You must work either on your own or with one partner. If you work with a partner, you must first register as a group in CMS and then submit your work as a group. Adhere to the Code of Academic Integrity. For a group, “you” below refers to “your group”. You may discuss background issues and general strategies with others and seek help from the course staff, but the work that you submit must be your own. In particular, you may discuss general ideas with others but you may not work out the detailed solutions with others. It is not OK for you to see or hear another student’s code and it is certainly not OK to copy code from another person or from published/Internet sources. If you feel that you cannot complete the assignment on your own, seek help from the course staff.

Objectives

Completing this project will solidify your understanding of user-defined functions and vectors (1-d arrays) and introduce you to more MATLAB graphic functions, including the histogram. Part A consists of a warm-up for user-defined functions and a simulation of the central limit theorem—a famous result in statistics and probability—that requires us to use arrays, functions, and subfunctions. In Part B you will perform a sensitivity analysis, which is an important concept and tool in engineering and computational science.

1 Your Own Star

Implement the following function as specified:

```matlab
function myStar(pts, inc, r)
    % Draw a star with pts points that lie on a circle with radius r.
    % The lines of the star connect the ith point with the (i+inc)th point.
    % Use markers to highlight the points of the star.

    close all % Close all figure windows
    figure % Start figure window
    bound = 1.1*r;
    axis([-bound bound -bound bound]) % Axis limits (center over origin)
    axis equal % Equal scaling for x and y axes
    hold on % Subsequent plot cmds use current axes

    for i = 1:pts
        plot([x(i) x((i+inc)mod(pts,pts))], [y(i) y((i+inc)mod(pts,pts))], 'o-')
    end
end
```

The function call `myStar(8,3,2)` should produce the figure shown on the right: an 8-pointed star with points that lie on a circle with radius 2 where each point is connected to the point 3 forward from it. The function call `myStar(5,2,1)` creates a traditional five-pointed star with points that lie on the unit circle. Choose marker and line colors as you wish. See Project 1 if you need a reminder on how to plot markers and lines, and you can also search MATLAB documentation (top right search box in the MATLAB desktop) for more information. The following graphics commands should be used to set up the figure window:

```matlab
close all % Close all figure windows
figure % Start figure window
bound = 1.1*r;
axis([-bound bound -bound bound]) % Axis limits (center over origin)
axis equal % Equal scaling for x and y axes
hold on % Subsequent plot cmds use current axes
```

Use the command `hold off` at the end of your function in order to return MATLAB to its default graphics state.

→ Add an extra comment line to the function comments of function `myStar` to answer this question: what shape is drawn if parameter `inc` has the value one?

You must implement and make effective use of the following function in drawing the star:

```matlab
function [x, y] = polarRad2xy(r,theta)
    % (x,y) are the Cartesian coordinates of polar coordinate r, theta.
    % theta is in radians.
```

Note the similarity between this function and `polar2xy` developed in lecture.

Submit your files `myStar.m` and `polarRad2xy.m` on CMS.
2 Simulating the Central Limit Theorem

The central limit theorem is an interesting idea in statistical theory which tells us that if we gather many samples of random numbers, and then calculate the average of each of the samples, these averages will follow a normal (i.e. bell-shaped) distribution. This is true regardless of how the original random numbers were distributed.

For example, you are now familiar with the `rand` command in MATLAB, which produces a random number between 0 and 1. We call this a uniform random number because all numbers in this range have an equal chance of being selected. If we generated 300 uniform random numbers using `rand` and drew a histogram of these numbers, we would get a plot that looks similar the one on the right; as expected, values in the range (0,1) appear with similar frequencies. Now let’s suppose we generate \( n \) different samples of 300 uniform random numbers and calculate the average of each of these samples. Interestingly, the histogram of these \( n \) averages looks like the symmetric, bell-shaped one below, which is called a normal distribution.

What is even more interesting is that this histogram of \( n \) sample means would have the same shape even if the random numbers were not at all uniform. For example, suppose we generated \( n \) samples of 300 exponential random numbers, which are skewed to one side because they include many values which are close to 0. The averages of these \( n \) samples of exponential random numbers still form a normal distribution.
In this project, you will generate samples of random numbers from different distributions: a uniform distribution, an exponential distribution, and a triangular distribution. You will produce a series of graphs that demonstrate that the central limit theorem is true for all of these distributions—see the graphic on page 5 for an idea of what the end result will look like. You will also explore how the distribution of sample averages is affected by variables such as the sample size (i.e. how many random numbers are in one sample) and the number of samples which were generated.

Keep in mind that we are only exploring these ideas of statistics and probability—we explain briefly the distributions and you “play” with them by writing code—you do not need to learn or memorize any details on the theory of probability and statistics! Of course, we do hope that in addition to practicing programming you will also find the statistical ideas presented here interesting. After all, the probability of your going on to take a course in statistics and subsequently using that knowledge is high.

Download the file SimulateCLT.m from the course website under “Projects”. This file contains a main function SimulateCLT, which will be responsible for producing the graphic and which already contains some code to help you test your work. The file also contains a set of subfunctions, which you will implement first, that generate random samples and calculate the averages of samples. When you are done, submit your completed SimulateCLT.m file on CMS.

Please do not use the built-in functions sum and mean in this project. You also should not use any vectorized code, meaning that all arithmetic operations should be done on scalars. The purpose of these rules is to ensure that you have the opportunity to practice using vectors with loops.

2.1 Three different distributions

2.1.1 Sample from a uniform distribution

The first function you write will generate a sample of uniform random numbers and store them in a vector. Implement this function as specified here:

```matlab
function u = GenerateUniformSample(sampleSize)
% u is a vector containing sampleSize uniform random numbers in the range (0,1).
% Assume sampleSize > 0.
```

Notice that this function is included in the file SimulateCLT.m, but it is not the main function; this function is called a subfunction which can be called from the main function, or from any other function in the file, to perform the task that it implements.

You should test that this function works properly before moving on. Notice the following lines in the main function SimulateCLT:
sampleSize = 150;
subplot(3,4,1)
hist(GenerateUniformSample(sampleSize))

These lines of code draw a histogram of the vector returned by the `GenerateUniformSample` function. If this subfunction has been implemented correctly, you should see a histogram which looks like Figure 1 in the upper left corner of the histogram grid when you run the program. You can run the program either by pressing the green “Run” button or by typing `SimulateCLT()` in the command window.

### 2.1.2 Sample from an exponential distribution

Next we would like to be able to generate a sample of exponential random numbers and store them in a vector. While MATLAB has built-in functions that can do this easily for us, we will instead use an interesting method called *inverse transformation sampling*, described below, to generate random numbers from almost any kind of distribution using only uniform numbers generated by `rand`.

One property of exponential random numbers is that they can be described by the following cumulative distribution function (CDF):

\[ y = 1 - e^{-\lambda x} \]

This function computes the probability (a number between 0 and 1) that any random exponential value is less than or equal to \( x \). \( \lambda \) is known as a parameter (a constant) of the exponential distribution; we will just assume for this project that \( \lambda = 1 \). If we invert the function \( y \), we obtain the following inverse cumulative distribution function:

\[ x = \frac{\ln(1-y)}{-\lambda} \]  

(1)

As stated above, \( y \) is a probability in the range \((0,1)\) and so it can be simulated by a uniform random number generated by `rand`. Substituting a uniform random number for \( y \) in equation (1) therefore produces an exponential random number.

Implement the following subfunction. You are welcome to use function `GenerateUniformSample` to complete this function, and you can test that it works in the same way; after completing function `GenerateExponentialSample`, you should see another histogram in the main simulation graphic when you click “Run”. This histogram should look like the histograms at the top of Figure 3.

```matlab
function e = GenerateExponentialSample(sampleSize)
    if sampleSize <= 0
        error('Sample size must be positive.');
    end
    u = rand(sampleSize, 1);
    e = -log(1 - u) / lambda;
end
```

### 2.1.3 Sample from a triangular distribution

The triangular distribution is another way of distributing random numbers, named for the roughly triangularly-shaped histogram that triangular random numbers produce (see Figure 4). We can use inverse transformation sampling again to generate triangular random numbers. The CDF for triangular random numbers is

\[ y = \frac{x^2}{2} \]

The inverse CDF we will use to generate a triangular random number \( x \), given a uniform random number \( y \) (generated by `rand`), is therefore:

\[ x = \sqrt{2y} \]  

(2)

Note that we are only using the positive square root to compute triangular random numbers. Implement the following subfunction, and you are again welcome to use your `GenerateUniformSample` function to do so. Finally, test your function as you did before: when you run the program, you should now see a third histogram which looks similar to Figure 4.

```matlab
function t = GenerateTriangularSample(sampleSize)
    if sampleSize <= 0
        error('Sample size must be positive.');
    end
    u = rand(sampleSize, 1);
    t = sqrt(2*u);
end
```
2.2 Calculating the averages of many samples

Now that you can use your three subfunctions to generate a sample of random numbers from each of three distributions, we would like to be able to generate many samples at once and compute the average of each sample. According to the central limit theorem, it is this group of averages which should follow a normal distribution (i.e. have a symmetric, bell-shaped histogram), regardless of whether the random numbers were from a uniform, exponential, or triangular distribution.

Implement the following subfunction, which should make use of the three subfunctions you have written already:

function [uM, eM, tM] = GenerateSampleMeans(numSamples, sampleSize)
% Creates three vectors of length numSamples that store the averages of samples of % random numbers. This function generates numSamples samples from each of the three % distributions, where each sample consists of sampleSize numbers. uM stores the % averages of each uniform random sample, eM stores the averages of each exponential % random sample, and tM stores the averages of each triangular random sample.

2.3 Simulation graphics

We are finally ready to implement the main function, SimulateCLT, which will draw a series of histograms to demonstrate that averages of many samples follow a normal distribution. An example of the expected end result is shown below:

The first column of histograms shows what one sample from each of the three distributions looks like. The second through fourth columns of histograms display the distribution of many averages of samples. Look at the title of each histogram to see how the number of samples generated from each distribution varies, as well as how the number of random values in each sample varies.

Complete the implementation of the following function. You will notice that it has already been started, since the code which draws the first column of histograms in the above figure has been written for you. Your task is to produce the rest of the histograms.
function SimulateCLT()
% Draws a 3-by-4 grid of histograms which demonstrate the Central Limit Theorem for three
% distributions of random numbers. The first row plots the uniform distribution, the
% second plots the exponential distribution, and the third plots the triangular
% distribution. The first column of each row (plots 1, 5, 9) shows a histogram of one
% sample from the distribution; subsequent columns show histograms of sample averages.

A few notes on how to implement this function:

- The three histograms in one column of the grid are produced by calling the GenerateSampleMeans function once; that is, if we were to write

\[ [uM, eM, tM] = \text{GenerateSampleMeans}(30, 50); \]

We would obtain the three vectors \( uM \), \( eM \), and \( tM \), which each contain the averages of 30 samples of uniform, exponential, and triangular numbers, respectively. Each of the 30 samples generated would consist of 50 random numbers.

- To draw the histogram that is in the first row and second column of the sample graphic, for example, we would next need to write

\[ \text{subplot(3,4,2)} \]
\[ \text{hist}(uM) \]

The \text{subplot} command specifies where on the grid we would like to place the histogram produced by the \text{hist} command; the “3,4” specifies that the grid of histograms has three rows and four columns, and the “2” specifies that the histogram is going in the second space on the grid. The grid is numbered from left to right such that the plot in the upper left is 1, the plot in the upper right is 3, the first plot on the second row is 4, and so on, until the plot in the lower right which is numbered 12. It is possible to draw the entire grid using nine pairs of \text{subplot} and \text{hist} statements, but this is quite repetitive. You should use a loop to draw at least three histograms on one iteration.

- Note that each histogram has a descriptive title that states how many sample averages it graphs and how many random numbers were included in each sample. You can use the \text{sprintf} and \text{title} commands which you have used before (review \text{clickLines.m} from Project 1) to add a title to each plot.

- Each column of histograms uses a different combination of the input parameter values \text{numSamples} and \text{sampleSize} of the \text{GenerateSampleMeans} function. Three different values for these two input parameters are stored in the vectors \text{numSamplesVec} and \text{sampleSizesVec}. The first column of histograms uses the first value of each vector as inputs to \text{GenerateSampleMeans}, the second column uses the second value of each vector, etc.

- You are welcome, but not required, to experiment with graphics commands to create histograms in different colors or that have different numbers of bars\(^1\), etc.

\subsection{2.4 Thinking about the central limit theorem}

Look closely at the histograms that your program generates and experiment with different values in the \text{numSamplesVec} and \text{sampleSizesVec} vectors. Then write answers to the following questions as comments in your code: 1–2 sentences for each question is sufficient.

1. What effect do you think the number of samples and the size of each sample have on the shape of the histograms?

2. Why do you think the averages of many samples tend to always be normally distributed, even when the random numbers come from a very differently shaped distribution (such as an exponential distribution)? As this is not a statistics course you simply need to explain your reasoning for this question—your best-effort reasoning suffices.

Project 3 Part B will appear in a separate document.

\(^1\text{Fun, optional challenge: The Freedman-Diaconis rule provides a way to determine a good “bin size” for a histogram, i.e. how many intervals along the x-axis there are. If you like you can research the Freedman-Diaconis rule and see if you can implement it in your histograms! It may help to write a subfunction to do the necessary calculations.} \)