Linh Tang

CEE/MAE M20

UID: 205542275

Nov 14, 2020

HOMEWORK 6

1. Numerical Integration with Monte Carlo

1.1 Introduction

The purpose of this problem is to approximate the numerical result of an integral using Monte Carlo method. Besides, the program plots the numerical results as a function of number samples picked in a rectangle region containing the function. Number of samples under the curve is also calculated.

1.2 Model and methods

Here is the integral needs to be calculated in this problem:

$$\int_0^{10} \frac{1}{x^3+1} \mathrm{d}x.$$

The first thing needs to be done is creating a function to calculate the numerical result of the integral of a function based on the number of samples picked. This function takes four inputs: a function, lower boundary, upper boundary, and number of samples. Then, the

function returns the numerical result of the integral. To calculate the integral over the interval [a, b] by using N samples; the codes are based on this following formula:

$$\frac{(b-a)}{N}\sum_{i=1}^{N} f(x_i)$$

The x value is generated randomly in the interval [0, 10] by using rand () function. Since rand() only generates numbers from 0 to 1, so x will be rand() *(10-0) + 0. The function integration is put at the bottom of the main script.

```
function [approx] = integration(f, L, U, samples)
    sum = 0;
    for k = 1: samples
        x = rand() *(U-L) + L; % random x values from 0-10
        f_value = f(x); % the value of function at that x.
        sum = sum + f_value; % summation area of rectangles
    end
    approx =((U -L)/samples)*sum;   % Using formula
end
```

At the beginning of the script, variables are defined; they are the number of samples, the boundary, the array of the results and the counter variable (counting number of samples under the curve).

```
N = 1000; % number of simulations
a = 0; % lower bound
b = 10; % upper bound
approx_arr = zeros(1, N);
under = 0; % counter
```

The function f(x) that the program needs to integrate is defined by using a function handle: `fx = @(x) 1/(x^3+1);`

A for loop is used to simulate the calculation with the number of samples picked.

Samples are picked in a rectangle containing the region of the function. Based on the function $\frac{1}{x^3+1}$, the maximum range is 1 with the interval [0,10]. The rectangle region has y- coordinate from 0 - 1, and x- coordinate from 0 - 10.

```
x = rand() *(b-a) + a; % generate randomly for x from 0-10
y = rand();  % generate randomly for y from 0-1
```

To check if one sample under the curve or not, the y- coordinate is compared with the expected range calculated from the function. The point is under the curve if its y-coordinate is less than f(x).

```
f_value = fx(x);
% If the sample point is under the curve
    if(y < f_value)
       under = under + 1;
    End
```

In the for loop, the function integration is called to get the numerical results; those results are appended to the array, so it can be used to plot the results later.

```
  samples = i; % number of samples
  approx = integration(fx, a, b, samples);
  approx_arr(i) = approx;
```

The expected result for the integral is calculated by the following codes:

```
syms x;
f = 1/(x^3 + 1);
exp = int(f, 0, 10);
```

For the graph, the expected answer and numerical results are plotted.

```
figure(1);
p = plot(1:N, approx_arr); % plot the approx results
hold on
plot ([1, N],[exp, exp], 'r'); % plot the expected answer
```

The number of samples under the curve and the probability are displayed at the command window.

```
fprintf('Number of samples under the curve is: %d .\n', under);
fprintf('The probability is: %.2f%% .\n', under/N*100);
```

1.3 Results

The expected answer is about 1.2; the numerical results converge to the expected answer.

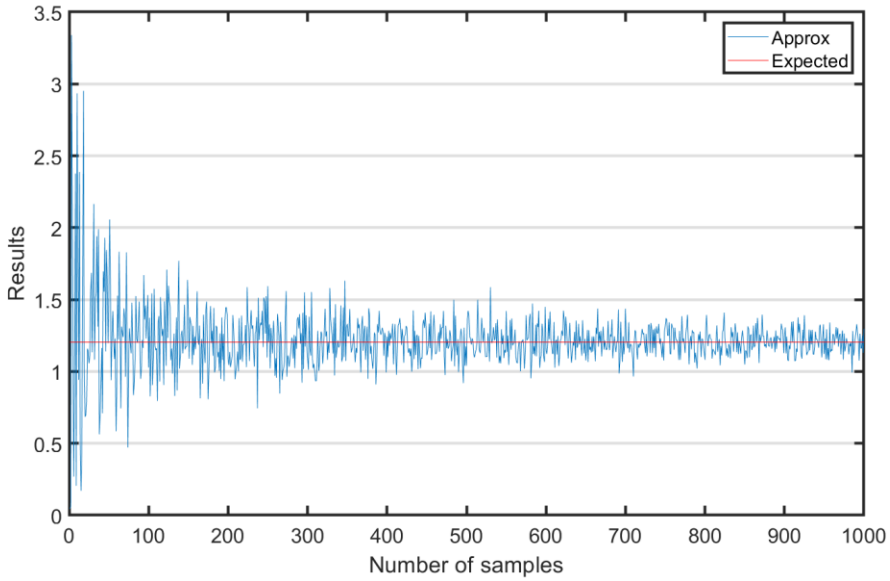


Figure 1. The graph of approximate and expected numerical results of the integral

```
Command Window
Number of samples under the curve is: 130 .
The probability is: 13.00% .
fx >>
```

1.4 Discussion

According to figure 1, the approximate results using Monte Carlo method converge to the expected answer. When the number of samples picked increases, the numerical results get more closed to the expected answer. With a small number of samples, the results are inaccurate when they are far away from the expected line. While running the program multiple times, there are a small number of samples, about 10 - 15% under the curve. In this problem, the challenging things are building the integration function and figuring out the algorithm to find samples under the curve.

2. Implementing Customized Probability Distributions

2.1 Introduction

The goal of this problem is to draw 10000 samples from a customized probability distribution function, and create a histogram of the results with the plot of the function. In this problem, the cumulative probability density function is explicitly calculated; besides, a function is built to return a random sample from the probability function.

2.2 Model and methods

In the problem, samples are drawn from this probability function:

$$p(x) = \begin{cases} -\frac{1}{2}x + 1 & \text{if } x \in [0, 2], \\ 0 & \text{otherwise.} \end{cases}$$

Based on the given function p(x), the cumulative probability density function needs to be explicitly calculated as following:

$P(x) = \int_{-\infty}^{} p(u)\,du = \int_{-\infty}^{0} p(x)\,dx + \int_{0}^{} p(x)\,dx = \int_{-\infty}^{0} 0\,dx + \int_{0}^{} \frac{-1}{2}x + 1\,dx = \frac{-x^2}{4} + x$.

The range of P(x) function is [0,1] due to $\int_{-\infty}^{+\infty} p(x)\,dx = 1$. Therefore, the range of P(x) which is y can be randomly generated using function rand(). Based on y value, a sample x is calculated using $x = P^{-1}(y)$. The inverse function of P(x) is determined by using an online tool.

$y = \frac{-x^2}{4} + x \Rightarrow x = -2\sqrt{-y+1} + 2, or\ x = 2\sqrt{-y+1} + 2.$

Since the range of y is [0, 1], the inverse of P(x) should be $x = -2\sqrt{-y+1} + 2$ which gives us x is [0, 2] (while $2 \le 2\sqrt{-y+1} + 2 \le 4$). Function myRand() generates y randomly from 0 to 1, and returns sample x with corresponding y.

```
function x = myRand()
y = rand();
% p(x)= x - (x^2/4);
% inverse = 2*(1 - x)^(1/2) + 2;
x = 2 - 2*(1 - y)^(1/2);
end
```

At the beginning of the script, a number of simulations and an array for x samples are defined. The function myRand() is called in for loop to draw 10000 samples from the function p(x).

```
N = 10000; % number of simulations

x = zeros(1,N); % array holds the values of N samples
for k = 1:N
```

```
    x(k) = myRand();
end
```

The histogram of results and the plot of function are displayed at the end with the codes:

```
figure (1)
histogram(x, 'Normalization', 'pdf');
hold on
% Plot the p(x) function
fplot(@(x) -1/2*x + 1,[0 2],'r')
xlabel ('x');
ylabel ('Random sample y');
set(gcf, 'Position', [100 40 1000 600]);
set(gca, 'LineWidth', 2, 'FontSize', 15);
ax = gca;
ax.XGrid = 'off';
ax.YGrid = 'on';
saveas(p, 'hw6_p2.png');
```

2.3 Results

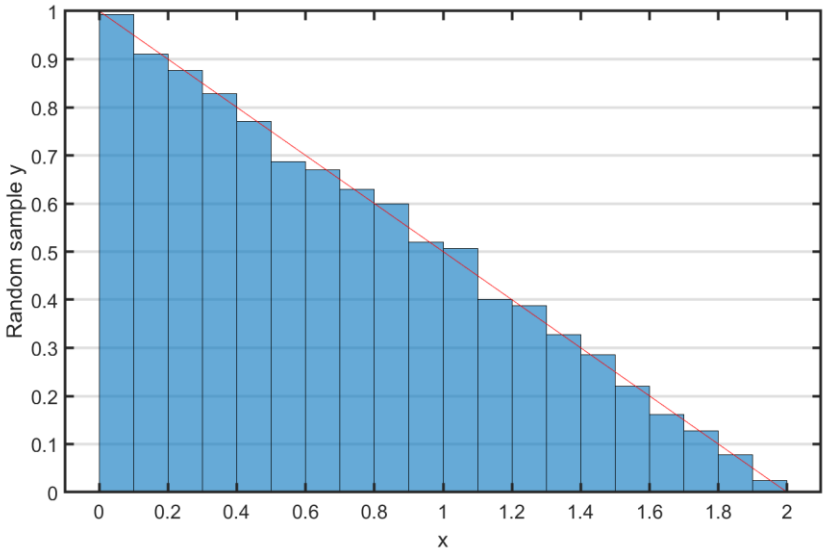


Figure 1. The probability distribution of the function p(x)

2.4 Discussion

According to figure 1, the probability distribution follows the trend line function p(x). The histogram decreases linearly in the interval [0,2]. The point of this problem is there are two inverse functions for the cumulative probability function, but there is only one right function for the range of x [0, 2]. One problem I encountered was when I tried to find the inverse function by using syms and the finverse() function; however, this function returns one reverse function and it was not the want I needed. Therefore, the inverse function was hard coded.

3. Birthday Paradox

3.1 Introduction

The main point of this problem is to answer the question "how many people need to be in a room such that there is a 50% probability that at least two people have the same birthday?" The number of birthdays of people is generated randomly in the range [1, 365] and the probability is calculated in 2000 simulation cycles.

3.2 Model and methods

At first, a number of simulations and an array holding a number of people in the room are defined. In this script, the number of people is from 2 to 101 people will go into the test to find which number in that range gives a 50% probability. There are 100 elements in the array, the array should start at 2 since the problem requires at least 2 people having the same birthday.

```
nTrials = 2000;
arr = linspace (2, 101, 100);
```

There are three for loops being used in this script. The first one is to go through the array containing the number of people and perform the calculation with each number of people. This for loop stops as the program figures out the number which gives the probability from 50.0 - 50.5% by using if condition. The for loop also breaks when it reaches 101 people but the probability is not 50% for that run.

```
for i = 1:length (arr)
    counter = 0; % counter when people have the same birthday
    for
      ....
    end
    % Calculate probability
    P = counter/nTrials;
    % when P is from 50.0- 50.9%
    if(abs(P - 0.5) <= 0.01)
         fprintf ("Number of people needed to be in a room : %d\n",
      length(group));
        break;
    end
    if (i == length(arr)&& abs(P - 0.5) > 0.01)
        disp("There is no number of people satisfying 50% probability
in this run.");
        break;
    end
end
```

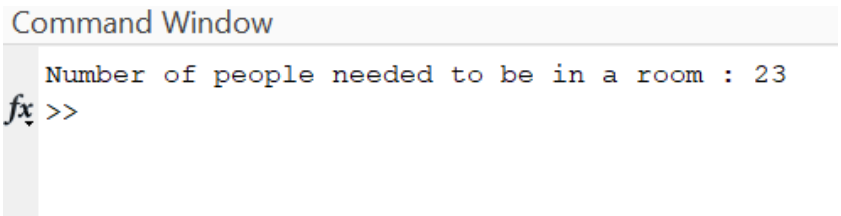
The second for loop is used to simulate the function 2000 times. In this for loop, an array is first defined to hold the number of birthdays when they are generated randomly and put into the array by using another for loop. The function randi() is called to get a random integer from 1 to 365.

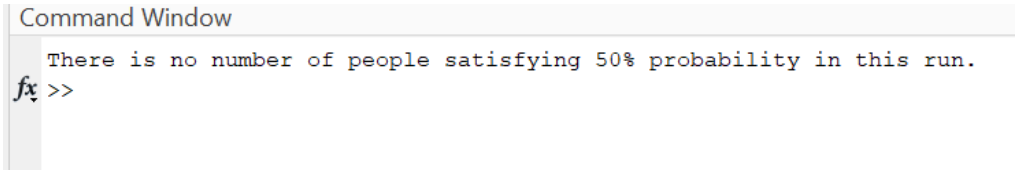
```
for k = 1:nTrials
        group = zeros(1, arr(i)); % the array holding the birthday
        for j = 1: arr(i)
            bday = randi([1 365]); % generate randomly the birthday
            group(j) = bday; % append bday to array


        end
      ...
   End
```

After that, the function sort is used to sort the array “group” in ascending order, then the function diff is called to determine the difference between adjacent elements in the sorted array with the purpose of finding whether two elements share the same value (at least two people have the same birthday). A counter variable is used to count whenever the difference 0 is found in the diff array by using the find() function.

```
        diff_group = diff(sort(group);
        if (find(diff_group == 0))
             counter = counter + 1;
        end
```

3.3 Results





3.4 Discussion

In this problem, the first challenging thing was how to approach the problem to compute the number of people instead of asking user input and checking for each value until 50% was obtained. That was why I chose the range for the number of people, checked if any number in that range satisfied 50% probability. However, the program gave me 22-23 people in some runs, others were unfounded when it exceeded the maximum number of the range. Besides, I encountered infinite loop when I used unidrnd instead of randi function.

If there are 23 people in the room, there will be $\frac{23(23-1)}{2}$= 253 pairs of people. The Poisson approximation for the binomial of that group will be: P = $\frac{253}{365}$= 0.693. The probability of no match will be $e^{-0.693}$; therefore the probability of at least one pair has same birthday is 1 - $e^{-0.693}$ = 0.50. By using Poisson approximation, 23 people gives us the probability that at least 2 people share a birthday is 50%.