Topic 1: Introduction to Images and MATLAB

1. Define a row vector *v*1 in MATLAB with entries [4 1 5]. Now, suppose we want to define a column vector *v*2 that has the exact same entries as *v*1. Let's consider three different ways we could accomplish this.

   a. Define a column vector at the prompt, similarly to how we defined *v*1.
   b. Open a new script and use the following for loop:

   ```
   for ind=1:3
           v2(ind,1) = v1(ind);
   end
   ```

   c. Take what is called the ***transpose*** of vector *v*1 by typing "v2=v1' " at the prompt.

   Which of these methods was the easiest to implement? Which was the hardest? Just like in this example, there are often many ways in MATLAB to accomplish the same task.

2. Earlier, we wrote the function "mysum" to calculate the sum of the integers from 1 to *n*, where *n* is an integer we specify as an input to it. A more common task we may want to accomplish is to find the sum of an arbitrary vector of any length.

   Let's look at how we can construct a program to do this by adapting the code we used for "mysum" (available on the website in the file "intro.m") and creating a new function called "myanysum".

   a. Start by copying the code from "mysum" into the new file. Change the name of the function in the first line of code.
   b. Instead of using *n* as an input, we want to input some vector *x*. Change this in the first line of code.
   c. In order to use the for loop, we still need to define *n* so that the computer knows when to stop the loop. To define *n*, we can use the command "length(x)", which gives us the number of entries in our vector. Before adding this line of code to the function, try using this command at the prompt with vectors of your choice that have varying numbers of entries to familiarize yourself with it.
   d. The last thing we need to do is adapt the line of code in the loop in which we update the variable mysum1 so that it adds the appropriate value at each iteration. To do this, recall that the 3rd entry of a vector *x* can be accessed by typing " x(3) ".
   e. Save the program and try running the function from the prompt for various vectors of your choice. Compare the results to those obtained using the built-in function "sum".

3. Earlier today, we wrote code to simulate *n*=100 coin flips and displayed the results using the command “hist”. The code for doing so is available on the website in the file “intro.m”. Let’s look at this code a bit further.

   a. Try changing the value of *n* to the following numbers: 10, 50, 200, 1000. How do the bars in the histogram change relative to each other as you increase the value of *n*?
   b. In the if-statement, we compare the value of “check” to 0.5. Let’s try changing that number to other numbers. With *n*=1000, try changing it to the following numbers: 0, 0.1, 0.25, 0.33, 0.67, 0.75, 0.9, 1. How does changing this value affect the histogram?

4. The if-else logic that we used in problem 3 provides a method for allowing one of two alternatives to occur. However, there are many occasions in which we have more than two alternatives we need to consider. Let’s look at how we can write code to simulate *n*=1000 observations from a spinner with three equally likely outcomes (See the image linked to on the website), marked 1, 2, or 3.

   To do this, we need to introduce a new command called “elseif”. This functions like the “else” command in that it will only occur if the condition in the “if” command did not work. However, unlike the “else” command, it will check a new condition. The “elseif” command can be used multiple times within a conditional statement to let us consider a large number of alternatives.

   a. Copy the code that we used for problem 3 into a new file. We will be basing this program on that, just altering it to suit our needs for this problem.
   b. Instead of seeing if “check” is less than 0.5, we want to see if it is less than 1/3. Also, we want *x* to take on the value 1 in this case, not 0. Make these changes to the code.
   c. Replace the “else” command with “elseif check<2/3” and change the assigned value of *x* to 2.
   d. Before using “end” to complete this logical statement, we now need to add in a new “else” statement that assigns a value of 3 to *x*.
   e. Use “hist” to plot the results of this code. Try running the code a few times.

5. Based off of your code in problem 4, write a program to simulate *n*=100 rolls of a standard 6-sided die (Each side is equally likely to be rolled). This will require you to use additional “elseif” statements.

6. Alter your code from part 5 to make rolling a 5 be **twice** as likely as rolling a 2 while keeping the other values equally likely. This will simulate rolling a loaded, or unfair die. HINT: More than two “elseif” statements will need to be changed.

7. On the website, there is a list of all of the grayscale/black & white images built-in to MATLAB. Try reading in many of them and take a look at them using the “imagesc”, “image”, and “imshow” commands. What are some readily apparent advantages and disadvantages to each of these commands for this type of image?

8. Repeat problem 7, but for the list of RGB images on the website.

9. Repeat problem 7 for the following small images:

$$A1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

$$A2 = \begin{pmatrix} 230 & 195 & 36 \\ 45 & 15 & 105 \end{pmatrix}$$