Homework 10
Code Breaking

I. Introduction

Overview and Background

Well, you made it to the Grand Challenge. In this assignment, we will write code that performs a variety of tasks related to cryptography, i.e. making codes and breaking them. The inspiration for this assignment comes from a book I really like, *The Code Book*, by Simon Singh. I highly recommend it as a summer read, as he does an amazing job of discussing the science and history of cryptography from ancient Egypt to modern quantum cryptography. There are some amazing historical anecdotes and intrigue, including the Navajo code talkers of WW2, cracking the German enigma code, and the Rosetta stone.

This assignment is quite long, and that is because I try to layout the requirements as clearly as possible, and because I need to provide sufficient background on the cryptographic techniques we will be using. I also include some scanned text from *The Code Book* in the Appendix. Before starting any coding, read this entire assignment top to bottom twice. I promise it will save you time.

Files for submission

Along with this assignment, there are two .mat files you will need, "cipher_text_test.mat" and "cipher_text_freq_analysis.mat". You can load the data contained in the .mat files by using the line:

```
load('cipher_text_test');
```

Notice that there is a new variable in the workspace, which should look like jibberish, i.e. it is text that has been enciphered, and is thus called cipher text.

You will submit either 7 m-files for this assignment, and a .doc file for the bonus if you choose to do it. These will be:

1. encipher_Caesar.m (a function, Part 1)
2. decipher_Caesar.m (a function, Part 1)
3. test_caesar_decipher.m (a script, Part 1)
4. encipher_Vigenere.m (a function, Part 2)
5. decipher_Vigenere.m (a function, Part 2)
6. encipher_Homophonoic.m (a function, Part 3)
7. decipher_Homophonic.m (a function, Part 3)
8. freq_analysis.doc (Bonus)

Submit your files with the names listed above but adding your initials to the end, e.g. freq_analysis_mal.m. For each file, make sure there is a proper header, sufficient comments, and readable code.

Terminology

We need to have some common vocabulary to complete the assignment, and some knowledge for how Matlab deals with text. When enciphered some text, we first start with the "plain text", which is simply the actual message we want to convey to our fellow spy. To make our life easier, we will require that our plain text is only capital letters, and we will use no punctuation or spaces. So an example plain text message would be: "THISASSIGNMENTDESCRIPTIONISWICKEDLONG". We will encipher this message into the cipher text, which is also comprised of capital letters only. Using the encryption method in part 1, the Caesar cipher, with a code word of "EXAMPLE", the cipher text is:

"DOQCECCQNVUPVDMPCABQYDQWVQCHQASPMTWVN"

Notice that the cipher text has the same number of letters as the plain text, but it is complete jibberish to anyone trying to read the message. Only with the code word, or by breaking the code, can someone determine the actual plain text message. There are a wide range of encryption methods, some much stronger than others, and they have evolved through history to become stronger and stronger.

ASCII format

In Matlab, we specify text using apostrophes, so if I wanted to create a variable for the plain text, I could use:

```
plain_text = 'THISASSIGNMENTDESCRIPTIONISWICKEDLONG'
```

Next, we will convert the text into ASCII format, which represents each letter in the alphabet as an integer. For capital letters, A-Z are represented by the integers from 65 to 90. To convert plain text into ASCII values, we just use:

```
pt = uint8(plain_text);
```

Try it out, you'll see that pt is not a row vector containing the same number of integers as the number of letters in the plain text, and all integers are between 65 and 90. To convert back from ASCII to letters, just use:

```
plain_text = char(pt);
```

Numbers are obviously easier to work with than letters in Matlab, so in general we will convert text into ASCII characters and manipulate them in the various encryption methods, and then convert back to letters at the end.

II. Part 1 – The Caesar Cipher

The Caesar cipher is named after Julius Caesar, who was a frequent user of ciphers for sending messages. The version we will implement is a "substitution cipher" in which every letter in the regular "plain" alphabet has a corresponding letter in the cipher alphabet. For example A could always be encrypted as G, B as U, C as E, etc. As long as the person who receives the message knows the cipher alphabet as well, then they can decipher the cipher text back into the plain text. See the appendix below for a few scanned pages on this topic, which gives additional background.

To create the cipher alphabet, the steps are as follows.

1. Choose a code word, for example "JULIUSCAESAR".
2. Remove any repeated letters in the code word, so our code word is now "JULISCAER" (I removed the repeated U, S, and A.
3. Use the code word as the beginning of the cipher alphabet.
4. The remainder of the cipher alphabet is then just the remaining letters of the alphabet in their regular order, starting where the keyword ends. Here it is for this example

   Plain: ABCDEFGHIJKLMNOPQRSTUVWXYZ
   Cipher: JULISCAERTVWXYZBDFGHKMNOPQ

Now we can encrypt plain text by substituting in the corresponding letter in the cipher alphabet, and any person with the code word can decipher back into plain text because they can determine the corresponding cipher alphabet.

For this part, you will create three m-files (2 functions, 1 script) related to the Caesar cipher.

encipher_Caesar.m

Output (1): [cipher_text] is a character string of enciphered text, e.g. cipher_text = 'DOQCECCQNVUPVDMPCABQYDQWVQCHQASPMTWVN'.

Inputs (2): [plain_text] is a character string of plain text, e.g. plain_text = 'THISASSIGNMENTDESCRIPTIONISWICKEDLONG'. [code_word] is a character string of plain text, e.g. code_word = 'EXAMPLE'.

This function should do the following:

1. Convert the plain text and code word into ASCII format using uint8. Call these variables pt and cw.
2. Eliminate any repeated letters in the code word. Hint, use the built in Matlab function "unique" and use the option 'stable'.
3. Initialize the plain alphabet in ASCII format (it is 65 – 90).
4. Initialize a variable for the cipher text in ASCII format, ct, which for now is a row vector of zeros of the same length as pt.
5. Check that the inputs are valid, i.e. that they are only capital letters with no spaces or punctuation.
6. Create the cipher alphabet using the code word.
7. Use a for loop to add values to the ct vector. Loop over each letter in pt and encipher each letter using the cipher alphabet. The corresponding enciphered letter will then be added to ct. At the end of the loop, all elements of ct will be integers between 65 and 90.
8. Create the cipher text from ct using the "char" function.

decipher_Caesar.m

Output (1): [plain_text] is a character string of deciphered text, e.g. plain_text = 'THISASSIGNMENTDESCRIPTIONISWICKEDLONG'.

Inputs (2): [cipher_text] is a character string of cipher text, e.g. cipher_text = 'DOQCECCQNVUPVDMPCABQYDQWVQCHQASPMTWVN'. [code_word] is a character string of plain text, e.g. code_word = 'EXAMPLE'.

This function should do the following:

1. Convert the cipher text and code word into ASCII format using uint8. Call these variables ct and cw.
2. Eliminate any repeated letters in the code word. Hint, use the built in Matlab function "unique" and use the option 'stable'.
3. Initialize the plain alphabet in ASCII format (it is 65 – 90).
4. Initialize a variable for the plain text in ASCII format, pt, which for now is a row vector of zeros of the same length as ct.
5. Check that the inputs are valid, i.e. that they are only capital letters with no spaces or punctuation.
6. Create the cipher alphabet using the code word.
7. Use a for loop to add values to the pt vector. Loop over each letter in ct and decipher each letter using the cipher alphabet. The corresponding deciphered letter will then be added to pt. At the end of the loop, all elements of pt will be integers between 65 and 90.
8. Create the plain text from pt using the "char" function.

You can text your functions by first enciphering and then deciphering text using the same code word. If done correctly, you will recover the original plain text.

test_caesar_decipher.m

This script should do the following:

1. Load the cipher_text_test.mat file using: load('cipher_text_test');. Once it is loaded, you will see that there is a new variable in the workspace called cipher_text, which is a string of characters of enciphered text that I created. I created it using the Caesar cipher, with a code word that is only two letters long.
2. Use two nested for loops to decipher the cipher text into plain text using your decipher_Caesar function. Use every possible code work (26^2=676 options), and store the output plain text in a cell of size 676x1. A cell is like a vector, but it stores text instead of numerical values. So within your loop, if c is the iteration number, you can store each possible plain text using: pt_test{c,1} = … Notice that we use "curly" brackets instead of regular parentheses.
3. Look through each of the 676 possible plain text options and determine which is the true plain text. In a comment, paste the true plain text and identify the code word that I used.
4. Using a log scale on the y-axis (use semilogy instead of plot), plot the number of possible code words as a function of the length of the code word, from 1 to 10 letters. Again use a comment to discuss the plot briefly in your script.

III. Part 2 – The Vigenere Cipher

The Caesar cipher is easy to implement, and relatively safe unless you are faced with a determined code breaker. Even with a sufficiently long code word that makes testing each possibility infeasible, it can be broken using frequency analysis. In any language, certain letters are more common than others. In English, E is the most common, followed by T, then A, etc. If the cipher text is sufficiently long, then a code breaker can analyze the frequency of each letter in the cipher alphabet and start to piece together the corresponding letter in the plain alphabet. They can also look for common words, like "the." Eventually they will break the code.

The weakness of the Caesar cipher is due to the fact that it is a "monoalphabetic" cipher, meaning there is a single cipher alphabet used for the entire message. A much stronger method uses a "polyalphabetic" cipher, which rotates between cipher alphabets for each letter in the plain text. A particularly elegant and effective version is the Vigenere cipher, named after a 16th century French diplomat, Blaise de Vigenere. The method is implemented as follows:

1. The first step is to create a Vigenere square, shown below, which has the plain alphabet at the top, followed by 26 cipher alphabets below, each shifted by 1 letter from the one above it. See the table below. The plain text could be enciphered with any of the 26 cipher alphabets.

Table 3 A Vigenère square.

Plain	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
1	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
2	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
3	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
4	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
5	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
6	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
7	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
8	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
9	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
10	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
11	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
12	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
13	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
14	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
15	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
16	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
17	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
18	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
19	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
20	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
21	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
22	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
23	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
24	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
25	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
26	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

2. Next, when enciphering plain text, a different row of the Vigenere square is used to encipher each letter of the plain text message. So the first letter could be encrypted using the 5th row, the second letter using the 14th, the third using the 21st, and so on.

3. To determine which row to use, a code word is chosen, for example JULISCAER (again we remove repeated letters first). When encrypting the first letter in the plain text, we use the cipher alphabet in the 9th row of the Vigenere square, because J is the first letter in the 9th row of the Vigenere square (note J is the 10th letter of the alphabet, but the first row starts with b not A). For the second letter, we use the 20th row (U is the 21st letter of the alphabet and starts the 20th row), then the 11th row (L is the 12th letter of the alphabet), etc. Since this code word is 9 letters long, once we get to the 10th letter in the plain text, we start back over with the 9th row is the square for J. So we have used 9 different cipher alphabets, which makes this method completely impregnable to frequency analysis.

For this part, you will create two m-files (both functions) related to the Vigenere cipher.

encipher_Vigenere.m

Output (1): [cipher_text] is a character string of enciphered text, e.g. cipher_text = 'CBTASUSMXWGPVLFEWTACABAQNMJFCNSWFLSEP'.

Inputs (2): [plain_text] is a character string of plain text, e.g. plain_text = 'THISASSIGNMENTDESCRIPTIONISWICKEDLONG'. [code_word] is a character string of plain text, e.g. code_word = 'JULISCAER'.

This function should do the following:

1. Convert the plain text and code word into ASCII format using uint8. Call these variables pt and cw.
2. Eliminate any repeated letters in the code word. Hint, use the built in Matlab function "unique" and use the option 'stable'.
3. Initialize the plain alphabet in ASCII format (it is 65 – 90).
4. Initialize a variable for the cipher text in ASCII format, ct, which for now is a row vector of zeros of the same length as pt.
5. Check that the inputs are valid, i.e. that they are only capital letters with no spaces or punctuation.
6. Create the Vigenere square in ASCII format, which is a 26x26 matrix. The first row is 66 – 90 and then 65, etc.
7. Use a for loop to add values to the ct vector. Loop over each letter in pt and encipher each letter using the appropriate cipher alphabet in the Vigenere square. The "mod" function will probably be useful for this. Keep in mind that you need to repeat the use of the code word several times based on the length of the plain text. The corresponding enciphered letter will then be added to ct. At the end of the loop, all elements of ct will be integers between 65 and 90.
8. Create the cipher text from ct using the "char" function.

decipher_Vigenere.m

Output (1): [plain_text] is a character string of deciphered text, e.g. plain_text = 'THISASSIGNMENTDESCRIPTIONISWICKEDLONG'.

Inputs (2): [cipher_text] is a character string of cipher text, e.g. cipher_text = 'CBTASUSMPHXMFVDIJLLTXLKORZBQTKCGDPFWA'. [code_word] is a character string of plain text, e.g. code_word = 'JULISCAER'.

This function should do the following:

1. Convert the cipher text and code word into ASCII format using uint8. Call these variables ct and cw.
2. Eliminate any repeated letters in the code word. Hint, use the built in Matlab function “unique” and use the option ‘stable’.
3. Initialize the plain alphabet in ASCII format (it is 65 – 90).
4. Initialize a variable for the plain text in ASCII format, pt, which for now is a row vector of zeros of the same length as ct.
5. Check that the inputs are valid, i.e. that they are only capital letters with no spaces or punctuation.
6. Create the Vigenere square in ASCII format, which is a 26x26 matrix. The first row is 66 – 90 and then 65, etc.
7. Use a for loop to add values to the pt vector. Loop over each letter in ct and decipher each letter using the appropriate cipher alphabet in the Vigenere square. The “mod” function will probably be useful for this. The deciphered letter will then be added to pt. At the end of the loop, all elements of pt will be integers between 65 and 90.
8. Create the plain text from pt using the “char” function.

You can text your functions by first enciphering and then deciphering text using the same code word. If done correctly, you will recover the original plain text.

IV. Part 3 – The Homophonic Cipher

Another approach to thwart frequency analysis is to use a homophonic cipher. In this approach, each letter of the alphabet has 1 or more unique numbers that correspond to it. The amount of numbers for a letter is proportional to the frequency that the letter occurs. Let us use 100 numbers total. Since E occurs approximately 12% of the time, there are 12 unique numbers between 1 and 100 that E can be enciphered into, for example [14 16 24 44 46 55 57 64 74 82 87 98] (these are completely arbitrary examples). Next, T occurs about 9% of the time, so there are 9 different numbers that correspond to T. The percentage for each letters from A to Z is summarized in the variable p, given here:

```
p = [8 2 3 4 12 2 2 6 6 1 1 4 2 6 7 2 1 6 6 9 3 1 2 1 2 1]';
```

Notice that it sums to 100. Also, be aware that for this cipher we are not using ASCII format for the cipher text. The cipher text is just a vector of numbers, with length equal to the number of letters in the plain text, and each number is an integer between 1 and 100. For this part, you will create two m-files (both functions) related to the homophonic cipher.

encipher_Homophonic.m

Outputs (2): [cipher_text] is a vector of numerical values. [Freq] is a 26x12 matrix that contains either zeros or the unique numbers corresponding to each letter. Each row corresponds to a letter.

Input (1): [plain_text] is a character string of plain text, e.g. plain_text = 'THISASSIGNMENTDESCRIPTIONISWICKEDLONG'.

This function should do the following:

1. Convert the plain text into ASCII format using uint8. Call this variables pt.
2. Initialize the plain alphabet in ASCII format (it is 65 – 90).
3. Initialize a variable for the cipher text, ct, which for now is a row vector of zeros of the same length as pt.
4. Check that the inputs are valid, i.e. that they are only capital letters with no spaces or punctuation.
5. Use a subfunction called homophonic_alp, which has no inputs and two outputs, Freq and p. In the subfunction:
    a. Define p as the vector given above.
    b. Initialize Freq as a 26x12 matrix of zeros.
    c. Use the randperm function to generate a vector r, 100x1, which contains a random ordering of the numbers between 1 and 100.
    d. Use a for loop, and in each row of the matrix, replace some of the zeros with values from r. In the first row, corresponding to A, replace the first 8 zeros, because p(1) = 8. In the second row, replace the first 2 zeros because p(2) = 2, etc.
    e. At the end of the for loop, Freq should contain all values of r. Each row should have unique numbers that are not repeated in any other row, and all values if Freq should be either the integers between 1 and 100, or zeros.
6. Back in the main function, use a for loop to add values to the ct vector. Loop over each letter in pt and encipher each letter using the appropriate values in the Freq matrix. For a given plain text letter, randomly select any of the corresponding numerical values in the associated row of Freq. So for E, there are 12 options, and you can randomly select any of them. The enciphered letter will then be added to ct. At the end of the loop, all elements of ct will be integers between 1 and 100.
7. Set cipher_text equal to ct.

decipher_Homophonic.m

Output (1): [plain_text] is a character string of deciphered text, e.g. plain_text = 'THISASSIGNMENTDESCRIPTIONISWICKEDLONG'.

Inputs (2): [cipher_text] is a character string of cipher text, e.g. cipher_text = 'DOQCECCQNVUPVDMPCABQYDQWVQCHQASPMTWVN'. [Freq] is the matrix you created in encipher_Homophonic.

This function should do the following:

1. Set the cipher_text variable equal to ct.
2. Initialize the plain alphabet in ASCII format (it is 65 – 90).
3. Initialize a variable for the plain text in ASCII format, pt, which for now is a row vector of zeros of the same length as ct.
4. Use a for loop to add values to the pt vector. Loop over each value in ct and decipher each number using the appropriate values in the Freq matrix. The deciphered letter will then be added to pt. At the end of the loop, all elements of pt will be integers between 65 and 90, i.e. ASCII capital letters.
5. Create the plain text from pt using the "char" function.

You can text your functions by first enciphering and then deciphering text using the same Freq matrix. If done correctly, you will recover the original plain text.

V. Bonus – Frequency Analysis

Using a Caesar cipher, I have enciphered the first chapter from a famous American novel. It is saved in the file cipher_text_freq_analysis.mat. I used the book title as the code word. After removing all spaces and punctuation, there are 9,622 characters in the cipher text. Your task is to use frequency analysis to break the code, allowing you to decipher the text and read the first chapter of the book. Use the frequency values given in Part IV Beyond that, the strategy is up to you.

In a word document, describe your approach, show any plots that may be helpful, identify the book title, and paste the deciphered plain text at the end. Your description can include any details on dead ends, breakthroughs, etc. You will be rewarded for a clear and interesting discussion, and docked for bad writing, insufficient detail, etc. Have fun with it.

A. Scanned Text on the Code Book