Benjamin Asdell
CEE/MAE M20
UID: 005114992

Homework 6

October 15, 2020

1 Implementing Custom Probability Distribution

1. Introduction

   The objective of this problem is to manipulate the built-in randomness functionality of MATLAB in order to approximate a custom probability distribution. The program samples from MATLAB's normal distribution function in order to determine an input for the inverse of the custom probability function. The program then returns a new random sample from this result and plots each of these in a histogram in addition to the actual custom probability function.

2. Models and Methods

   The custom probability density function is given as:

$$p(x) = \begin{cases} \frac{1}{2}x + \frac{1}{2} & -1 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

   MATLAB's `rand` function only samples from a normal distribution on the interval [0,1]. Therefore, a slight workaround is implemented in order to use the custom distribution. A y value is sampled from the normal `rand` distribution using the logic that the range of the custom probability density function is [0,1]. By hand, the cumulative probability density function $P(x)$ is calculated in Eq. 2:

$$P(x) = \int_{-\infty}^{x} p(u)du = \frac{1}{4}x^2 + \frac{1}{2}x + \frac{1}{4} \tag{2}$$

   However, this function takes in an x for the input, so the inverse of the function needs to be calculated in order to use the previously selected y.

$$x = P^{-1}(y) = -1 + 2\sqrt{y} \tag{3}$$

   Substituting in the y value into this inverse function results in the desired sample from the custom distribution. This process of selected a y and calculating x is done 10000 times, and a histogram is plotted to show the frequency of these experimental probabilities. The actual probability function in Eq. 1 is also plotted to compare the experimental probabilities to the mathematical true value.

3. Calculations and Results

   When the program is executed, the following plot is generated:

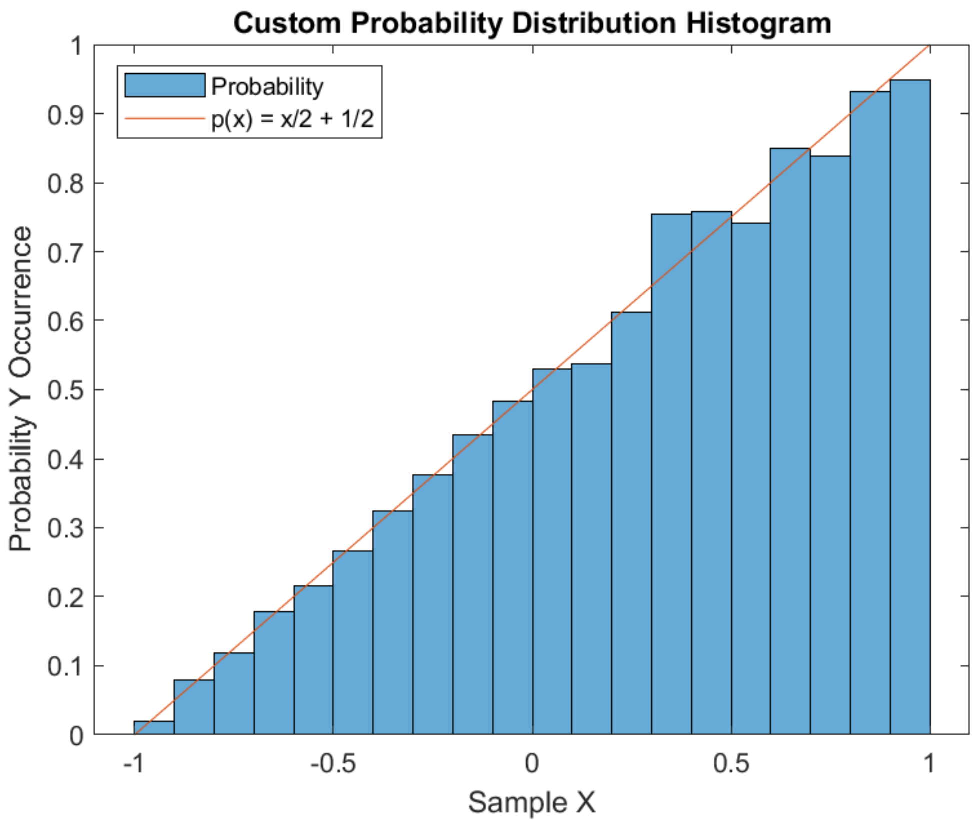

Figure 1: Each bin in the histogram represents the probability occurrence corresponding to the randomly selected x values from the custom distribution. Since there is a finite number of samples, the experimental probabilities do not perfectly fit the probability density function.

4. Discussion

    This program is an interesting statistical problem and the logic of manipulating MATLAB to use the custom distribution was thoroughly intriguing. This method demonstrates MATLAB's flexibility in solving mathematical quandaries outside its built-in functionality. While the histogram clearly does not entirely fit the custom probability density function, it does indicate that running this program with larger sample sizes will most likely converge to a near perfect fit to the line. The bin sizes are also too large to allow the histogram to visibly show such a fit. Another interesting note is that this histogram changes with each execution of the program, and some results are more visibly convergent to the line than others. The randomness of the program's logic is the reason behind this inconsistency, but this is to be expected from any random distribution problem.

2 Monty Hall

1. Introduction

   The goal of this problem is to simulate the classic Monty Hall problem involving three doors, one with a prize car and two with goats behind them. The program is designed to randomly select the player's initial door choice and the winning door, then runs a simulation of the player's choice next move and its result. This process is completed once with the player always switching their choice and a second time with the player always staying with their initial choice.

2. Models and Methods

   As mentioned, the Monty Hall problem is based on an old game show hosted by the namesake of the problem. In the game, there are three doors of which one hides a prize car and the other hides two goats. Obviously, the goal of the game is to choose the door with the car to win. The twist is that after the player makes a selection, the host reveals a goat behind one of the other doors. This leaves two doors with either a goat or car behind them. The player then has a choice to stay with their initial selection or switch to the remaining door. This program simulates the Monty Hall problem using the Monte Carlo method.

3. Calculations and Results

   The results of the two simulations are printed to the command window when the program is executed:

   ```
   The probability of winning when switching is: 0.6696
   The probability of losing when switching is: 0.3304
   The probability of winning when staying is: 0.3401
   The probability of losing when staying is: 0.6599
   ```

   These results are experimental and vary slightly with each program execution. However, the consistency between the two simulations and the overall consistency between trials demonstrates that these experimental values are very close to the true values for the probabilities. Note that the probabilities of winning when switching and losing represent the same concept, and the other two are similarly linked.

4. Discussion

I had heard of this problem on the Internet and in mathematical discussions, so I was very interested in seeing MATLAB simulate this. Many mathematical proofs and arguments have shown that the player has a greater chance of winning the car if they switch their answer after the host reveals a goat, regardless of their initial guess. In fact, it has been shown that there is a 2/3 chance of winning using this logic. The program matches with this expected result very much, the experimental probabilities of winning when switching (and losing when staying) is very close to 2/3. I found that the best way to explain this counterintuitive finding is that always switching from the initial selection inverts the incentive of the entire problem. If the player knows they are going to switch, then their incentive at the start of the problem is to choose a goat door. That way, the host reveals the other goat door, and the player switches to the winning door. This means that the player wants to choose either of the two goat doors to start; in other words, they have a 2/3 chance of winning using this logical strategy.

3 Numerical Integration

1. Introduction

   The purpose of this problem is to once again implement a Monte Carlo, but this time to approximate an integral over a given interval. The program randomly selects points from a normal distribution, then calculates how many of those points fall under the function curve. The probability of those points being under the curve is graphed with respect to how many sample points are generated in addition to the true value of the integral.

2. Models and Methods

   The function being approximated is

   $$\int_0^5 \frac{1}{x^2+1} \tag{1}$$

   Most of the problem's functionality is explained in the Introduction section, but some clarifications are made. The points are selected using MATLAB's `rand` function on the interval [0,5] for x and [0,1] for y. The points are then compared to the value of the function curve (the integrand) at that x value. If the points falls under the curve, a counter is incremented to keep track of the total number of points below the curve. The counter is then divided by the total number of samples generated in that specific trial, and that value is plotted as a point on the final plot. The true value of the integral is also displayed on the plot in order to compare the number of points to the curve as they converge to the true value.

3. Calculations and Results

   When the program is executed, the following plot is generated:

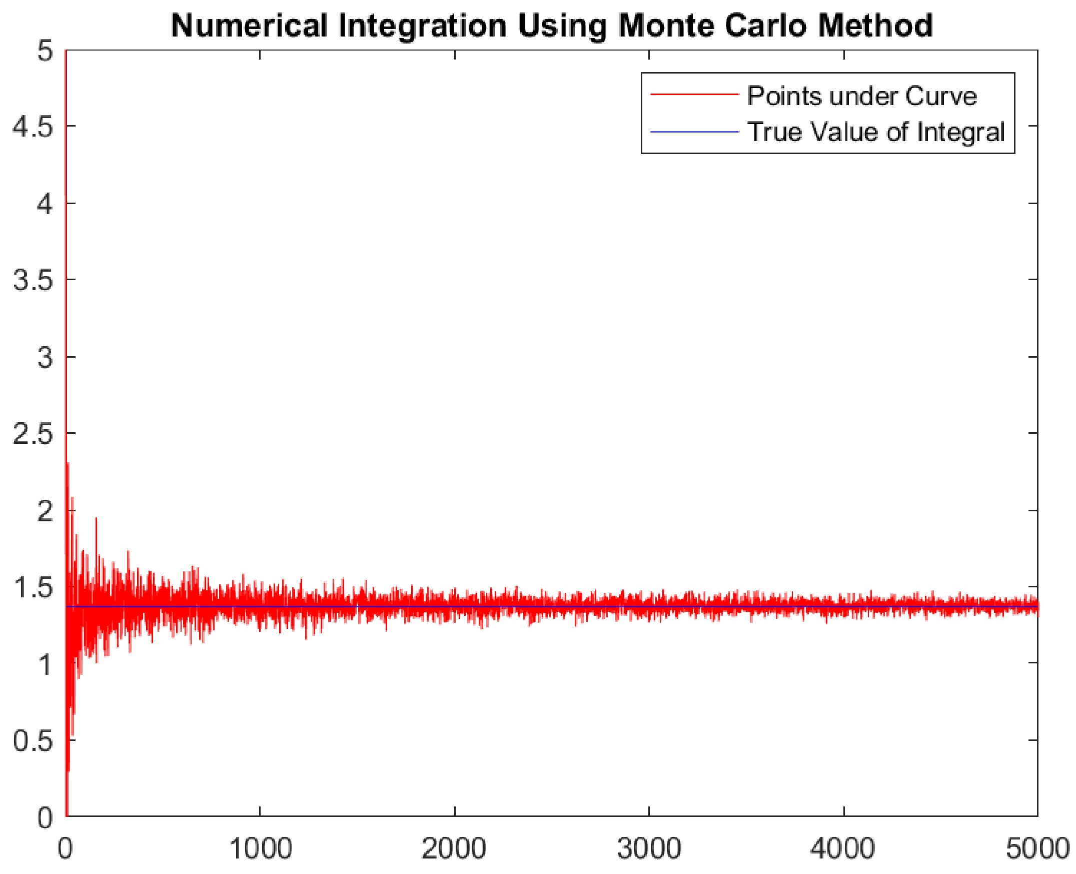

Figure 2: The scatter plot clearly converges to the true value of the integral as the number of samples per trial increases. The deviance from the true value at very small sample sizes is extremely large, while the points near the maximum sample size of 5000 are essentially at the true value. There is still some noise in the data as it converges, but this is expected due to the randomness of the normal distribution from which the samples are taken.

4. Discussion

   This program demonstrates MATLAB's ability to handle mathematical processes and calculations that would be extremely inefficient to compute by hand. It also provides insight into how MATLAB can be used to approximate mathematical calculations using alternative yet rigorous methods. The plot clearly demonstrates that the Monte Carlo method for approximating the integral does converge to the true value of that integral. Even running this program with 10000 or 100000 sample points does not increase the visible or numerical convergence; there is simply too much noise from the randomness of the sampling to converge any further. However, the signficant decrease in deviation from the true value on the plot is evidence enough to confirm this method of numerical integration is a good approximation. As for the utility of a program like to compute integrals, it is difficult to tell because the convergence is not tight enough to easily determine the value to which the points are converging. However, performing this method in MATLAB on a complex integral may be worthwhile to check an answer or give insight into the true value of that integral if the actual value is not as necessary.

4 Dropping Needles on a Grid

1. Introduction

   This problem is designed to simulate dropping needles of varying length onto a tiled grid to determine how many at each length will cross a grid line. The program depends on a dropNeedle function to determine the position of one endpoint of the needle and its angle with the horizontal x-axis. The program then calculates the second endpoint and compares both endpoints to determine if they are in the same tile of the grid. If they are not, then the program realizes the needle has crossed a grid line; the probability of a needle being marked this way is then plotted with respect to the needle length.

2. Models and Methods

   The dropNeedle function that forms the main functionality of the program returns the (x,y) coordinate of one endpoints and its corresponding angle θ to the horizontal x-axis. Using trigonometric functions to find the (x2,y2) position of the second endpoint, the dropNeedle function ensures the needle is entirely within the grid (not the same tile). If not, the function simulates another needle until the boundary condition is satisfied. The main script then recalculates the (x2,y2) and compares the results of the floor function on both endpoints. If either set of floor(x_i) or floor(x_2) fails, the program counter for the number of crosses increments (grid lines occur at integer values). This is an implementation of the Monte Carlo, as the needle position and angle is randomly selected for each trial. There are 10000 needle trials dropped per needle length, and there are 16 different needle lengths tested on the interval [0.1, 1.6].

3. Calculations and Results

   When the program is executed with a 10x10 grid, the following plot is generated:

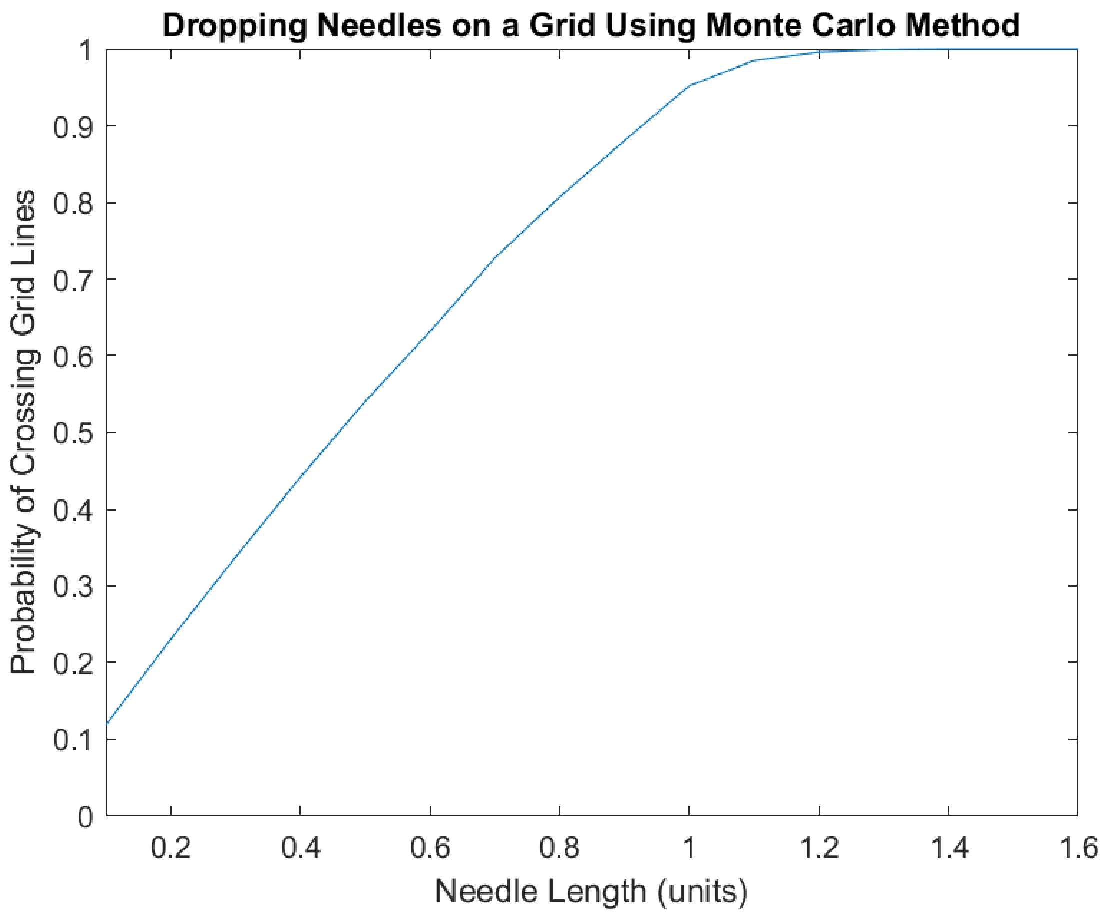

Figure 3: Visibly, as the needle length increases, the greater the probability that a needle will cross a grid line. The values for needle lengths of 1.5 and 1.6 are the equal at 1, meaning all needles dropped will cross a grid line. This is expected because no matter the angle or position of the needle at these lengths, it is too long to stay entirely in one tile. Note that the first point on the plot is at a needle length of 0.1.

4. Discussion

   This program is an excellent example of MATLAB's strength in simulating physical experiments. This experiment done by hand would take an immeasurable amount of time. Instead, this program shows the results of such a thought experiment in mere seconds. It is interesting because this program is entirely expandable by increasing the size of the grid within the code. Another notable result from the simulation is the significant slope of the line as the needle length increases, meaning that seemingly small increases by 0.1 units of length contribute to much higher probabilities of crossing a grid line with those needles. This implies that the possible combinations of positions and angles quickly decreases as the length of the needle grows.