Probability & Statistics Project

Shamsa Hafeez Dawoodani - sd06162
Alisha Momin - am05757
Umema Zehra - uz05607

January 22, 2022

Dhanani School of Science and Engineering

1 Random Walk

Solution:

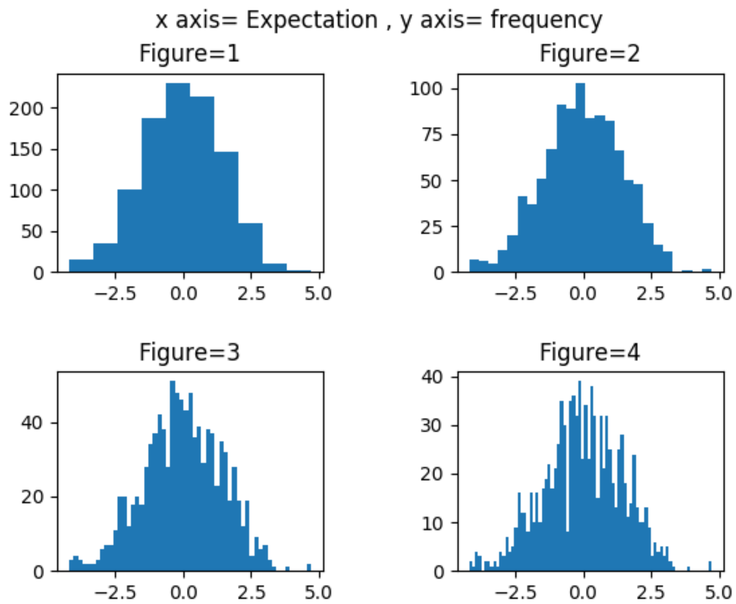
Task 1.1:

This function that takes in two parameters n and p. Where n is the number of steps left or right, as dictated by the probabilities and p is the probability of steps in the right direction. In our code, the total number of experiment were led for 100 steps each is 1000. We set the starting position equal to 0. Over here we generated an array, filled with zero, of the size of experiment i.e 1000. Our function generates a number in the range of 0.0 to 1.0 randomly till $h < n$. Then it checks if $x <$ P(moving right) then it will move in a right direction otherwise it will move in left direction. We store the expected position in a *position_list* and then add all the element of *position_list* in total and then we found the average of the total and store that in numpyarray.The experiment is rehashed several time using iteration and we created one figure containing 4 histograms with a different number of bins i.e. 10, 25, 55, 80 bins for the better result.

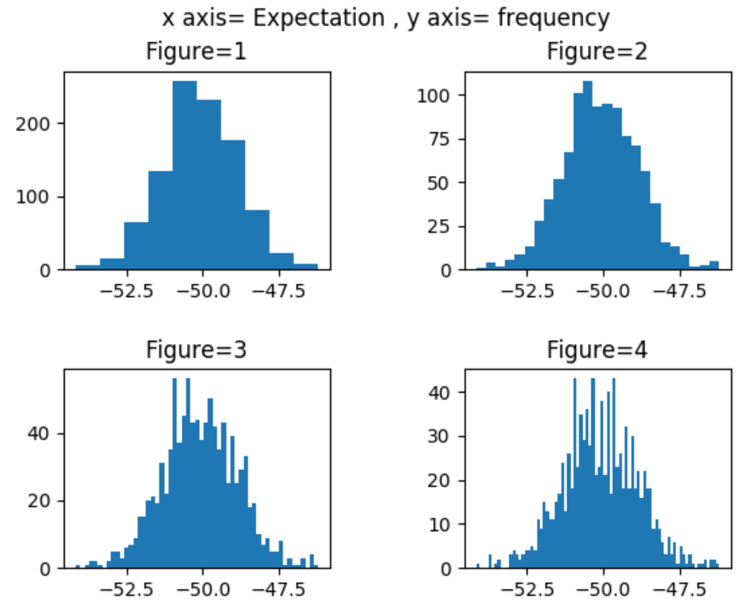
```python
# Task 1.1
import numpy as np
import math
import matplotlib.pyplot as plt
from matplotlib.widgets import TextBox
import random
numpyarray = np.zeros((1000))  # 1000= number of experiment
def q1(n, p):
    global numpyarray
    y = 0
    # we will use the while loop. it will run in the range of no of exp
    while y < 1000:
        position_lst = []
        z = 0
        # overhere 50 is the average number. from here onwards the loop will run in the range of 50
        while(z < 50):
            position = 0
            # it will run till h<n
            for h in range(n):
                # https://www.techiedelight.com/generate-random-float-python/
                # generate random from 0.0 to 1.0
                x = random.uniform(0, 1)
                # if the generated random number is less than the P(moving right) then:
                if x < p:
                    # moving right
                    position += 1
                else:
                    # moving left
                    position -= 1
            # append all the position in the position_lst
            position_lst.append(position)
            z += 1
        # add all the elements present in position_lst
        total=0
        for ele in position_lst:
            total+=ele
        # taking the average of position_lst by dividing it by 50
        numpyarray[y] = total / 50
        y += 1
def hist():
```

```python
            y += 1
def hist():
    global numpyarray
    q1(100, 0.25)
    # Stacking subplots in two directions:
    # https://matplotlib.org/devdocs/gallery/subplots_axes_and_figures/subplots_demo.html
    fig, axes = plt.subplots(nrows=2, ncols=2)
    fig.suptitle('x axis= Expectation , y axis= frequency')
    plt.subplots_adjust(wspace=0.5, hspace=0.5)
    axes[0, 0].hist(numpyarray, bins=10)
    axes[0, 1].hist(numpyarray, bins=25)
    axes[1, 0].hist(numpyarray, bins=55)
    axes[1, 1].hist(numpyarray, bins=80)
    axes[0, 0].set_title("Figure=1")
    axes[0, 1].set_title("Figure=2")
    axes[1, 0].set_title("Figure=3")
    axes[1, 1].set_title("Figure=4")
    plt.show()


hist()
```

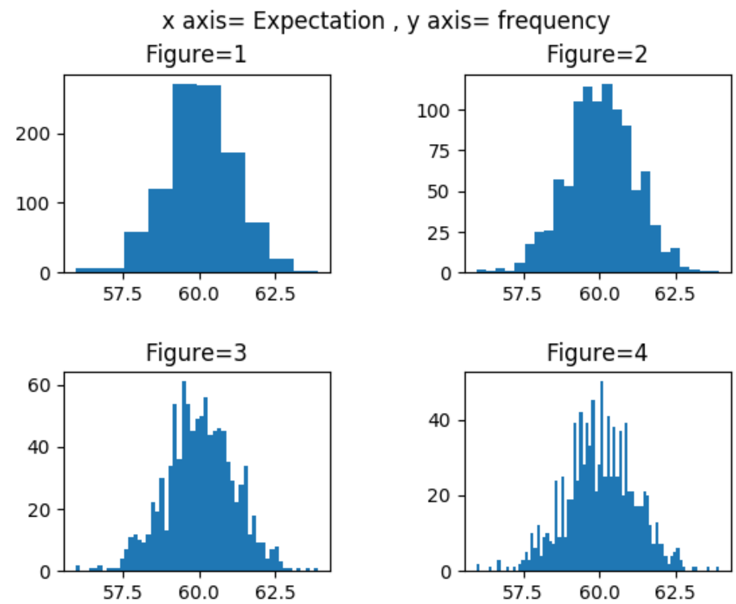
When we put n=100 and p=0.5, it give us the expected value of zero that is:



When we put n=100 and p=0.25, it give us the expected value with the most probability of moving in left direction:



When we put n=100 and p=0.8, it give us the expected value with the most probability of moving in right direction:



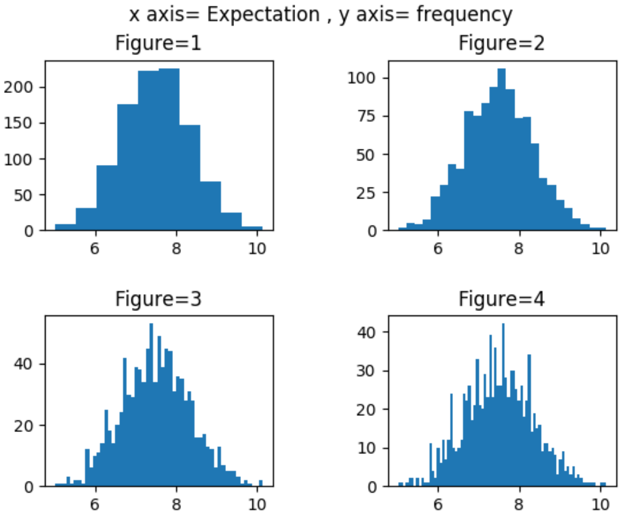
Task 1.2:

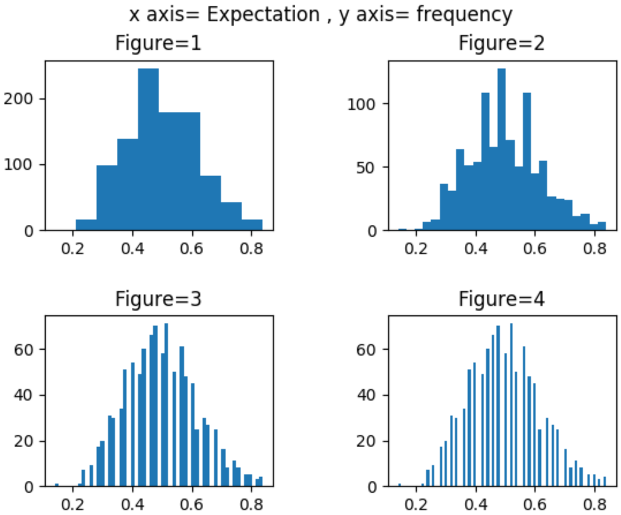
This function that takes in two parameters n and p. Where n is the number of steps left or right, as dictated by the probabilities and p is the probability of steps in the right direction. In our code, the total number of experiment were led for 100 steps each is 1000. We set the starting position equal to 0. Over here we generated an array, filled with zero, of the size of experiment i.e 1000. Our function generates a number in the range of 0.0 to 1.0 randomly till $h < n$. Then it checks if $x <$ P(moving right) then it will move in a right direction otherwise if the current position is greater than zero and $x \geq p$ then it will move in left direction. We store the expected position in a *position_list* and then add all the element of *position_list* in total and then we found the average of the total and store that in numpyarray.The experiment is rehashed several time using iteration and we created one figure containing 4 histograms with a different number of bins i.e. 10, 25, 55, 80 bins for the better result.

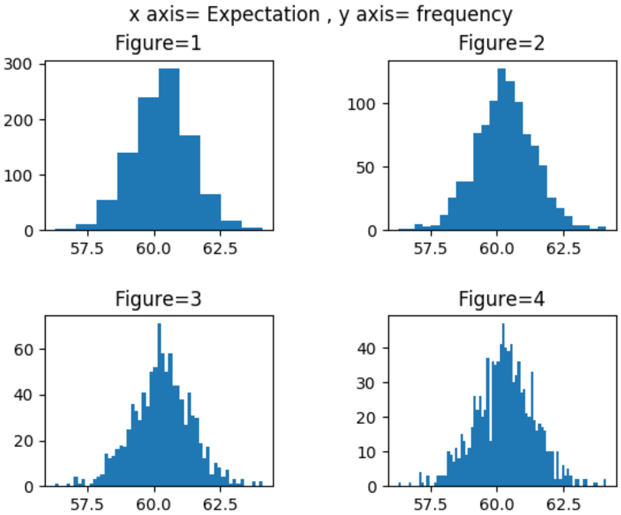
```python
# # task 1.2
import numpy as np
import math
import matplotlib.pyplot as plt
from matplotlib.widgets import TextBox
import random
numpyarray = np.zeros((1000))  # 1000= number of experiment
def q1(n, p):
    global numpyarray
    y = 0
    # we will use the while loop. it will run in the range of no of exp
    while y < 1000:
        position_lst = []
        z = 0
        # overhere 50 is the average number. from here onwards the loop will run in the range of 50
        while(z < 50):
            position = 0
            # it will run till h<n
            for h in range(n):
                # https://www.techiedelight.com/generate-random-float-python/
                # generate random from 0.0 to 1.0
                x = random.uniform(0, 1)
                # if the generated random number is less than the P(moving right) then:
                if x < p:
                    position += 1 # moving right
                if x >= p and position > 0:
                    # if current position is greater than 0 and ->
                    # -> if x is greater than p then move left
                    position -= 1
            # append all the position in the position_lst
            position_lst.append(position)
            z += 1
        # add all the elements present in position_lst
        total=0
        for ele in position_lst:
            total+=ele
        # taking the average of position_lst by dividing it by 50
        numpyarray[y] = total / 50
        y += 1
def hist():
```

```python
        y += 1
def hist():
    global numpyarray
    q1(100, 0.8)
    # Stacking subplots in two directions:
    # https://matplotlib.org/devdocs/gallery/subplots_axes_and_figures/subplots_demo.html
    fig, axes = plt.subplots(nrows=2, ncols=2)
    fig.suptitle('x axis= Expectation , y axis= frequency')
    plt.subplots_adjust(wspace=0.5, hspace=0.5)
    axes[0, 0].hist(numpyarray, bins=10)
    axes[0, 1].hist(numpyarray, bins=25)
    axes[1, 0].hist(numpyarray, bins=55)
    axes[1, 1].hist(numpyarray, bins=80)
    axes[0, 0].set_title("Figure=1")
    axes[0, 1].set_title("Figure=2")
    axes[1, 0].set_title("Figure=3")
    axes[1, 1].set_title("Figure=4")
    plt.show()
hist()
```

When we put n=100 and p=0.5, p=0.25 and p=0.8 respectively give us the expected value:







Task 1.3:

This function that takes in 4 parameters, the start position of object 1, the start position of object 2, the probability of object 1 to move right and the probability of object 2 to move right. In our code, the total number of experiment were led for 100 steps each is 1000 with the average of 50. Over here we generated an array, filled with zero, of the size of experiment i.e 1000. Our function generates two number in the range

of 0.0 to 1.0 randomly. Then it checks 4 if conditions and run until $i_1$is not equal to i_2. Once the position of both the objects are equal i.e $i_1$is equal to i_2 then the while loop is break. We store the steps in a list and then add all the element of list in total and then we found the average of the total and store that in numpyarray.The experiment is rehashed several time using iteration and we created one figure containing 4 histograms with a different number of bins i.e. 10, 25, 55, 80 bins for the better result.
Note: This function works similarly like task 1.1. we only modify few conditions for two objects along with their 2 probability of moving right.

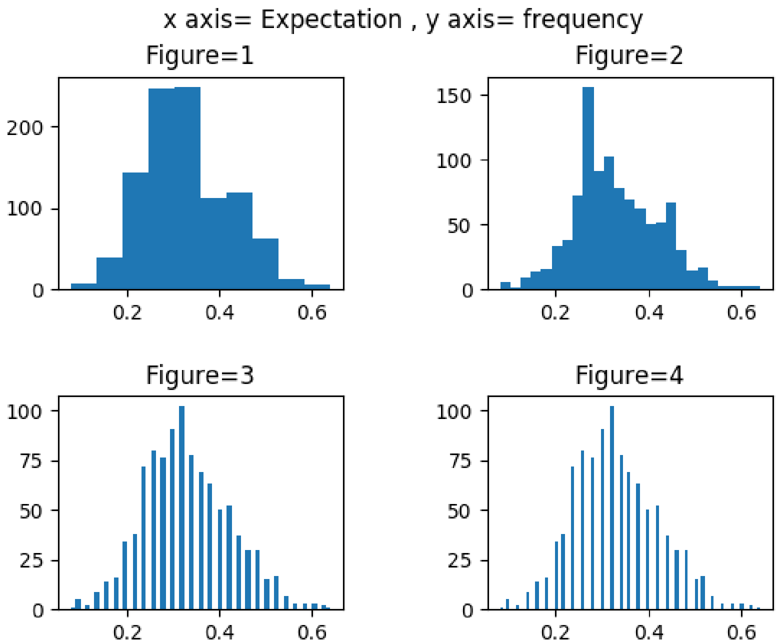
```
# # Task 1.3:
import numpy as np
import math
import matplotlib.pyplot as plt
from matplotlib.widgets import TextBox
import random
numpyarray = np.zeros((1000))  # 1000= number of experiment
def q1(i1,i2,p1,p2):
    global numpyarray
    # updating back to the origional
    temp1 = i1
    temp2 = i2
    # var = 0 # -> checker
    for i in range(1000):  # i<1000 -> no of experiment
        pos_list=[]
        for _ in range(50):
        # updating back to the origional
            i1 = temp1
            i2 = temp2
            step=0
            while i1!=i2:
                pos1=i1
                pos2=i2
                # print(i1,i2)
                # print(pos1,pos2)
                # https://www.techiedelight.com/generate-random-float-python/
                # generate 2 random numbers from 0.0 to 1.0
                x1 = random.uniform(0, 1)
                x2 = random.uniform(0, 1)
                # print(x1,x2)
                if (x1 < p1 and x2 < p2):
                    # check if x1<P1(moving right)-> move right
                    # check if x2<P2(moving right)-> move right
                    # if these are true then it check the below conditions
                    if (i1 + 1 == pos2 or i2 + 1 == pos1):

                        break
                    else:
                        # move right
```

```
            while i1!=i2:
                pos1=i1
                pos2=i2
                # print(i1,i2)
                # print(pos1,pos2)
                # https://www.techiedelight.com/generate-random-float-python/
                # generate 2 random numbers from 0.0 to 1.0
                x1 = random.uniform(0, 1)
                x2 = random.uniform(0, 1)
                # print(x1,x2)
                if (x1 < p1 and x2 < p2):
                    # check if x1<P1(moving right)-> move right
                    # check if x2<P2(moving right)-> move right
                    # if these are true then it check the below conditions
                    if (i1 + 1 == pos2 or i2 + 1 == pos1):

                        break
                    else:
                        # move right
                        i1 = i1+1
                        i2 = i2+1
                if (x1 < p1 and x2 > p2):
                    # check if x1<P1(moving right)-> move right
                    # check if x2>P2(moving right)-> move left
                    # if these are true then it check the below conditions
                    if (i1+1 ==pos2 or i2-1==pos1):
                        break
                    else:
                        i1 = i1+1 # move right
                        i2 = i2-1 # move left
                if (x1 > p1 and x2 > p2):
                    # check if x1>P1(moving right)-> move left
                    # check if x2>P2(moving right)-> move left
                    # if these are true then it check the below conditions
                    if (i1-1 == pos2 or i2-1==pos1):
                        break
                    else:
                        # move left
                        i1 = i1-1
                        i2 = i2-1
```

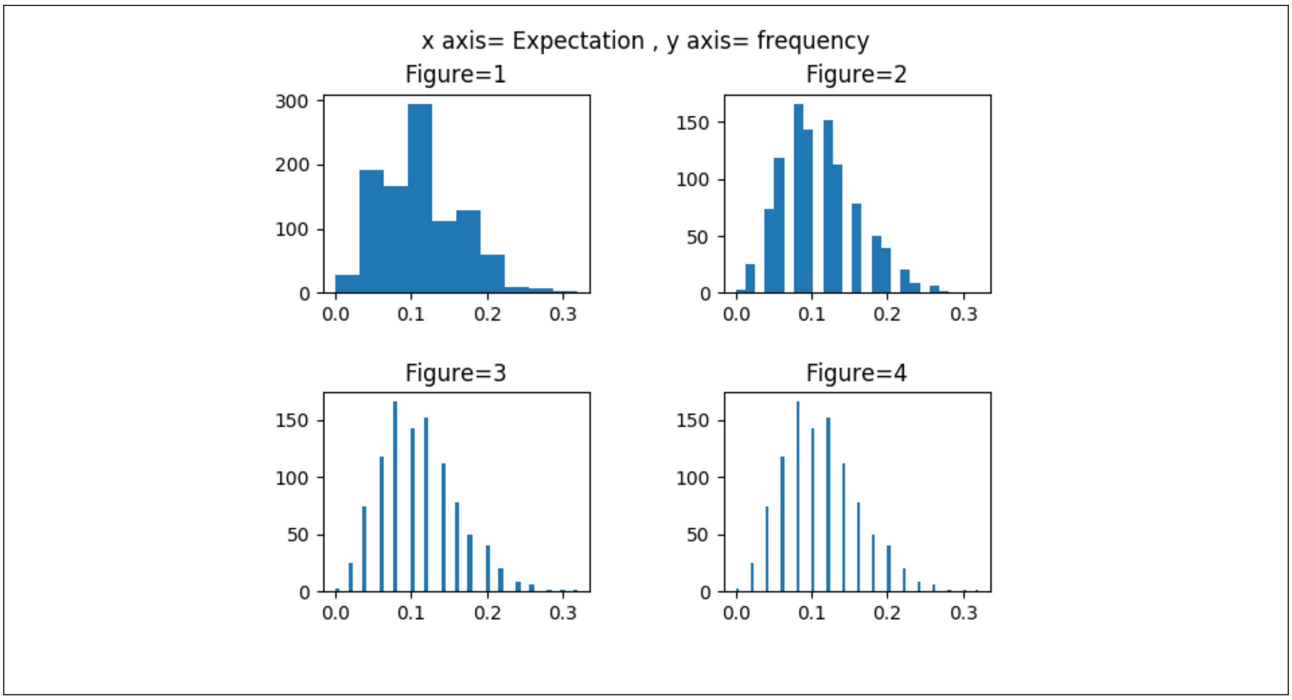
```
                if (x1 < p1 and x2 > p2):
                    # check if x1<P1(moving right)-> move right
                    # check if x2>P2(moving right)-> move left
                    # if these are true then it check the below conditions
                    if (i1+1 ==pos2 or i2-1==pos1):
                        break
                    else:
                        i1 = i1+1 # move right
                        i2 = i2-1 # move left
                if (x1 > p1 and x2 > p2):
                    # check if x1>P1(moving right)-> move left
                    # check if x2>P2(moving right)-> move left
                    # if these are true then it check the below conditions
                    if (i1-1 == pos2 or i2-1==pos1):
                        break
                    else:
                        # move left
                        i1 = i1-1
                        i2 = i2-1
                if (x1 > p1 and x2 < p2):
                    # check if x1>P1(moving right)-> move left
                    # check if x2<P2(moving right)-> move right
                    # if these are true then it check the below conditions
                    if (i1-1 == pos2 or i2+1==pos1):
                        break
                    else:
                        i1 = i1-1 #move left
                        i2 = i2+1 #move right
                else:
                    break
                step += 1
        # print(step)
            pos_list.append(step)
        total = 0
        for ele in pos_list:
            total+=ele
        numpyarray[i]=total/50
        # var += 1 #checker print
        # print(var)
def hist():
```

```python
        # print(step)
            pos_list.append(step)
        total = 0
        for ele in pos_list:
            total+=ele
        numpyarray[i]=total/50
        # var += 1 #checker print
        # print(var)
def hist():
    global numpyarray
    q1(6,10,0.5,0.5)
    # print(numpyarray)
    # Stacking subplots in two directions:
    # https://matplotlib.org/devdocs/gallery/subplots_axes_and_figures/subplots_demo.html
    fig, axes = plt.subplots(nrows=2, ncols=2)
    fig.suptitle('x axis= Expectation , y axis= frequency')
    plt.subplots_adjust(wspace=0.5, hspace=0.5)
    axes[0, 0].hist(numpyarray, bins=10)
    axes[0, 1].hist(numpyarray, bins=25)
    axes[1, 0].hist(numpyarray, bins=55)
    axes[1, 1].hist(numpyarray, bins=80)
    axes[0, 0].set_title("Figure=1")
    axes[0, 1].set_title("Figure=2")
    axes[1, 0].set_title("Figure=3")
    axes[1, 1].set_title("Figure=4")
    plt.show()


hist()
```

When we put i_1=6 ,i_2=10 and p_1, p_2=0.5



When we put i_1=-2 ,i_2=5 and $p_1 = 0.9, p_2 = 0.5$


x axis= Expectation , y axis= frequency
Figure=1
Figure=2
Figure=3
Figure=4
300
200
100
150
50
0
0.0
0.1
0.2
0.3

2 Simulating Distribution

Look at the following algebra and examine the accompanying code.
Let X follow a uniform distribution between 0 and 1. The probability that X is less than some number, x, is $P(X < x) = x$. Suppose we want Y to follow a random distribution for which we do not have any in built functions. Let Y follow the distribution $f_Y(y) = e^{-y}$ for $y \geq 0$. The following trick is used to derive the relation between X and Y.

$$P(Y < y) = P(X < x)$$

$$\int_0^y e^{-y}\, dy = x$$

$$1 - e^{-y} = x$$

$$y = -\ln(1 - x)$$

```
1  y = []
2
3  for i in range(100000):
4      x = np.random.random()
5      y.append(- np.log(1-x))
6
7  bins = 20
8  binWidth = (max(y) - min(y)) / bins
9  plt.hist(y, bins=bins, weights=np.ones(len(y))/(len(y)*binWidth))
10 values = np.linspace(min(y), max(y), 50)
11 plt.plot(values, np.exp(-values))
12 plt.show()
```

2.1

Does the code accomplish simulating the distribution? Which distribution does it follow? Try running the code with different number of bins. Attach plots and discuss your results.

Solution:
Yes, the above code accomplishes the distribution.
Reason: Given that X is a **Uniform Distribution** between 0 and 1 such that $P(X < x) = x$. A random number generator generates a (pseudo) Random value from the standard uniform distribution [1]. So to generate a uniform distribution between 0 and 1, we generate a random number x (see Line 4).
Since we have found that the relation between x and y is :

$$y = -ln(1 - x)$$

Line 5 assure that list y stores all the values of y.
Line 9 plots the histogram which represents the distribution (the blue plots in the diagram)...
On Line 10, $values$ consists of 50 evenly spaced values of y with $min(y)$ as starting point and $max(y)$ as endpoint (2).
Finally, on Line 11, it plots $values$ on the x axis and their corresponding values, obtained from exponential function, on y-axis such that if $y_1 \in values$, it is plotted on x axis and e^{-y_1} plots on y axis.

So we have successfully plotted the distribution e^{-y} for $y \geq 0$ as shown by the red curve.
Note that $y \geq 0$ is assured since the relationship between x and y is derived after applying the lower limit and upper limit as 0 and y respectively.

$$
\begin{gathered}
\int_0^y e^{-y}\, dy = x \\
-\left[e^{-y}\right]_0^y = x \\
-(e^{-y} - e^{-0}) = x \\
-e^{-y} + e^{-0} = x \\
1 - e^{-y} = x
\end{gathered}
$$

Now, let us also discuss the results of running the code with different number of bins.

```
import numpy as np
from matplotlib import pyplot as plt
y = []

for i in range (100000) :
    x = np.random.random()
    y.append(- np.log(1-x))

bins = 10
binWidth = (max(y) - min(y)) / bins
plt.hist (y, bins =bins , weights =np.ones (len (y))/( len (y)* binWidth ))
values = np.linspace ( min(y), max(y), 50)
plt.plot(values , np.exp (- values ))
plt.show ()
```

Figure 1: Code given in Figure 1 with bins = 10

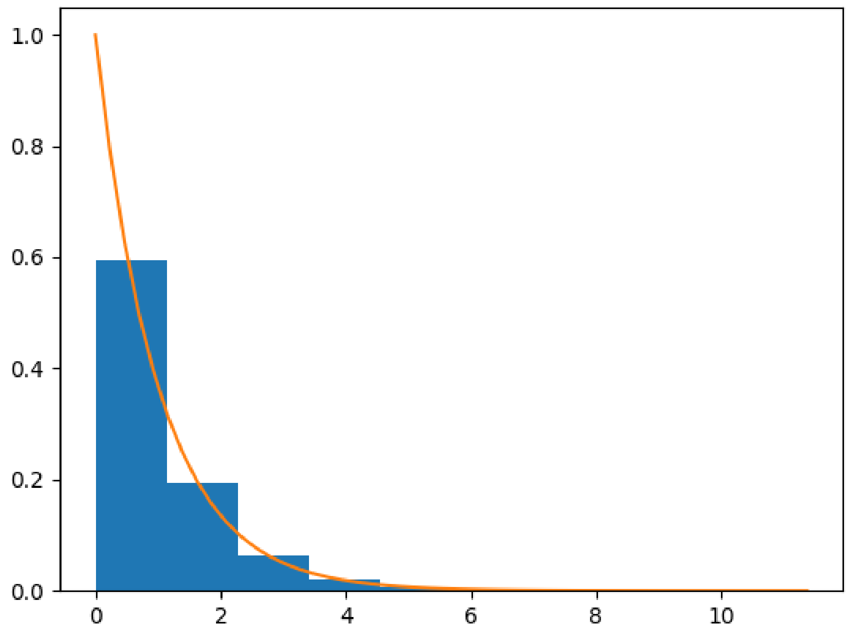


Figure 2: Results obtained from code in figure 2 i.e., bins = 10

```python
import numpy as np
from matplotlib import pyplot as plt
y = []

for i in range (100000) :
    x =np.random.random()
    y.append(- np.log(1-x))

bins = 20
binWidth = (max(y) - min(y)) / bins
plt.hist (y, bins =bins , weights =np.ones (len (y))/( len (y)* binWidth ))
values = np.linspace ( min(y), max(y), 50)
plt.plot(values , np.exp (- values ))
plt.show ()
```

Figure 3: Code given in Figure 1 with bins = 20

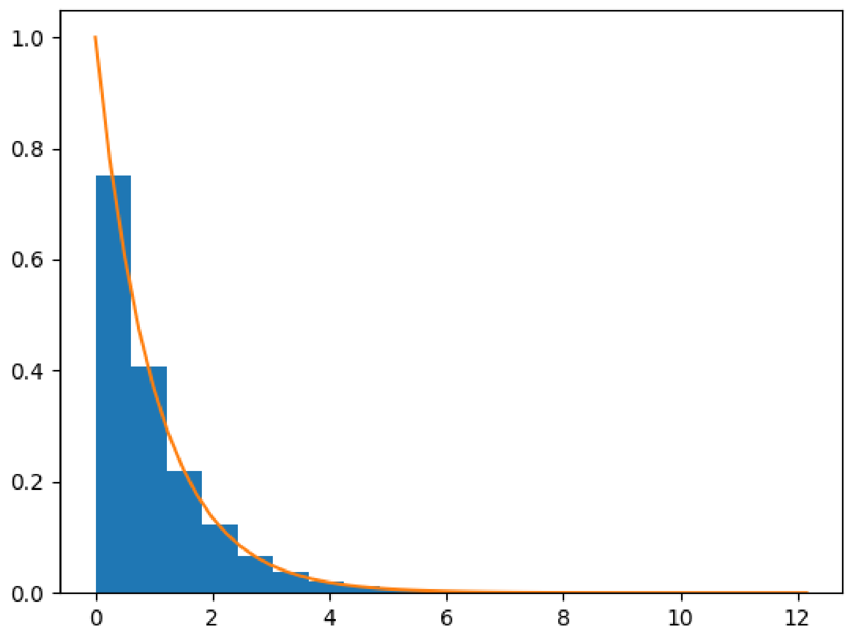


Figure 4: Results obtained from code in figure 4 i.e., bins = 20

```python
import numpy as np
from matplotlib import pyplot as plt
y = []

for i in range (100000) :
    x =np.random.random()
    y.append(- np.log(1-x))

bins = 30
binWidth = (max(y) - min(y)) / bins
plt.hist (y, bins =bins , weights =np.ones (len (y))/( len (y)* binWidth ))
values = np.linspace ( min(y), max(y), 50)
plt.plot(values , np.exp (- values ))
plt.show ()
```

Figure 5: Code given in Figure 1 with bins = 30

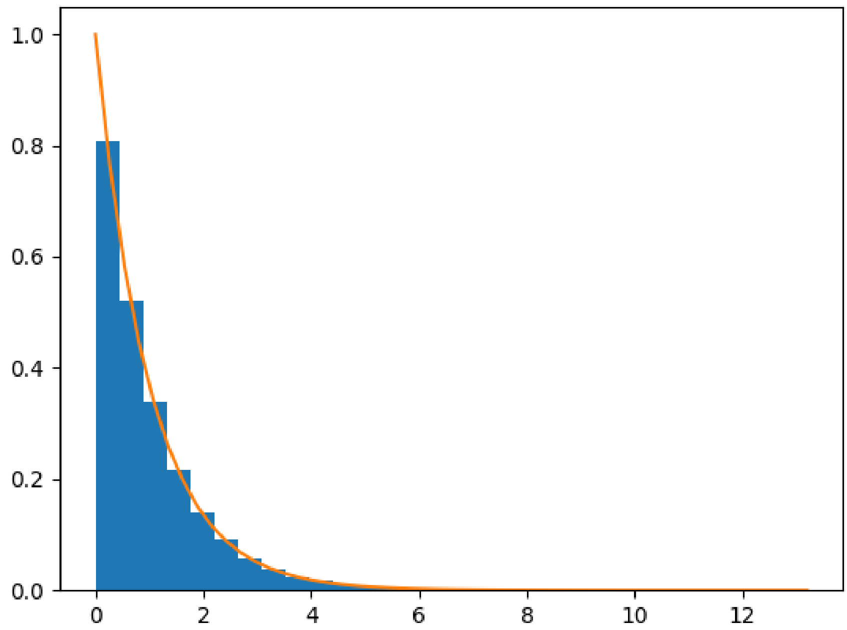


Figure 6: Results obtained from code in figure 6 i.e., bins = 30

```
import numpy as np
from matplotlib import pyplot as plt
y = []

for i in range (100000) :
    x =np.random.random()
    y.append(- np.log(1-x))

bins = 50
binWidth = (max(y) - min(y)) / bins
plt.hist (y, bins =bins , weights =np.ones (len (y))/( len (y)* binWidth ))
values = np.linspace ( min(y), max(y), 50)
plt.plot(values , np.exp (- values ))
plt.show ()
```

Figure 7: Code given in Figure 1 with bins = 50

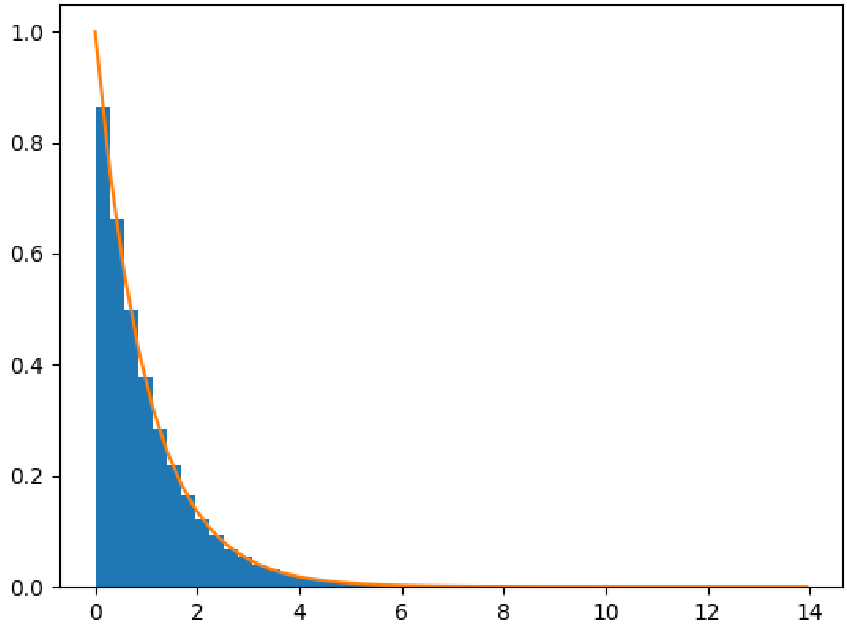


Figure 8: Results obtained from code in figure 8 i.e., bins = 50

```python
import numpy as np
from matplotlib import pyplot as plt
y = []

for i in range (100000) :
    x =np.random.random()
    y.append(- np.log(1-x))

bins = 100
binWidth = (max(y) - min(y)) / bins
plt.hist (y, bins =bins , weights =np.ones (len (y))/( len (y)* binWidth ))
values = np.linspace ( min(y), max(y), 50)
plt.plot(values , np.exp (- values ))
plt.show ()
```

Figure 9: Code given in Figure 1 with bins = 100

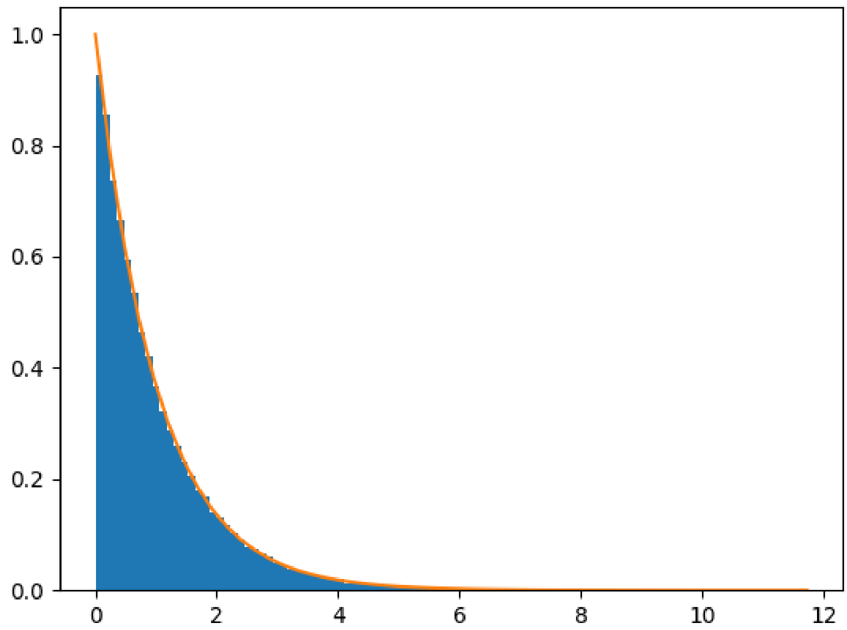


Figure 10: Results obtained from code in figure 10 i.e., bins = 100

```
import numpy as np
from matplotlib import pyplot as plt
y = []

for i in range (100000) :
    x =np.random.random()
    y.append(- np.log(1-x))

bins = 1000
binWidth = (max(y) - min(y)) / bins
plt.hist (y, bins =bins , weights =np.ones (len (y))/( len (y)* binWidth ))
values = np.linspace ( min(y), max(y), 50)
plt.plot(values , np.exp (- values ))
plt.show ()
```

Figure 11: Code given in Figure 1 with bins = 1000

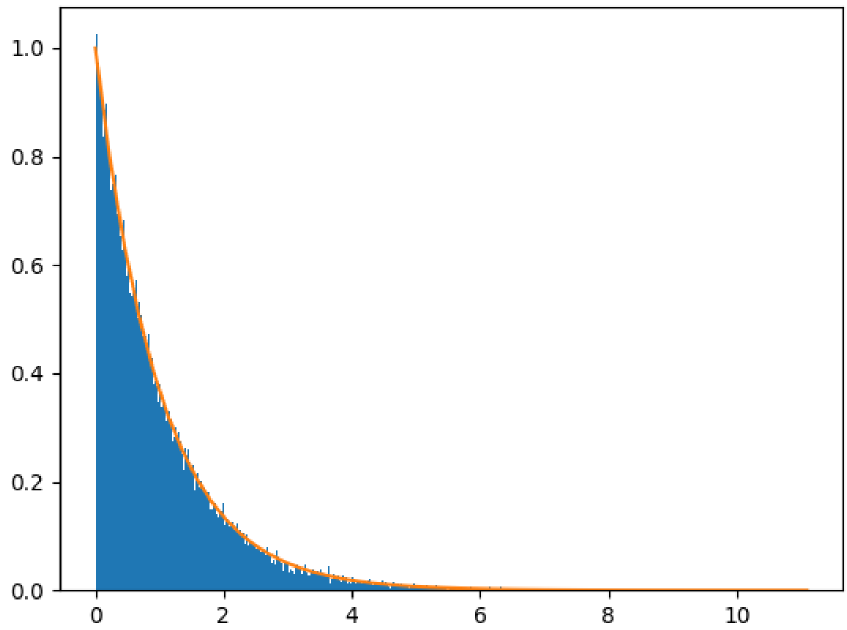


Figure 12: Results obtained from code in figure 12 i.e., bins = 1000

Finding the type of distibution:

We know that a random variable Z is said to have an exponential distribution with parameter λ ($\lambda > 0$) if the PDF of z is :

$$f_Z(z) = \begin{Bmatrix} \lambda e^{-\lambda z} & \text{for } z \geq 0 \\ 0 & \text{otherwise} \end{Bmatrix}$$

By comparing the definition with $f_Y(y) = e^{-y}$ for $y \geq 0$ we can conclude that it is an **Exponential Distribution** with $\lambda = 1$ and Y is an Exponential Random Variable,

Results:

First, let us try to understand what are bins. A histogram displays numerical data by grouping data into "bins" of equal width. Each bin is plotted as a bar whose height corresponds to how many data points are in that bin [3].

Now, consider the following:

$x \in [0, 1]$

$y \in [-ln(1), -ln(0)]$ Apply limit instead of 0

$y \in [0, \infty)$

So, the continuous Random variable Y takes in $y \in [0, \infty)$ [6].

Since our range is now from $[0, \infty)$ it is difficult to plot 100000 possibly different values on histogram so we regularize it and divide it into bins. We divide our range into discrete number of intervals and we count how many of our samples are in each of these discrete ranges [7]. This implies that, increasing the number of bins draws more bars in the histogram and makes it more precise [5].

Notice that the results show that as the number of bins increases, the height of the histogram that was plotted using uniform random variable X (and the relation between x and y) , shown in blue color, traces the plot of exponential random variable, as shown in red color. This shows that we can simulate different distributions by mapping from the uniform distribution. So, yes the code accomplishes the distribution.

2.2

Examine the following section of code and mathematically deduce what distribution Y follows (Try working the above trick in reverse starting with the last statement). Show all required working.

```
1 y = []
2 for i in range(100000):
3     x = np.random.random()
4     y.append(1 / (1-x))
5 y.sort()
6 ind = (np.array(y) > 30).tolist().index(1)
7 y = y[:ind]
8 bins = 100
9 binWidth = (max(y) - min(y)) / bins
10 plt.hist(y, bins=bins, weights=np.ones(len(y))/(len(y)*binWidth))
11 values = np.linspace(min(y), max(y), 50)
12 plt.plot(values, (1 / values ** 2))
13 plt.show()
```

Why are the lines 5-7 important. What does removing them do?

Let us first deduce the distribution that Y follows. Let X follow a uniform distribution between 0 and 1. The probability that X is less than some number,x, is $P(X < x) = x$. From Figure 14 Line 4 we can find the relation between x and y:

$y = \frac{1}{1-x}$
$1 - x = \frac{1}{y}$
$x = 1 - \frac{1}{y}$
$x \in [0, 1]$ and $y \in [1, y]$
$x = \int_1^y \frac{d}{dy}(1 - \frac{1}{y})\ dy$
$x = \int_1^y -(\frac{d}{dy}y^{-1})\ dy$
$x = \int_1^y (-1)(-1)y^{-2}\ dy$
$x = \int_1^y \frac{1}{y^2}\ dy$
So, Y follows the distribution $f_Y(y) = e^{-y}$ for $y \geq 1$.
Let us now understand why Line 5-7 are important.
Line 5 sorts the list the list of y in ascending order.
For example: If $y = [5, 4, 80, 1]$ then after $y, sort()$, $y = [1, 4, 5, 80]$
Line 6 stores the index of smallest element (in the sorted list of y) that is greater than 30 in variable ind. For example: If $y = [1, 2, 40, 80]$ so $ind = 2$
Line 7 updates y such that sorted y is sliced up till the smallest element in y that is less than or equal to 30. For Example: Just before executing Line 7 $y = [1, 4, 5, 30, 80]$ and after Line 7 $y = [1, 4, 5, 30]$
Its importance lies in the fact that $y = \frac{1}{1-x}$
So, as $x \to 1$, $y \to \infty$
So, if we remove Line 5 - 7 , the maximum value of y, can be infinitely large.
$max(y) \to \infty$
The range of y has increased just because of at least a single large outcome of y. A zoomed out version of the graph would be obtained. As a matter of fact, the declining slope will look almost flat since the range of y has increased and needs to be accommodated in the graph. In fact, the histogram seems to have been disappeared due to this reason!

```
1  import numpy as np
2  from matplotlib import pyplot as plt
3
4  y = []
5  for i in range (100000) :
6      x = np.random.random()
7      y.append (1  / (1-x))
8
9  y.sort()
10 ind = (np.array (y) > 30).tolist().index(1)
11 y = y[: ind]
12
13 bins = 100
14 binWidth = (max(y) - min(y)) / bins
15 plt.hist(y, bins =bins , weights =np. ones (len (y))/( len (y)* binWidth ))
16 values = np. linspace ( min(y), max(y), 50)
17 plt . plot (values , (1 / values ** 2))
18 plt.show ()
```

Figure 13: Code with Line 5-7

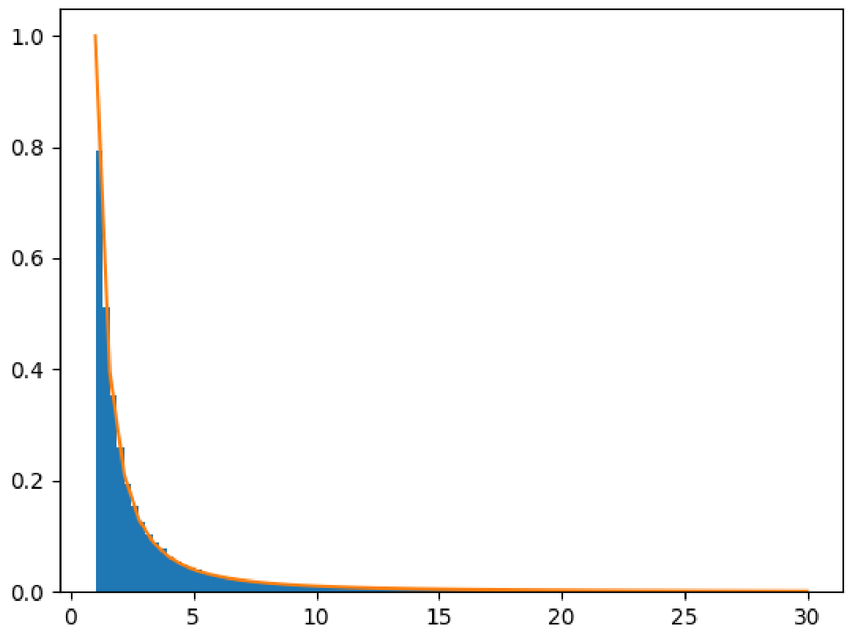


Figure 14: Graph with Line 5 - 7

```python
1  import numpy as np
2  from matplotlib import pyplot as plt
3
4  y = []
5  for i in range (100000) :
6      x = np.random.random()
7      y.append (1  / (1-x))
8
9  bins = 100
10 binWidth = (max(y) - min(y)) / bins
11 plt.hist(y, bins =bins , weights =np. ones (len (y))/( len (y)* binWidth ))
12 values = np. linspace ( min(y), max(y), 50)
13 plt . plot (values , (1 / values ** 2))
14 plt.show ()
```

Figure 15: Code without Line 5-7

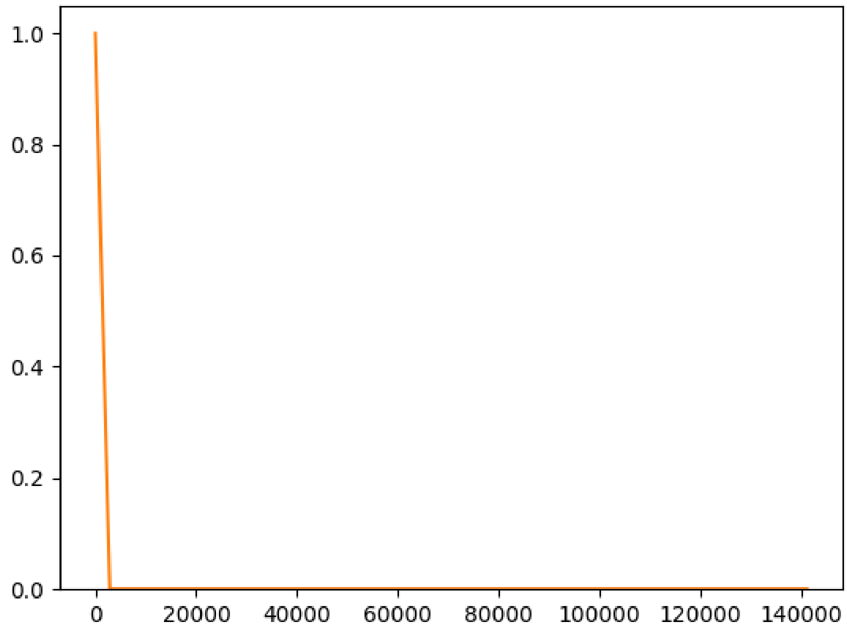


Figure 16: Graph without Line 5 - 7

So, when Line 5 - 7 are removed the distribution curve does not seem to satisfy the appropriate distribution function from which it is calculated. Line 5 - 7 restricts the possible range of values of y so that the histogram gives a local view of the Probability Distribution Function that justifies $f_Y(y) = e^{-y}$ for $y \geq 0$ (OR 1)

2.3

Implement a function that returns a random variable from the distribution,

$$f_Y(y) = \frac{1}{y^3} \text{ for } y \geq \sqrt{\frac{1}{2}}$$

Use it to produce a histogram and line plot like the above code.
Implement a different function that calculates the expected value using the experiments and iterations approach and plots the set of expected values obtained. You may need to utilize the trick pointed to in the above lines and choose an appropriate cutoff for both of these.

First let us implement a function that returns a random variable from the distribution,
$f_Y(y) = \frac{1}{y^3}$ for $y \geq \sqrt{\frac{1}{2}}$
Let X be a Random Variable that follows a uniform distribution between 0 and 1. The probability that X is less than some number, x, is $P(X < x) = x$.
Using the same trick as in 2.1 to find the relation between X and Y
$P(Y < y) = P(X < x)$
$\int_{\sqrt{0.5}}^{y} \frac{1}{y^3}\, dy = x$
$\int_{\sqrt{0.5}}^{y} y^{-3}\, dy = x$
$\left[\frac{y^{-3+1}}{-3+1}\right|_{\sqrt{0.5}}^{y} = x$
$-\frac{1}{2}\left[y^{-2}\right|_{\sqrt{0.5}}^{y} = x$
$-\frac{1}{2}((y^{-2}) - (\sqrt{\frac{1}{2}})^{-2}) = x$
$-\frac{1}{2}(\frac{1}{y^2} - 2) = x$
$\frac{-1}{2y^2} + \frac{2}{2} = x$
$1 - \frac{1}{2y^2} = x$
$\frac{1}{2y^2} = 1 - x$
$2y^2 = \frac{1}{1-x}$
$y^2 = \frac{1}{2(1-x)}$
$y = \sqrt{\frac{1}{2(1-x)}}$

Following is the code used to produce histogram and line plot like the previous parts:

```
1   import numpy as np
2   from matplotlib import pyplot as plt
3
4   y = []
5   for i in range (10000) :
6       x = np.random.random ()
7       y.append ((1  / (2 * (1-x)))** (1/2))
8
9   ####### Since y can have an infinitely large value #############
10  y.sort()
11  ind = (np.array (y) > 30).tolist().index(1)
12  y = y[: ind]
13  ################################################################
14
15  bins = 100
16  binWidth = (max(y) - min(y)) / bins
17  plt.hist(y, bins =bins , weights =np. ones (len (y))/( len (y)* binWidth ))
18  values = np. linspace ( min(y), max(y), 50)
19  plt . plot (values , (1 / values ** 3))
20  plt.show ()
```

Figure 17: Code used to produce histogram and line plot for question 2.3

Notice that Line 10 - 12 are important for the same reason as Line 5 - 7 were in question 2.2. Since $y \geq \sqrt{\frac{1}{2}}$ in $f_Y(y)$, therefore, the list y in the code above can have some values that are too large. The presence of even one such value makes it difficult to visualize the histogram as per the function provided. So, we have eliminated such large values. We are restricting the maximum possible value of y such that $y \leq 30$. This is the reason why the x-axis in the histogram above graphs values from 0 to 30

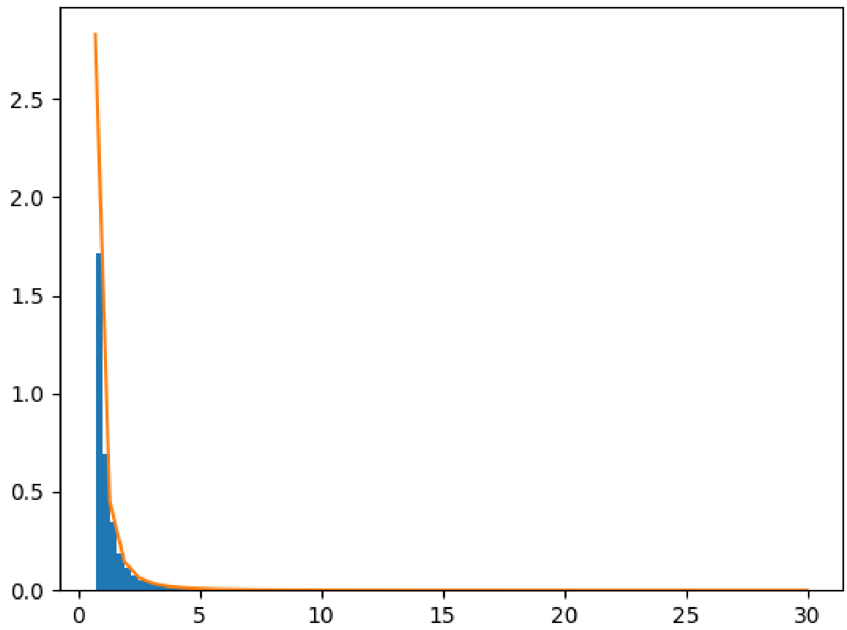


Figure 18: Histogram obtained from the distribution $f_Y(y) = \frac{1}{y^3}$ for $y \geq \sqrt{0.5}$

Finding Expected Value:
We know that expected value is basically the average value of a random variable.

```
import numpy as np
from matplotlib import pyplot as plt
import math
def average():
    # Return the average / expected value of y by iterating over 1000 times
    y = []
    for i in range (10000) :        # Fins 10000 values of y using x
        x = np.random.random()
        y.append ((1  / (2 * (1-x)))** (1/2))
    return sum(y) / (len(y) + 1)    #Returns average y

def draw_histogram():               # Draws histogram
    expected_values = []            # Finds 500 average / expected values of y
    for i in range(500):
        expected_values.append(average())
    bins = 50                       # draw the histogram
    binWidth = (max(expected_values) - min(expected_values)) / bins
    plt.hist(expected_values, bins =bins,weights =np.ones(len (expected_values))/(len(expected_values)* binWidth))
    values = np.linspace ( min(expected_values), max(expected_values), 50) # draw the line plot
    plt.plot(values , np.exp (- values ))
    plt.show ()
draw_histogram()
```

Figure 19: Code used to find the expected value of $f_Y(y) = \frac{1}{y^3}$ for $y \geq \sqrt{0.5}$

This produced the following histogram:

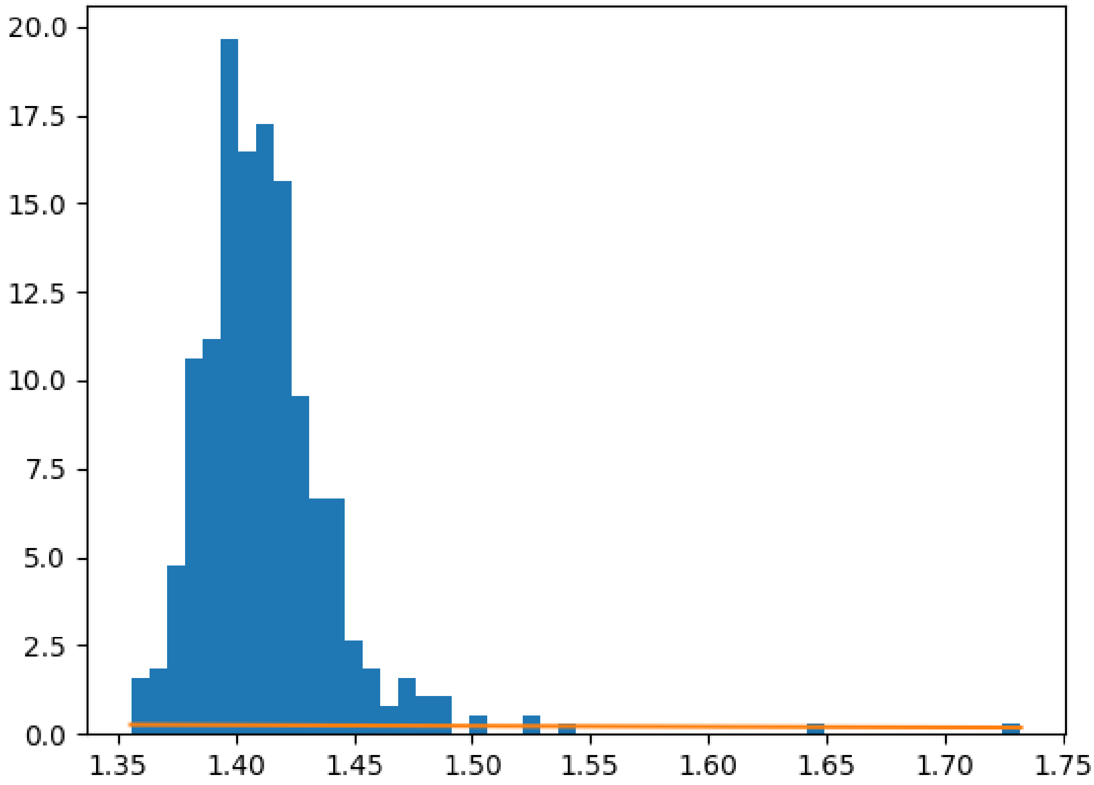


Figure 20: Histogram of expected value of $f_Y(y) = \frac{1}{y^3}$ for $y \geq \sqrt{0.5}$ using experiments and iterations

Let us verify if we have obtained the correct results:
$E[Y] = \int_{\sqrt{0.5}}^{\infty} y \, \frac{1}{y^3} \, dy$
$E[Y] = \int_{\sqrt{0.5}}^{\infty} \frac{1}{y^2} \, dy$
$E[Y] = \left[\frac{Y^{-1}}{-1} \right|_{\sqrt{0.5}}^{\infty}$
$E[Y] = \lim_{y \to \infty} \frac{-1}{y} - (-\frac{1}{\sqrt{0.5}})$
$E[Y] = 0 + \sqrt{2}$ $E[Y] = \sqrt{2}$

Notice that the center of the histogram that has the highest frequency is approximately 1.4. This shows that our histogram is displaying correct results

3 Picking a random point correctly

3.1

For this question, you have to pick random points in a circle in a uniform manner. The most intuitive approach for this is usually to pick a random number, r, from the uniform distribution between 0 and R, where R is the radius of the circle. Similarly, one can pick the angle θ in a similar manner and generate x, y coordinates from them.

Implement a function that takes in a radius, R, and samples a large number of points in the described manner. The function should generate a scatter plot containing all the sampled points, as well as plotting a circle of the appropriate radius, Find and mention the variation in the x-coordinates as well.

Solution: Following is our code. Note that since the question has not mentioned to keep the center of the circle into consideration we have assumed the center of the circle to be at origin. i.e., (0, 0)

```python
import numpy as np
import math
import matplotlib.pyplot as plt
import numpy as np

def UniformPoint_1(R):
    """
    Finds x , y coordinate uniformly on circle with radius R with center (0, 0)
    Args:
    - R: Radius of the circle
    Returns:
    - (x, y): x and y coordinates by uniform distribution on circle with radius R and center (0, 0)
    """
    r = np.random.uniform(0, R)                    # Uniform distribution between 0 and R
    theta = np.random.uniform(0, 2 * math.pi)      # Uniform distribution between 0 and 2 pi
    x = r * math.cos(theta)                        # Find x coordinate
    y = r * math.sin(theta)                        # Find y coordinate
    return x , y

def plot(R):
    """
    Plots the circle with radius R and draws the scatter plot using uniformly distributed x and y
    Prints the variance of x coordinates
    Args: None
    Ref:
    - https://moonbooks.org/Articles/How-to-plot-a-circle-in-python-using-matplotlib-/
    Returns:
    None
    """
    # Draw the circle of radius 1
    x = np.linspace(-R - 0.5, R + 0.5)
    y = np.linspace(-R - 0.5, R + 0.5)
    X, Y = np.meshgrid(x,y)
    F = X**2 + Y**2 - R ** 2
    fig, ax = plt.subplots()
    ax.contour(X,Y,F,[0])
    ax.set_aspect(1)

    # Scatter plot
    y_list, x_list = [] , []
    for i in range(1000):                              # Find x, y 1000 timees
        x, y = UniformPoint_1(R)
        y_list.append(y)
        x_list.append(x)
    plt.scatter(x_list, y_list, s = 5)                # Plot in scatter plot

    print("Variance is : " , np.var(x_list))          # print the Variance
    plt.show()

def main():
    R = 1  # Radius of the circle
    plot(R)
main()
```

Figure 21: Code for 3.1

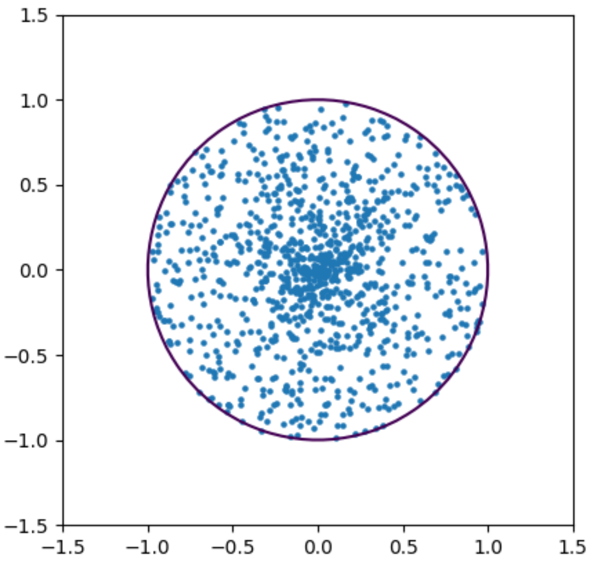


Figure 22: Scattered plot obtained using strategy mentioned in 3.1 on a circle with radius 1 centered at origin

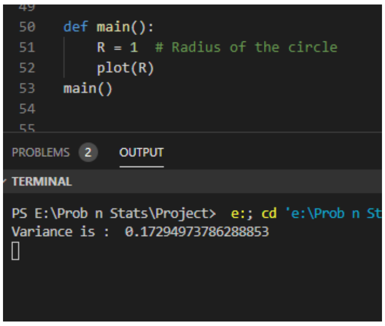


Figure 23: variance of x-coordinates obtained using strategy mentioned in 3.1 on a circle with radius 1 centered at origin

So, the variation in the x-coordinates is 0.17294973786288853

3.2

This, however, does not result in a uniform pick. You may spot this from the plot which should have points concentrated more towards the center rather then points being uniformly spread out across the circle. Change the number points you are plotting if you do not observe this trend. Now instead of generating r and θ values we will generate x and y values uniformly. To generate random points on a circle of radius, R, pick both x and y independently and uniformly from the range [-R, R] to obtain a point. If the distance of this point from the origin is more than R, discard it and generate a new point in its place.

Implement a function that takes in a radius, R, and samples a large number of points in the described manner. The function should generate a scatter plot containing all the sampled points, as well as plotting a circle of the appropriate radius, Find and mention the variation in the x-coordinates as well. Comment on why this found variation is different or same as in the previous part.

```python
import numpy as np
import math
import matplotlib.pyplot as plt
import numpy as np

def UniformPoint_2(R):
    """
    Finds x , y coordinate uniformly on circle with radius R with center (0, 0)
    Args:
    - R: Radius of the circle
    Returns:
    - (x, y): x and y coordinates by uniform distribution on circle with radius R and center (0, 0)
    """
    x = np.random.uniform(-R, R)
    y = np.random.uniform(-R, R)

    distance_from_origin = math.sqrt((x ** 2) + (y ** 2))
    if distance_from_origin > R or distance_from_origin < - R:
        return UniformPoint_2(R) # discard and generate again
    else:
        return x , y


def plot(R):
    """
    Plots the circle with radius R and draws the scatter plot using uniformly distributed x and y
    Prints the variance of x coordinates
    Args: None
    Ref:
    - https://moonbooks.org/Articles/How-to-plot-a-circle-in-python-using-matplotlib-/
    Returns:
    None
    """
    # Draw the circle of radius 1
    x = np.linspace(-R - 0.5, R + 0.5)
    y = np.linspace(-R - 0.5, R + 0.5)
    X, Y = np.meshgrid(x,y)
    F = X**2 + Y**2 - R ** 2
    fig, ax = plt.subplots()
    ax.contour(X,Y,F,[0])
    ax.set_aspect(1)

    # Scatter plot
    y_list, x_list = [] , []
    for i in range(1000):                             # Find x, y 1000 timees
        x, y = UniformPoint_2(R)
        y_list.append(y)
        x_list.append(x)
    plt.scatter(x_list, y_list, s = 5)               # Plot in scatter plot

    print("Variance is : " , np.var(x_list))        # print the Variance
    plt.show()

def main():
    R = 1  # Radius of the circle
    plot(R)
main()
```

Figure 24: Code for 3.2

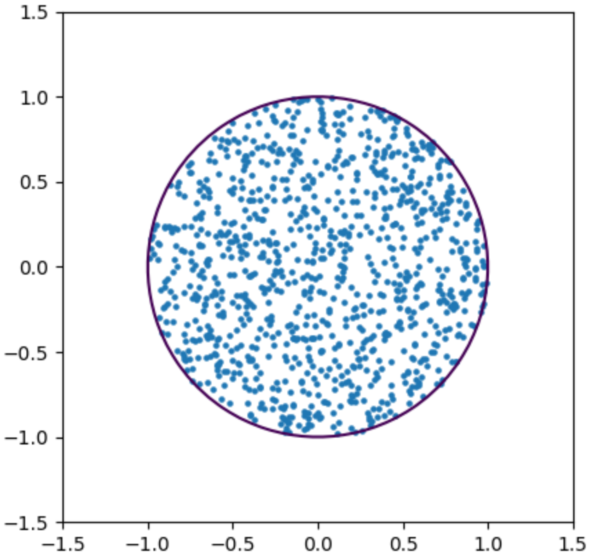


Figure 25: Scattered Plot obtained using strategy mentioned in 3.2 on a circle with radius 1 centered at origin

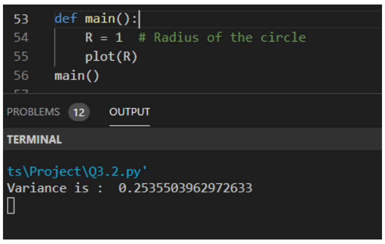


Figure 26: variance of x-coordinates obtained using strategy mentioned in 3.2 on a circle with radius 1 centered at origin

So, the variation in the x-coordinates is 0.2535503962972633

Comment:
Notice that the x coordinates' variation obtained is approximately 0.25 which is approximately 0.42 more than the variance obtained in previous approach i.e., approach used in part 3.1 (whose x coordinates' variance approximately was 0.17) although both are obtained on a circle with radius 1 centered at origin. Question arises why is that so?
In the first approach, points obtained are not equally spaced with respect to the center The points are the center of the circle are more dense. as we go away from the center, the density of points goes on decreasing. So this implies that the randomly generated point is more likely to be near the center than as compared to near the circumference. The points closer to the center has more probability of generation and as we go further and further away from the center this probability goes on decreasing. Thus the points obtained in 3.1 were not Uniformly distributed. However, when we picked points by a uniform random distribution of in terms of x and y coordinates the resulting points were Uniform as each point is equally likely to be obtained. Note that we are discarding a point (x, y) if it is outside the circle implying that the points to be obtained are to be limited to the area under consideration i.e., circle. The variance has increased. i.e., the degree of the spread of the data has increased increased since this is a more uniformly distributed plot

3.3

To get an intuition of why the first approach does not result in a uniform pick imagine a circle of radius 1 embedded in a circle of radius 2 as shown in Fig.2. If points are picked randomly, the probability of the point

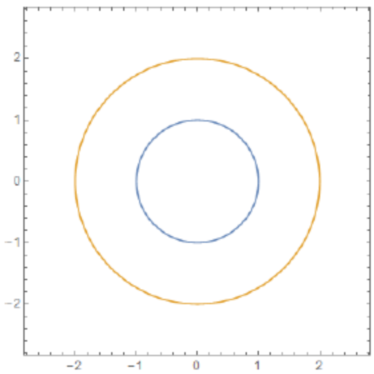


Figure 27: Comparison of area. Circles of radius 1 and 2

lying inside the larger circle should be 4 times than the smaller one. Does this hold when the above described method to pick r and θ is used? Explain in report with working.
In this part, modify one or both of the ways to pick r and θ such that the points are sampled in a uniform manner and a plot similar to that in part 2 is obtained. Implement a similar function as the above part. The plot generated this time should contain points that are uniformly spread across the circle. Describe how are you picking the random variables and find the variance of the x-coordinates once again and comment on your results.
If you feeling up to it or for a bonus then you may derive the distribution from the following two facts. The probability of a point to lie inside a circle of radius, $r \leq R$ is proportional to its area. i.e. $P(r \leq R) = k\,\pi r^2$. The probability of it lying inside the outermost circle of radius, R should be 1 i.e. $P(R \leq R) = 1$.
After finding the distribution that r follows, you may then generate the values of r appropriately by mapping from the uniform random distribution as in the previous questions. Show all mathematical working.

Solution:
Yes, if the above described method of r and θ are used then the probability of a randomly picked point to lie in the outside circle is 4 times more than the probability of the point to lie in the smaller circle (blue one). To understand this we know that the radius of outside circle is 2. Let this radius be r. Consequently, the radius of the inside circle is $\frac{r}{2}$. Let A be the event of a randomly picked point (picked using randomly generated r and θ as described in first approach) to lie inside/on the blue circle. Let B be the event of a randomly picked point (picked using randomly generated r and θ as described in first approach) lie in the orange region.

$P(A) = \frac{\pi\,(\frac{r}{2})^2}{\pi\,r^2}$
$P(A) = \frac{\pi\,\frac{r^2}{4}}{\pi\,r^2}$
$P(A) = \frac{1}{4} = 0.25$

Note that since the outer circle includes the orange as well as blue region , $P(A)$ tells us that the probability of the point lying inside the larger circle is 4 times than the smaller one.

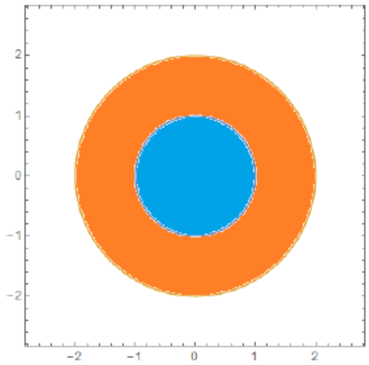


Figure 28: Comparison of area. Circles of radius 1 and 2

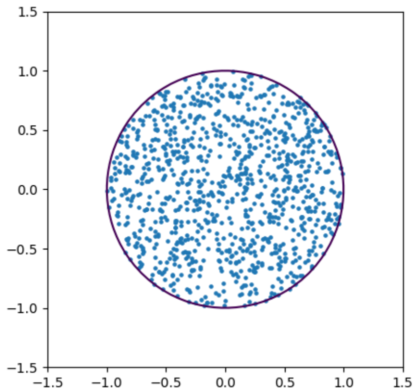
Thus it is proved that randomly generated r and θ does not result in a uniform probability

Given that $P(r \leq R) = k\,\pi r^2$
We want it to be uniformly distributed.So we will map it to uniformly selected x such that $x \in [0, 1]$
$P(r \leq R) = k\,\pi r^2 = x$
$k\,\pi r^2 = x$
$r^2 = \frac{x}{k\pi}$
$r = \sqrt{\frac{x}{k\pi}}$................(i)
When $r = R$ it is given that $P(R \leq R) = 1$
$k\pi(R^2) = 1$
$k\pi = \frac{1}{R^2}$
Put in equation (i):
$r = \sqrt{\frac{x}{\frac{1}{R^2}}}$
$r = \sqrt{R^2 x}$
$r = R\sqrt{x}$

```python
import numpy as np
import math
import matplotlib.pyplot as plt
import numpy as np

def UniformPoint_3(R):
    """
    Finds x , y coordinate uniformly on circle with radius R with center (0, 0)
    Args:
    - R: Radius of the circle
    Ref:
    - https://stackoverflow.com/questions/5837572/generate-a-random-point-within-a-circle-uniformly
    Returns:
    - (x, y): x and y coordinates by uniform distribution on circle with radius R and center (0, 0)
    """
    r = R * math.sqrt(np.random.uniform(0, 1))
    theta = np.random.uniform(0, 1) * 2 * math.pi
    x = r * math.cos(theta)
    y = r * math.sin(theta)
    return x , y

```

```python
def plot(R):
    """
    Plots the circle with radius R and draws the scatter plot using uniformly distributed x and y
    Prints the variance of x coordinates
    Args: None
    Ref:
    - https://moonbooks.org/Articles/How-to-plot-a-circle-in-python-using-matplotlib-/
    Returns:
    None
    """
    # Draw the circle of radius 1
    x = np.linspace(-R - 0.5, R + 0.5)
    y = np.linspace(-R - 0.5, R + 0.5)
    X, Y = np.meshgrid(x,y)
    F = X**2 + Y**2 - R ** 2
    fig, ax = plt.subplots()
    ax.contour(X,Y,F,[0])
    ax.set_aspect(1)

    # Scatter plot
    y_list, x_list = [] , []
    for i in range(1000):                              # Find x, y 1000 timees
        x, y = UniformPoint_3(R)
        y_list.append(y)
        x_list.append(x)
    plt.scatter(x_list, y_list, s = 5)               # Plot in scatter plot

    print("Variance is : " , np.var(x_list))        # print the Variance
    plt.show()

def main():
    R = 1  # Radius of the circle
    plot(R)
main()
```

Figure 29: Code for 3.3




```
def main():
    R = 1  # Radius of the circle
    plot(R)
main()

PROBLEMS 4   OUTPUT
TERMINAL
------------------------------------------------
PS E:\Prob n Stats\Project>  e:; cd 'e:\Prob n St
Variance is :  0.2597108325345517
```

Figure 31: Variance as per 3.3

4 Saying random is not enough - Approaches effect distributions

In this question we are going to observe the distribution followed by the length of a random chord picked from a circle of radius r. The difficulty of the question lies in how to pick a random chord in a circle. For each of the described approaches implement a different function that takes in radius, r and plots a histogram of the length of chords with an appropriate number of bins, with proportion (probability) of values in the bin on y-axis instead of counts. Include mathematical calculation of chord lengths in all parts.

4.1

For the first approach we imagine the circle centred on the origin of the Cartesian plane. The $\theta = 0$ ray/line is defined as starting at the origin and pointing in the direction of increasing x, and θ increasing counter clockwise. We pick two angles θ_1 and θ_2 uniformly between 0 and 2π, and our random chord is the chord between the points of the circle defined by those two angles.

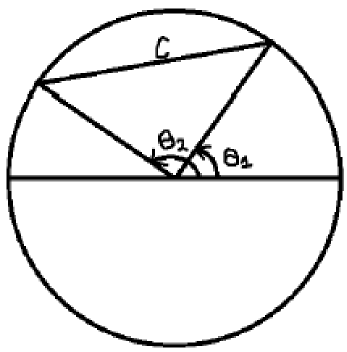


Figure 32: Picking a chord through 2 random angles

Solution: Using a circle centered at origin with radius 1.

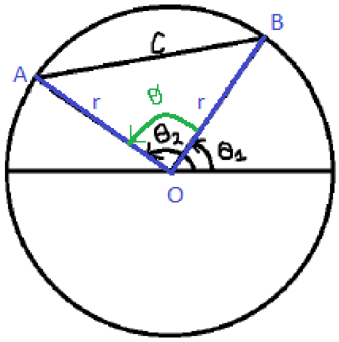


Figure 33: Finding a chord through 2 random angles

Deriving the value of C from θ_1 and θ_2

$$\theta = \theta_2 - \theta_1$$

We know that the distance from the center of the circle till any point on the circumference of the circle is equal to the radius r of the circle. Therefore,

$$\overline{OA} = \overline{OB} = r$$

Observe that $\triangle OBA$ is formed with sides AB, AO and OB, The angle opposite to side AB is θ
According to Law of Cosine:

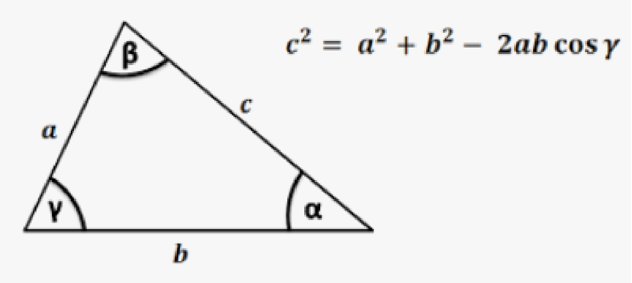


Figure 34: Law of cosine [8]

Using Law of cosine:
$C = \sqrt{r^2 + r^2 - 2(r)(r)(cos\,\theta)}$
$C = \sqrt{2r^2 - 2(r^2)(cos\,\theta)}$
$C = \sqrt{2r^2(1 - cos\,\theta)}$
$C = r\sqrt{2(1 - cos\,\theta)}$
Using double angle formula:
$cos\,2\theta = 1 - 2\,sin^2\,\theta$
$cos\,\theta = 1 - 2\,sin^2\,\frac{\theta}{2}$
$1 - cos\,\theta = 2\,sin^2\,\frac{\theta}{2}$
Plugging it to obtain C:
$C = r\sqrt{2(2\,sin^2\,\frac{\theta}{2})}$
$C = r\sqrt{4sin^2\,\frac{\theta}{2}}$
$C = 2r\,sin\,\frac{\theta}{2}$

```
1  import numpy as np
2  import math
3  import matplotlib.pyplot as plt
4  import numpy as np
5  
6  def chord_4_1(radius):
7      """
8      Stores the lengths of random chords by picking theta1 and theta2 uniformly
9      as described in approach 4.1
10     Args:
11     - Radius: Radius of the circle
12     Reurns:
13     - chords: A list containing integer as length of random chords
14     """
15     chords = []              # List of the length of chords
16     for i in range(100000):
17         # theta1 and theta2 unifromly distributed between 0 and 2 pi
18         theta1 = np.random.uniform(0, 2 * math.pi)
19         theta2 = np.random.uniform(0, 2 * math.pi)
20         if theta2 < theta1:                # theta2 must always be greater than or equal to theta1
21             theta2, theta1 = theta1, theta2
22         theta = theta2 - theta1
23         chords.append(2 * radius * math.sin(theta / 2))
24     return chords
25 
26 def histogram(radius):
27     """
28     Makes the histogram
29     Args:
30     - radius: integer radius of the circle
31     Ref:
32     - https://stackoverflow.com/questions/38650550/cant-get-y-axis-on-matplotlib-histogram-to-display-probabilities
33     Return:
34     none
35     """
36     chords = chord_4_1(radius)
37     bins = 100                              # using bin = 100
38     n, bin_edges = np.histogram(chords, bins)
39     bin_probability = n / (n.sum())
40     binWidth = (max(chords) - min(chords)) / bins
41     bin_middles = (bin_edges[1:] + bin_edges[:-1]) / 2
42     plt.bar(bin_middles, bin_probability, width = binWidth)
43     plt.show()
44 histogram(1)
```

Figure 35: Code for question 4.1

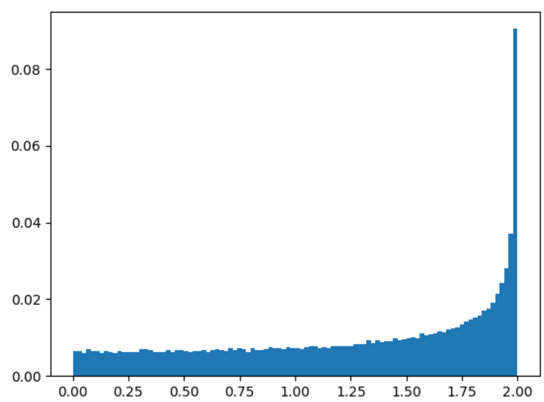


Figure 36: Histogram for part 4.1

Notice that in line 22 - 23, we interchange the angle if $\theta_2 > \theta_1$. This is because we do not want the length of our chord to be negative as length is always positive. This implies that θ_1 is greater than θ_2

4.2

For the second approach we imagine the circle in a similar manner. Then we pick a random direction, θ, and draw a line from the center of the circle to its boundary such that the angle from the ray $\theta = 0$ to this line, measured counter clockwise is θ. To create a random chord, we pick a point along this line and construct the perpendicular bisector of the line at this point. The perpendicular bisector can be extended to touch the boundary of the circle at either ends to obtain a chord.

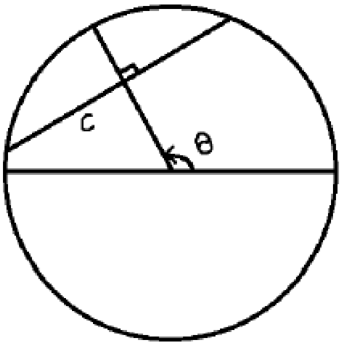


Figure 37: Picking a chord as bisector of some ray

Solution: Using a circle centered at origin with radius 1.

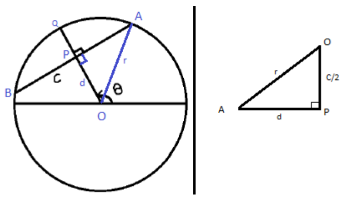


Figure 38: Chord as bisector of some ray

As given in the question, we can **pick a point** on line $\overline{OQ}$. Let this point be called as P. Let d be the distance between O and P such that :
$d \in [0, r]$
$P = (d\cos\theta, d\sin\theta)$

Furthermore,
$\overline{AB} = \overline{BP} + \overline{PA}$
$\overline{AB} = C$
Since $\overline{OP}$ is perpendicular bisector of $\overline{AB}$,
$\overline{BP} = \overline{PA} = \frac{C}{2}$

Since $\overline{OP} \perp \overline{PA}$, $\triangle OPA$ is a right angled triangle
Applying Pythagoras Theorem:
$r^2 = d^2 + \frac{C^2}{4}$
$r^2 - d^2 = \frac{C^2}{4}$
$C^2 = 4(r^2 - d^2)$
$C = \sqrt{4(r^2 - d^2)}$
$C = 2\sqrt{r^2 - d^2}$

```python
import numpy as np
import math
import matplotlib.pyplot as plt
import numpy as np

def CalculateChordLength(r, d):
    """
    Return the length of chord with a perpendicular distance d
    from the center of the circle of radius r.
    Args:
    - r: Radius of the circle
    - d: Perpendicular distance of the chord from the center of the circle
    Returns:
    - Length of the chord
    """
    return 2 * math.sqrt((r ** 2) - (d ** 2))

def chord_4_2(radius):
    """
    Stores the lengths of random chords by picking a point along this line at an angle
    and constructing perpendicular bisector of the line at this point. The perpendicular
    bisector can be extended to obtain the chord as described in approach 4.2
    Args:
    - Radius: Radius of the circle
    Reurns:
    - chords: A list containing integer as length of random chords
    """
    chords = []
    for i in range(100000):                    # List of the length of chords
        d = np.random.uniform(0, radius)      # distance from center to circumference
        chords.append(CalculateChordLength(radius, d))
    return chords

```

```python
def histogram(radius):
    """
    Makes the histogram
    Args:
    - radius: integer radius of the circle
    Ref:
    - https://stackoverflow.com/questions/38650550/cant-get-y-axis-on-matplotlib-histogram-to-display-probabilities
    Return:
    none
    """
    chords = chord_4_2(radius)
    bins = 100                                  # using bin = 100
    n, bin_edges = np.histogram(chords, bins)
    bin_probability = n / (n.sum())
    binWidth = (max(chords) - min(chords)) / bins
    bin_middles = (bin_edges[1:] + bin_edges[:-1]) / 2
    plt.bar(bin_middles, bin_probability, width = binWidth)
    plt.show()
histogram(1)

```

Figure 39: Code for question 4.2

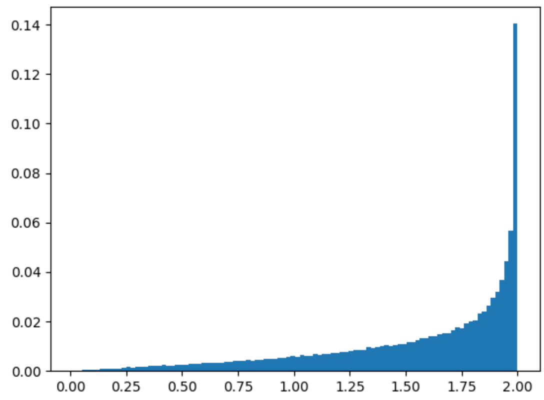
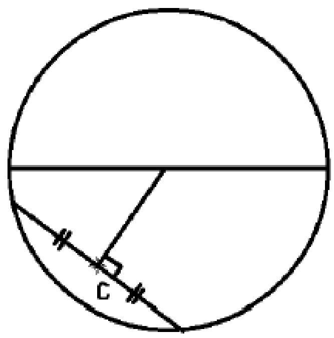


Figure 40: Histogram for question 4.2

4.3

For the third approach we again visualize the circle as before. This time we pick a random point uniformly from the circle as we did in the previous question. You may use any helper functions you may have developed in the previous part for this. After picking a point we find the chord which will have this point as it midpoint and this will be our random chord. There will be only one such chord.



Solution: Using a circle centered at origin with radius 1. Let (x, y) be be a point that we pick uniformly from the circle.
Using Pythagoras Theorem:
$r^2 = (\frac{c}{2})^2 + d^2$
$4r^2 = c^2 + 4d^2$
$C = \sqrt{4r^2 - 4d^2}$
$C = 2\sqrt{r^2 - d^2}$
$C = 2\sqrt{r^2 - x^2 - y^2}$
$C = 2\sqrt{r^2 - (x^2 + y^2)}$
$C = 2\sqrt{r^2 - d^2}$
When d is the perpendicular distance from the center of the circle till point (x, y)

```python
import numpy as np
import math
import matplotlib.pyplot as plt
import numpy as np

def CalculateChordLength(r, d):
    """
    Return the length of chord with a perpendicular distance d
    from the center of the circle of radius r.
    Args:
    - r: Radius of the circle
    - d: Perpendicular distance of the chord from the center of the circle
    Returns:
    - Length of the chord
    """
    return 2 * math.sqrt((r ** 2) - (d ** 2))

def UniformPoint_3(R):
    """
    Finds x , y coordinate uniformly on circle with radius R with center (0, 0)
    Args:
    - R: Radius of the circle
    Ref:
    - https://stackoverflow.com/questions/5837572/generate-a-random-point-within-a-circle-uniformly
    Returns:
    - (x, y): x and y coordinates by uniform distribution on circle with radius R and center (0, 0)
    """
    r = R * math.sqrt(np.random.uniform(0, 1))
    theta = np.random.uniform(0, 1) * 2 * math.pi
    x = r * math.cos(theta)
    y = r * math.sin(theta)
    return x , y
```

```python
def chord_4_3(radius):
    """
    Stores the lengths of random chords by
    Args:
    - Radius: Radius of the circle
    Reurns:
    - chords: A list containing integer as length of random chords
    """
    chords = []
    for i in range(100000):                    # List of the length of chords
        x, y = UniformPoint_3(radius)
        d = (x ** 2 + y ** 2) ** 0.5
        chords.append(CalculateChordLength(radius, d))
    return chords

def histogram(radius):
    """
    Makes the histogram
    Args:
    - radius: integer radius of the circle
    Ref:
    - https://stackoverflow.com/questions/38650550/cant-get-y-axis-on-matplotlib-histogram-to-display-probabilities
    Return:
    none
    """
    chords = chord_4_3(radius)
    bins = 100                                 # using bin = 100
    n, bin_edges = np.histogram(chords, bins)
    bin_probability = n / (n.sum())
    binWidth = (max(chords) - min(chords)) / bins
    bin_middles = (bin_edges[1:] + bin_edges[:-1]) / 2
    plt.bar(bin_middles, bin_probability, width = binWidth)
    plt.show()
histogram(1)
```

Figure 41: Code for question 4.3

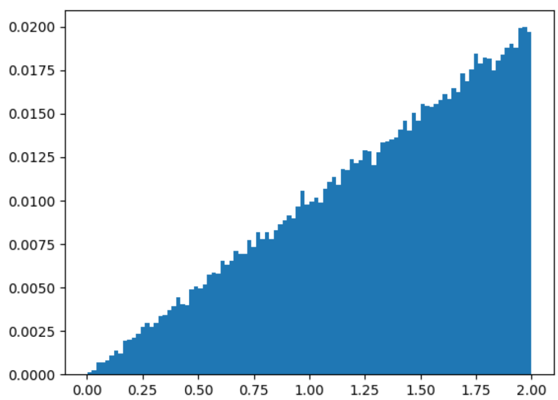


Figure 42: Histogram for question 4.3

4.4

You will notice that all of these approaches results in a different distribution. Which of these do you think corresponds most to our goal, which was to find the distribution of the length of a random chord.

Solution: Firstly, notice that in all three distributions, the maximum length of the chord is 2. This is because we testing a circle of radius 1 in all 3 of them. A circle of radius 1 has diameter 2. So, the maximum length of the chord would be 2. Similarly, the minimum length of the chord would be zero. There is no such chord with length zero. However, the histogram of part 4.1 shows that the distribution of the length of chord is such that there is some probability (greater than zero) for a chord to have zero length. So the strategy in 4.1 is perhaps not corresponding most to our goal.
Now we need to decide which distribution does the length of a chord follow. The distribution of the length of the chord should be such that as the length of chord increases. The probability of the length of chord also increases. Both 4.2 and 4.3 strategies satisfies this condition.
Notice that the (almost) linear graph and the strategy employed in 4.3 implies that there exists only one chord which has a randomly chosen point as its mid point. However, this is not true. Take for example the center of the circle. There must exist infinitely many chords passing through it (i.e., through the same midpoint).So, the histogram must not be linear. By randomly selecting a θ instead of selecting a point, we remove this problem. Thus, it is established that 4.2 corresponds most to our goal, which was to find the distribution of the length of a random chord.

5 Hypothesis Testing

Intuition - If someone hands you a coin, and tells you its fair, you toss it 15 times and get 15 heads, you are going to be skeptical. That is the essence of hypothesis testing, we make a certain assumption, and then sample some data. If the sum of probability of obtaining the observed data or data less or equally likely is less than a certain threshold then we conclude our assumption to be false. The statement 'or data less likely' is a little vague and more importantly problem dependent. Let use look at a concrete example.

Suppose you have a coin which we do not know as fair or not. We assume that the coin is fair. This is known as the null hypothesis. The alternative hypothesis is that the coin is not fair. We then set a certain threshold, and declare that given our assumption if the observed data or data less or equally likely has a total probability less than this threshold we will reject our assumption. Let us set the threshold at 0.05.

We toss the coin 15 times and obtain 15 heads. The probability of this happening given that the coin is fair is $(\frac{1}{2})^{15} \approx 0 : 00003$. An event that is less or equally like is getting 15 tails, with a probability of 0.00003 as well. The cumulative of these is 0:00006 which is less than our threshold, therefore we reject the null hypothesis.

Suppose instead that we had tossed the coin 10 times and obtained 2 heads, while using the threshold of 0:1. The events equally or less likely are getting 2 or less heads and 2 or less tail, the sum of whose probabilities is 0:109375, which is greater than our threshold. Therefore, we declare that the null hypothesis is valid, and an unlikely but not too unlikely possibility has occurred.

It may be argued that in the former case as well, the coin could have been fair and it was only that an unlikely possibility had occurred. The argument is valid, and when it comes to simulations, one can rectify this problem by repeating several times to obtain an expected value, and then repeating the entire experiment multiple times, to get a distribution of the expected values as we did in the previous questions. In real life, however, we hardly have such liberties, such as when conducting surveys, and therefore hypothesis testing remains a reliable method. Of course, we could be wrong sometimes to reject the null hypothesis but we would be right most of the time. That's just how probability works.

5.1

Implement a function that simulates the behavior of a fair coin, you may choose return types as you see fit. Implement another function that uses the above function to simulate 10 coin tosses multiple times and finds the expected number of times the null hypothesis is rejected even though it is true. Use the several experiments each having several iterations approach to generate a histogram of expected values. Mathematically and simulation-wise, what is the probability we will reject the null hypothesis even though it is true. Explain both approaches in your report. Use a threshold of 0:05. Reach out if you have confusions but not at the 11^{th} hour.

Mathematical Calculation:
[13] [14]

$$P(X = x) = {}^{n}C_r \, p^{n-r} \, (1-p)^r$$

$p = \frac{1}{2}$ (Since it is a fair coin)
Probability of getting zero heads:
$P(H = 0) = {}^{10}C_0 \, (0.5)^0 \, (0.5)^{10-0}$
$P(H = 0) = \frac{1}{1024} \approx 0.0009765625$
Probability of getting one heads:
$P(H = 1) = {}^{10}C_1 \, (0.5)^1 \, (0.5)^{10-1}$
$P(H = 1) = \frac{5}{512} \approx 0.009765625$
Probability of getting two heads:
$P(H = 2) = {}^{10}C_2 \, (0.5)^2 \, (0.5)^{10-2}$
$P(H = 2) = \frac{45}{1024} \approx 0.0439453125$
Probability of getting three heads:
$P(H = 3) = {}^{10}C_3 \, (0.5)^3 \, (0.5)^{10-3}$
$P(H = 3) = \frac{15}{128} \approx 0.1171875$
Probability of getting four heads:
$P(H = 4) = {}^{10}C_4 \, (0.5)^4 \, (0.5)^{10-4}$
$P(H = 4) = \frac{105}{512} \approx 0.205078125$
Probability of getting five heads:
$P(H = 4) = {}^{10}C_4 \, (0.5)^4 \, (0.5)^{10-4}$
$P(H = 4) = \frac{105}{512} \approx 0.205078125$
Notice that asymmetrically, the number of heads whose appearance makes the probability lie outside our favourable region is 0, 1 and 2
$P(H = 0) + P(H = 1) + P(H = 2)$
$= \frac{1}{1024} + \frac{5}{512} + \frac{45}{1024}$
$= 0.0546875$

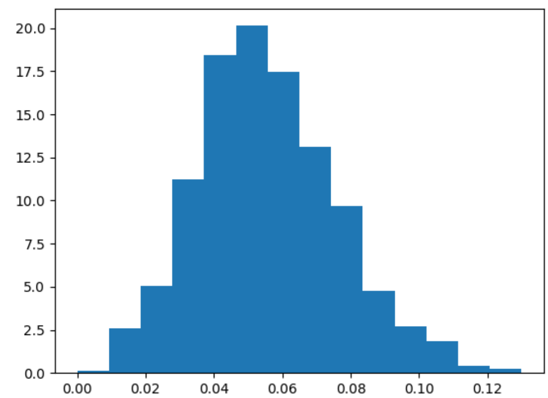
So, the probability that we will reject the null hypothesis even though it is true is 0.0546875

Simulation:
The code of simulation is given below :

```python
from scipy.stats import binom
import numpy as np
from matplotlib import pyplot as plt

def coinFlip():
    """
    Returns the result of the flip of coin
    Args:
    - None
    Ref:
    - https://towardsdatascience.com/how-to-code-a-fair-coin-flip-in-python-d54312f33da9
    - https://numpy.org/doc/stable/reference/random/generated/numpy.random.binomial.html
    Returns:
    - returns 1 if the flip of a coin is heads and 0 in case of tails
    """
    return np.random.binomial(1, 0.5)

def flip10times():
    """
    Finds the number of times head appears when a coin is flipped 10 times
    Args:
    - None
    Ref:
    - https://towardsdatascience.com/how-to-code-a-fair-coin-flip-in-python-d54312f33da9
    Returns:
    - The number of times coin shows head when flipped 10 times
    """
    n = 10
    fullResults = np.arange(n)
    for i in range(n):
        fullResults[i] = coinFlip()    # Flip the coin n number of times
    return np.count_nonzero(fullResults == 1)

```

```python
def reject_H0():
    """
    Returns the expected number of times null hypothesis is rejected
    Args: None
    Ref:
    - https://stackoverflow.com/questions/46678622/binomial-distribution-cdf-using-scipy-stats-binom-cdf
    Returns:
    the average number of times null hypothesis is rejected when tested 100 times
    """
    rejected = 0  # Bumber of times H0 gets rejected
    n = 100
    for i in range(n):
        cum_probab = 1 - (binom.cdf(flip10times(), 10, 0.5)) # using sdf of binomial random variable
        if cum_probab < (0.05 / 2): # cumulative probability should be less than threshold / 2
            rejected += 1
    return rejected / n

def make_histogram():
    """
    Displays a histogram of the expected number of times the null hypothesis is rejected
    eventhough it is true
    Args:
    None
    Returns:
    None
    """
    n = 1000
    expectation = [reject_H0() for i in range(n)] # List of expected number of times H0 gets rejected
    bins = 14
    binWidth = (max(expectation) - min(expectation)) / bins
    plt.hist(expectation, bins = bins, weights = np.ones(len (expectation))/( len (expectation)* binWidth ))
    plt.show()

make_histogram()
```

Histogram:



Notice that the center of the Histogram is 0.05 which is the highest too. This verifies that our threshold is indeed the probability that we will reject the null hypothesis even though it is true.

5.2

You are out fishing. The length of fish in your fishing area follows a normal distribution. You are trying to prove or disprove what someone said to you to about the mean length of the fishes. Unfortunately, you do not have access to the lengths of every fish in the area, which would allow you to calculate the population mean and the population variance. The best you can do is to catch a small sample, find the sample mean and the sample variance, and make some simplifying assumptions. You are provided some code files which can be used in the following way to catch a single sh and measure its length

```
import fishCImport as f
length = f.fish()
```

5.2.1

Suppose that the mean length you have been told is 23, and the size of the sample i.e. the number of fish you decide to catch, n, is 30. You simplified your problem by stating that the means of samples follow the normal distribution with mean u_0 which is the population mean, and standard deviation $\frac{\sigma}{\sqrt{n}}$, where σ is the standard deviation of your sample, and n the size of your sample.

$$S \sim N(u_0, (\frac{sigma}{\sqrt{n}})^2) \tag{1}$$

You may start of by declaring that the null hypothesis is that the population mean, u_0, is exactly 23. Conduct hypothesis testing several times with a threshold of 0.05. Measure the proportion / expected number of times, the null hypothesis is rejected. Conduct the experiment several times to and several values of this proportion. Plot these as a histogram.
Implement a function that takes in u_0 and n and conducts a single hypothesis test and returns its result. A single hypothesis test here constitutes catching a sample of 30 fish, finding the sample mean, u, the sample variance σ, and using the above specified normal distribution with the population mean, u_0, as 23, assumed through the null hypothesis, to find the probability of obtaining the sample mean or a mean with an absolute difference greater or equal to $|u - u_0|$. Mathematically, if $a = |u - u_0|$ then the null hypothesis is rejected if

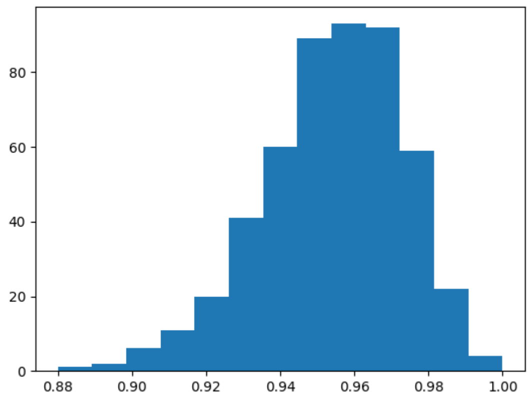
$$P(|S - u_0| \geq a) < Threshold \tag{2}$$

Implement another function that utilizes the above function or otherwise, performs several experiments, each with several hypothesis tests and plots a histogram of the proportion of times the null hypothesis is rejected.
Comment on whether it would have been sufficient to accept or reject the null hypothesis based on a single hypothesis test

In the first part of the question we are asked to find the proportion of the number of times the null hypothesis is rejected. So lets do that first!
Let the null hypothesis H_0 and the alternate hypothesis H_1 be as follows :
H_0: Population mean is exactly 23
H_1: Population mean is not equal to 23
Threshold = 0.05

```python
from scipy import stats
import numpy as np
import fishCImport as f
import math
from matplotlib import pyplot as plt

def RejectOrAccept():
    """ This function catches 30 fishes and tells if the null hypothesis is accepted or not
    Args: None
    Returns:
    - int: 1 if the null hypothesis is False and 0 if the null hypothesis is True
    """
    n = 30                                        # Total number of fishes in our sample
    threshold = 0.05                              # Theshold
    u_0 = 23                                      # Mean length given
    fish_lengths = [f.fish() for i in range(n)]   # List of lengths of 30 fishes
    stdev = np.std(fish_lengths)                  # Sandard Deviation of sample
    u = np.mean(fish_lengths)                     # Mean length of fishes catched
    cdf = stats.norm(u_0 , stdev / math.sqrt(n)).cdf(u) # Finds the probability of P (S <= u)
    if u > u_0 :                                  # If u is on the right of u_0 then cater the right region
        cdf = 1 - cdf
    if cdf < (threshold / 2) :                    # compare tail region (right OR left) with threshold / 2
        return 1                                  # return 1 if rejected , 0 if accepted
    return 0

def reject_H0():
    """ Finds the expected number of times null hypothesis gets rejected
    Args:
    None
    Return:
    The probability of null hypothesis being rejected
    """
    expected = 0
    for i in range(100):                          # Experiments 100 times
        expected += RejectOrAccept()
    return (expected / 100)

```

```python
def histogram():
    """ Draws the histogram that plots the measure of proportion/expected number of times
    the null hypothesis is rejected
    Args: None
    Returns: None
    """
    n = 500
    # Conduct the experiment 500 times to find and several values of this proportion.
    expectation = [reject_H0() for i in range(n)]
    bins = 13
    plt.hist(expectation, bins = bins )
    plt.show()

histogram()


```

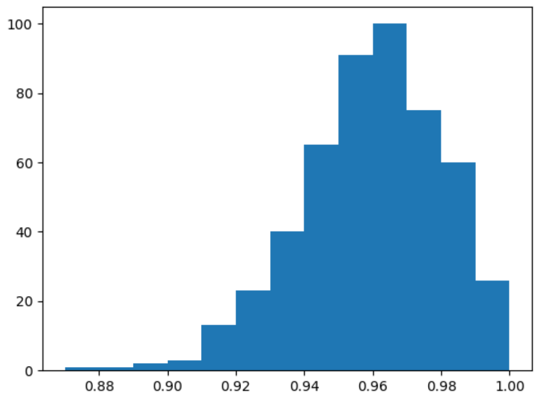


Comment: The the null hypothesis is rejected since the mean expected value of the number of times it was rejected was 0.96

In the second part of the question we have to implement single hypothesis test. Before coding, let us first look into some basic computations:
We have to find $P(|S - u_0| \geq a)$ and then compare it with the threshold.
$P(|S - u_0| \geq a)$ where $a = |u - u_0|$
$= 1 - P(|S - u_0| < a)$
$= 1 - P(-a < S - u_0 < a)$
$= 1 - (P(S - u_0 < a) - P(S - u_0 < -a))$
$= 1 - P(S - u_0 < a) + P(S - u_0 < -a)$
$= (1 - P(S - u_0 < a)) + P(S - u_0 < -a)$
$= P(S - u_0 \geq a) + P(S - u_0 < -a)$

```
from scipy import stats
import numpy as np
import fishCImport as f
import math
from matplotlib import pyplot as plt

def RejectOrAccept(u_0 , n):
    """ This function catches 30 fishes and tells if the null hypothesis is accepted or not
    Args: None
    Returns:
    - int: 1 if the null hypothesis is False and 0 if the null hypothesis is True
    """
    threshold = 0.05                                 # Theshold
    fish_lengths = [f.fish() for i in range(n)]      # List of lengths of 30 fishes
    stdev = np.std(fish_lengths)                     # Sandard Deviation of sample
    u = np.mean(fish_lengths)                        # Mean length of fishes catched
    a = abs(u - u_0)

    right = 1 - (stats.norm(u_0, stdev / math.sqrt(n))).cdf(a + u_0)  # right tail
    left = stats.norm(u_0, stdev / math.sqrt(n)).cdf(- a + u_0)       # Left tail
    total = left + right

    if total < threshold:
        return 1                                     # Return 1 if null hypothesis is rejected
    return 0                                         # Return 0 if null hypothesis is accepted
```

```
def reject_H0():
    """ Finds the expected number of times null hypothesis gets rejected
    Args:
    None
    Return:
    The probability of null hypothesis being rejected
    """
    expected = 0
    for i in range(100):                            # Experiments 100 times
        expected += RejectOrAccept(23, 30)
    return (expected / 100)

def histogram():
    """ Draws the histogram that plots the measure of proportion/expected number of times
    the null hypothesis is rejected
    Args: None
    Returns: None
    """
    n = 500
    # Conduct the experiment 500 times to find and several values of this proportion.
    expectation = [reject_H0() for i in range(n)]
    bins = 13
    plt.hist(expectation, bins = bins )
    plt.show()

histogram()
```



Comment: The mean of the expected number of times the null hypothesis was rejected was approximately 0.96.

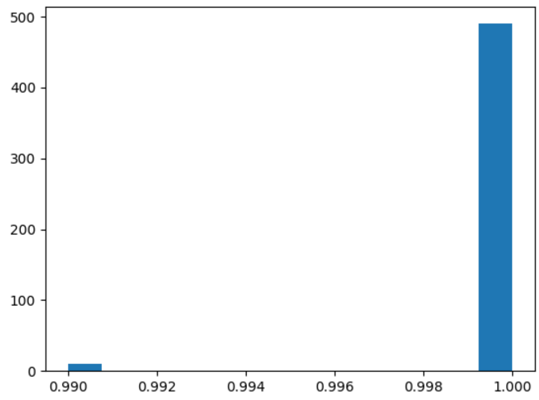
Based on only one such hypothesis test we cannot tell whether out null hypothesis is True or not since there is a probability of 0.05 that we will reject the null hypothesis even though it is True (as found in 5.1). Therefore, to remove to reduce the chances of obtaining incorrect results regarding the correctness or incorrectness of the null hypothesis and to declare whether we accept or reject the null hypothesis we have to perform the hypothesis test many times.

5.2.2

Conduct the same experiments with same u0 and n = 70. Implement a different function for this and generate a similar histogram plot of proportion of times the null hypothesis is rejected.
Comment on what increasing the value of n accomplishes and whether it would have been sufficient to accept or reject the null hypothesis based on a single hypothesis test in this case.

```python
from scipy import stats
import numpy as np
import fishCImport as f
import math
from matplotlib import pyplot as plt

def RejectOrAccept(u_0 , n):
    """ This function catches 30 fishes and tells if the null hypothesis is accepted or not
    Args: None
    Returns:
    - int: 1 if the null hypothesis is False and 0 if the null hypothesis is True
    """
    threshold = 0.05                                  # Theshold
    fish_lengths = [f.fish() for i in range(n)]       # List of lengths of 30 fishes
    stdev = np.std(fish_lengths)                      # Sandard Deviation of sample
    u = np.mean(fish_lengths)                         # Mean length of fishes catched
    a = abs(u - u_0)

    right = 1 - (stats.norm(u_0, stdev / math.sqrt(n))).cdf(a + u_0)  # right tail
    left = stats.norm(u_0, stdev / math.sqrt(n)).cdf(- a + u_0)       # Left tail
    total = left + right

    if total < threshold:
        return 1                                      # Return 1 if null hypothesis is rejected
    return 0                                          # Return 0 if null hypothesis is accepted

def reject_H0():
    """ Finds the expected number of times null hypothesis gets rejected
    Args:
    None
    Return:
    The probability of null hypothesis being rejected
    """
    expected = 0
    for i in range(100):                              # Experiments 100 times
        expected += RejectOrAccept(23, 70)            # n = 70
    return (expected / 100)

```

```python
def histogram():
    """ Draws the histogram that plots the measure of proportion/expected number of times
    the null hypothesis is rejected
    Args: None
    Returns: None
    """
    n = 500
    # Conduct the experiment 500 times to find and several values of this proportion.
    expectation = [reject_H0() for i in range(n)]
    bins = 13
    plt.hist(expectation, bins = bins )
    plt.show()

histogram()

```



Comment:

Note that in part 5.2.1 since the mean of this bell shaped curve is approximately 0.97, therefore, we can conclude that the portion/ expected number of times the null hypothesis is rejected is 0.97.
As the sample size increased in this part, the probability that we will accept a False null hypothesis will decrease. [15] So it is sufficient to reject the NULL hypothesis, However, we can still reject a True Null Hypothesis, [15]. So it is still not sufficient to accept the NULL Hypothesis. Note that the expected number of times the null hypothesis is rejected is approximately 1 when $n = 70$. So, we can certainly reject the null hypothesis.

5.2.3

In 5.1 we saw that proportion of times the null hypothesis is rejected despite being true is close to the threshold we choose. In normal distributions it is exactly equal to the threshold. Experimentally or mathematically, determine the least value of n (or close enough) to ensure that the null hypothesis is not wrongly rejected more than 10 percent of the time. You may use a sample standard deviation of 3, if you decide to approach mathematically. If you decide to approach simulation wise you will have to define your own fish function which returns a random variable from the normal distribution

```
from scipy import stats
import numpy as np
import fishCImport as f
import math |
from matplotlib import pyplot as plt

def fish():
    # Select a fish length randomly with mean 23 and standard deviation 3
    f = np.random.normal(23, 3)
    return f

def run(n):              # n : Number of samples to be taken
    l = [np.random.normal(23, 3) for i in range(n)]
    sd = np.std(l)
    threshold = (50 / 100)
    sample_mean = 23
    mean = np.mean(l)
    if mean >= sample_mean:
        cum_probab = stats.norm(sample_mean , sd / math.sqrt(n)).cdf(mean)
    elif  mean < sample_mean:
        cum_probab = 1 - stats.norm(sample_mean , sd / math.sqrt(n)).cdf(mean)
    if cum_probab > threshold / 2 :
        return 1
    return 0
```

```
def reject_H0(n):    # Finds the expected value of rejection
    expected = 0
    for i in range(100):
        expected += run(n)
    return expected / 100


def make_histogram(): # Plot the histogram
    n = 1000
    expectation = [ reject_H0(30) for i in range(n) ]
    bins = 50
    plt.hist(expectation, bins = bins )
    plt.show()

make_histogram()
```

Figure 43: Code with which we have tested different n values

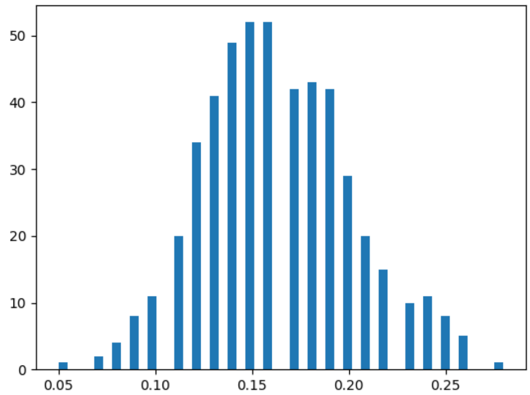


Figure 44: n = 9

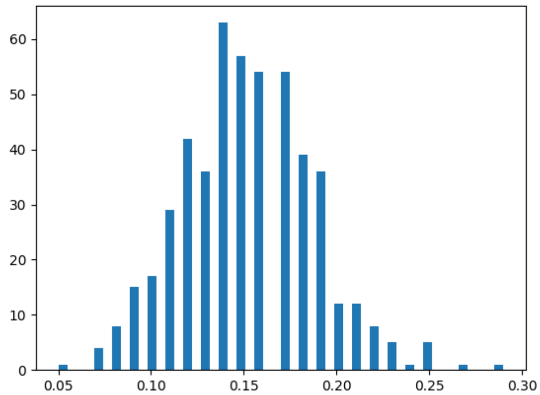


Figure 45: n = 10

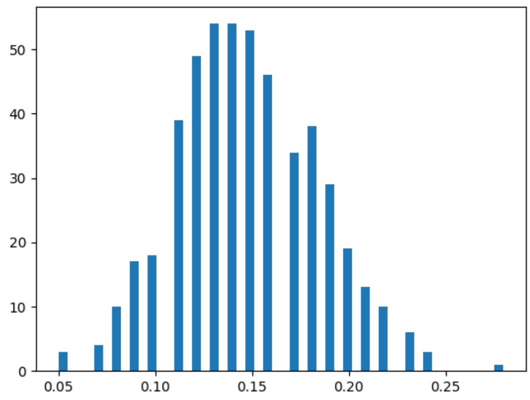


Figure 46: n = 11

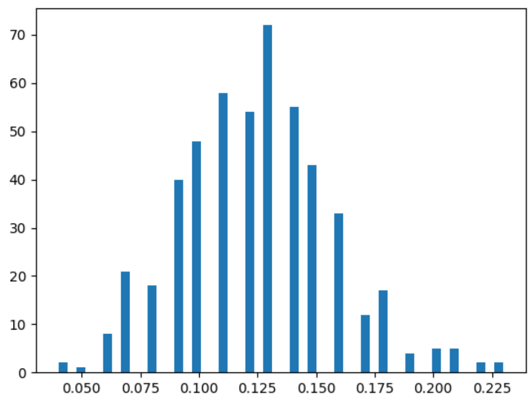


Figure 47: n = 20

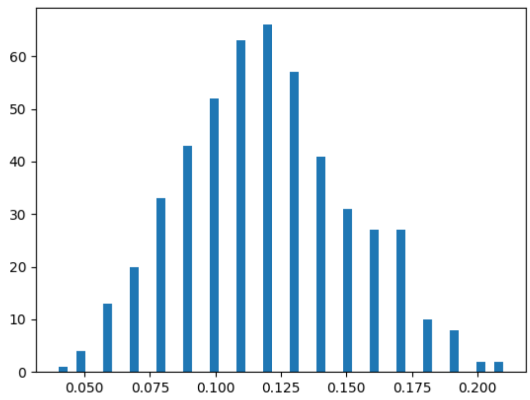


Figure 48: n = 29, bin = 50

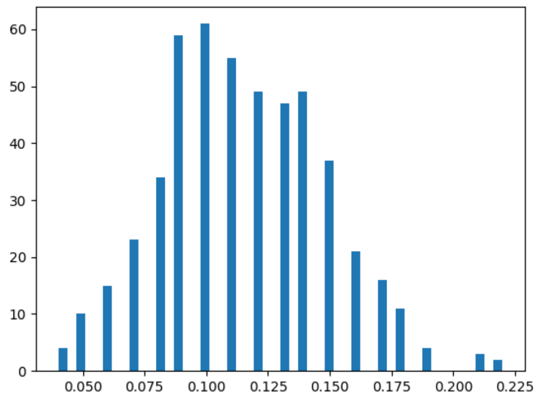


Figure 49: n = 30, bin = 50

Answer: n = 30 is the minimum n

6 References:

References

[1] "Generating Random Variables from Standard Uniform Distribution on $(1, 0)$." Mathematics Stack Exchange, 1 Aug. 1961, math.stackexchange.com/questions/241525/generating -random-variables-from-standard-uniform-distribution-on-1-0.

[2] "Numpy.linspace" Numpy.linspace - NumPy v1.20 Manual, numpy.org/doc/stable/reference/ generated/numpy.linspace.html.

[3] "Histograms Review (Article)." Khan Academy, Khan Academy, www.khanacademy.org/math /statistics-probability/displaying-describing-data/quantitative-data-graphs/a/histograms-review# :~:text=A%20$histogram$%20$displays$%20$numerical$%20$data$, %22%2C%20or%20%22$buckets$ %22.

[4] "How to Choose Bins in Matplotlib Histogram."Stack Overflow, 1 Aug. 1964, stackoverflow.com/questions/33458566/how-to-choose-bins-in-matplotlib-histogram/33459231.

[5] "More Precise Histogram in Python." Stack Overflow, 1 Sept. 1968, stackoverflow.com/questions/59226828/more-precise-histogram-in-python.

[6] gtribello. "Generating Uniform Continuous Random Variables Using Python." YouTube, YouTube, 6 Aug. 2020, www.youtube.com/watch?v=0ydYnya_wIo.

[7] gtribello. "Estimating the Probability Density Function by Calculating a Histogram." YouTube, YouTube, 18 Aug. 2020, www.youtube.com/watch?v=-aS_CrskEYE.

[8] "Law of Cosines Calculator." Omni Calculator, Omni Calculator, 4 Dec. 2020, www.omnicalculator.com/math/law-of-cosines.

[9] https://www.quora.com/A-point-is-selected-randomly-from-the-interior-of-a-circle- The-probability-that-the-point-is-closer-to-the-center-than-the-boundary-of-circle-is

[10] Data to Fish,datatofish.com/plot-histogram-python/.

[11] MIT, web.mit.edu/urban_or_book/www/book/chapter7/7.1.3.html.

[12] jaradniemi. “Inverse CDF Method.” YouTube, YouTube, 1 Mar. 2013, www.youtube.com/watch?v=TR0biDues7k.

[13] https://stats.stackexchange.com/questions/348807/find-probability-of-rejecting-a-true-null-hypothesis

[14] https://www.csus.edu/indiv/j/jgehrman/courses/stat50/hypthesistests/9hyptest.html

[15] https://www.bmj.com/content/349/bmj.g4287/rr