EE 381 Lab 3 Report

Uniform Distribution

Daniel Duong and Long Nguyen

Instructor: Dr. Duc Tran

Due 10/29/20

Table of Contents

Introduction

The purpose of this lab was to test the manual calculation of probability density function and cumulative distribution function using the given functions and compare that to the built-in PDF and CDF functions. The lab also helps us learn how to use the built-in function to calculate probability and cumulative value. The purpose of Question 2 purpose was for us to simulate an experiment that generates 3 random points on the circle for every random experiment and then to calculate the probability that points lie within the same semi-circle.

Procedure

1. **Question 1:**
    a. We initialized our variables and the constant to be 1/9th. We plotted 1000 points
    b. Then created an array named xValues of randomly spaced out numbers from range 0 to 11 with the incrementer of 0.0001 and an empty list named theList that will hold the probabilities.
    c. We iterated through the xValues array and iteratively checked if the value of xValues were less than or equal to 1 or if the value of xValues were greater than equal to 10, so that way we added 0 as an integer to the numList list.
    d. If the values of the xValues were not less than or equal to to 10 or not greater than or equal to 10, then we iteratively checked if the values of xValues is greater than 1 and if the values of xValues is less than 10, so that way we added constant to the list.
    e. We labeled the probability's function's graph x axis as "X Values" and the y axis as "f(x)" and we passed in the xValues list and the listNums which contains the list of probabilities to the plot function.
    f. For the Cumulative Distribution Function, we iterated through the xValues array and iteratively checked if the values of xValues is less than or equal to a. If so, we added 0 to the cdDistrib list.
    g. Then we iteratively checked if the value of xValues were greater than a and if the value of xValues were less than b. If so, we added (xValues[j] - a) / (b - a) to the cdfDistrib list.

h. Then we iteratively checked if the values of Xvalues was greater than or equal to b. If so, we added 1 to the cdfDistrib list.

i. We plotted the Xvalues array and the cdfDisturb.

j. We assigned f1 to be the uniform.pdf which the computer automatically gives you the array of values. We plotted the Xvalues array, f1, and passed in ***'r'*** which means to make the color of the graph red. Labeled the x axis as "X Values" and the y axis as "f(x)"

k. We assigned F1 to be uniform.cdf which the computer automatically gives you the array of values. We plotted the xValues array, F1, and passed in ***'r'*** which means to make the color of the graph read. Labed the x axis as "X Values" and the y axis as "F(x)".

l. We used sublot to arrange all four graphs into one tab rather than 4 different tabs.

2. Question 2:

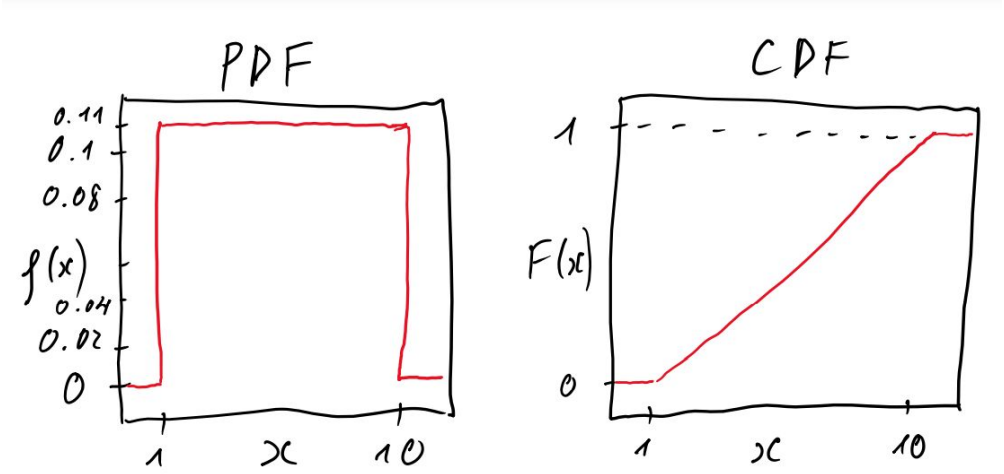
1. We saved the value of a specific radius to a variable and use that to calculate the value of the Circumference of the circle using C = 2*pi*r.
2. Then we generate 3 random numbers between 0 and the circumference of the above circle.
3. Add the 3 random values to a list and sort that list.
4. Then we calculate the difference between the largest element in the list and the smallest element in the list. If that value is less than or equal to circumference / 2 then the 3 points are in the same semi circle. Increase the counter

5. If the difference between the biggest and the smallest element is more than circumference / 2 we check the position of the value in the middle to see which semi-circle it is currently in. If the value in the middle of the array is less than circumference / 2. The we calculate the distance between the largest point and the middle point which is the section with the smallest point in between. If that value is less than circumference / 2 than the 3 points would be in the same semi circle.
6. Else, we check the case where the largest point is in between the first point and the middle point where the middle point value is more than circumference / 2. If the value between the middle point and the first point is less than the semi-circle value. Then we increase the counter by 1.

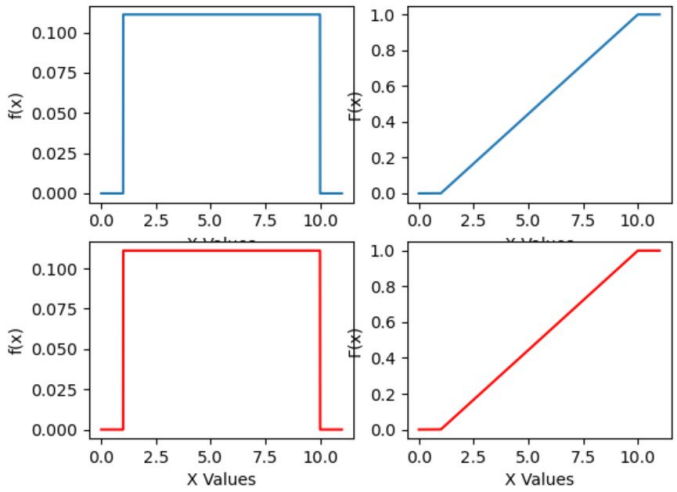
Results and Discussions

Lab 3 Q1 Trials:

Theoretical Results:



1. Our Results:



The graph from question a and the the graph from question b are identical where question a is manually calculated and question b are the built-in functions.

NOTE: The graph marked in blue is our manual calculations; while the red graph represents the computer's automatic calculation.

Lab 3 Q2 Results:

Theoretical Results:

$$\text{Probability} = 3 \times \frac{1}{2} \times \frac{1}{2} = \frac{3}{2^2} = \frac{3}{2^{3-2}} = \boxed{\frac{3}{4}}$$

1. The probability is: 0.74866
2. The probability is: 0.75064
3. The probability is: 0.74791

Conclusion:

We learnt how to use the built-in probability density function and the cumulative distribution function as well as how to graph them. We also learnt to simulate the value calculated by these 2 function using the given equation from the lab prompt. We learned that the third parameter for the function call plot() in python let us input the color we want our graph to be as a single character string(r - red, g-green, …). We differentiated the manual graph and the built-in functions graph by setting the color of the built-int function to red. We found that coming up with the conditions to check to see if the randomly generated values are on the same semi-circle were very challenging. We got stuck at checking to see if the 3 randomly generated values are on the same semicircle. Since we were missing some conditions we were missing some cases therefore we were getting 0.5 as our probability multiple times. We were looking into Pythagorean theorem and law of sines and cosines in order to solve the problem but those methods would require us to generate 2 different set of values to simulate the coordinates plane; however, it sounded like it was going to be a very tedious and more frustrating than our original plan which was to calculate the circumference and distance between the points and to determine the appropriate conditions to determine the probability of 3 randomly selected points lie on the same semi-circle. In question 2, we were able to apply the classical approach whenever the distance between two points were considered by our conditions and iteratively incrementing the counter by 1 and after 10000 experiments, we took the counter divided the number of experiments to simulate the probability of getting 3 randomly selected points that lie within the same semi-circle.

Appendix

Lab3Q1.py

```
#Authors: Daniel Duong and Long Nguyen
from random import uniform

import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import uniform

print("<<< Part A >>>: PDF")

a = 1
b = 10
constant = (1/(b-a))
numPoints = 1000

xValues = np.arange(0, 11, 0.0001)
theList = []

#print(xValues)

for i in range(np.size(xValues)):
    if xValues[i] <= 1 or xValues[i] >= 10:
        theList.append(0)

    elif 1 < xValues[i] and xValues[i] < 10:
        theList.append((constant))

#print("xValues: ", xValues)
#print("size of xValues: ", np.size(xValues))
#print()
#print("theList: ", theList)
#print("size of theList: ", len(theList))
```

```
plt.subplot(221)
#print(type(plt))
plt.plot(xValues, theList)
plt.xlabel("X Values")
plt.ylabel("f(x)")
#plt.show()

print("<<< Part 2: CDF >>>")

cdfDistrib = []

for j in range(np.size(xValues)):
    if xValues[j] <= a:
        cdfDistrib.append(0)

    elif xValues[j] > a and xValues[j] < b:
        cdfDistrib.append((xValues[j] - a) / (b - a))

    elif xValues[j] >= b:
        cdfDistrib.append(1)

plt.subplot(222)
plt.plot(xValues, cdfDistrib)
plt.xlabel("X Values")
plt.ylabel("F(x)")
#plt.show()

print("<<< Part B >>>")

''' Computer's Calculation to compare side'''

f1 = uniform.pdf(xValues, 1, 9)
#print("f1: ", f1)
#print("type of f1: ", type(f1))
plt.subplot(223)
plt.plot(xValues, f1, 'r')
plt.xlabel("X Values")
plt.ylabel("f(x)")
#plt.show()

F1 = uniform.cdf(xValues, 1, 9)
plt.subplot(224)
plt.plot(xValues, F1, 'r')
plt.xlabel("X Values")
plt.ylabel("F(x)")
plt.show()
```

Lab3Q2.py

```
#Authors: Daniel Duong and Long Nguyen
from random import uniform
import random

import numpy as np

numExperiments = 100000
counter = 0

for i in range(numExperiments):
    radius = 5
    a = 0
    circumference = 2 * 3.14 * radius

    x = random.uniform(a, circumference)
    y = random.uniform(a, circumference)
    z = random.uniform(a, circumference)

    points = []
    points.append(x)
    points.append(y)
    points.append(z)
    points.sort()
    #print("Arrays:", points)
    '"print("radius: ", radius)
    print("x: ", x)
    print("y: ", y)
    print("z: ", z) '"

    #biggest = max(x, y, z)
    #print("largest: ", biggest)
    #smallest = min(x, y, z)
    #print("smallest: ", smallest)
```

```
    difference1 = points[2] - points[0]       #when c is biggest and b is sandwiched between a and c
    difference2 = points[2] - points[1]
    difference3 = points[1] - points[0]
    #print("difference: ", difference)
    #print("semicircle value: ", (circumference / 2))
    semi_c = circumference / 2
    # if c - a is less than circumference semicircle with b sandwiched between a and c increase
counter
    if difference1 <= semi_c:
        counter += 1
    elif (difference1 > semi_c):
        if(difference3 <= semi_c):
            if(circumference - difference2 <= semi_c):
                counter += 1
        else:
            if (circumference - difference3) <= semi_c:
                counter += 1

    #elif difference >= (circumference / 2):

print("radius: ", radius)
print("circumference: ", circumference)
#print("x: ", x)
#print("y: ", y)
#print("z: ", z)
#print("largest: ", biggest)
#print("smallest: ", smallest)
#print("difference: ", difference)
#print("semicircle value: ", (circumference / 2))
print("Counter is at: ", counter)
print("The probability is: ", (counter / numExperiments))
```