School of Electronic and Electrical Engineering

UNIVERSITY OF LEEDS

ELEC1401 Communications Networks and Signals

MATLAB Sheet III

This practice sheet has been designed to help you better understand A-to-D conversion and Monte Carlo simulations. You will practice writing MATLAB functions and employ modular programming techniques. It is important that, in each case, you first think of a proper algorithm for your code, write the algorithm down on paper, and then implement it.

The tasks in this worksheet are for learning purposes, the marks associated with the tasks are for guidance only and they will not contribute to your final module mark. However, completing these tasks is essential to ensure that you understand the material as Matlab will be assessed in the January assessment (Assessment 2). You are expected to finish the tasks by the end of Unit 2 of this module. The sheet includes empty boxes below each task to write your answers, this should be useful when you review the Matlab sheet before the final assessment. You might find some of the tasks already explained in the screencasts, they are repeated here for completeness and for you to try it yourself if you have not done so.

MATLAB Help: In MATLAB you can get help in various ways. The simplest way, if you know the exact name of a particular function, is to type "help name_of_function" in the MATLAB command line. Alternatively, you can browse MATLAB help to find more about what you are looking for. In any of the problems below, if anything is unclear, or you don't know how to use a function, you should first try MATLAB Help. MATLAB commands and built-in functions appear in red in this sheet.

****

Section 1: A-D Conversion

Recommended time for completing the tasks in this section: 90 minutes

We have seen that analogue-to-digital conversion would consist of two parts: sampling (dealt with in the previous Matlab sheet), and then quantization. Quantization is the act of approximating the sample value, and then encoding it to a series of bits. In this section, we try to write a series of MATLAB functions that take care of quantization. The particular quantizer that we develop in this section is a uniform quantizer that takes values between a start point A and an end point B > A. If the input point to the quantizer is greater than B, your code would replace the input with B. Similarly, if the input to the quantizer is smaller than A, it would be replaced with A.

a. Suppose, your quantizer has n bits. How many bands do we need to partition values between A and B?

b. Write a function (call it ADBoundary) that finds the boundaries of each band. The inputs to this function are A, B, and n; the output is a vector with all boundary values.

Task 1: Write your code in the below box:

[1 mark]

These marks are for guidance only and they will **not** contribute to your final module mark

c. Write a function that accepts, as input, the vector generated by ADBoundary and finds the representative value for each band in your quantizer, that is, the middle point in each band. Call this new function ADmid.

d. Now write a function (called ADquant) that takes a real number r, as the sample value, as well as inputs A, B, and n, and gives back the quantized value, i.e, the middle point in the band that r would lie in).

Task 2: Sketch your algorithm (how you plan to write your code, step by step) in a modular way below. [1 mark]

Task 3: Run your code and make sure that it works. Pay attention to input values outside the A to B range. Then write your code below.

[1 mark]

e. Optional: Write a code that corresponds each quantized value to an n-bit sequence.

f. Optional: Read any of the wave files in Matlab Sheet 2, and plot it versus time. What are the maximum and minimum values that this signal can take? Using your code in part (d), quantize each sample in your wave file using an n-bit quantizer, and make a new wave file. Vary n from 1 to 8 and listen to the generated wave file. How the quality changes once you vary n?

Section 2: Risk Game

Recommended time for completing the tasks in this section: 90 minutes

In this section, we are going to simulate part of a board game known as Risk. In Risk, players conquer countries on the planet by attacking their neighboring enemies. The way an attack is implemented in the game is as follows:

The attacker attacks with three soldiers, each represented by a red die; let's refer to them by R1, R2 & R3. The defender defends with two soldiers, each represented by a white die; let's denote them by W1 & W2. For every attack, the attacker and the defender roll all their dice and sort them in descending orders. Without loss of generality, assume R1 >= R2 >= R3 and W1 >=W2. Then, if R1 > W1, a defender soldier is considered killed, otherwise the attacker has lost a soldier. Similarly, if R2 > W2, a defender soldier is considered killed, otherwise the attacker has lost a soldier. For example, suppose R1 = 5, R2 = 3, R3=3, W1 = 4, and W2 =3. Then because R1 > W1, one white soldier is killed, but because R2 is not strictly greater than W2, a red soldier is also killed. So both sides lose one soldier. Other possible scenarios are when red kills two white soldiers or the opposite. Here we want to find out what are the chances for instance that the attacker kills two white soldiers in an attack. In this section, we calculate this probability in two ways: an exact way and an approximate way. We then compare the two results.

Exact solution:

Once we roll 5 dice, there are 6 x 6 x 6 x 6 x 6 possibilities. If we recreate all these cases, using a code, we can count the number of cases in which the attacker kills 2 white soldiers. The probability of this event will then be given by

(#cases where attackers kills 2) / (total number of possibilities).

a. Write a function that accepts 5 inputs R1, R2, R3, W1, and W2, and, as the output, specifies how many white soldiers are killed. Note that these numbers are not necessarily in order. Your function should take care of sorting if needed. Before writing the code, think of different things that needs to be done and come up with an algorithm.

Task 4: write your function code below [1 mark]

b. Now, using for loops, create all possible values for your 5 dice, and use your function in part (a) to see in which cases two white soldiers are killed. Calculate the probability of killing two white soldiers with your code and write the answer below:

The chance of killing two white soldiers =

Approximate solution (Monte Carlo simulation):

In the Monte Carlo simulation, we instead of looking at all possible cases (which could be quite a large number), we only look at a randomly selected set of cases, and based on that set approximate the probability of interest. This way we can avoid large computational tasks. In some cases, an exact solution may simply not exist, in which case the Monte Carlo simulation is our best option. Let's apply this method to the Risk game.

c. Write a function that generates a random number out of 1, 2, 3, 4, 5, 6. The chance of getting each of these numbers must be the same. This would simulate the process of rolling a die. There are various ways to do this job. You may check how the built-in function rand would help you with this, or find other relevant built-in functions that may directly do the job (randi, for instance).

d. Using the functions in parts (a) and (c), you can generate 5 random die values and specify how many white soldiers are killed. Now, write a function and repeat this process n times. n could be an input to your function. Calculate the chance of killing two white soldiers based on the result of these n rounds. That is the output of your function will be #successful attacks/n.

Task 5: Vary n from 1 to 1000 and plot the chance of success versus n. Show your results below. How would you interpret the results? How many random trials would be sufficient in order to estimate the probability of interest? Compare it with the number of cases considered in part (b).

[1 mark]