

▼ 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 theorem
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 your own code to obtain the results

▼ 1. Sampling from uniform distribution

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

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 N = 10 # Number of points (Example = 10)
5 X = np.random.uniform(0.0,1.0,N) # Generate N points from a uniform distribution range from [0 1] #
6 print(X)

[0.23545539 0.98240639 0.66763773 0.62811326 0.73568258 0.93165055
 0.82028595 0.32487405 0.8466936 0.10132811]

```

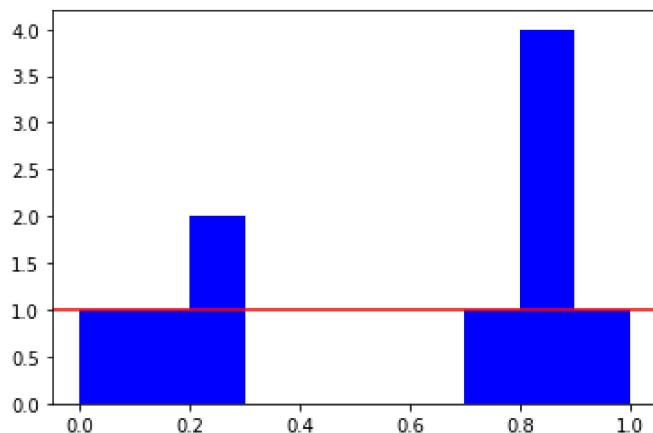
- b) Show with respect to no. of sample, how the sampled distribution converges to parent distribution.

```

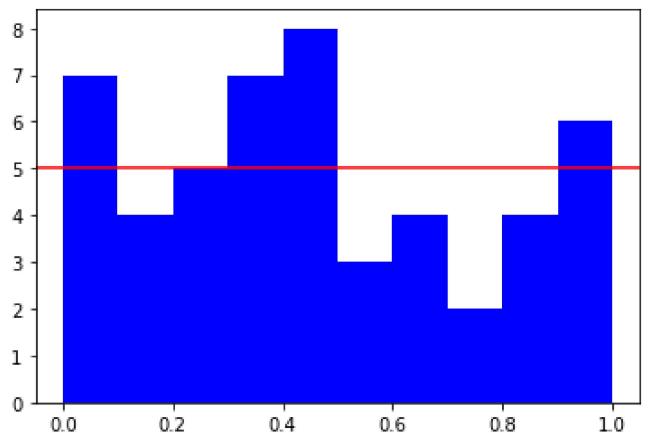
1 arr = [10, 50, 100, 500, 1000, 5000, 10000, 50000] # Create a numpy array of different values of no.
2
3 for i in arr:
4     x = np.random.uniform(0.0,1.0,i) # Generate i points from a uniform distribution range from [0 1]
5
6     plt.hist(x, color="blue", range=(0,1)) # Write the code to plot the histogram of the samples for all
7     plt.axhline(y = (i/10), color = 'r', linestyle = '-')
8     print("_____")
9     print("For number of samples = "+str(i))
10    plt.show()
11 print("The red line shows the level at which all the bars would have to be, for an 'ideal' uniform di

```

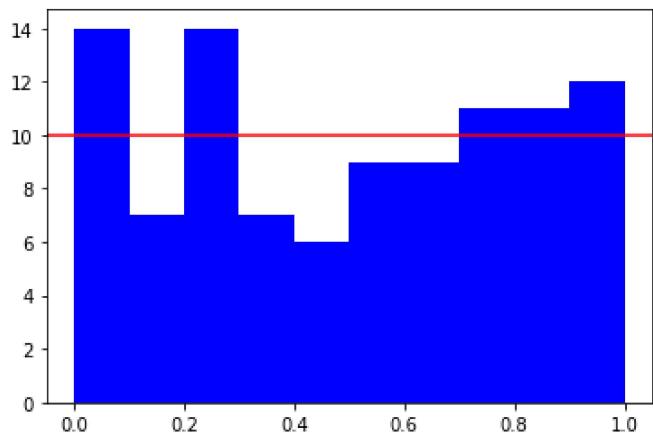
For number of samples = 10



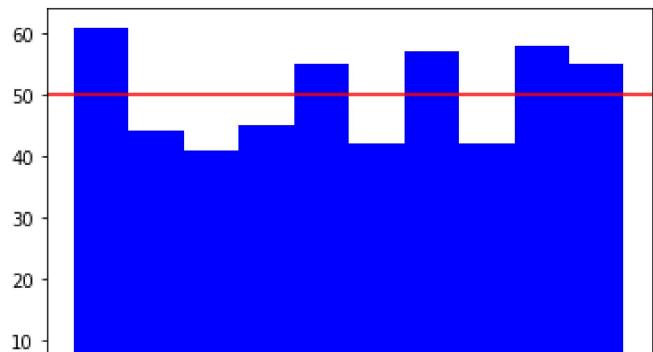
For number of samples = 50

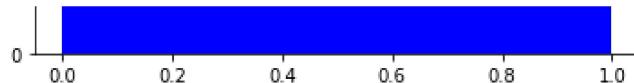


For number of samples = 100

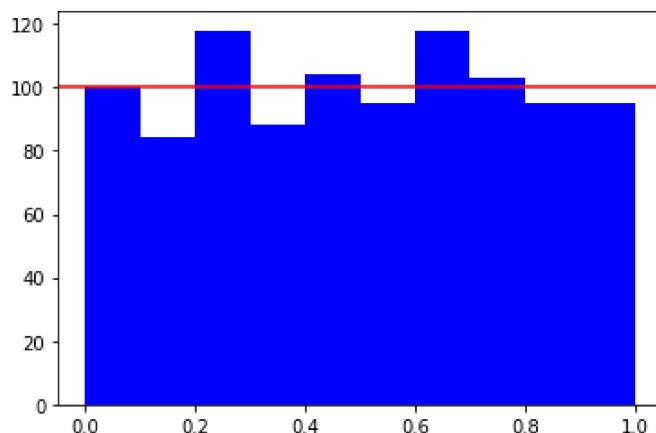


For number of samples = 500

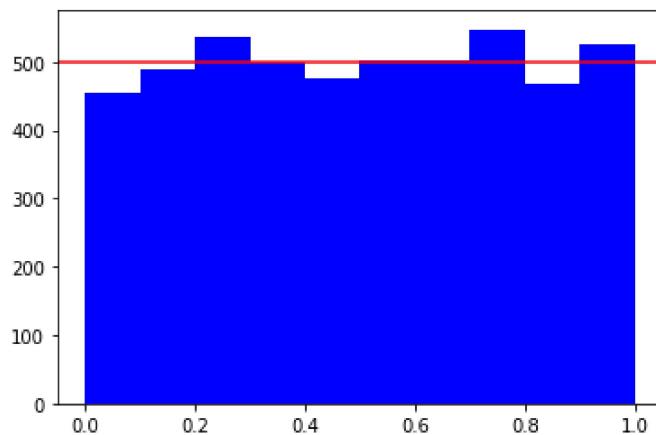




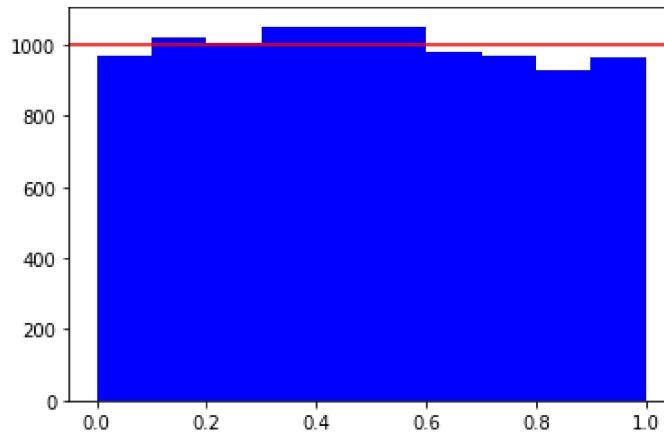
For number of samples = 1000



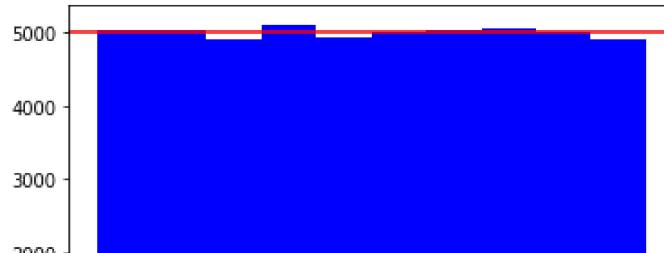
For number of samples = 5000



For number of samples = 10000



For number of samples = 50000



c) Law of large numbers: $\text{average}(x_{\text{sampled}}) = \bar{x}$, where x is a uniform random variable of range $[0,1]$, thus

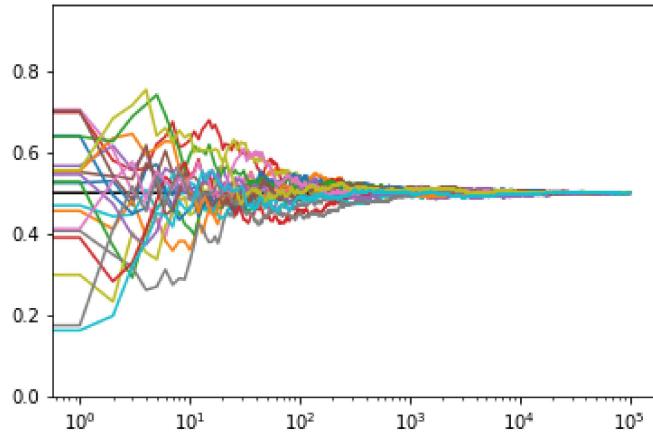
$$\bar{x} = \int_0^1 xf(x)dx = 0.5$$

u.u u.u u.u u.u u.u

```

1 N = 100000 # Number of points (>10000)
2 k = 20 # set a value for number of runs
3
4 ## Below code plots the semilog scaled on x-axis where all the samples are equal to the mean of distr
5 m = 0.5 # mean of uniform distribution
6 m = np.tile(m,x.shape)
7 plt.semilogx(m,color='k') # Ref : https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.semilog
8
9 for j in range(k):
10
11 i = np.arange(1, N+1, 1) # Generate a list of numbers from (1,N) # Ref : https://numpy.org/doc/sta
12 x = np.random.uniform(0.0,1.0,N) # Generate N points from a uniform distribution range from [0 1]
13 mean_sampled = np.cumsum(x)/(i) # Ref : https://numpy.org/doc/stable/reference/generated/numpy.cums
14 ## Write code to plot semilog scaled on x-axis of mean_sampled, follow the above code of semilog fo
15 plt.semilogx(mean_sampled)
16
17

```



▼ 2. Sampling from Gaussian Distribution

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

```

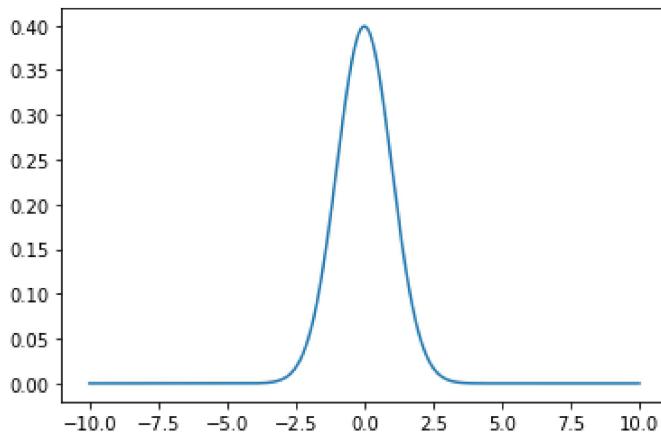
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 X = np.linspace(-10, 10, 1000) # Generate 1000 points from -10 to 10 # Ref : https://numpy.org/doc/s
5 # Define mean and variance
6 mean = 0
7 variance = 1
8
9 X_n = ((X-mean)**2)/(2*variance)
10 gauss_distribution = (1/np.sqrt(variance*np.pi*2))*np.exp(-X_n) # Define univariate gaussian distrib
11
12 ## Write code to plot the above distribution # Ref : https://matplotlib.org/stable/api/_as_gen/matplc
13 plt.plot(X,gauss_distribution)

```

```

14 plt.show()
15
16 print("The above is a 'normal' distribution, with mean zero and unit variance. It is a special case of the univariate Gaussian distribution")

```



The above is a 'normal' distribution, with mean zero and unit variance. It is a special case of the univariate Gaussian distribution.

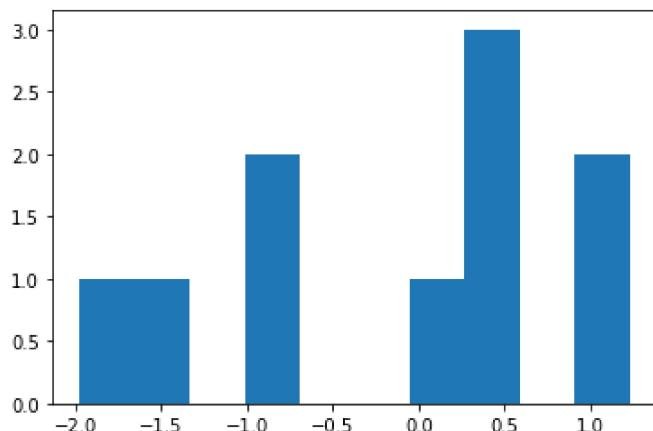
b) Sample from a univariate Gaussian distribution, observe the shape by changing the no. of sample drawn.

```

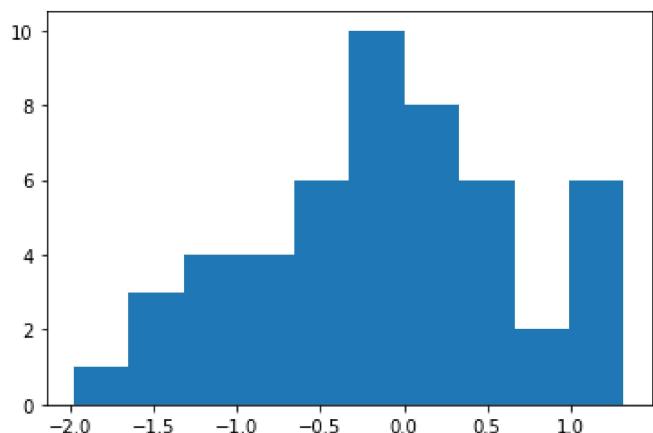
1 from random import gauss
2
3
4 arr = [10, 50, 100, 500, 1000, 5000, 10000, 50000] # Create a numpy array of different values of no. of samples
5
6 for i in arr:
7     x_sampled = np.random.normal(size=i) # Generate i samples from univariate gaussian distribution
8
9     variance = np.var(x_sampled)
10
11    # optimal bin width
12    range_x = max(x_sampled) - min(x_sampled)
13    num_bins = int(range_x*(x_sampled.size**((1/3)))/(3.49))
14    bin_arr = np.linspace(int(min(x_sampled)), int(max(x_sampled)), num_bins)
15
16    # write the code to plot the histogram of the samples for all values in arr
17    print("For number of samples = " + str(i))
18    if(i < 250):
19        plt.hist(x_sampled, bins=10)
20    else:
21        plt.hist(x_sampled, bins = bin_arr, range = (-variance, variance))
22
23 plt.show()

```

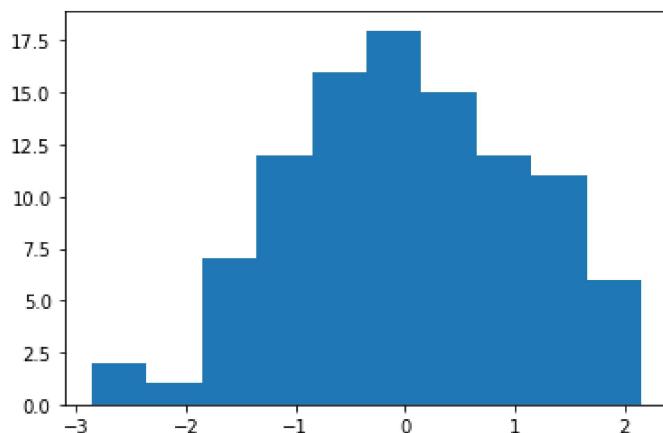
For number of samples = 10



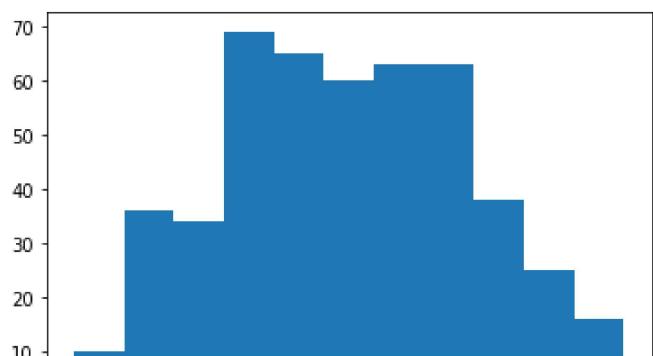
For number of samples = 50

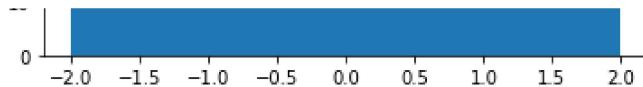


For number of samples = 100

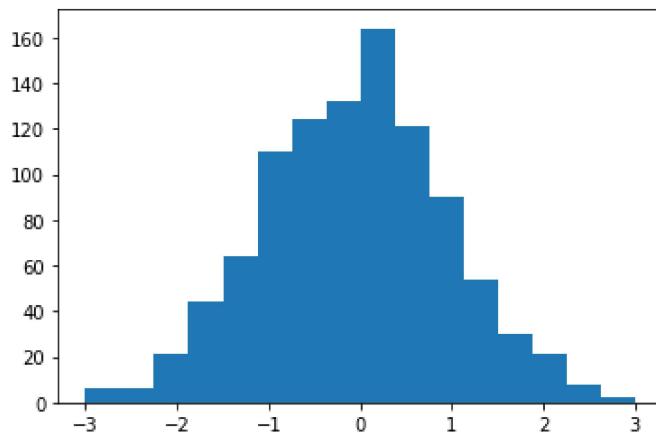


For number of samples = 500

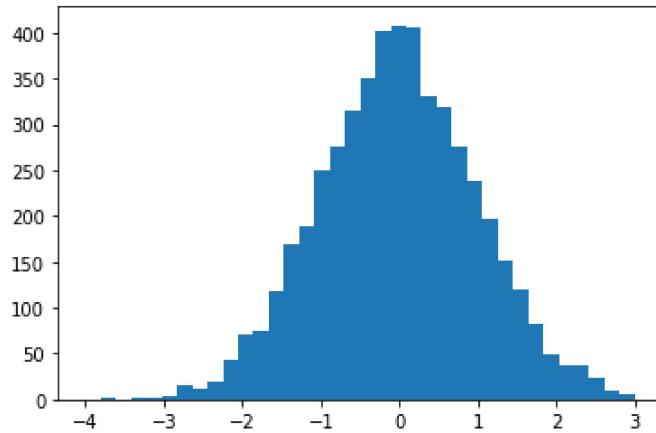




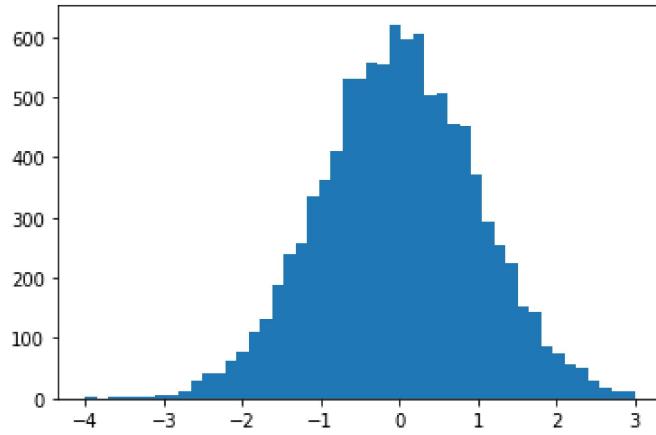
For number of samples = 1000



For number of samples = 5000



For number of samples = 10000



For number of samples = 50000



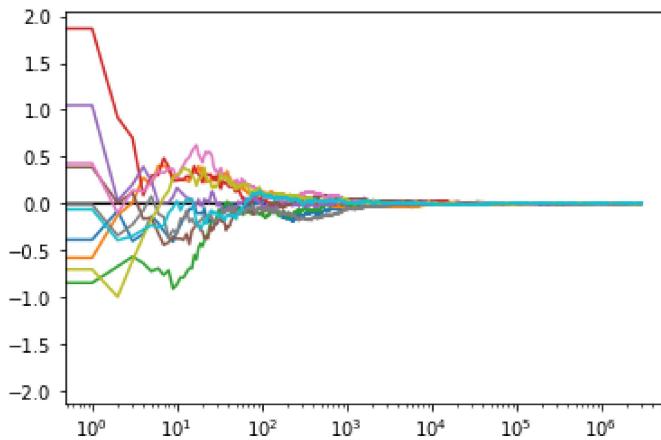
c) Law of large number

```
1 N = 3000000 # Number of points (>1000000)
```

```

2 k = 10 # set a value for number of distributions
3
4 ## Below code plots the semilog when all the samples are equal to the mean of distribution
5
6 m = np.tile(mean,x.shape)
7 plt.semilogx(m,color='k')
8
9 for j in range(k):
10
11 i = np.arange(1, N+1, 1) # Generate a list of numbers from (1,N)
12 x = np.random.normal(0.0, 1.0, N) # Generate N samples from univariate gaussian distribution # Ref
13 mean_sampled = np.cumsum(x)/(i)# insert your code here (Hint : Repeat the same steps as in the uni
14
15 ## Write code to plot semilog scaled on x axis of mean_sampled, follow the above code of semilog fo
16 plt.semilogx(mean_sampled)
17
18

```



▼ 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 occurrences and divide by the number of total draws for 3 scenarios :
1. $p_0 < prob_0$
 2. $p_1 < prob_1$
 3. $p_2 < prob_2$

```

1 n = 2000000 # Number of points (>1000000)
2 y = np.random.uniform(0, 1, n) # Generate n points from uniform distribution range from [0 1]
3 x = np.arange(1, n+1)
4 prob0 = 0.3
5 prob1 = 0.6
6 prob2 = 0.1
7
8 # count number of occurrences and divide by the number of total draws
9 p0 = np.cumsum([1 if (k <= prob0) else 0 for k in y])/(x) # insert your code here

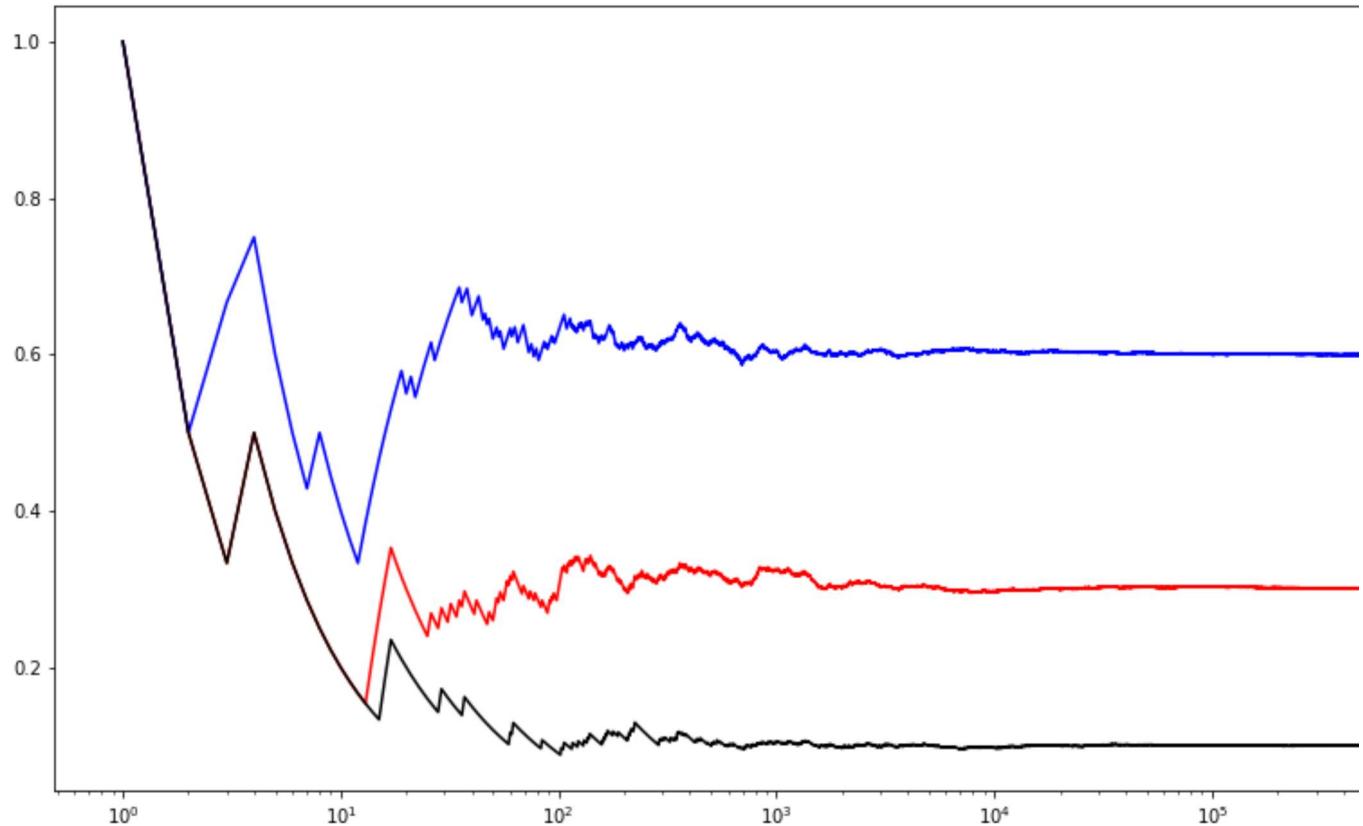
```

```

10 p1 = np.cumsum([1 if (k <= prob1) else 0 for k in y])/(x) # insert your code here
11 p2 = np.cumsum([1 if (k <= prob2) else 0 for k in y])/(x) # insert your code here
12
13
14 plt.figure(figsize=(15, 8))
15 plt.semilogx(x, p0,color='r')
16 plt.semilogx(x, p1,color='b')
17 plt.semilogx(x, p2,color='k')
18 plt.legend(['-p0-','-p1-','-p2-'])
19

```

<matplotlib.legend.Legend at 0x7f3a4a11b290>



▼ 4. Central limit theorem

a) Sample from a uniform distribution (-1,1), some 10000 no. of samples 1000 times ($u_1, u_2, \dots, u_{1000}$). show addition of iid random variables converges to a Gaussian distribution as number of variables tends to infinity.

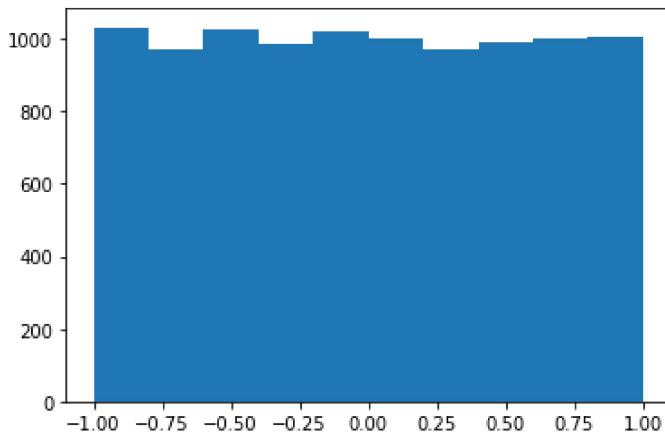
```

1 x = np.random.uniform(-1, 1, (10000, 1000)) # Generate 1000 different uniform distributions of 1000 s
2
3 # one uniform RV
4 plt.figure()
5 plt.hist(x[:,0])
6 print("_____")
7 print("1 Uniform RV:")
8 plt.show()
9
10 # iid = Independent and identically distributed
11 # addition of 2 random variables
12

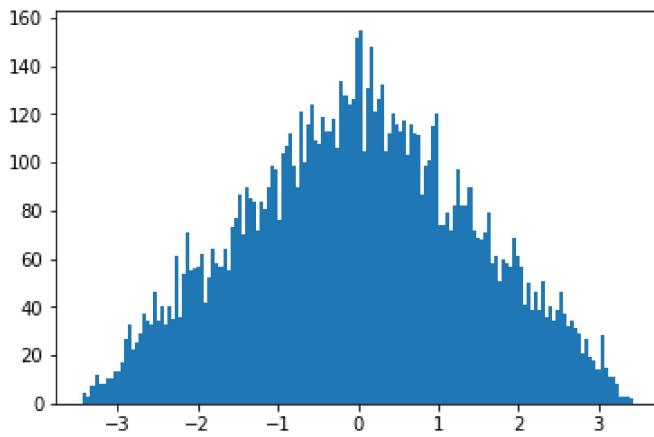
```

```
13 tmp2=np.sum(x[:,0:2],axis=1)/(np.std(x[:,0:2]))
14 plt.figure()
15 plt.hist(tmp2,150)
16 print("____")
17 print("Sum of 2 IID RVs:")
18 plt.show()
19
20 # Repeat the same for 100 and 1000 random variables
21
22 # addition of 100 random variables
23 # start code here
24
25 tmp3=np.sum(x[:,0:100],axis=1)/(np.std(x[:,0:2]))
26 plt.figure()
27 plt.hist(tmp3,150)
28 print("____")
29 print("Sum of 100 IID RVs:")
30 plt.show()
31
32 # addition of 1000 random variables
33 # start code here
34
35 tmp4=np.sum(x[:,0:1000],axis=1)/(np.std(x[:,0:2]))
36 plt.figure()
37 plt.hist(tmp4,150)
38 print("____")
39 print("Sum of 1000 IID RVs:")
40 plt.show()
41
```

1 Uniform RV:



Sum of 2 IID RVs:



Sum of 100 IID RVs:

250 ↴

▼ 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:

$$\frac{\text{No of points } (x^2 + y^2 \leq 1)}{\text{Total no. generated points}} = \frac{\pi r^2}{(2r)^2}$$

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

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 fig = plt.gcf()
4 ax = fig.gca()
5
6 radius = 1
7
8 # for 10^7, takes 30s, gets pi = 3.14136
9 # for 10^6, takes 3s, gets pi = 3.14292

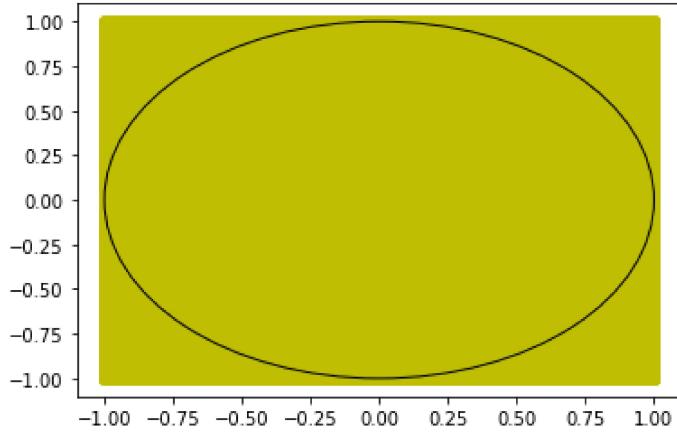
```

```

10 # for 10^5, takes .5s, gets pi = 3.13424
11
12 n = 1000000 # set the value of n (select large n for better results)
13 x = np.random.uniform(-1,1,(n,2)) # Generate n samples of 2D data from range
14
15 ax.scatter(x[:,0],x[:,1],color='y') # Scatter plot of x
16
17 # find the number points present inside the circle
18 x_cr = sum((x[:,0]**2 + x[:,1]**2 ) <= 1) # insert your code here
19
20 circle1 = plt.Circle((0, 0), 1, fc='None', ec='k')
21 ax.add_artist(circle1) # plotting circle of radius 1 with centre at (0,0)
22
23 pi = x_cr*4/n # calculate pi value using x_cr and radius
24
25 print('computed value of pi=',pi)
26
27
28

```

computed value of pi= 3.144936



▼ 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 switching 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

```
1 """
2 Function
3 -----
4 simulate_prizedoor
5
6 Generate a random array of 0s, 1s, and 2s, representing
7 hiding a prize between door 0, door 1, and door 2
8
9 Parameters
10 -----
11 nsim : int
12     The number of simulations to run
13
14 Returns
15 -----
16 sims : array
17     Random array of 0s, 1s, and 2s
18
19 Example
20 -----
21 >>> print simulate_prizedoor(3)
22 array([0, 0, 2])
23 """
24 def simulate_prizedoor(nsim):
25
26     answer = np.random.randint(0,3,nsim)
27
28     return answer
```

```
1 """
2 Function
3 -----
4 simulate_guess
5
6 Return any strategy for guessing which door a prize is behind. This
7 could be a random strategy, one that always guesses 2, whatever.
8
9 Parameters
10 -----
11 nsim : int
12     The number of simulations to generate guesses for
13
14 Returns
15 -----
16 guesses : array
17     An array of guesses. Each guess is a 0, 1, or 2
18
19 Example
20 -----
21 >>> print simulate_guess(5)
22 array([0, 0, 0, 0, 0])
23 """
```

```

24 #your code here
25
26 def simulate_guess(nsim):
27
28     answer = np.random.randint(0,3,nsim)
29
30     return answer

1 """
2 Function
3 -----
4 goat_door
5
6 Simulate the opening of a "goat door" that doesn't contain the prize,
7 and is different from the contestants guess
8
9 Parameters
10 -----
11 prizedoors : array
12     The door that the prize is behind in each simulation
13 guesses : array
14     The door that the contestant guessed in each simulation
15
16 Returns
17 -----
18 goats : array
19     The goat door that is opened for each simulation. Each item is 0, 1, or 2, and is different
20     from both prizedoors and guesses
21
22 Examples
23 -----
24 >>> print goat_door(np.array([0, 1, 2]), np.array([1, 1, 1]))
25 >>> array([2, 2, 0])
26 """

27 # write your code here # Define a function and return the required array
28 def goat_door(prizedoors, guesses):
29     goats = []
30     for (x,y) in zip(prizedoors, guesses):
31         if(x != y):
32             goats.append(int(3-x-y))          # reason : firstly, the property of choice is commutative
33                                         # (1,2) , (2,1) => 0 ; (1,0) , (0,1) => 2 ; (0,2) , (2,0) => 1
34         else:                            # adding ... 3 => 0,
35             init_array = [0,1,2]           # could've used the removal method for the x!=y case
36             init_array.remove(x)
37             i = np.random.randint(0,2)
38             goats.append(i)
39     return goats
40

```

```

1 """
2 Function
3 -----
4 switch_guess
5
6 The strategy that always switches a guess after the goat door is opened
7

```

```

8 Parameters
9 -----
10 guesses : array
11     Array of original guesses, for each simulation
12 goatdoors : array
13     Array of revealed goat doors for each simulation
14
15 Returns
16 -----
17 The new door after switching. Should be different from both guesses and goatdoors
18
19 Examples
20 -----
21 >>> print switch_guess(np.array([0, 1, 2]), np.array([1, 2, 1]))
22 >>> array([2, 0, 0])
23 """
24 # write your code here # Define a function and return the required array
25 def switch_guess(guesses, goatdoors):
26     switched = []
27     for (x,y) in zip(guesses, goatdoors):
28         switched.append(int(3-x-y))
29     return switched

```

```

1 """
2 Function
3 -----
4 win_percentage
5
6 Calculate the percent of times that a simulation of guesses is correct
7
8 Parameters
9 -----
10 guesses : array
11     Guesses for each simulation
12 prizedoors : array
13     Location of prize for each simulation
14
15 Returns
16 -----
17 percentage : number between 0 and 100
18     The win percentage
19
20 Examples
21 -----
22 >>> print win_percentage(np.array([0, 1, 2]), np.array([0, 0, 0]))
23 33.333
24 """
25
26 def win_percentage(guesses, prizedoors):
27
28     answer = 100 * (guesses == prizedoors).mean()
29
30     return answer

```

```

1 ## Put all the functions together here
2 # + list of func: simulate_prizedoor, simulate_guess, goat_door, switch_guess, win_percentage

```

```
<# T1ST OT THS: simulate_prizedoor, simulate_guess, goat_door, switch_guess, win_percentage
3
4 # 10^7 simulations takes ~15s
5 nsim = 10000000 # Number of simulations
6
7 ## case 1 : Keep guesses
8 # write your code here (print the win percentage when keeping original door)
9 prizedoors = simulate_prizedoor(nsim)
10 guesses = simulate_guess(nsim)
11 # goats = goat_door(prizedoors, guesses)           # this line is actually useless here !!!
12 w_stay = win_percentage(guesses, prizedoors)
13
14 print("On staying, win percentage is: "+str(w_stay))
15
16 ## case 2 : switch
17 # write your code here (print the win percentage when switching doors)
18 prizedoors = simulate_prizedoor(nsim)
19 guesses = simulate_guess(nsim)
20 goats = goat_door(prizedoors, guesses)
21 new_guess = switch_guess(guesses, goats)
22 w_switch = win_percentage(new_guess, prizedoors)
23 print("On switching, win percentage is: "+str(w_switch))
```

On staying, win percentage is: 33.33109

On switching, win percentage is: 72.21873

1