ECEN310 Communication Engineering (XMUT) Lab 1 (2%)

The purpose of this lab is to help you with revisions on using MATLAB (or similar software, e.g. OCTAVE). Part 1 is purely focused on the details of MATLAB; you are advised to go through the materials although you might be competent in MATLAB, and in Part 2 there are 3 exercises designed to help you practise using MATLAB by investigating (a) **the concept of entropy**, (b) **Shannon's channel capacity**, and (c) **Fourier Transform of a sine wave**.

You are not required to submit any written reports, BUT you need to execute your MATLAB coding and show the final results to the teacher in order to get the 2%.

Part 1: Revisions on Matlab

MATLAB is a very useful tool for analyzing and designing communication systems using the personal computer (PC). It has become widely used in engineering courses and is often used in industry for system design and simulation.

MATLAB is an acronym for Matrix Laboratory sold by MathWorks. It treats all constants and variables as matrices, row vectors, or column vectors. Consequently, the default operations in MATLAB are matrix operations. For example, a * b is the matrix multiplication of a and b.

This means that MATLAB program code can be very concise. Complicated operations can be expressed using very few lines of code with no "do" loops. The results can be computed very efficiently with the PC. However, this can make the code hard to understand, although it is compact and computationally efficient. The use of matrices and vectors also allows for powerful graphical capabilities.

It is advisable that you type and store your written MATLAB in a .m files extension; this is known as the M-files. You can use any text editor to type the codes, but save as .m file. What follows below will give you a quick revision on MATLAB.

1-1 Quick Start on Running MATLAB

1. Click on the MATLAB icon on your desktop.

2. The default MATLAB window will appear (unless you previously changed the default to another selection). Note the Start icon in the left-hand lower corner of the default window. This will allow you to select Toolboxes, Simulink, Demos, etc. Toolboxes and Simulink are not used by any of the M-files for this book because we are interested in learning how to create basic M-files that will be used to solve communication examples and problems.

3. Click on the New Script icon that is located just below File at the top of the default MATLAB window. The Editor window should appear. (Alternately, in the default window, click on the Desktop pull-down menu and select Editor.)

4. At the top of the Editor window, click on the folder icon and go to the folder in which your stored M-files are located. Select Example1_1.m. In the Editor window, lines of code for this Example1_1.m file should appear.

5. To run this MATLAB file, click on the green (run) arrow at the top of the Editor window. Alternately, you can run a M-file by typing the file name (without the file extention, .m) at the MATLAB prompt in the Command window.

6. Statements that you type into the Command window are executed immediately when you hit the Enter key. Statements that you type into the Editor window, along with any other statements in the Editor window, will be executed when you click the green arrow.

1-2 Programming in MATLAB

There are many sources that will help you to write the lines of code for MATLAB script files. The first is the built-in MATLAB help. Click on the fx icon on the command-prompt line of the Command window to browse the descriptions of MATLAB functions. More general help can be obtained by clicking on Help at the top of the default window to obtain a pulldown list of help sources. Help can also be obtained from the MathWorks Web site located at http://www.mathworks.com.

Second, there are many excellent publications on MATLAB programming. One of the best for beginners is the MATLAB Primer [Davis, 2011]. This is a pocket-sized book, around 230 pages in length, that concisely describes the basics of MATLAB programming. It includes an appendix listing the descriptions of the MATLAB top 500 commands and functions. Another excellent publication is Mastering MATLAB 7 [Hanselman and Littlefield, 2011]. This paperback book, around 850 pages in length, proceeds from the basics of MATLAB to more advanced topics. The basics are covered in the first few chapters and then one can skip to topics of interest in the latter chapters.

As indicated previously, the basic type of variable used in MATLAB is the matrix. This sets the stage, so to speak, for the notation used in MATLAB programs. The list below summarizes the basic notation used in MATLAB programs; try it out yourself!

1. A row vector is created by a comma-delimited or a space-delimited list. For example, enter M1 = [1,2,3,4,5,6] or M1 = [1 2 3 4 5 6]. This is a six-element row vector, or a 1 * 6 matrix (1 row * 6 columns).

2. A column vector is created by a semicolon-delimited list. For example, M2 = [1;2;3;4;5;6] is a six-element column vector, or a 6 * 1 matrix.

3. Enter M3 = [1 2 3; 4 5 6]. This creates a 2 * 3 matrix. To view the element in the 2nd row and 3rd column, enter M3(2,3). It has a value of 6.

4. A scalar is a 1 * 1 matrix. For example, enter M4 = 2.5.

5. Enter M5 = 0.5 + 2j. This specifies a complex number that is a 1 * 1 matrix.

6. The colon operator is used to index arrays and to create elements of vectors. The notation used is start-value:skip-increment:end-value. For example, enter t = 1:2:6. This creates the row vector t = [1 3 5]. If the skip increment is deleted, the default increment is 1. For example, u = 1:6 is the row vector u = [1 2 3 4 5 6]. The colon can also serve as a wild card. For example, in Item 3, M3 (1,3) denotes the element in the 1st row and 3rd column, but M3 (1,:) would denote the row vector corresponding to the 1st row of the matrix M3.

7. Once a variable (i.e., a matrix) is defined, it remains in memory until it is cleared. That is, its value can be obtained by typing in its symbol. For example, enter M3 at the MATLAB prompt.

8. Enter whos. Whos is the MATLAB command that lists all variables (i.e., matrices) that are stored in memory. It also shows the size of each matrix.

9. MATLAB is case sensitive. That is, M3 is a different variable from the variable m3, which has not been defined.

10. The transpose operator is ’. For example, enter M1’, which is the transpose of M1.

11. Insert a semicolon at the end of a MATLAB statement to suppress the display of the computed result for that statement. For example, enter y = 6*3;. The computed result is not displayed on the screen. However, you can display the computed result by entering y.

12. Multiple statements may be entered on the same line if the statements are separated by commas or semicolons. For example A = 1, B = 5.

13. The usual elementary functions (i.e., trigonometric and logarithmic) are built into MATLAB. For a listing, enter help elfun. For a list of specialized functions, such as Bessel functions, enter help specfun. User defined functions can also be created via M-files; enter help function for details.

14. The function plot (t, w) gives a plot of the waveform, vector w, as a function of time vector, t. Multiple plots in a single window can be obtained by using the subplot function. For example, multiple plots arranged in a four-row * one-column array can be obtained by using subplot (4, 1, x), where x is the plot number within the four possible plots. Alternately, the notation subplot(41x) could be used. In another example, multiple plots arranged in a three-row * two-column array can be obtained by using subplot (3, 2, x) where x is the plot number within the six possible plots. That is, subplot (3, 2, 3) would be the third of six plots, where the third plot appears in the first column of the second row.

15. The % sign is used to define a comment statement. All text on a line that follows a % sign is interpreted as a comment statement.

Part 2: Exercises using MATLAB

What follows are three (3) exercises to help you practise using MATLAB, as well as to help you understand the communication concepts learnt in our lectures.

A. The concept of Entropy

In Lecture 1 you have been introduced to the information measure of a wireless transmission, where the information sent from a digital source when the *j*th message is transmitted is given by

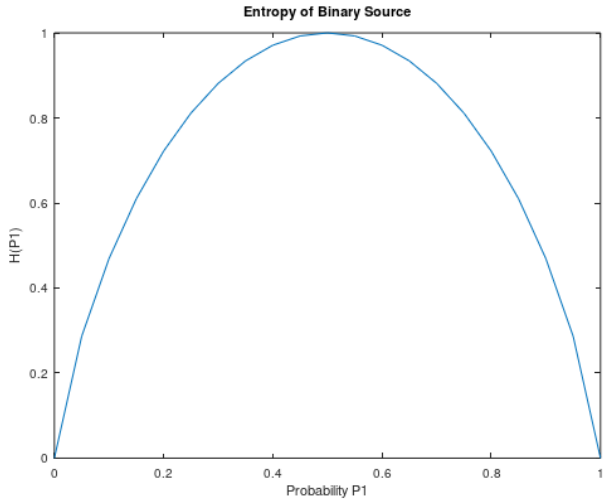
$$I_j = \log_2\left(\frac{1}{P_j}\right) \text{bits}$$

where P_j is the probability of transmitting the *j*th message. And the *average information* measure of a digital source is

$$H = \sum_{j=1}^{m} P_j I_j = \sum_{j=1}^{m} P_j \log_2\left(\frac{1}{P_j}\right) \text{bits}$$

where *m* is the number of possible different source messages and P_j is the probability of sending the *j*th message (*m* is finite because a digital source is assumed). This average information is called **entropy**.

Exercise: For a case of a binary problem, examine by using MATLAB, how the entropy changes as a function of P_1 where P_1 is the probability of obtaining a binary "1" where it ranges from 0 to 1. You should get a plot that looks like this:



B. Shannon's Channel Capacity

In Lecture 1, we have also come across the concept of channel capacity.

For digital systems, the optimum system might be defined as the system that minimizes the probability of bit error at the system output subject to constraints on transmitted energy and channel bandwidth.

This raises the following question: Is it possible to invent a system with no bit error at the output even when we have noise introduced into the channel?

This question was answered by Claude Shannon in 1948–1949.The answer is yes, under certain assumptions.

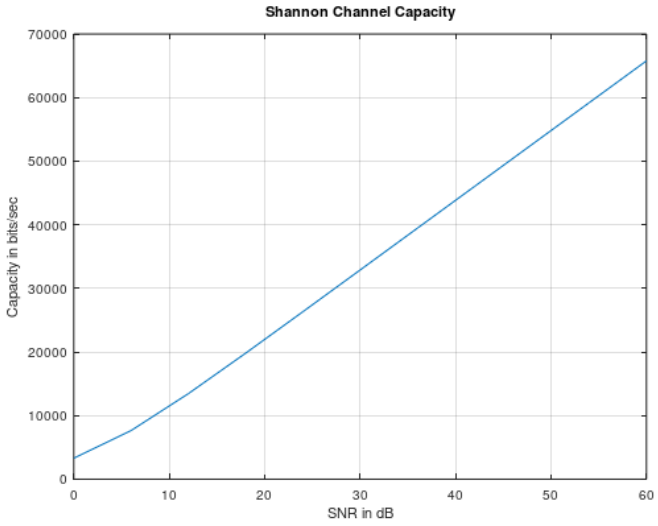
Shannon showed that (for the case of signal plus white Gaussian noise), a **channel capacity** C (bits/s) could be calculated such that if the rate of information R (bits/s) was less than C, the probability of error would approach zero. The equation for C is

$$C = B \log_2\left(1 + \frac{S}{N}\right)$$

where B is the channel bandwidth in hertz (Hz) and S/N is the signal-to-noise power ratio (watts/watts, not dB) at the input to the digital receiver.

Shannon does not tell us how to build this system, but he proves that it is theoretically possible to have such a system. Thus, Shannon gives us a theoretical performance bound that we can strive to achieve with practical communication systems. Systems that approach this bound usually incorporate error-correction coding.

Exercise: For a case of telephone line with a bandwidth of 3,300 Hz, write a MATLAB code to plot the channel capacity of the telephone line as a function of the S/N over a range of 0 – 60 dBs. You should get a plot that looks like this:



C. Spectrum of a pure Sine Wave

In the course of XMUT220 Signals and Systems, we have learnt that a time domain signal can be converted into frequency domain by using Fourier Transform. The frequency spectrum (i.e. frequency domain representation) of the same time domain signal will give us the distribution of the harmonic contents of the signal and their relative power. This is also known as the *harmonic analysis*. In this part, we will look at how to generate such a spectrum.

You will plot a time domain of a pure single-tone 100 Hz sine wave (with an amplitude of 1) for a 2 second period. Remember to generate enough points within this 2 second (i.e. set the time resolution to be sufficiently small, eg 0.002, so that we get 1000 sample points of the sine wave). Then execute the Fourier Transform of the signal by using fft( ) command of MATLAB. Because we have N (i.e. 1000) sample points, the resulting fft( ) is an accumulation of N iterations, so we need to normalise its values by dividing with N (for curiosity, you can check the differences with and without this dividing).

Note: fft stands for Fast Fourier Transform, and it is a computing method to execute Fourier Transform by using the discrete Fourier Transform (DFT). Plot the resulting spectrum obtained with fft. You need to generate a frequency index (say from 0 to 500 Hz, with sufficient frequency resolution between points, say 0.5) in order to plot the fft.

Note: the Fourier Transform of a time domain signal is in general a complex-valued function of frequency, so you have to consider its magnitude (i.e. the absolute-valued amplitude) when doing the plot.

Notice that the spectrum shows two peaks at both ends of the frequency scale. This is because the “negative frequency” has been folded and appended to the positive frequency part. To correct this anomaly, we can use the fftshift( ) command of MATLAB after executing the fft( ). This will bring the DC (i.e. 0 Hz) to the middle of the plot so that both the negative and positive frequencies are displayed properly.

What is the magnitude of the resulting Fourier Transform of the sine wave? Does it agree with the theory? (answer: yes it should, which is $A^2/2$, where A is the amplitude of the time domain sine wave)

You should get a final plot that looks like this: