Lab 1 : Probability Theory

1. Sampling from uniform distribution
2. Sampling from Gaussian distribution
3. Sampling from categorical distribution through uniform distribution
4. Central limit theoram
5. Law of large number
6. Area and circumference of a circle using sampling
7. Fun Problem

There are missing fields in the code that you need to fill to get the results but note that you can write you own code to obtain the results

1.Sampling from uniform distribution

a) Generate N points from a uniform distribution range from [0 1]

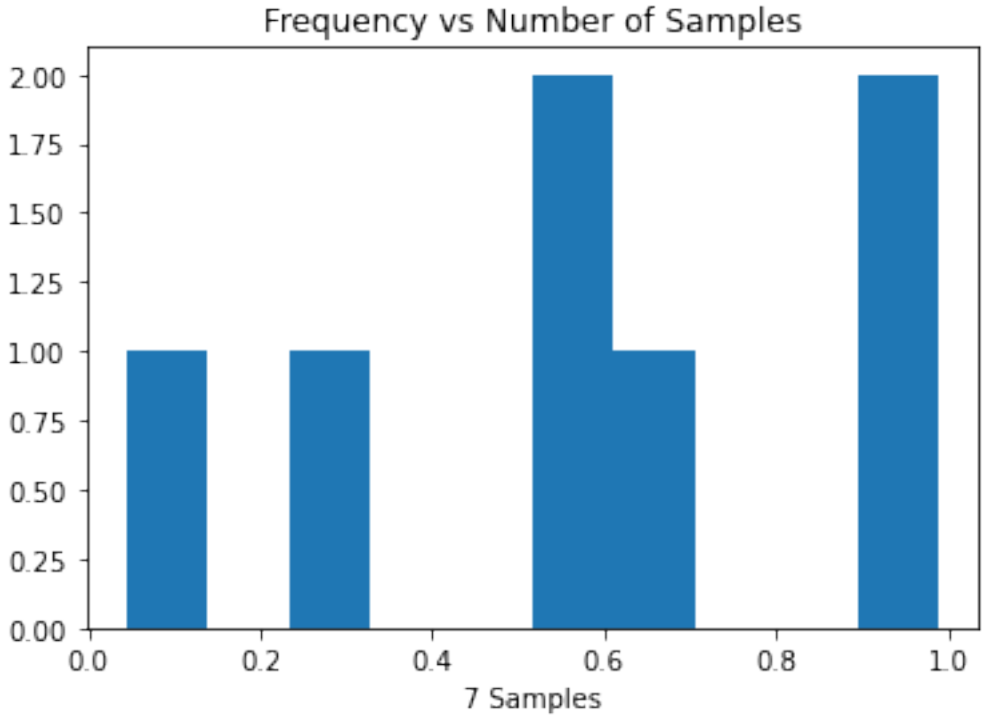
```
import numpy as np
import matplotlib.pyplot as plt

N =  100 #  Number of points (Example = 10)
X =  np.random.uniform(0,1,N)#  Generate N points from a uniform distribution range from [0 1] # Ref : https://numpy.org/doc/stable/reference/random/generated/numpy.random.uniform.html
print(X[:10])
```

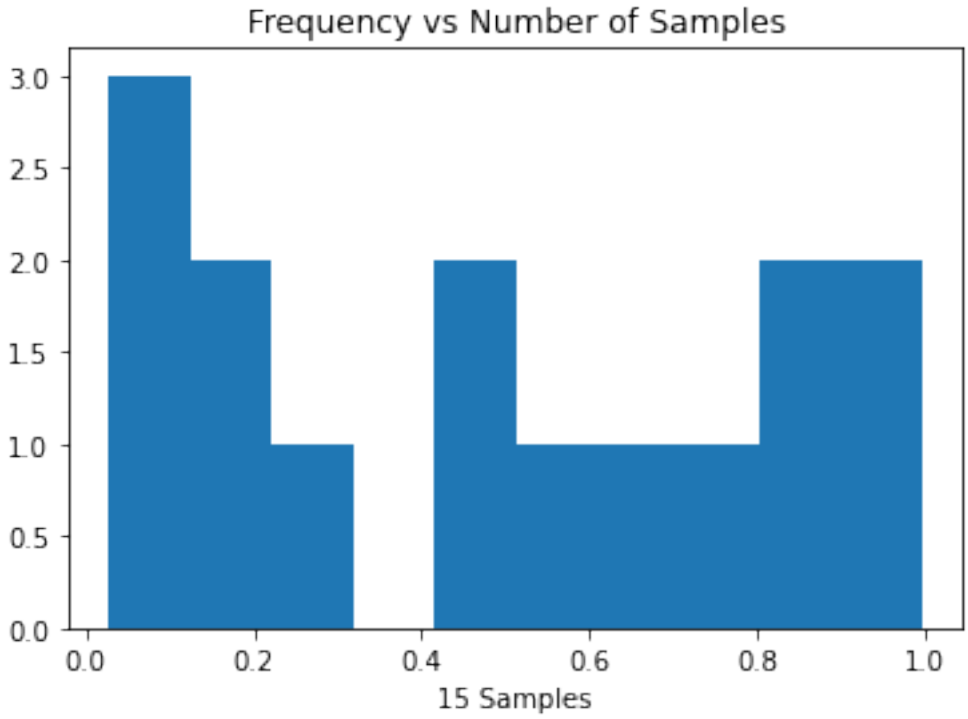
```
[0.22697721 0.06498763 0.70209712 0.52723525 0.7640076  0.10032463
 0.35071077 0.39478703 0.5429365  0.28295515]
```

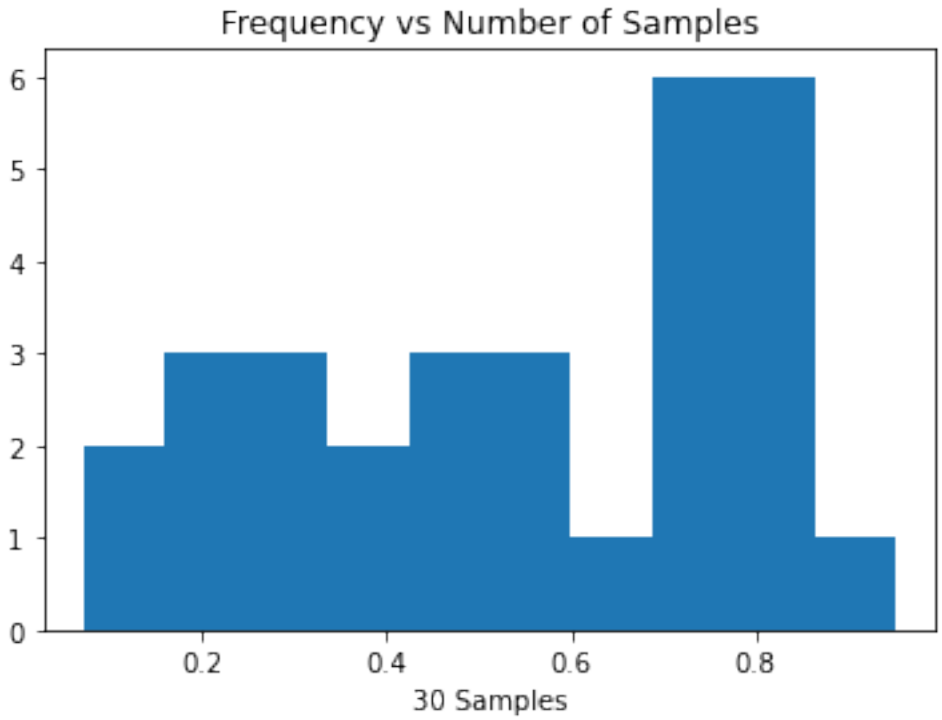
b) Show with respect to no. of sample, how the sampled distribution converges to parent distribution.

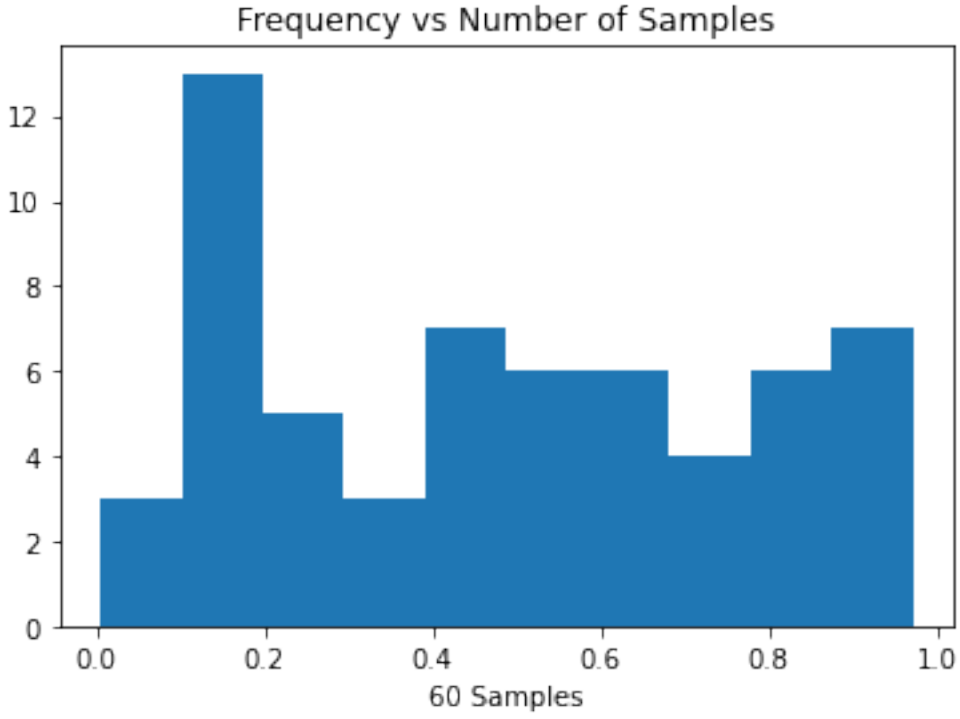
```
arr =  np.array([7, 15, 30, 60, 100, 500, 1000, 5000, 10000])# Create a numpy array of different values of no. of samples # Ref : https://numpy.org/doc/stable/reference/generated/numpy.array.html

for i in arr:
  x = np.random.uniform(0,1,i) # Generate i points from a uniform distribution range from [0 1]
  plt.hist(x)
  plt.xlabel(str(i)+" Samples")
  plt.title(f"Frequency vs Number of Samples")
  plt.show()

  # write the code to plot the histogram of the samples for all values
```

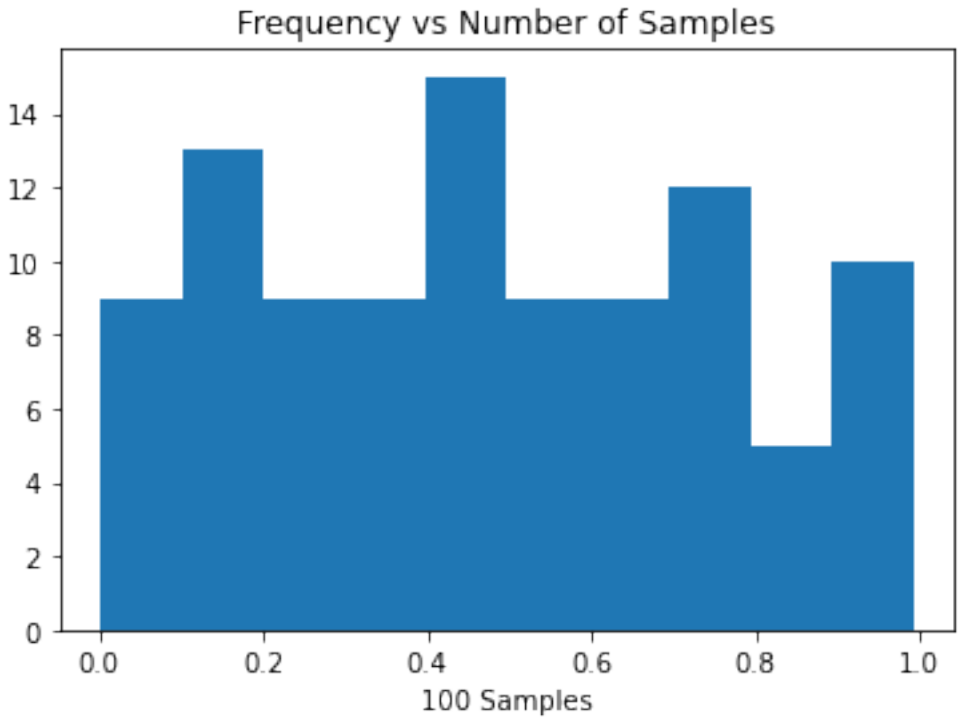
```
in arr # Ref :
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.hist.html
```

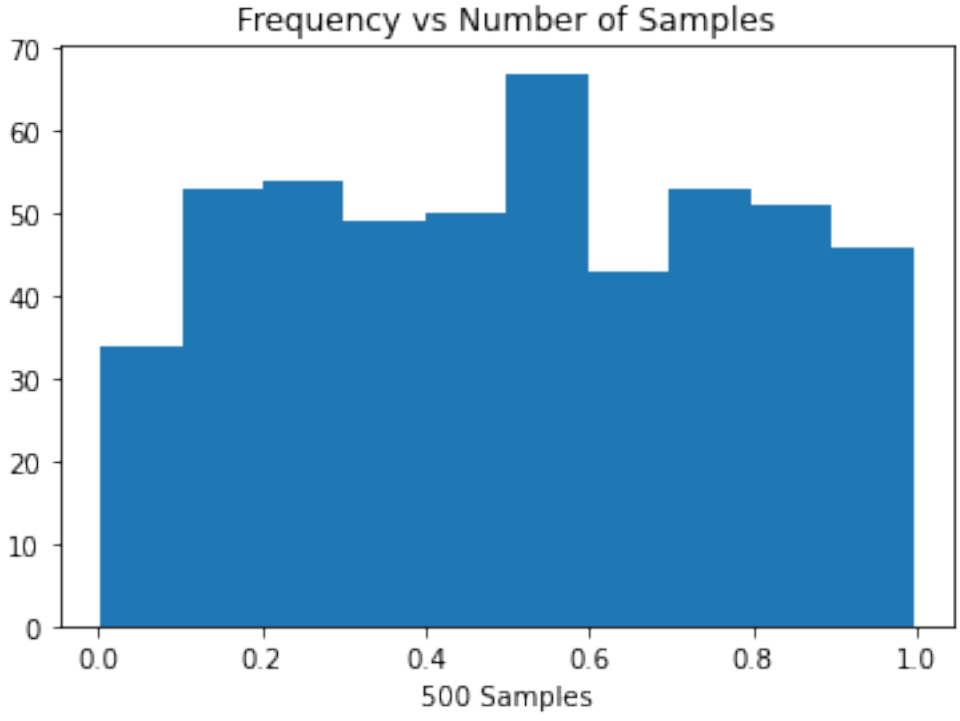


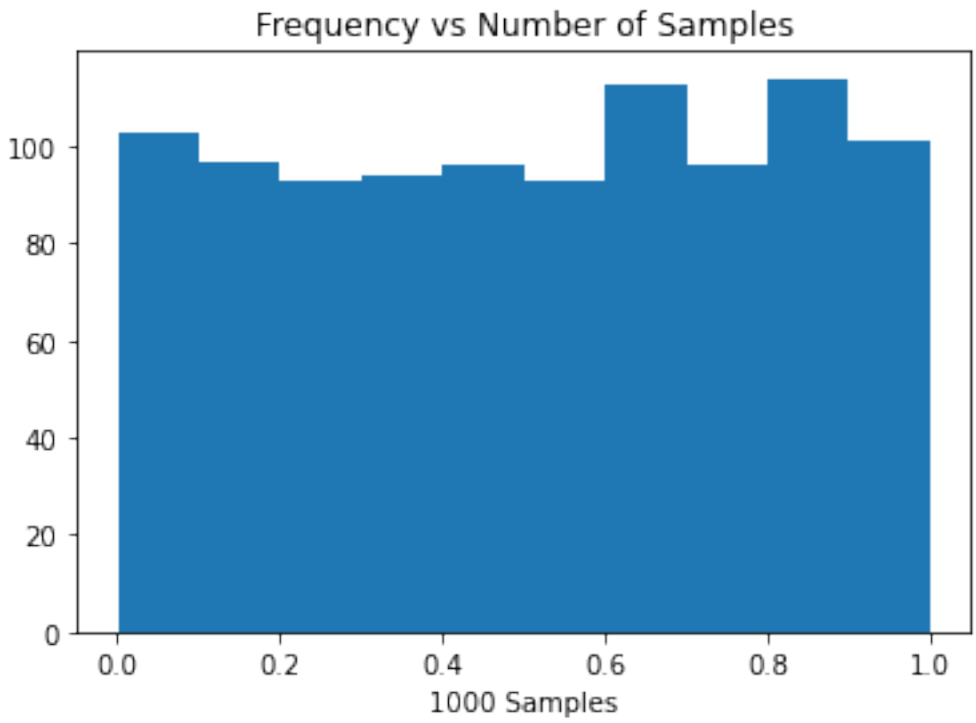


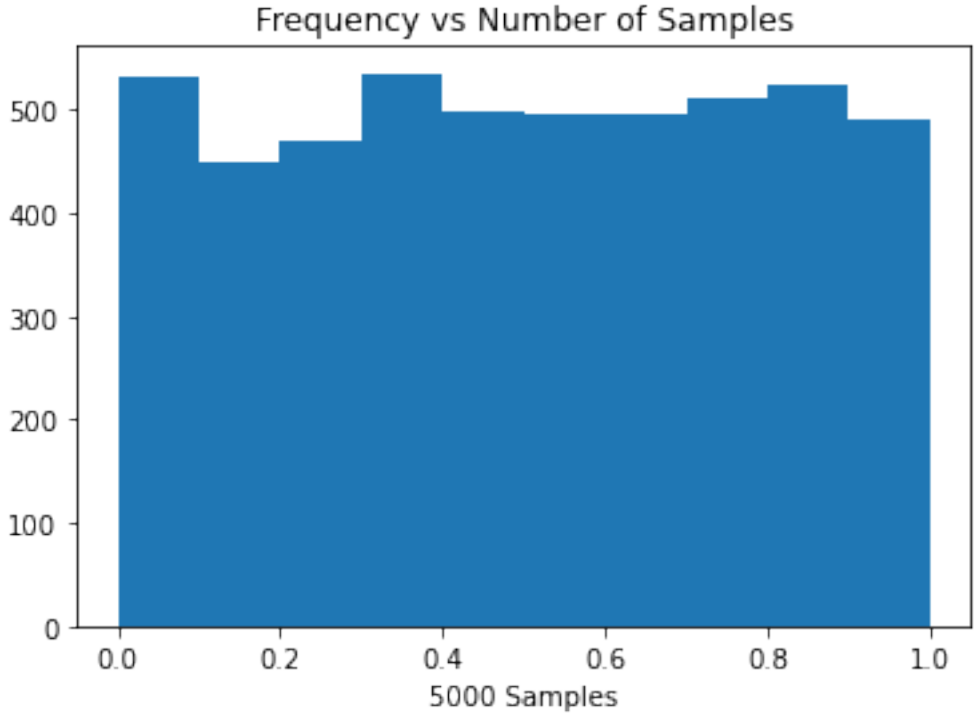

Frequency vs Number of Samples
0
1
2
3
4
5
6
0.2
0.4
0.6
0.8
30 Samples

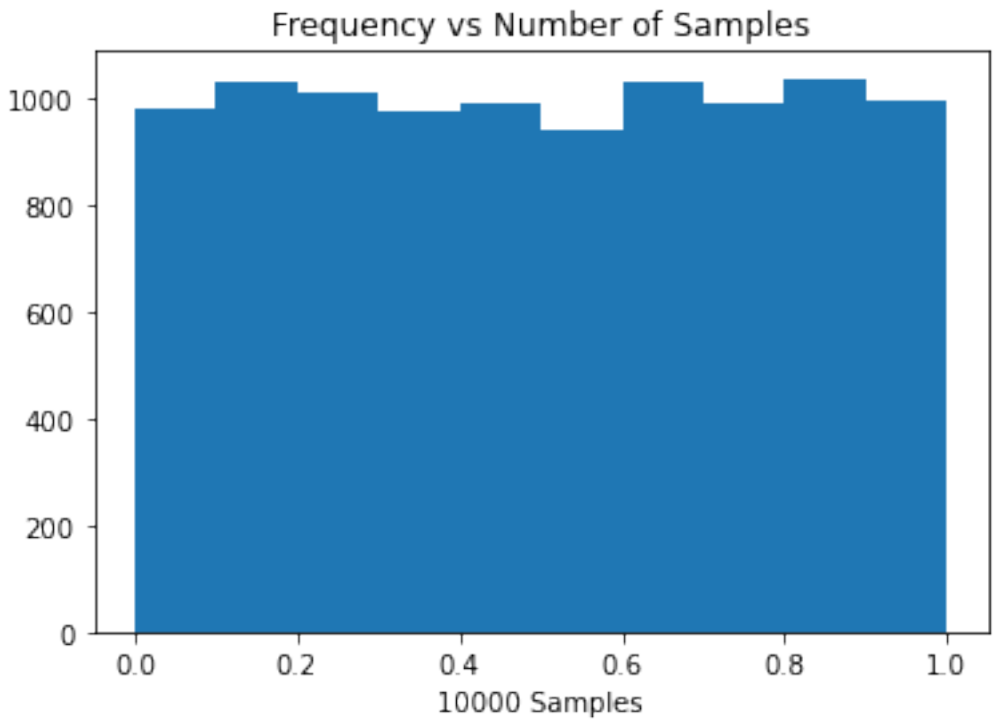

Frequency vs Number of Samples
0
2
4
6
8
10
12
0.0
0.2
0.4
0.6
0.8
1.0
60 Samples


Frequency vs Number of Samples
14
12
10
8
6
4
2
0
0.0
0.2
0.4
0.6
0.8
1.0
100 Samples


Frequency vs Number of Samples
70
60
50
40
30
20
10
0
0.0
0.2
0.4
0.6
0.8
1.0
500 Samples


Frequency vs Number of Samples
100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0
1000 Samples

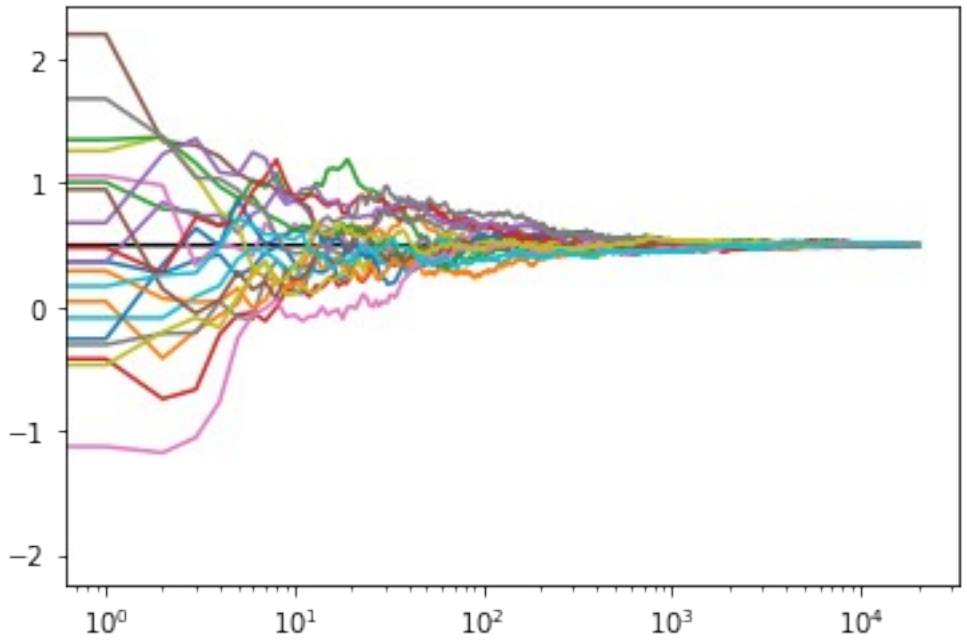

Frequency vs Number of Samples
500
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
1.0
5000 Samples



c) Law of large numbers: $average(x_{sampled}) = \acute{x}$, where x is a uniform random variable of range [0,1], thus $\acute{x} = \int_0^1 x f(x) d x = 0.5$

```
N =  20000 # Number of points (>10000)
k =  20 # set a value for number of runs

## Below code plots the semilog scaled on x-axis where all the samples
are equal to the mean of distribution
m = 0.5 # mean of uniform distribution
m = np.tile(m,x.shape)
#print(x.shape)
plt.semilogx(m,color='k') # Ref :
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.semilogx.h
tml


for j in range(k):

  i = np.arange(1,N+1) # Generate a list of numbers from (1,N) # Ref :
https://numpy.org/doc/stable/reference/generated/numpy.arange.html
  x = np.random.normal(0.5,1,N) #  Generate N points from a uniform
distribution range from [0 1]
```

```
  mean_sampled = np.cumsum(x)/(i) # Ref :
https://numpy.org/doc/stable/reference/generated/numpy.cumsum.html
  plt.semilogx(mean_sampled)
  ## Write code to plot semilog scaled on x-axis of mean_sampled,
follow the above code of semilog for reference
```



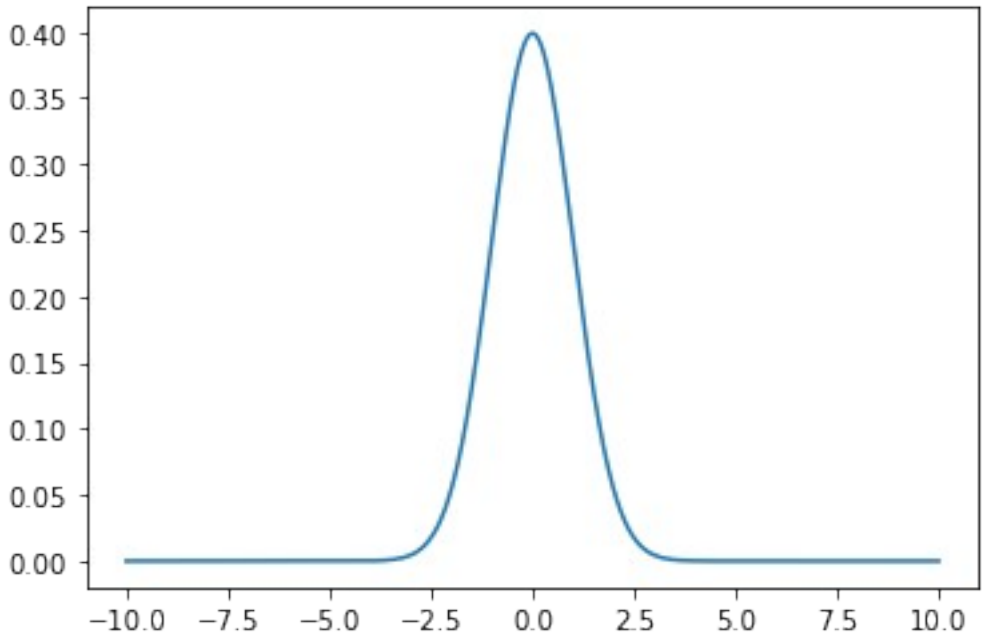
2. Sampling from Gaussian Distribution

a) Draw univariate Gaussian distribution (mean 0 and unit variance)

```
from math import e, pi, sqrt
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-10,10,1000) # Generate 1000 points from -10 to 10 #
Ref :
https://numpy.org/doc/stable/reference/generated/numpy.linspace.html

# Define mean and variance
mean = 0
variance = 1

X_square = X * X

#print(X_square.shape)
gauss_distribution =(1/sqrt(2*pi)) * np.power(np.array([e]),(-
1/2)*X_square) # Define univariate gaussian distribution (Hint :
Probabilty Distribution Function of normal distribution)
```

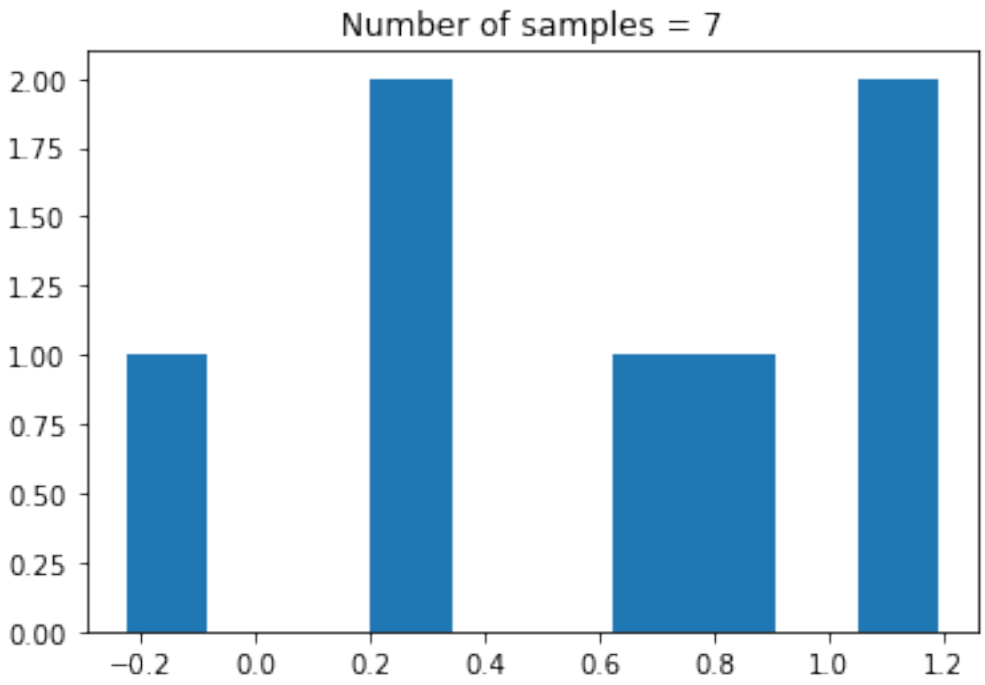
```
plt.plot(X,gauss_distribution)

## Write code to plot the above distribution # Ref :
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html
```

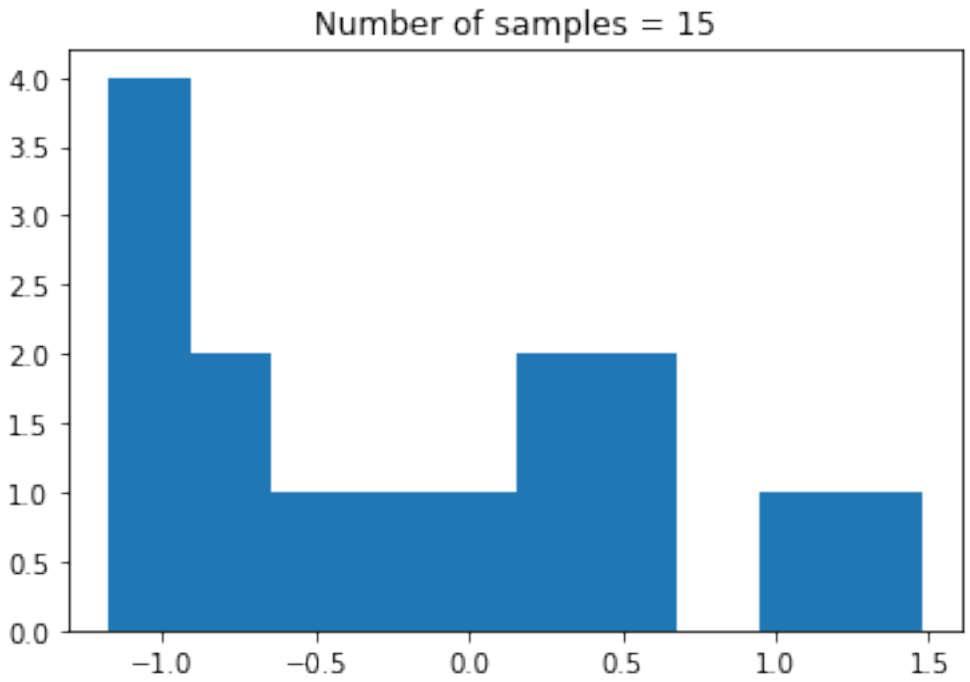
```
[<matplotlib.lines.Line2D at 0x7f0d4f134520>]
```

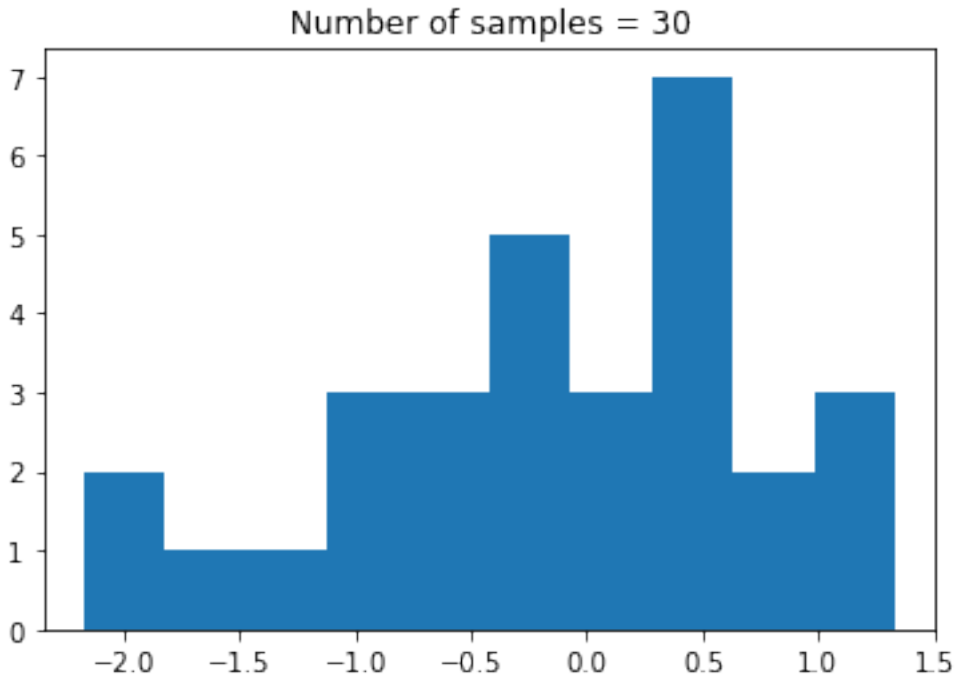


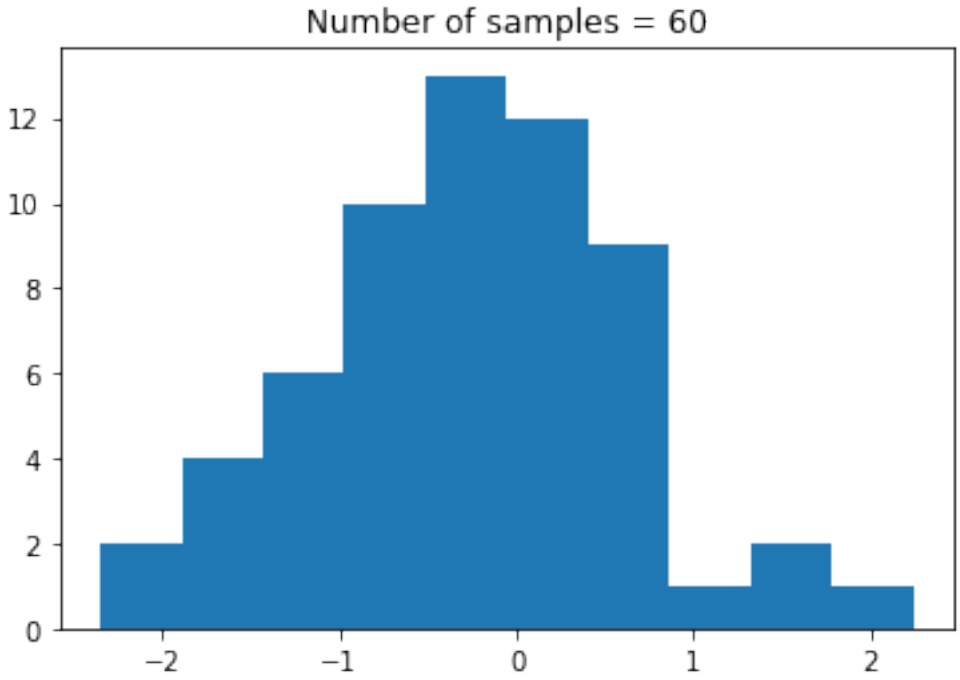
b) Sample from a univariate Gaussian distribution, observe the shape by changing the no. of sample drawn.

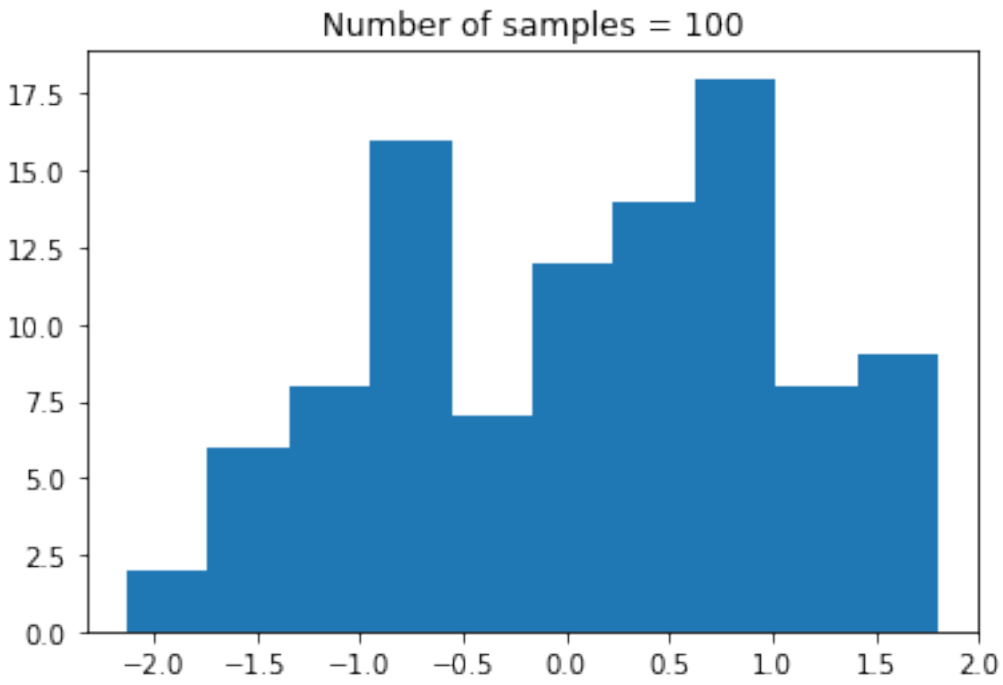
```
arr = np.array([7, 15, 30, 60, 100, 500, 1000, 5000, 10000, 30000]) #
Create a numpy array of differnt values of no. of samples and plot the
histogram to show the above

for i in arr:
  x_sampled = np.random.normal(0,1,i) # Generate i samples from
univariate gaussian distribution
  plt.hist(x_sampled)
  plt.title(f"Number of samples = {i}")
  plt.show()
  # write the code to plot the histogram of the samples for all values
in arr
```

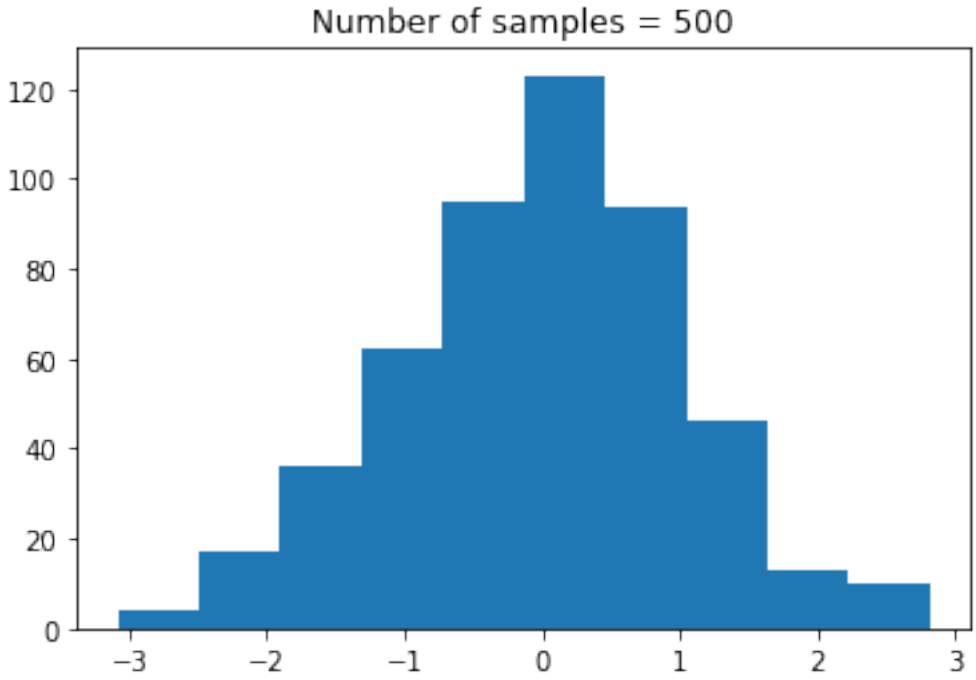

Number of samples = 7
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2

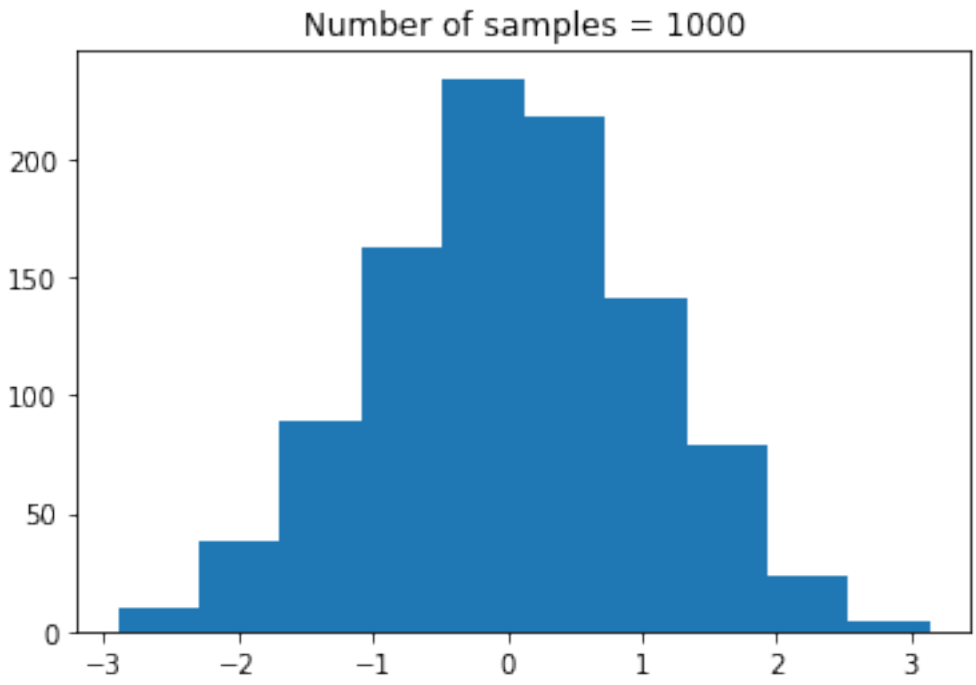

Number of samples = 15
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−1.0
−0.5
0.0
0.5
1.0
1.5

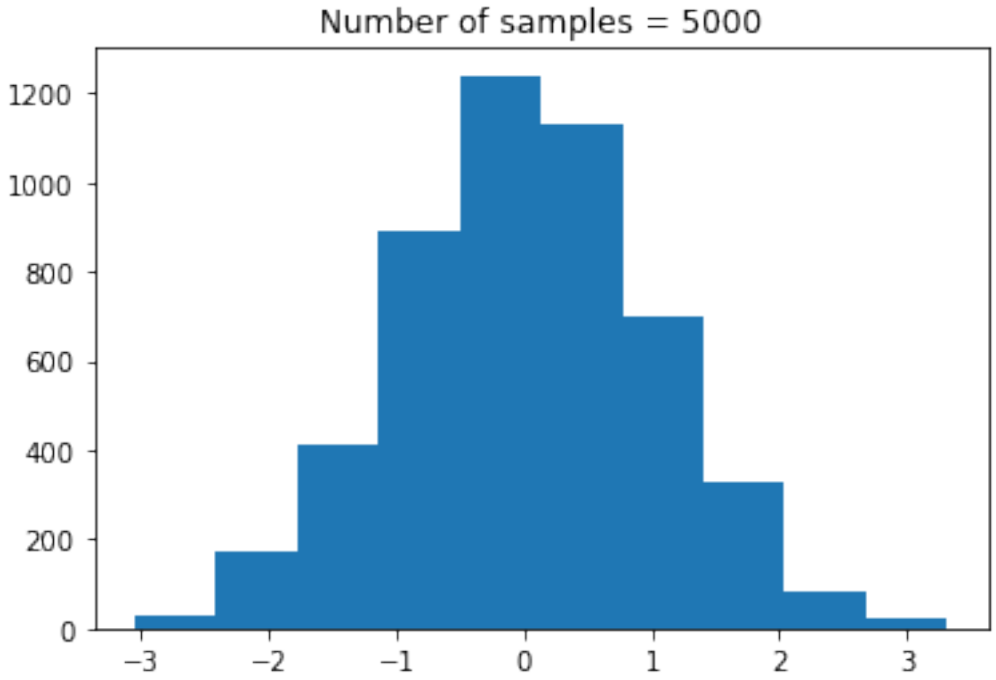

Number of samples = 30
7
6
5
4
3
2
1
0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

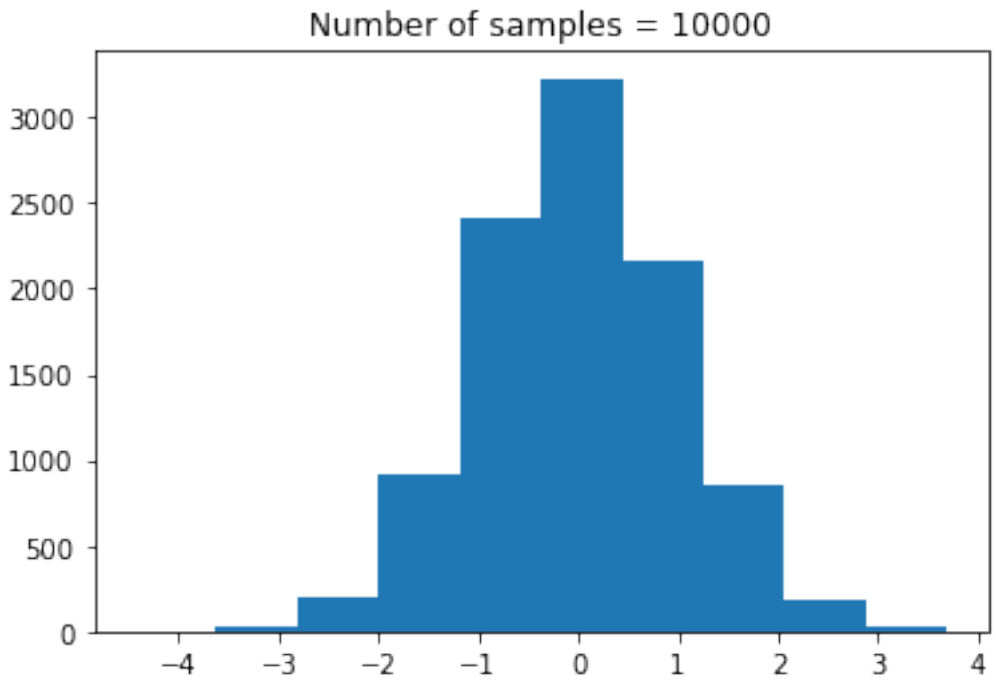

Number of samples = 60
12
10
8
6
4
2
0
−2
−1
0
1
2

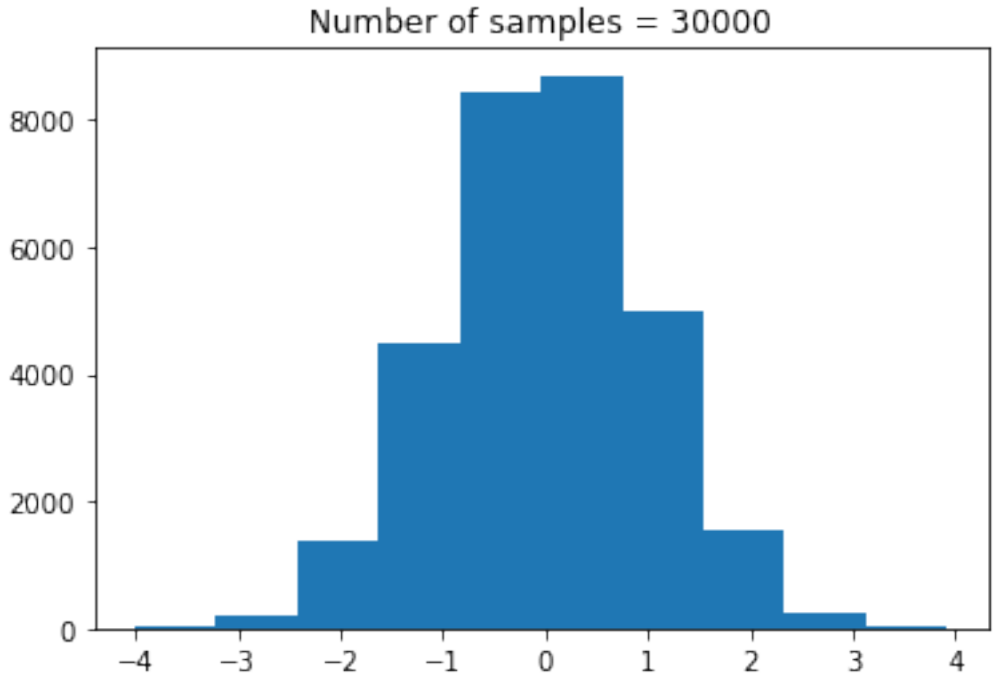

Number of samples = 100
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Number of samples = 500
120
100
80
60
40
20
0
−3
−2
−1
0
1
2
3


Number of samples = 1000
200
150
100
50
0
−3
−2
−1
0
1
2
3


Number of samples = 5000
1200
1000
800
600
400
200
0
−3
−2
−1
0
1
2
3

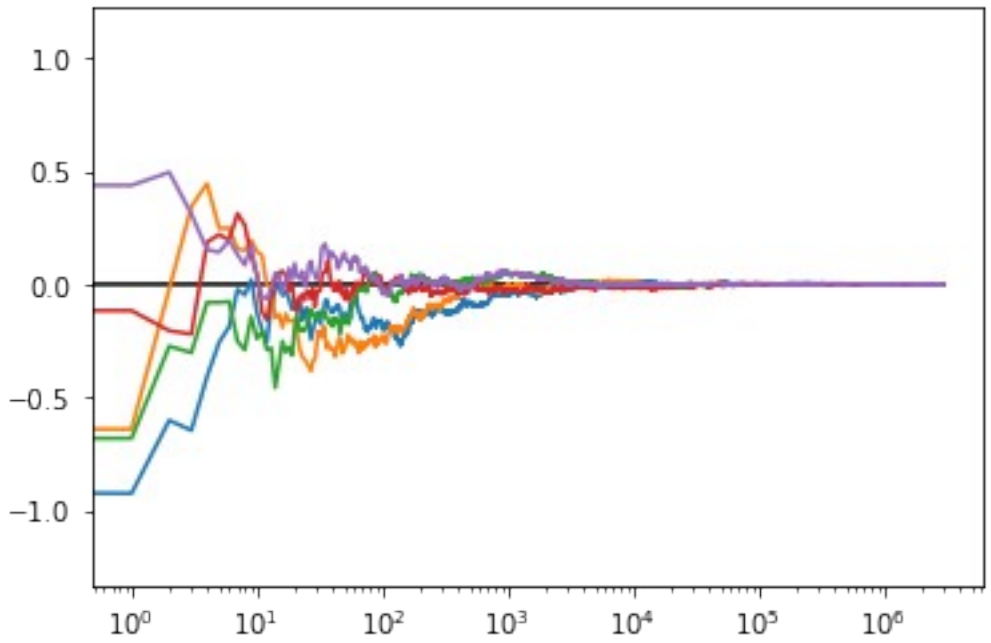




c) Law of large number

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

N = 3000000  #  Number of points (>1000000)
k = 5 # set a value for number of distributions
```

```
## Below code plots the semilog when all the samples are equal to the
mean of distribution

m = np.tile(mean,x.shape)
#print(x.shape)
plt.semilogx(m,color='k')

for j in range(k):

  i = np.arange(1,N+1) # Generate a list of numbers from (1,N)
  x = np.random.normal(0,1,N) # Generate N samples from univariate
gaussian distribution # Ref :
https://numpy.org/doc/stable/reference/random/generated/numpy.random.n
ormal.html
  mean_sampled = np.cumsum(x)/(i) # Ref :
https://numpy.org/doc/stable/reference/generated/numpy.cumsum.html
  plt.semilogx(mean_sampled) # insert your code here (Hint : Repeat
the same steps as in the uniform distribution case)

  ## Write code to plot semilog scaled on x axis of mean_sampled,
follow the above code of semilog for reference
```



3.Sampling of categorical from uniform

i) Generate n points from uniforms distribution range from [0 1] (Take large n)

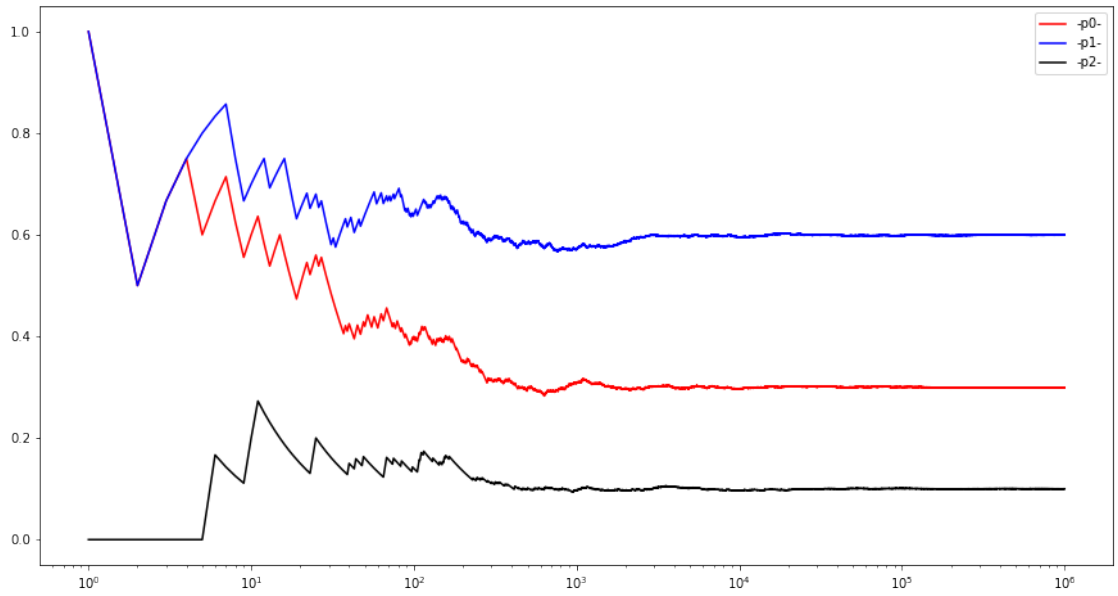
ii) Let $prob_0$=0.3, $prob_1$=0.6 and $prob_2$=0.1

iii) Count the number of occurences and divide by the number of total draws for 3 scenarios :

1. $p_0 : ¿\ prob_0$
2. $p_1 : ¿\ prob_1$
3. $p_2 : ¿\ prob_2$

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

n = 1000000 # Number of points (>1000000)
y = np.random.uniform(0,1,n)# Generate n points from uniform
distribution range from [0 1]
x = np.arange(1, n+1)
prob0 = 0.3
prob1 = 0.6
prob2 = 0.1

# count number of occurrences and divide by the number of total draws
# p0 = y[y < prob0]
#p0 = y[0 if y >= prob0 else 1]
p0 = np.array([ 0 if x >= prob0 else 1 for x in y ]) # insert your
code here
p0 = np.cumsum(p0)/x
#p1 = y[y < prob1]/n # insert your code here
p1 = np.array([ 0 if x >= prob1 else 1 for x in y ])
p1 = np.cumsum(p1)/x
#p2 = y[y < prob2]/n # insert your code here
p2 = np.array([ 0 if x >= prob2 else 1 for x in y ])
p2 = np.cumsum(p2)/x
#print(p0.shape)

plt.figure(figsize=(15, 8))
plt.semilogx(x, p0,color='r')
plt.semilogx(x, p1,color='b')
plt.semilogx(x, p2,color='k')
plt.legend(['-p0-','-p1-','-p2-'])
```

```
<matplotlib.legend.Legend at 0x7f0d4e61b340>
```



4. Central limit theorem

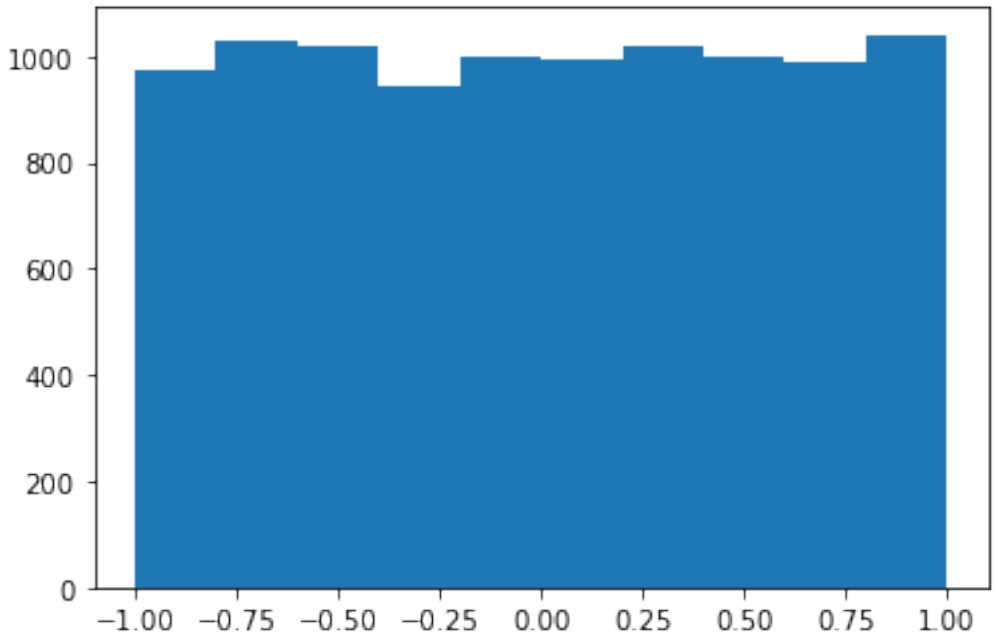
a) Sample from a uniform distribution (-1,1), some 10000 no. of samples 1000 times (u1,u2,....,u1000). show addition of iid rendom variables converges to a Gaussian distribution as number of variables tends to infinity.

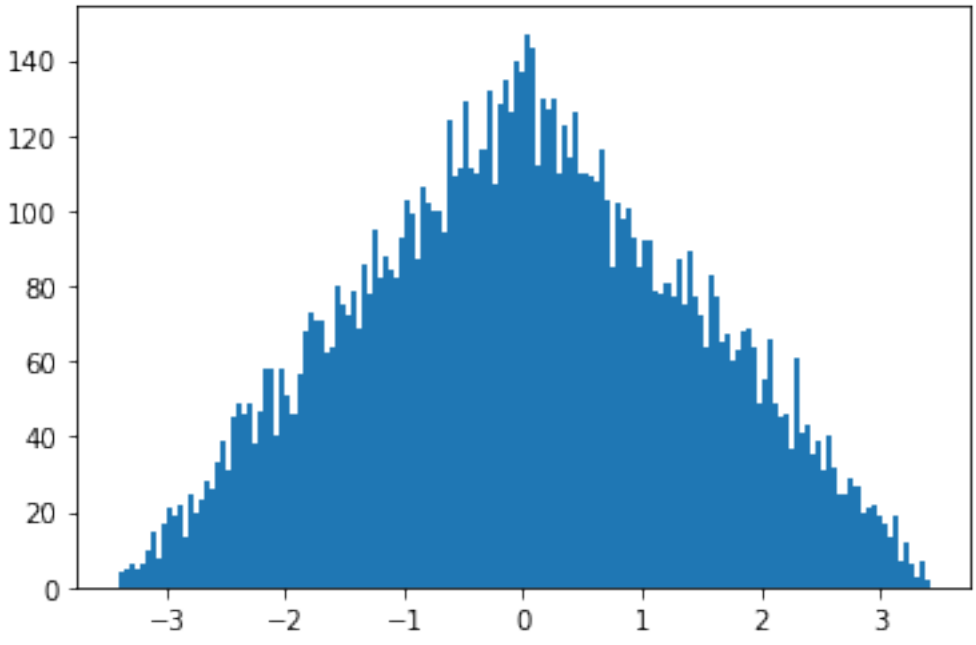
```
x = np.array( [np.random.uniform(-1,1,10000) for _ in range(10000)])# Generate 1000 diferent uniform distributions of 10000 samples each in range from [-1 1]

plt.figure()
plt.hist(x[:,0])

# addition of 2 random variables
Add_2_RV=np.sum(x[:,0:2],axis=1)/(np.std(x[:,0:2]))
plt.figure()
plt.hist(Add_2_RV,150)

# Repeat the same for 100 and 1000 random variables

# addition of 100 random variables
# start code here
Add_100_RV = np.sum(x[:,0:100],axis=1)/(np.std(x[:,0:100]))
plt.figure()
plt.hist(Add_100_RV,150)

# addition of 1000 random variables
# start code here
Add_1000_RV = np.sum(x[:,0:1000],axis=1)/(np.std(x[:,0:1000]))
plt.figure()
plt.hist(Add_1000_RV,150)
```

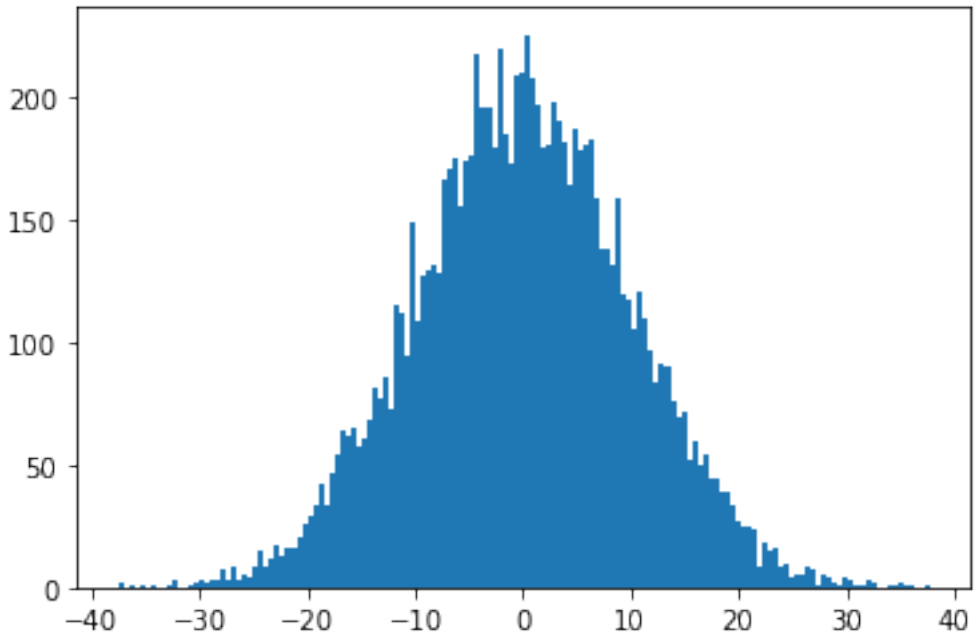
```
# addition of 1000 random variables
# start code here
Add_10000_RV = np.sum(x[:,0:10000],axis=1)/(np.std(x[:,0:10000]))
plt.figure()
plt.hist(Add_10000_RV,150)
```

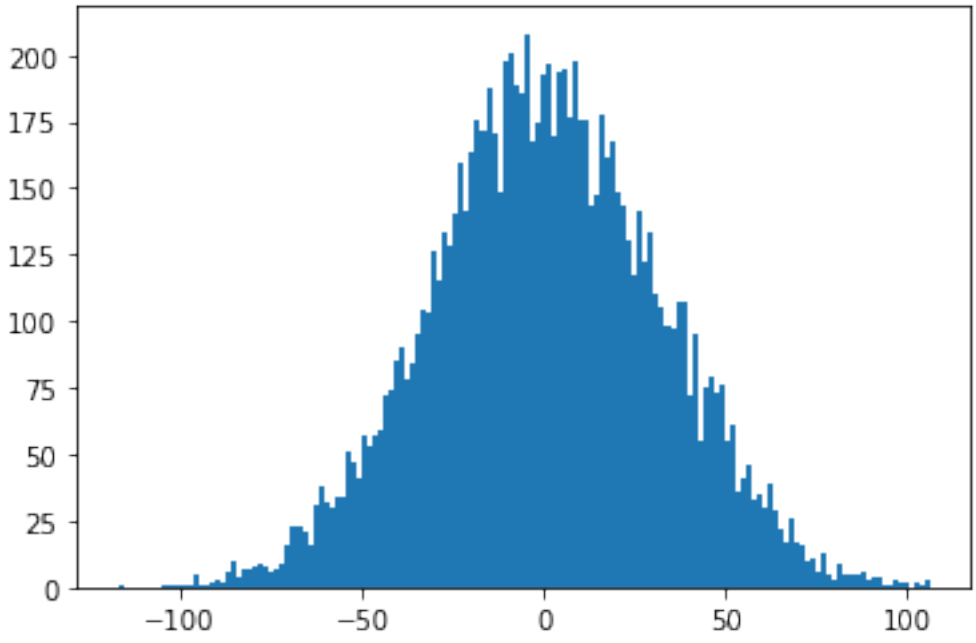
```
(array([  1.,   0.,   0.,   1.,   0.,   0.,   0.,   0.,   2.,   0.,
2.,
         2.,   2.,   0.,   2.,   4.,   2.,   4.,   2.,   4.,   3.,
3.,
         2.,   5.,   4.,   7.,   4.,  12.,  12.,  23.,  15.,  14.,
21.,
        35.,  28.,  26.,  36.,  43.,  39.,  25.,  40.,  48.,  69.,
66.,
        54.,  84.,  99.,  90.,  67., 105., 106.,  97., 109., 114.,
133.,
       127., 115., 148., 152., 134., 153., 182., 201., 178., 184.,
215.,
       204., 181., 199., 189., 201., 185., 218., 225., 231., 193.,
208.,
       208., 199., 201., 180., 178., 194., 167., 182., 203., 181.,
159.,
       168., 126., 148., 152., 132., 120., 111.,  91., 100.,  94.,
97.,
        73.,  84.,  71.,  56.,  67.,  64.,  50.,  54.,  42.,  46.,
30.,
        28.,  33.,  23.,  22.,  16.,  19.,  19.,  17.,  20.,  12.,
12.,
         9.,   3.,   7.,   9.,   5.,   4.,   3.,   1.,   3.,   4.,
3.,
         0.,   0.,   2.,   1.,   1.,   1.,   1.,   2.,   0.,   1.,
0.,
         1.,   0.,   0.,   0.,   0.,   0.,   1.]),
 array([-380.06509944, -374.88653565, -369.70797187, -364.52940808,
        -359.35084429, -354.1722805 , -348.99371671, -343.81515292,
        -338.63658913, -333.45802535, -328.27946156, -323.10089777,
        -317.92233398, -312.74377019, -307.5652064 , -302.38664261,
        -297.20807883, -292.02951504, -286.85095125, -281.67238746,
        -276.49382367, -271.31525988, -266.13669609, -260.95813231,
        -255.77956852, -250.60100473, -245.42244094, -240.24387715,
        -235.06531336, -229.88674957, -224.70818579, -219.529622  ,
        -214.35105821, -209.17249442, -203.99393063, -198.81536684,
        -193.63680305, -188.45823927, -183.27967548, -178.10111169,
        -172.9225479 , -167.74398411, -162.56542032, -157.38685653,
        -152.20829275, -147.02972896, -141.85116517, -136.67260138,
        -131.49403759, -126.3154738 , -121.13691001, -115.95834622,
        -110.77978244, -105.60121865, -100.42265486,  -95.24409107,
         -90.06552728,  -84.88696349,  -79.7083997 ,  -74.52983592,
         -69.35127213,  -64.17270834,  -58.99414455,  -53.81558076,
```

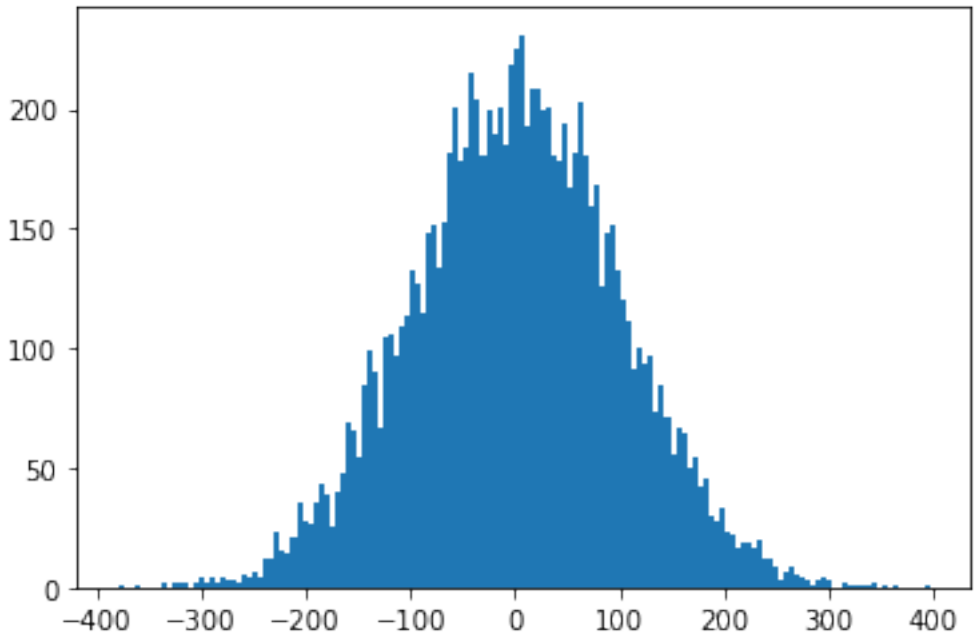
```
       -48.63701697,  -43.45845318,  -38.2798894 ,  -33.10132561,
       -27.92276182,  -22.74419803,  -17.56563424,  -12.38707045,
        -7.20850666,   -2.02994288,    3.14862091,    8.3271847 ,
        13.50574849,   18.68431228,   23.86287607,   29.04143986,
        34.22000364,   39.39856743,   44.57713122,   49.75569501,
        54.9342588 ,   60.11282259,   65.29138638,   70.46995016,
        75.64851395,   80.82707774,   86.00564153,   91.18420532,
        96.36276911,  101.5413329 ,  106.71989668,  111.89846047,
       117.07702426,  122.25558805,  127.43415184,  132.61271563,
       137.79127942,  142.9698432 ,  148.14840699,  153.32697078,
       158.50553457,  163.68409836,  168.86266215,  174.04122594,
       179.21978972,  184.39835351,  189.5769173 ,  194.75548109,
       199.93404488,  205.11260867,  210.29117246,  215.46973624,
       220.64830003,  225.82686382,  231.00542761,  236.1839914 ,
       241.36255519,  246.54111898,  251.71968276,  256.89824655,
       262.07681034,  267.25537413,  272.43393792,  277.61250171,
       282.7910655 ,  287.96962928,  293.14819307,  298.32675686,
       303.50532065,  308.68388444,  313.86244823,  319.04101202,
       324.21957581,  329.39813959,  334.57670338,  339.75526717,
       344.93383096,  350.11239475,  355.29095854,  360.46952233,
       365.64808611,  370.8266499 ,  376.00521369,  381.18377748,
       386.36234127,  391.54090506,  396.71946885]),
<BarContainer object of 150 artists>)
```




140
120
100
80
60
40
20
0
−3
−2
−1
0
1
2
3


200
150
100
50
0
−40
−30
−20
−10
0
10
20
30
40





5. Computing π using sampling

a) Generate 2D data from uniform distribution of range -1 to 1 and compute the value of π.

b) Equation of circle

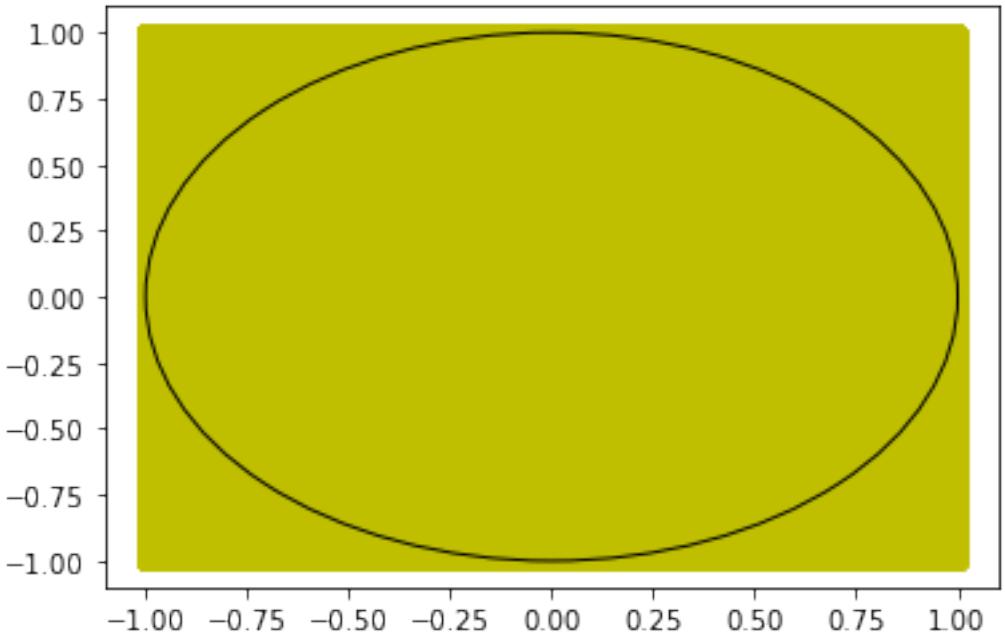
$$x^2 + y^2 = 1$$

c) Area of a circle can be written as:

$$\frac{No\,of\,points\left(x^2+y^2<¿1\right)}{Total no.generated\,points}=\frac{\pi r^2}{(2r)^2}$$

where r is the radius of the circle and $2r$ is the length of the vertices of square.

```
import numpy as np
import matplotlib.pyplot as plt
fig = plt.gcf()
ax = fig.gca()

radius = 1

n = 10000000 # set the value of n (select large n for better results)
x = np.random.uniform(-1,1,(n,2)) # Generate n samples of 2D data from
uniform distribution from range -1 to 1 (output will be a (n X 2)
matrix) (Ref =
https://numpy.org/doc/stable/reference/random/generated/numpy.random.u
niform.html )
ax.scatter(x[:,0],x[:,1],color='y') # Scatter plot of x

# find the number points present inside the circle



x_cr = [0 if x[i][0]**2 + x[i][1]**2 >= 1 else 1 for i in range(n)] #
insert your code here
x_cr = sum(x_cr)

circle1 = plt.Circle((0, 0), 1,fc='None',ec='k')
ax.add_artist(circle1) # plotting circle of radius 1 with centre at
(0,0)

pi = 4 * x_cr / n  # calculate pi value using x_cr and radius

print('computed value of pi=',pi)



computed value of pi= 3.141736
```



6. Monty Hall problem

Here's a fun and perhaps surprising statistical riddle, and a good way to get some practice writing python functions

In a gameshow, contestants try to guess which of 3 closed doors contain a cash prize (goats are behind the other two doors). Of course, the odds of choosing the correct door are 1 in 3. As a twist, the host of the show occasionally opens a door after a contestant makes his or her choice. This door is always one of the two the contestant did not pick, and is also always one of the goat doors (note that it is always possible to do this, since there are two goat doors). At this point, the contestant has the option of keeping his or her original choice, or swtiching to the other unopened door. The question is: is there any benefit to switching doors? The answer surprises many people who haven't heard the question before.

Follow the function descriptions given below and put all the functions together at the end to calculate the percentage of winning cash prize in both the cases (keeping the original door and switching doors)

Note : You can write your own functions, the below ones are given for reference, the goal is to calculate the win percentage

Try this fun problem and if you find it hard, you can refer to the solution here

```
"""
Function
--------
simulate_prizedoor

Generate a random array of 0s, 1s, and 2s, representing
```

```
hiding a prize between door 0, door 1, and door 2

Parameters
----------
nsim : int
    The number of simulations to run

Returns
-------
sims : array
    Random array of 0s, 1s, and 2s

Example
-------
>>> print simulate_prizedoor(3)
array([0, 0, 2])
"""
def simulate_prizedoor(nsim):

  answer = np.random.randint(0,3,nsim) # write your code here

  return answer

"""
Function
--------
simulate_guess

Return any strategy for guessing which door a prize is behind. This
could be a random strategy, one that always guesses 2, whatever.

Parameters
----------
nsim : int
    The number of simulations to generate guesses for

Returns
-------
guesses : array
    An array of guesses. Each guess is a 0, 1, or 2

Example
-------
>>> print simulate_guess(5)
array([0, 0, 0, 0, 0])
"""
#your code here

def simulate_guess(nsim):
```

```
  answer = np.ones(nsim)

  return answer
"""
Function
--------
goat_door

Simulate the opening of a "goat door" that doesn't contain the prize,
and is different from the contestants guess

Parameters
----------
prizedoors : array
    The door that the prize is behind in each simulation
guesses : array
    THe door that the contestant guessed in each simulation

Returns
-------
goats : array
    The goat door that is opened for each simulation. Each item is 0,
1, or 2, and is different
    from both prizedoors and guesses

Examples
--------
>>> print goat_door(np.array([0, 1, 2]), np.array([1, 1, 1]))
>>> array([2, 2, 0])
"""
# write your code here # Define a function and return the required
array

def monty_sim(prize_door, guess_door):
    answer = np.array([0]*prize_door.shape[0])

    for i in range(answer.shape[0]):
        possible_values = [0, 1, 2]
        possible_values.remove(prize_door[i])
        if prize_door[i] == guess_door[i]:
            answer[i] = possible_values[0]
        else:
            possible_values.remove(guess_door[i])
            answer[i] = possible_values[0]

    return answer
```

```
"""
Function
--------
switch_guess

The strategy that always switches a guess after the goat door is
opened

Parameters
----------
guesses : array
     Array of original guesses, for each simulation
goatdoors : array
     Array of revealed goat doors for each simulation

Returns
-------
The new door after switching. Should be different from both guesses
and goatdoors

Examples
--------
>>> print switch_guess(np.array([0, 1, 2]), np.array([1, 2, 1]))
>>> array([2, 0, 0])
"""
# write your code here # Define a function and return the required
array

def switch_the_door(guess_door, monty_door):
     return np.array([3 - guess_door[i] - monty_door[i] for i in
range(guess_door.shape[0])])

"""
Function
--------
win_percentage

Calculate the percent of times that a simulation of guesses is correct

Parameters
-----------
guesses : array
    Guesses for each simulation
prizedoors : array
    Location of prize for each simulation

Returns
--------
percentage : number between 0 and 100
    The win percentage
```

```
Examples
---------
>>> print win_percentage(np.array([0, 1, 2]), np.array([0, 0, 0]))
33.333
"""

def win_percentage(guesses, prizedoors):

  answer = 100 * (guesses == prizedoors).mean()

  return answer

## Put all the functions together here

nsim = 500000 # Number of simulations

## case 1 : Keep guesses
# write your code here (print the win percentage when keeping original
door)

def keep_guess():
    prz_door = simulate_prizedoor(nsim)
    guss_door = simulate_guess(nsim)
    ans = win_percentage(guss_door, prz_door)
    print(ans)


## case 2 : switch
# write your code here (print the win percentage when switching doors)

def switch_guess():
    prz_door = simulate_prizedoor(nsim)
    guss_door = simulate_guess(nsim)
    mnty_door = monty_sim(prz_door,guss_door)
    swtch_door = switch_the_door(guss_door,mnty_door)
    ans = win_percentage(swtch_door, prz_door)
    print(ans)


keep_guess()
switch_guess()
```

```
33.379999999999995
66.51
```