EECS 70 Discrete Mathematics and Probability Theory
Fall 2014 Anant Sahai

Homework 6

This homework is due October 13, 2014, at 12:00 noon.

1. Try It Out (optional)

Please do the online problems on FLT and RSA. Give us your brief comments on how you found them and how you think that they helped you understand the material better for this homework.

2. RSA Lab

In this week's Virtual Lab, we will implement a toy RSA cryptosystem. We will start by generating the public and private keys for RSA using functions we implemented in the past two weeks (`mod_exp`, `egcd`, etc.). Then, we will encrypt and decrypt messages with the RSA function (for example, you will decrypt a secret and tell us in your written homework what the original message is). Finally, we will transition into Polynomials and implement Lagrange Interpolation for a degree 2 polynomial.

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

(a) *Warmup*: Implement the function `pairwise_coprime`, which returns True if every pair of numbers in the input list is coprime and False otherwise.

(b) In this part, we will give you sample implementations of the functions `is_probable_prime` (which uses the Miller-Rabin algorithm) and `gen_prime`, which generates a large pseudorandom prime.

Implement the function `gen_key`, which generates a pair of public (N, e) and private keys (d) for RSA. Refer to Lecture Note 6 for more details on how RSA key generation works.

For simplicity, you can generate e as a random integer between 1 and $(p-1)(q-1)$ as long as it's coprime with $(p-1)(q-1)$ and return the keypair as a triple (N, e, d).

In practice, e is usually chosen to be the minimum integer coprime with $(p-1)(q-1)$*, however.*

(c) Implement the functions `encrypt_integer` and `decrypt_integer`, which encrypts and decrypts a positive integer x, respectively, using the RSA function. Make sure to test your implementation by checking that

```
decrypt_integer(encrypt_integer(x, N, e), N, d) == x
```

for some positive integer x between 2 and $N-1$.

(d) We now want the ability to encrypt or decrypt actual text messages, instead of just positive integers. In this part, we will give you the code to convert text messages into lists of integers based on ASCII values (between 0 and 127). These numbers are then combined into blocks, each of size n, using base 256, which you can then apply the RSA function on.

Implement the `encrypt` and `decrypt` functions for text messages. Then answer the two subparts below. For both subparts, we will give you the values of N, e, and d that we want you to work with.

- What's the secret of your class log-in (*cs70-XYZ*) using a block size of 4? If your login only has two letters, treat the last letter (Z) as a whitespace to keep the message length at 8 characters. Everyone's answer to this question should be quite different from one another!

- What's the original message of the following secret? You will need to figure out the block size yourself.
  Hint: the length of the original message is 126 (counting all characters, including whitespace). To figure out the block size, check the numbers from `range(1, len(message))` that divides the message's length.
  The values of N, e, and d, as well as the secret, can be found at `http://pastebin.com/BpesLp6d`. They are also embedded inside the code skeleton.
  Please report the final answers to both parts by copying and pasting from the notebook. Make sure you keep the original message for the second subpart intact. Tell us what block size you used as well (there is only one correct answer).

(e) Recall from lecture that given three data points (x_i, y_i) where all the x_i are distinct, we can find a unique degree (at most) 2 polynomial $p(x)$ such that $p(x_i) = y_i$.

The three Δ_i functions can be found as follows.

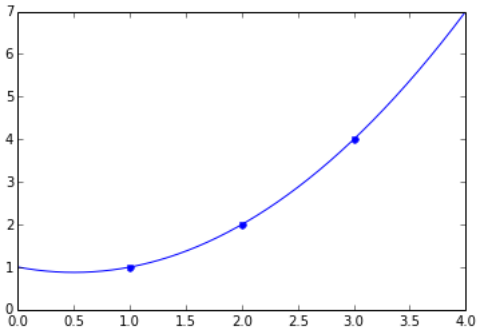
$$\Delta_1(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}, \tag{1}$$

$$\Delta_2(x) = \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}, \tag{2}$$

$$\Delta_3(x) = \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)} \tag{3}$$

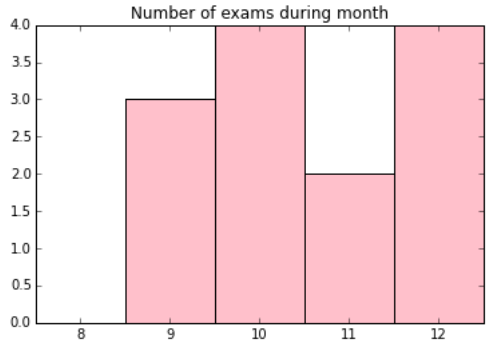
Implement the function `interpolate_2D` to calculate the polynomial $p(x)$ using Lagrange Interpolation.

Test your implementation with the example in Note 7. If your implementation is correct, when you run the plotting code cell in the skeleton, you should see that the polynomial passes through the three given points $(1,1), (2,2)$ and $(3,4)$.



(f) Last week, we learned how to plot a simple curve using Matplotlib. This week, we will learn to plot a bar chart. By the end of this question, you will plot a bar chart grouping the EECS 70 lecture notes by their last modified months!

Let's first look at an example showing the number of exams a student typically has per month during the Fall semester (this is just a toy dataset!).



The code to make the above plot turns out to be very simple.

Listing 1: Simple Bar Chart Code

```
width = 1
y = [0, 3, 4, 2, 4]
x = np.arange(8, 13)
plt.bar(x, y, width=width, color="pink")
plt.xticks(x + width*0.5, x)  # center the xticks
plt.title('Number of exams during month')
```

We have also introduced four new commands that you will need for this week's lab:

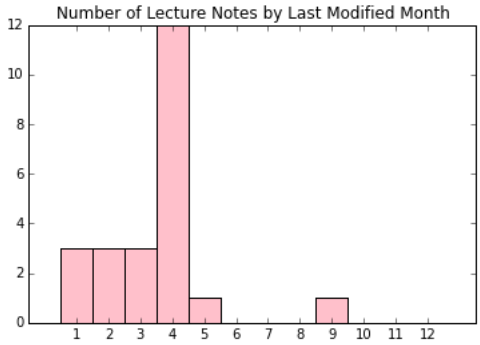
- `np.arange` is very similar to Python's `range` or `xrange`. The main difference is that `np.arange` returns a vectorized array. For example, scalar addition doesn't work on a normal Python's list, but it will add the same scalar to every element in the vectorized array. Those of you who are familar with Matlab will find this very familiar.
- `plt.bar` plots a bar chart, similar to `plt.plot` which plots a line curve from last week.
- `plt.xticks` sets the x-limits of the current tick locations and labels.
- `plt.title` gives the plot a meaningful title.

Believe it or not, you now know more than enough to complete this question! We will give you the code to recursively fetch the lecture note pdfs on the course website, as well as the Python commands to extract the last modified months. **You can tackle this problem in any creative way you want**, but here's our recommended approach.

- Group the lecture notes by month. (*Hint: use a counter dictionary, where the keys are 1 to 12 (the month), and the values are the number of notes that were last modified during that month*).
- Convert the counter dictionary into two lists and make sure to preserve the order. For example, there are three lecture notes that were last modified in January, so 1 should be the first value of the list of months, and 3 should be the first value of the list of counts.
- Plot the list values as a bar chart.

Important*: the dataset we're working with is very small, and it is possible to create the counter dictionary by inspection. Please do NOT do this. Write the code to convert the dataset into a format usable for* `plt.bar`*.*

To help you understand what we expect, the final plot should at least look very similar to this.



Be as creative as you want, and feel free to add axes' labels, the count above each bar, etc. to your heart's content.

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 `hw6.zip`.

3. The Enemy

The land of the Golden Bears has been conquered by an enemy who shall not be named, and even the bears can't bear it. A brave bear decides to gather her fellow bears together in secret to plan an attack against The Enemy. To announce the meeting info, she leaves messages written in secret codes that only bears can decipher. You are a young, strong bear, and you definitely want to join the revolution!

Here are the basic decrypting rules you learned from your *Freshbear Bootcamp*.

- Encryption is done on uppercase letters A to Z. If there are any other characters in the code, remove them, then add them back at their original positions after decoding the rest.
- To get the original text back, you map the letters A to Z to numbers 0 to 25, apply a corresponding decoding function D on this array of numbers, and map the output numbers back to alphabets.

You recall the three encoding functions that you were taught. Let a be the array of numbers, and a_i be the i^{th} element in a, $1 \leq i \leq n$, where n is the number of elements. The functions are,

- *Caesar cipher* with offset c: $E_{Caesar}(a_i, c) = (a_i + c) \mod 26$,
- *Accumulating cipher* with offset c: $E_{Acc}(a_i, c) = (\sum_{j=1}^{i} a_j + c) \mod 26$,
- *RSA*: $E_{RSA}(a_i, p, q, e) = a_i^e \mod (pq)$.

Unfortunately, you can only remember two decoding functions out of the three,

- *Caesar cipher* with offset c: $D_{Caesar}(a_i, c) = (a_i - c) \mod 26$,
- *RSA*: $D_{RSA}(a_i, p, q, e) = a_i^d \mod (pq)$, where d is the multiplicative inverse of e modulo $(p-1)(q-1)$,

but that is fine, you can figure out the decoding function for the *Accumulating cipher* by yourself anyway. You then revisit how to actually use these decoding functions. To decrypt "VZQL TO" that is encrypted with Caesar cipher with offset -1:

1. Strip out non-uppercase characters: "VZQL TO" $\rightarrow$ "VZQLTO"
2. Convert the string of characters to an array of numbers: "VZQLTO" $\rightarrow [21, 25, 16, 11, 19, 14]$

3. Apply the decoding function D_{Caesar} to each element:

$$\begin{aligned}
&D_{Caesar}([21,25,16,11,19,14],-1)\\
&= [D_{Caesar}(21,-1), D_{Caesar}(25,-1), D_{Caesar}(16,-1), D_{Caesar}(11,-1), D_{Caesar}(19,-1), D_{Caesar}(14,-1)]\\
&= [(21+1) \bmod 26, (25+1) \bmod 26, (16+1) \bmod 26, (11+1) \bmod 26, (19+1) \bmod 26, (14+1) \bmod 26]\\
&= [22,0,17,12,20,15]
\end{aligned}$$

4. Convert the array of numbers back to a string of characters: $[22,0,17,12,20,15] \rightarrow$ "WARMUP"
5. Insert the stripped out non-uppercase characters back: "WARMUP" $\rightarrow$ "WARM UP"

Now you are ready to go! Find out the name of the enemy, the gathering place, and a secret message from the encoded messages below. Show your work.

(a) The Enemy's name: "OVTLDVYR", encrypted with a Caesar cipher ($c = 7$).

(b) The gathering place: "OVZ VJIVDDN YMGTZD", encrypted with an Accumulating cipher ($c = -5$).

(c) The secret message: "QKHH JONK!", encrypted with RSA ($p = 2,\ q = 13$, and $e = 5$).

4. Poker mathematics

A *pseudo-random number generator* is a way of generating a large quantity of random-looking numbers, if all we have is a little bit of randomness (known as the *seed*). One simple scheme is the *linear congruential generator*, where we pick some modulus m, some constants a, b, and a seed x_0, and then generate the sequence of outputs $x_1, x_2, x_3, x_4 \ldots$ according to the following equation:

$$x_{t+1} = (ax_t + b) \bmod m$$

(Notice that $0 \leq x_t < m$ holds for every t.)

You've discovered that a popular web site uses a linear congruential generator to generate poker hands for its players. For instance, it uses x_0 to pseudo-randomly pick the first card to go into your hand, x_1 to pseudo-randomly pick the second card to go into your hand, and so on. For extra security, the poker site has kept the parameters a and b secret, but you do know that the modulus is $m = 2^{31} - 1$ (which is prime).

Suppose that you can observe the values x_0, x_1, x_2, x_3, and x_4 from the information available to you, and that the values $x_5, \ldots, x_9$ will be used to pseudo-randomly pick the cards for the next person's hand. Describe how to efficiently predict the values $x_5, \ldots, x_9$, given the values known to you.

5. RSA with three primes

Show how you can modify the RSA encryption method to work with three primes instead of two primes (i.e. $N = pqr$ where p, q, r are all prime), and prove the scheme you come up with works in the sense that $D(E(x)) \equiv x \bmod N$.

6. $d+2$ points vs. a polynomial of degree d

(a) Given 3 points $(0,1)$, $(1,1)$, and $(2,3)$, use Lagrange interpolation to construct the degree-2 polynomial which goes through these points.

(b) Given 4 points $(0,1)$, $(1,1)$, $(2,3)$, and $(-1,3)$, does there exist a degree-2 polynomial which goes through these points? If yes, find the polynomial; if no, explain why none exists.

(c) Given 4 points $(0,1)$, $(1,1)$, $(2,3)$, and $(-1,0)$, does there exist a degree-2 polynomial which goes through these points? If yes, find the polynomial; if no, explain why none exists.

(d) Design a machine (i.e. give the pseudocode for an algorithm) with the following function: given four points $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)$ with all the x_i distinct, the machine outputs YES if there exists a polynomial $p(x)$ of degree at most 2 such that $p(x_i) = y_i$ for all i; otherwise, it outputs NO.

7. Because the Moth just doesn't cut it

Gandalf the Grey (a good wizard) wanders about on his merry adventures but frequently runs into some troubles with goblins and orcs along the way. Always being the well-prepared wizard that he is, Gandalf has enlisted the service of the Great Eagles to fly him out of sticky situations at a moment's notice. To do this, he broadcasts a short message detailing his dilemma and a nearby eagle will come to his aid.

While this is all well and good, Saruman the White (an evil wizard) wants in on this eagle concierge service. The eagles can no longer trust just any distress call they receive! Gandalf needs you (a cryptography master) to help him devise a simple scheme that will allow the eagles to verify his identity whenever he broadcasts a message out. Not only that, but the eagles need to know when the message they receive from Gandalf has been tampered with.

Once you have devised this scheme, Gandalf will tell it to the eagle lord Gwaihir, who will relay it out to the rest of the world (they are loudmouths so they can't keep a secret).

To summarize:

(a) Gandalf broadcasts a message m to all of Middle-Earth.

(b) He is able to attach to the message an extra piece of information s that verifies his identity (i.e. cannot be forged) to whomever receives it.

(c) If the message has been modified in transit, recipients of the modified message should be able to detect that it is not original.

(d) Everyone in Middle-Earth knows the scheme (i.e. the algorithm itself is not a secret)

Your job in this problem is to devise an algorithm (like RSA) that meets the above criteria. In your answer, you should formally prove that Gandalf's messages can be successfully verified. You do not need to formally prove (though it should still be the case) that it is difficult to forge/tamper with messages, but you should provide some informal justification.

8. Breaking RSA

(a) Eve is not convinced she needs to factor $N = pq$ in order to break RSA. She argues: "All I need to know is $(p-1)(q-1)$... then I can find d as the inverse of e mod $(p-1)(q-1)$. This should be easier than factoring N". Prove Eve wrong, by showing that finding $(p-1)(q-1)$ is at least as hard as factoring N (that is, show that if she knows $(p-1)(q-1)$, she can easily factor N). Assume Eve has a friend Wolfram, who can easily return the roots of polynomials over $\mathbb{R}$ (this is, in fact, easy).

(b) When working with RSA, it is not uncommon to use $e = 3$ in the public key. Suppose that Alice has sent Bob, Carol, and Dorothy the same message indicating the time she is having her birthday party. Eve, who is not invited, wants to decrypt the message and show up to the party. Bob, Carol, and Dorothy have public keys $(N_1, e_1), (N_2, e_2), (N_3, e_3)$ respectively, where $e_1 = e_2 = e_3 = 3$. Furthermore assume that N_1, N_2, N_3 are all different. Alice has chosen a number $0 \leq x < \min\{N_1, N_2, N_3\}$ which indicates the time her party starts and has encoded it via the three public keys and sent it to her three friends. Eve has been able to obtain the three encoded messages. Prove that Eve can figure out x. First solve the problem when two of N_1, N_2, N_3 have a common factor. Then solve it when no two of them have a common factor. Again, assume Eve is friends with Wolfram as above.

9. Coin tosses over text messages

Alice and Bob want to flip a fair coin. However, they are on different continents, so they try to flip a coin by exchanging text messages. Neither one trusts the other, so they want to make sure that neither Alice nor Bob can affect the result of the coin toss (being able to affect the result would be obviously problematic if they are betting on the result of the random bit, for instance). Obviously, one person cannot simply toss the coin and "report" the results.

Bob proposes the following: "Each of us flips our own coin, with some result in $\{0,1\}$. Then we will send our result to the other, and agree that the final coin should be the XOR of our individual coins"

(a) What is wrong with this method?

(b) Propose a secure coin-flipping protocol using RSA. You do not need to give a formal proof of security, but explicitly list all properties you require from RSA, and why you need them (such that, given these properties, you could carry through a proof of security). *Be careful! Remember your protocol should be secure against cheating at any stage.*

(c) Find at least one flaw in the following alternate protocol:

1. Alice: Publish a public-key (N_A, e_A).
2. Bob: Publish a public-key (N_B, e_B).
3. Alice: Generate a 1000-bit string r_A by flipping 1000 coins. Send the encryption $y_A = E_{e_A}(r_A)$ to Bob.
4. Bob: Generate a 1000-bit string r_B by flipping 1000 coins. Send the encryption $y_B = E_{e_B}(r_B)$ to Alice.
5. Alice: Reveal r_A to Bob.
6. Bob: Reveal r_B to Alice.
7. Alice: Confirm $y_B = E_{e_B}(r_B)$.
8. Bob: Confirm $y_A = E_{e_A}(r_A)$.

Then they agree to use the XOR of all 2000 bits in r_A, r_B as their fair coin.

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?