficient of x_0.

- **Sender:** Interpolate n data points to find $P(x)$. Create a degree-n polynomial $\hat{P}(x) = xP(x)$ to let c occupy the coefficient of x^0. Send $n+1$ packets, $(x_1,\hat{P}(x_1)),(x_2,\hat{P}(x_2)),\ldots,(x_n,\hat{P}(x_n))$, and $(x_{n+1},\hat{P}(x_{n+1}))$.

- **Receiver:** Interpolate the points to find $\hat{P}(x) = b_n x^n + \ldots + b_1 x + b_0$. Let $c = b_0$ and recover the original message $(y'_1 - c)(x'_1)^{-1}$, $(y'_2 - c)(x'_2)^{-1}$, ..., $(y'_n - c)(x'_n)^{-1}$.

Both of the schemes need 1 additional packet = $(n+1)$ minimum packets in total.

Method 3: (Student's solution from HW Party) Swap x's and y's so the noise perturbs x values instead.

- **Sender:** Interpolate n data points to find $P(x)$. Send $(P(x_1), x_1), (P(x_2), x_2), \ldots, (P(x_n), x_n)$.
- **Receiver:** Get the original message $x'_1, x'_2, \ldots, x'_n$.

This method only needs n packets. It is worth noting that this only works because the offset is constant and doesn't affect the packet ordering. □

(c) Polly changes her mind again. Adding a constant offset c is too simple. She now picks a random po(l)lynomial $N(x)$ of degree $\leq d$ and adds it to the y values instead, i.e., each packet (x, y) becomes $(x, y + N(x))$. Note that $N(x)$ can have higher degree than $P(x)$. Suppose Alex and Barb know d, provide a scheme for them to reliably communicate using the minimum number of packets. You only need to give a brief justification.

This is an extension of the solutions for part (b).

Method 1: Send $d+1$ additional packets at the end with $y = 0$ to figure out $N(x)$ to subtract back.

- **Sender:** Interpolate n data points to find $P(x)$.
  Send $(x_1, P(x_1)), (x_2, P(x_2)), \ldots, (x_n, P(x_n)), (x_{n+1}, 0), (x_{n+2}, 0), \ldots, (x_{n+d+1}, 0)$.
- **Receiver:** Interpolate for $N(x)$ from points $(x'_{n+1}, y'_{n+1}), (x'_{n+2}, y'_{n+2}), \ldots, (x'_{n+d+1}, y'_{n+d+1})$.
  Recover the original message $y'_1 - N(x'_1), y'_2 - N(x'_2), \ldots, y'_n - N(x'_n)$.

Method 2: Shift $P(x)$'s degree by $d+1$ places.

- **Sender:** Interpolate n data points to find $P(x)$. Create a degree-$n+d$ polynomial $\hat{P}(x) = x^{d+1} P(x)$ to let $N(x)$ occupy the coefficients of x^d to x^0.
  Send $n+d+1$ packets, $(x_1, \hat{P}(x_1)), (x_2, \hat{P}(x_2)), \ldots, (x_{n+d}, \hat{P}(x_{n+d}))$, and $(x_{n+d+1}, \hat{P}(x_{n+d+1}))$.
- **Receiver:** Interpolate the points to find $\hat{P}(x) = b_{n+d} x^{n+d} + \ldots + b_1 x + b_0$. Let $N(x) = b_d x^d + \ldots + b_1 x + b_0$ and recover the original message $y_i = (y'_i - N(x'_i))(x'^{d+1}_i)^{-1}$.

Note that Method 3 from part (b) doesn't work because it can give wrong or undefined ordering of the message. For example, let $P(x) = 2x$, $N(x) = -x$, and $n = 3$. Let the original packets from sender before the x-y swap be $(1,2), (2,4), (3,6)$. The packets that arrive Polly will be $(2,1), (4,2), (6,3)$. Polly then delivers $(2, 1+N(2)), (4, 2+N(4)), (6, 3+N(6)) = (2,-1), (4,-2), (6,-3)$, effectively reversing the order of the message. The receiver swaps x and y back, sorts by ascending x values and gets 6, 4, 2 as the message instead of 2, 4, 6.

Next, we show that $n+d+1$ *is indeed the minimum number of packets*. Let

$$P(x) = a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_1 x + a_0, \tag{9}$$

$$N(x) = b_d x^d + b_{d-1} x^{d-1} + \ldots + b_1 x + b_0. \tag{10}$$

Let $Q(x)$ denote the received polynomial, i.e., $Q(x) = P(x) + N(x)$,

$$Q(x) = a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_1 x + a_0 + b_d x^d + b_{d-1} x^{d-1} + \ldots + b_1 x + b_0. \tag{11}$$

$Q(x)$ consists of $n+d+1$ unknown variables, $a_0, a_1, \ldots, a_{n-1}$, and $b_0, b_1, \ldots, b_d$. Although we only need to know either $a_0, a_1, \ldots, a_{n-1}$ or $b_0, b_1, \ldots, b_d$, we need at least $n+d+1$ pairs of $(x, Q(x))$ to generate $n+d+1$ equations to solve the system. Hence, the minimum number of packets is $n+d+1$.

Then what about just sending $d+1$ extra packets, constructing $n+d+1$ equations and solving them? There will be cases where the $n+d+1$ packets offer redundant data, leading to degenerated system of equations, and we won't be able to figure out the unique original message. For example, consider $N(x) = -P(x)$, the received packets will be $(x_1, P(x_1)+N(x_1)), \ldots, (x_{n+d+1}, P(x_{n+d+1}) + N(x_{n+d+1})) = (x_1, 0), \ldots, (x_{n+d+1}, 0)$, which are not useful at all. □

(d) Extend the scheme in part (c) to account for an additional k_e erasure errors. Does your scheme still use the minimum number of packets? If not, come up with a new scheme that does. You should give a brief justification.

From part (c), if we use Method 1 , we would need $2k_e$ extra packets, k_e for $N(x)$ and k_e for the actual message. But we know from the lecture we can guard against k_e erasures using just k_e extra packets. Can we send less? Yes, by merging all the information into one polynomial. We can do that for Method 1 with a slight modification (or just a different interpretation of the packets, really.) Method 2 works right away.

Method 1: (Student's solution from HW Party) Add $d+1$ data points with $y = 0$.

- **Sender:** Add $d+1$ data points, $(x_{n+1}, 0), \ldots, (x_{n+d+1+k_e}, 0)$. Interpolate to find $P(x)$ of degree up to $n+d$ that passes through all $n+d+1$ data points. Use $P(x)$ to generate and send $n+d+1+k_e$ packets, $(x_1, P(x_1)), (x_2, P(x_2)), \ldots, (x_n, P(x_n)), (x_{n+1}, 0), (x_{n+2}, 0), \ldots, (x_{n+d+1+k_e}, 0)$.
- **Receiver:** The received packets will be of form $(x'_i, P(x'_i) + N(x'_i))$. Let $R(x) = P(x) + N(x)$. Since $R(x)$ can be of at most degree $n+d$, we can interpolate the received $n+d+1$ packets to find $R(x)$. Then we can generate all missing packets, then use $(x'_{n+1}, y'_{n+1}), \ldots, (x'_{n+d+1}, y'_{n+d+1}) = (x'_{n+1}, N(x'_{n+1})), \ldots, (x'_{n+d+1}, N(x'_{n+d+1}))$ to generate $N(x)$. Finally, recover the original message $y'_1 - N(x'_1), \ldots, y'_n - N(x'_n)$.

Method 2: Shift $P(x)$'s degree by $d+1$ places.

- **Sender:** Interpolate n data points to find $P(x)$. Create a degree-$n+d$ polynomial $\hat{P}(x) = x^{d+1}P(x)$ to let $N(x)$ occupy the coefficients of x^d to x^0.
  Send $n+d+1+k_e$ packets, $(x_1, \hat{P}(x_1)), \ldots, (x_{n+d+k_e}, \hat{P}(x_{n+d+k_e}))$, and $(x_{n+d+k_e+1}, \hat{P}(x_{n+d+k_e+1}))$.
- **Receiver:** Interpolate the $n+d+1$ points received to find $\hat{P}(x) = b_{n+d}x^{n+d} + \ldots + b_1 x + b_0$. Let $N(x) = b_d x^d + \ldots + b_1 x + b_0$ and recover the original message $y_i = (y'_i - N(x'_i))(x'^{d+1}_i)^{-1}$.

□

(e) Modify the scheme in part (d) to account for an additional k_g general errors instead of erasure errors.

Again, from the lecture, we should need $2k_g$ extra packets to recover from general errors. Fortunately, both methods in part (d) work. We can just send $n+d+1+2k_g$ packets instead of $n+d+1+k_e$ and then run Berlekamp-Welch algorithm to detect the error and recover the right polynomial. □

7. Error-Detecting Codes

In the realm of error-correcting codes, we usually want to recover the original message if we detect any errors, and we want to provide a guarantee of being able to do this even if there are k general errors. Suppose that instead we are satisfied with detecting whether there is any error at all and do not care about the original message if we detect any errors. In class you saw that for recovering from at most k general errors when transmitting a message of length n you need to extend your message by $2k$ symbols and send a message of length $n+2k$. But since we don't require recovering the original message, it is conceivable that we might need less symbols.

Formally, suppose that we have a message consisting of n symbols that we want to transmit. We want to be able to detect whether there is any error if we are guaranteed that there can be at most k general errors. That

is, your receiver should be able to say either 'this message is completely correct' and decode it, or say 'this message has at least one error' and throw it away. How should we extend our message (i.e. by how many symbols should we extend, and how should we get those symbols) in order to be able to detect whether our message has been corrupted on its way? You may assume that we work in $GF(p)$ for a very large prime number p. Show that your scheme works, and that adding any lesser number of symbols is not good enough.

We claim that we need to extend our message by k symbols in order to be able to detect up to k errors. Suppose that the message is extended to $m_1, m_2, \ldots, m_k, m_{k+1}, \ldots, m_{k+n}$. The encoding procedure is exactly the same as what we saw in the lecture. We do the following detection algorithm. Find the unique polynomial of degree $n-1$ that passes through points $(1, m_1)$ up to (n, m_n). Call this polynomial to $P(x)$. If all of the points corresponding to the extended symbols $(n+1, m_{n+1})$ up to $(n+k, m_{n+k})$ lie on $P(x)$ then we declare that there are no errors. Otherwise the message is corrupted.

Now we argue that why the above extended message and the detection algorithm work:
We know that there are at most k errors. In other words, at least n of the points are still correct. We know that n points are enough to fully determine a degree at most $n-1$ polynomial, and since these points are all still "correct", the only polynomial of degree $n-1$ that goes through them is the original polynomial that interpolated the n original symbols. So if any of the other points have been changed, and therefore do not lie on the original polynomial, then the original polynomial would not match these points, so there would be no degree $n-1$ polynomial that matches all the points sent. Otherwise, if all the points are unchanged, it is clear that by our construction that we will get the original polynomial going through the points.

Now we prove that if we extend the message by any number of symbols less than k, then the adversary can perturb the symbols such that the detector does not find it out. Suppose that the message is extended by $k-1$ symbols, m_{n+1} up to m_{n+k-1}. Then the adversary can change symbol m_n to $\tilde{m}_n$, and find the polynomial that passes through the points $(1, m_1)$ up to $(n, \tilde{m}_n)$. Let this polynomial be $\tilde{P}(x)$. Then the adversary can change the extended symbols such that all of the points $(n+1, \tilde{m}_{n+1})$ up to $(n+k-1, \tilde{m}_{n+k-1})$ lie on $\tilde{P}(x)$. (Note that the adversary can perturb up to k symbols) Therefore, the detector cannot detect that the message is corrupted. This shows that we need at least k extra symbols to detect k errors.

8. Orpheus' Adventures in the Halls of Time

You're designing a new role-playing game for a mathematically themed production house. Your eccentric colleague comes to you with an idea for a key scene and he wants you to think about it.

The backstory is that the mortal Orpheus wants to gain knowledge of the dates of certain key events in the year to come: call these the prophecies of interest. He has heard that in the Halls of Time, these things are already known so he quests through the underworld until he comes upon them.

In the Halls of Time, he encounters the Guardians. They have access to the knowledge of the Fates.

The game behaves as follows. There are 12 guardians (corresponding to the 12 constellations of the Zodiac or the 12 months) and each knows all the prophecies, but they have a peculiar property. Half of them are honest and answer questions posed to them exactly. One quarter of them consider mortals to be beneath them and will simply say "Begone mortal!" And one quarter despise mortals and will answer maliciously.

But mortals do not know the secret forms of the guardians and so Orpheus doesn't know who he is talking to.

On this setting, Orpheus can only ask questions (he can invoke arithmetic operations in $GF(367)$ if he wants) whose answer is a number from $\{0, 1, 2, \ldots, 366\}$.

(The prophecies he wants are answers to questions like: "When will my child be born?" The answers can be viewed as numbers: 1, . . . , 365 for the days in the coming year. 0 for the past. 366 to represent the future beyond this coming year. Fortunately for Orpheus, 367 happens to be prime.)

All guardians are good at math and can answer any question as long as the answer is from 0 to 366 (not limited to just a simple answer to a prophecy). Orpheus can only ask any individual guardian one question. After that, that particular guardian will magically leave the room. He gets to question all 12 guardians.

How many prophecies can Orpheus reliably extract from the 12 guardians? How can he do it? (Be explicit) Why will this work?

Orpheus can reliably extract 3 prophecies.
Suppose the answers of the three prophecies Orpheus is interested are a_1, a_2 and a_3 (note that Orpheus does not know the answers in advance). Orpheus can ask the Guardians by either ways as follows:

- **Method 1:** Orpheus can ask the i-th Guardian to form a polynomial of degree≤ 2 over $GF(367)$ with a_1, a_2, a_3 as coefficients, i.e. $P(x) = a_3x^2 + a_2x + a_1$, and to tell him what is $P(i)$.
- **Method 2:** Orpheus can ask the i-th Guardian to form a polynomial $P(x)$ of degree≤ 2 over $GF(367)$ such that $P(1) = a_1, P(2) = a_2$ and $P(3) = a_3$, and to tell him what is $P(i)$.

Let r_i be the i-th answer Orpheus got from the Guardians. He knows that among all r_i's, there are only 9 valid answers because 3 of the Guardians will always answer 'Begone mortal'. Among the 9 valid answers, there are 3 malicious answers. Collecting all the answers r_i's from every Guardian, he can recover the polynomial $P(x)$ by ignoring the 3 missing answers and using Berlekamp-Welch algorithm with the remaining 9 answers (3 of them may be wrong).

If Orpheus uses **method 1**, he can find out a_1, a_2, a_3 by just looking at the coefficients of $P(x)$.
If Orpheus uses **method 2**, he can find out a_1, a_2, a_3 by $a_1 = P(1), a_2 = P(2), a_3 = P(3)$.

We can analogize the 12 answers as a message of 12 packets which subjects to $k_e = 3$ erasure errors and $k_g = 3$ general errors, so the above framework can recover a polynomial of degree at most $n = 12 - k_e - 2k_g = 3$.

9. Reed-Solomon and Reliable Computation

In this question, we will see how error correction can help with faulty computations. Let us first establish a useful fact:

(a) Suppose we are using a Reed-Solomon code over $GF(p)$ guarding for k transmission errors. Let $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$ be two n-packet messages. Show that the Reed-Solomon codeword for message $a + b = (a_1 + b_1, \ldots, a_n + b_n)$ is the same as the sum of the Reed-Solomon codewords of a and b. In other words, the RS codeword of the element-wise sum is the element-wise sum of the RS codewords.

Let A, B and C be the message polynomials associated with messages a, b and $a + b$ respectively. By definition, A, B and C are of degree at most $n - 1$ and are such that $A(i) = a_i$, $B(i) = b_i$ and $C(i) = a_i + b_i$ for i in range $1, \ldots, n$. We have to show that $C(i) = A(i) + B(i)$ for all i in range $1, \ldots, n + 2k$.

First, we have $C(i) = a_i + b_i = A(i) + B(i)$ for $1 \leq i \leq n$. What about the remaining $2k$ points? $A + B - C$ is a polynomial of degree at most $n - 1$ with n distinct roots $1, \ldots, n$. Hence $A + B - C$ must be the zero polynomial ($A + B - C = 0$) which we can rewrite as $C = A + B$, thus finishing the proof.

Suppose you have invented a machine for doing additions extremely fast. Your invention takes a list of pairs of numbers as input and returns the list of the pairs sums. Although the machine is blazing fast, it is at the same time prone to mistakes. Luckily, you can bound the number of mistakes: over all of the n outputs returned, you know that at most $\max(1, \lfloor n/4 \rfloor)$ outputs have an error. For example, if we feed the machine $((2,3),(4,3),(0,7),(4,2))$ we might get back output $(5,7,4,6)$, where $4 \neq 0 + 7$ is a mistake.

You want to sell your invention, but none of your potential clients is interested in an error-prone device like this. They feel the speed benefit does not compensate for the unreliability of the results.

(b) Show that you can augment your machine with a Reed-Solomon encoding and decoding scheme such that no wrong outputs are ever returned. Your clients can use the machine the exact same way as before, but they no longer experience erroneous results.

Let n the number of input pairs $(a_1,b_1),\ldots,(a_n,b_n)$. The idea is to first encode $a=(a_1,\ldots,a_n)$ into a' and $b=(b_1,\ldots,b_n)$ into b' using an $(n,2k)$-Reed-Solomon code, feed the original machine with the encoded pairs $(a'_1,b'_1),\ldots,(a'_{n+2k},b'_{n+2k})$ to get the faulty summations, and then decode the obtained message and return the error-free solution. We are indeed guaranteed to get back the actual summations $(a_1+b_1,\ldots,a_n+b_n)$ by part (a) which shows that the sum of the Reed-Solomon codes corresponds to the Reed-Solomon code of the sums. Now, the only question is to find the number of errors k we need to account for in the Reed-Solomon scheme.

We have to be careful about the max condition in the error bound. There is at least 1 error independently of the input size n, which means we need $k \geq 1$. Furthermore, when $n \geq 4$ at most a fraction $\frac{1}{4}$ of the packets can get corrupted. Hence, by Problem 6 on the previous homework, we need:

$$k \geq \frac{n\frac{1}{4}}{1-2\frac{1}{4}} = \frac{n}{2}$$

when $n \geq 4$. Combining these two constraints and ignoring the $n \geq 4$ condition (which does not impact the correctness), we get the following bound:

$$k \geq \max(1, \frac{n}{2})$$

so we can use $2\max(1, \lfloor \frac{n}{2} \rfloor)$ additional inputs to make the scheme work.

10. Write your own problem

Write your own problem related to this week's material and solve it. You may still work in groups to brainstorm problems, but each student should submit a unique problem. What is the problem? How to formulate it? How to solve it? What is the solution?