Cryptography Exercise 2

For this exercise I was asked to write a Java program that accepts a user defined password in order to encrypt and decrypt text using a Vigenère cipher. I also had to write a cryptanalysis program to analyse cipher text encrypted with a Vigenère cipher in order to assist with recovering the original plaintext and the encryption key.

# Vigenère Cipher

First off I had to create functions for encrypting and decrypting using Vigenère cipher. I recreated the cipher in the method “encrypt” by looping through the text and when encountering a letter shifting it by the current key index value, then performing a modulo operation to account for wrapping back to the start of the alphabet, I then incremented onto the next index of the key and repeated this process, in my “decrypt” method I perform a similar operation, only reversing the shift. I then used them to encrypt the text found at <http://homepages.cs.ncl.ac.uk/feng.hao/teaching/pg1661.txt> with the key “ncluni”. I then ran a frequency analysis on the cipher text produced.

The table on the left shows the output of my frequency analysis, looking at the number of occurrences of each letter and their respective occurrence percentage it is obvious that the usual pattern found in by analysing normal text or text encrypted with a monoaplabetic cipher has completely disappeared. The values show that the encryption has diffused the frequency of letter across the alphabet with the values taking a much smaller range than what is seen in English text the highest being g at 6.045% and the lowest being d at 1.443%.

I have produced a graph below that visualises the diffusion you can see that after encryption the values letter frequencies are completely changes and less extreme.

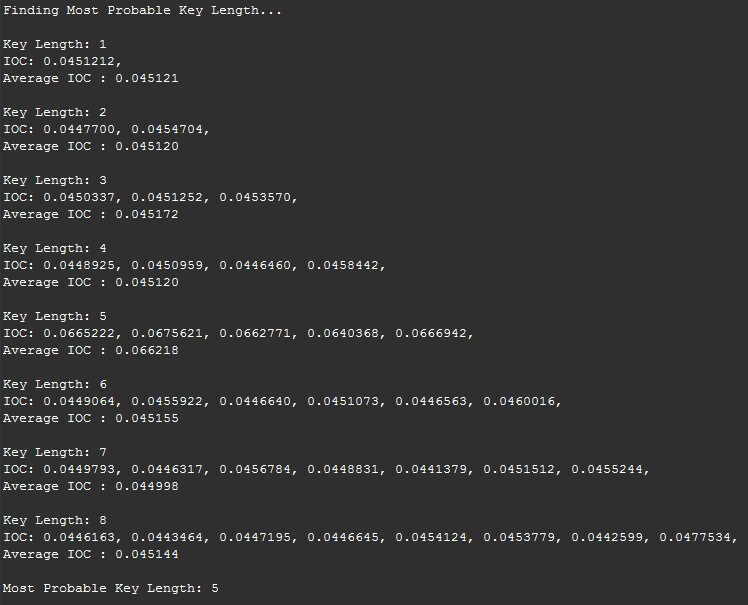
# Cryptanalysis of Vigenère Cipher

I was then given a piece of cipher text and asked to recover the plaintext and key given the knowledge that it was encrypted with Vigenère Cipher. Since the text was encrypted with Vigenère Cipher a simple frequency analysis would not have been enough to break it, the frequencies had been diffused by the cipher and since Vigenère Cipher is a polyalphabetic cipher the plaintext could not be recovered by performing a shift cipher on the text.

So instead I started by attempting to find the key length used to encipher the text. Since Vigenère Cipher is a polyalphabetic cipher which is in turn made up of several monoalphabetic ciphers finding the key length will find out how many monoalphabetic ciphers were used in the cipher.

To find the key length I created a method “findMostProbableKeyLength” that would calculate the index of coincidence on the cipher text for a range of key lengths. The method takes in a minimum key length and a maximum key length “i” and for each key length puts every ith letter into a group, this index of coincidence for this group is then calculated. The function then returns the key length whose group had the index of coincidence most similar to that of normal English, this key length would have the highest probability of being the key length used to encipher the text.

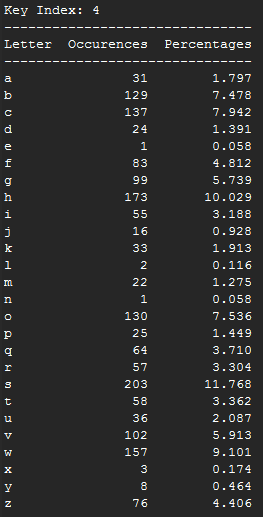
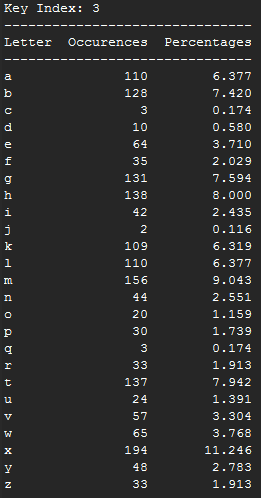
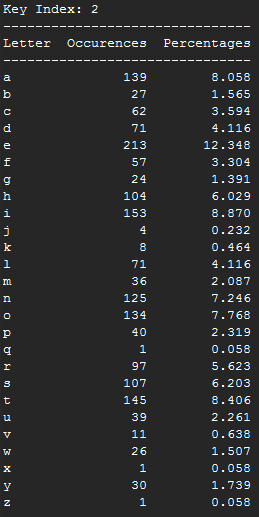
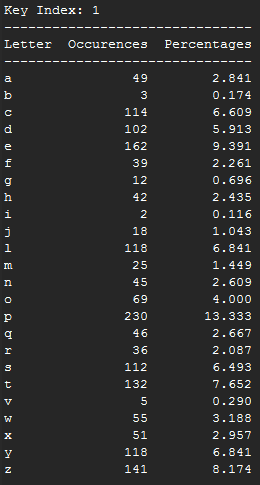
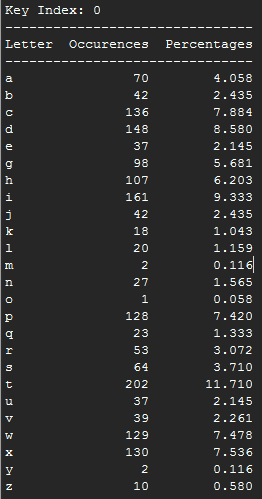
I ran this function on the cipher text using 1 as my minimum key length and 8 as my maximum key length, I have included one so I can view the index of coincidence of the cipher text itself:

 The screenshot on the left show the output of running this function. It has listed all key lengths and their respective index of coincidence. Keys of length 2, 3 and 4 all had an index of coincidence very similar to the cipher text itself this means that the frequency distribution of every 2nd, 3rd and 4th letter was similar to the cipher text and so a key of these lengths was not used to encrypt the text. Next we have a key length of 5 with an index of coincidence of 0.665.This looks much more promising as it is very close to the index of coincidence of normal English text (0.065). I also noticed that all key lengths that were multiples of 5 had a similar index of coincidence, which strongly suggests that 5 is in fact the key length. My function agreed telling me that a key length of 5 was most probable due it having the most ‘English like’ index of coincidence.

Now that I had the key length it was time to create a function that would assist in working out the key.

The function I created takes in some cipher text and a key length it then groups all the letters by their index in relation to the key length. For a key length of 5 it would go over the first 5 letters and put the first letter in group 1, the second in group two and so on until it put the fifth letter in group 5 then it would loop back around and do the same to the next 5 letters in the text. Splitting the text up like this allows me to start analysing the individual monoalphabetic ciphers of each letter in the key.

Now that I have each letter of the text split into groups corresponding to the letter of the key that was used to encrypt it I can start working out what the key is itself.

To do this I performed a frequency analysis on each group corresponding to each index of the key.

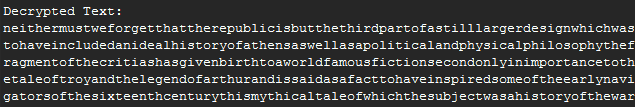
I noticed from the tables above that the frequency distribution patterns found in normal English text have returned, and that all have them have undergone a monoalphabetic cipher. Using this information I can figure out how much the text has been shifted, I modified my code to assist with this. The function now assumes that the character with the most occurrences in the cipher text was originally the letter ‘e’, it then calculates how many places ‘e’ must have been shifted by to get to the letter which now has the most occurrences.

Using this value “x” I can figure out the letter that was used to encrypt this section of the text. I do this by calculating what letter is x shifts away from ‘a’. For index 2 of the key ‘e’ is the most common letter, ‘e’ is 0 shifts away from ‘e’ and so the letter used to encrypt it was ‘a’ . Though in this case any text encrypted with this letter from the key will be exactly the same when encrypted as the text will be shifted 0 places.

I ran the program again, this time returning the key used to encipher the text.

It calculated that the key used to encipher the text was “plato”. I could see from the frequency tables that this was the correct result, provided that ‘e’ was the most frequent letter.

To test if my program had worked I tried to decrypt the cipher text using the key “plato”. The following is a screenshot of some of the deciphered text.



As shown above the text has correctly been deciphered back into English, there are no spaces in the text because they were not present in the cipher text, my program preserves all non-letter characters in a piece of text. I added a function to my program that will automate the process of deciphering text based on the same assumption of ‘e’ being the most common letter in a section of text. I tested the program with text encrypted with multiple Vigenère ciphers, it was able to decipher the text only when the second encryption was of length 1 or equal to the first. My automatic version also has difficulty when looking at key lengths that are multiples of the actual key as the will all have similar index of coincidences to normal English. It may for example think “platoplato” is the key instead of just “plato”. Though this should have no effect in the decryption, the smaller the groups of letters I perform the frequency analysis on the higher the chance of an error as ‘e’ will be less likely to be the most common letter.

# Conclusion

This exercise looked into polyalphabetic ciphers, specifically the Vigenère cipher and how to break them through cryptanalysis. My results have shown that these ciphers not only introduce confusion into the text but also introduce an element of diffusion, making them less vulnerable to frequency analysis. Though they are safer than a monoalphabetic cipher they can still easily be broken with a through the use of index of coincidence and a computer program.

# 