Math Primer

1. Solution of the system of equations
2. Gradient descent
3. Sampling

1) Solution of the system of equations (Ax=b)

a. Unique solution (A is full rank and m=n)

```
Reference: https://docs.scipy.org/doc/numpy-1.10.1/reference/routines.lin
alg.html
```

If A is full rank (i.e determinant is non zero) and m=n (i.e a square matrix), ***x***=A^{-1}***b***

Ex-a:i) Equations to solve:

$$x + y = 5$$
$$2x + 4y = 4$$

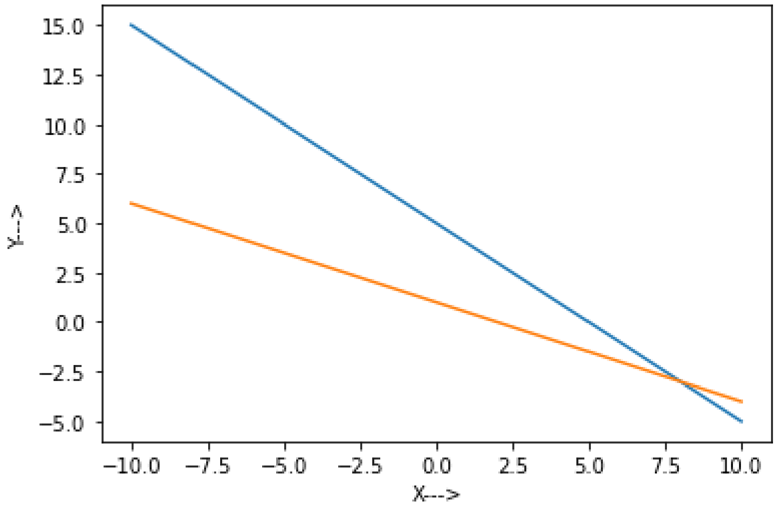
Ex-a:ii) Equations to solve:

$$x + y + z = 5$$
$$2x + 4y + z = 4$$
$$x + 3y + 4z = 4$$

In [ ]:

```
# Ex-a:i)
import numpy as np
import matplotlib.pyplot as plt

# Defining matrixes
A = np.matrix([[1,1], [2,4]])
b = np.matrix([[5], [4]])
print('A=',A,'\n')
print('b=',b,'\n')
# Taking Determinant of A (to show full rank)
Det=#  write your code to find the determinant of A here (use the help of refere
nce link )
print('Determinant=',Det,'\n')
#or direct matrix rank
print('Matrix rank=',np.linalg.matrix_rank(A),'\n')
# Taking inverse of A
A_inverse = # write your code to find the Inverse  of matrix A here
print('A_inverse=',A_inverse,'\n')

# Solving for X
x_op = # insert your code here (x_op=A^{-1}*b)
print('x=',x_op,'\n')

#ploting
def f(x,A_coff,b_coff):
  return ((b_coff-A_coff[0]*x)/A_coff[1]).T ## (b-A)


X=np.linspace(-10,10,100)
Y1=f(X,A[0,:].T,b[0]) # transpose is used to make same size
Y2=# insert your code to find Y2 values, (similar to Y1)
%matplotlib inline
plt.plot(X,Y1)
plt.plot(X,Y2)
plt.xlabel('X--->')
plt.ylabel('Y--->');

# validate
error=# insert your code here   error=(b-A*x)
print('error=',error,'\n')
```

```
A= [[1 1]
 [2 4]]

b= [[5]
 [4]]

Determinant= 2.0

Matrix rank= 2

A_inverse= [[ 2.  -0.5]
 [-1.   0.5]]

x= [[ 8.]
 [-3.]]

error= [[0.]
 [0.]]
```



In [ ]:

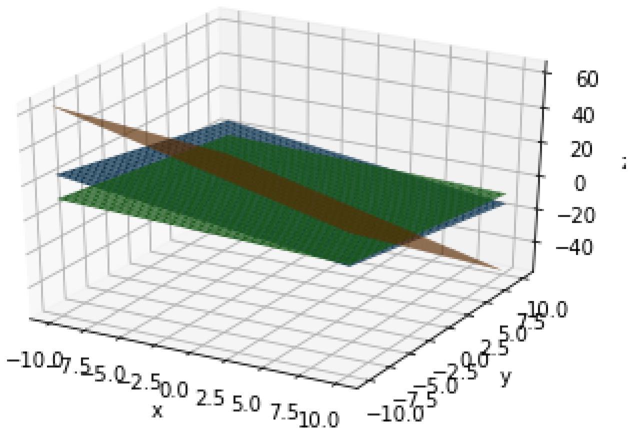
```
# Ex-a:ii)
import numpy as np


# Defining matrixes
A = np.matrix([[1,1,1], [2,4,1],[1, 3, 4]])
b = np.matrix([[5], [4],[4]])
print('A=',A,'\n')
print('b=',b,'\n')
# Taking Determinant of A (to show full rank)
Det=#  write your code to find the determinant of A here (use the help of refere
nce link )
print('Determinant=',Det,'\n')
#or direct matrix rank
rank=# write your code to find the rank of matrix A here
print('Matrix rank=',rank,'\n')
# Taking inverse of A
A_inverse = # write your code to find the Inverse  of matrix A here
print('A_inverse=',A_inverse,'\n')

# Solving for X
x_op = # insert your code here (x_op=A^{-1}*b)
print('x=',x_op,'\n')

# validate
error=# insert your code here   error=(b-A*x)
print('error=',error,'\n')
```

```
A= [[1 1 1]
 [2 4 1]
 [1 3 4]]

b= [[5]
 [4]
 [4]]

Determinant= 7.999999999999998

Matrix rank= 3

A_inverse= [[ 1.625 -0.125 -0.375]
 [-0.875  0.375  0.125]
 [ 0.25  -0.25   0.25 ]]

x= [[ 6.125]
 [-2.375]
 [ 1.25 ]]

error= [[0.]
 [0.]
 [0.]]
```

Plotting in a 3D plane:

In [ ]:

```
from mpl_toolkits import mplot3d
# %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
def f(x, y,coff):

    return # Write your expression here # ax+by+cz=d, z=(d-ax-by)/c

x = np.linspace(-10, 10, 100)
y = np.linspace(-10, 10, 100)

X, Y = np.meshgrid(x, y)
coff1=np.array([1,1,1,5])
coff2=np.array([2,4,1,4])
coff3=np.array([1,3,4,4])
Z1 = f(X, Y,coff1)
Z2= f(X, Y,coff2)
Z3= # insert your code here (like Z1 and Z2)
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z1)
ax.plot_surface(X, Y, Z2)
ax.plot_surface(X, Y, Z3)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z');
```



b. A is full rank, m>n(3,2) (b lies in the span,unique solution else, No solution, least square solution)

Ex-b:i)

$$x + z = 0$$
$$x + y + z = 0$$
$$y + z = 0$$
$$z = 0$$

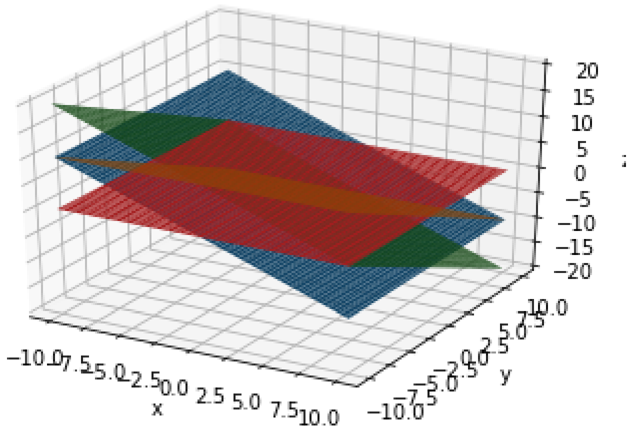
Ex-b:ii)

$$x + y + z = 35$$
$$2x + 4y + z = 94$$
$$x + 3y + 4z = 4$$
$$x + 9y + 4z = -230$$

In [ ]:

```
#Ex-b:i)
# Defining matrixes
A = np.matrix([[1,0,1], [1,1,1],[0, 1, 1],[0, 0,1]])
b = np.matrix([[0], [0],[0], [0]])
print('A=',A,'\n')
print('b=',b,'\n')


#or direct matrix rank
rank=# write your code to find the rank of matrix A here
print('Matrix rank=',rank,'\n')

# Taking inverse of A
A_inverse =  # write your code to find the psudoinverse  of matrix A here # not
 square so psudoinverse
print('A_inverse=',A_inverse,'\n')

# Solving for X
x_op = # insert your code here (x_op=A^{-1}*b)
print('x=',x_op,'\n')

# validate
error=# insert your code here   error=(b-A*x)
print('error=',error,'\n')

# plotting
# from mpl_toolkits import mplot3d
# %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
def f(x, y,coff):

    return # Write your expression here # ax+by+cz=d, z=(d-ax-by)/c

x = np.linspace(-10, 10, 100)
y = np.linspace(-10, 10, 100)

X, Y = np.meshgrid(x, y)
coff1=np.array([1,0,1,0])
coff2=np.array([0,1,1,0])
coff3=np.array([1,1,1,0])
coff4=np.array([0,0,1,0])
Z1 = f(X, Y,coff1)
Z2= f(X, Y,coff2)
Z3= f(X, Y,coff3)
Z4=f(X,Y,coff4)
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z1)
ax.plot_surface(X, Y, Z2)
ax.plot_surface(X, Y, Z3)
ax.plot_surface(X, Y, Z4)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z');
```

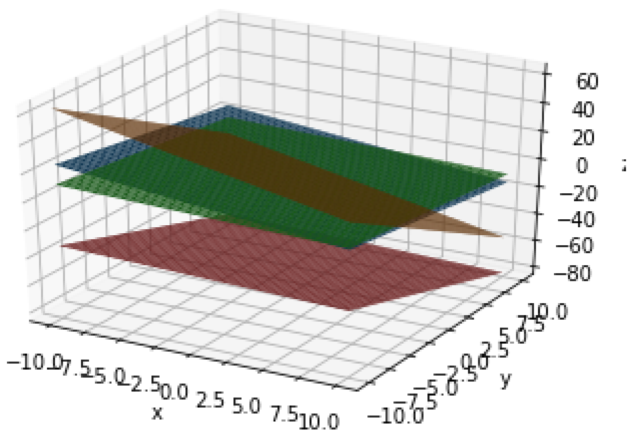
```
A= [[1 0 1]
 [1 1 1]
 [0 1 1]
 [0 0 1]]

b= [[0]
 [0]
 [0]
 [0]]

Matrix rank= 3

A_inverse= [[ 0.5   0.5  -0.5  -0.5 ]
 [-0.5   0.5   0.5  -0.5 ]
 [ 0.25 -0.25  0.25  0.75]]

x= [[0.]
 [0.]
 [0.]]

error= [[0.]
 [0.]
 [0.]
 [0.]]
```



In [ ]:

```
#Ex-b:ii) least square solution

# Defining matrixes
A = np.matrix([[1,1,1], [2,4,1],[1, 3, 4],[1,9,4]])
b = np.matrix([[5], [4],[4],[-230]])
print('A=',A,'\n')
print('b=',b,'\n')


#or direct matrix rank
rank=# write your code to find the rank of matrix A here
print('Matrix rank=',rank,'\n')

# Taking inverse of A
A_inverse =  # write your code to find the psudoinverse  of matrix A here # not
 square so psudoinverse
print('A_inverse=',A_inverse,'\n')

# Solving for X
x_op = # insert your code here (x_op=A^{-1}*b)
print('x=',x_op,'\n')

# validate
error=# insert your code here   error=(b-A*x)
print('error=',error,'\n')

# plotting
# from mpl_toolkits import mplot3d
# %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
def f(x, y,coff):

    return # Write your expression here # ax+by+cz=d, z=(d-ax-by)/c

x = np.linspace(-10, 10, 100)
y = np.linspace(-10, 10, 100)

X, Y = np.meshgrid(x, y)
coff1=np.array([1,1,1,5])
coff2=np.array([2,4,1,4])
coff3=np.array([1, 3, 4,4])
coff4=np.array([1,9,4,-230])
Z1 = f(X, Y,coff1)
Z2= f(X, Y,coff2)
Z3= f(X, Y,coff3)
Z4=f(X,Y,coff4)
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z1)
ax.plot_surface(X, Y, Z2)
ax.plot_surface(X, Y, Z3)
ax.plot_surface(X, Y, Z4)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z');
```

```
A= [[1 1 1]
 [2 4 1]
 [1 3 4]
 [1 9 4]]

b= [[   5]
 [   4]
 [   4]
 [-230]]

Matrix rank= 3

A_inverse= [[ 0.27001704  0.45570698  0.07666099 -0.25809199]
 [-0.06558773  0.02810903 -0.14480409  0.15417376]
 [ 0.04429302 -0.16183986  0.31856899 -0.03918228]]

x= [[ 62.8407155 ]
 [-36.25468484]
 [  9.86030664]]

error= [[-31.44633731]
 [ 13.4770017 ]
 [ 10.48211244]
 [ -5.98977853]]
```



c. A is not full rank (if b lies in the span, multiple solution else No solution (i.e. least square solution)).

Ex-c:i)

$$x + y + z = 0$$
$$3x + 3y + 3z = 0$$
$$x + 2y + z = 0$$

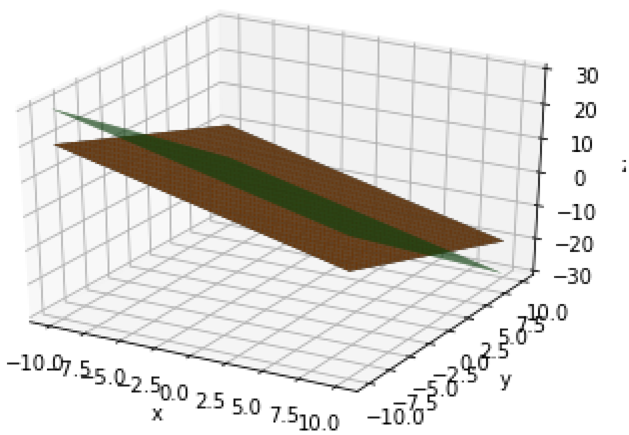
Ex-c:ii)

$$x + y + z = 0$$
$$3x + 3y + 3z = 2$$
$$x + 2y + z = 0$$

In [ ]:

```
#Ex-c:i) multiple solution

# Defining matrixes
A = np.matrix([[1,1,1], [3,3,3],[1, 2, 1]])
b = np.matrix([[0], [0],[0]])
print('A=',A,'\n')
print('b=',b,'\n')


#or direct matrix rank
rank=# write your code to find the rank of matrix A here
print('Matrix rank=',rank,'\n')

# Taking inverse of A
A_inverse =  # write your code to find the psudoinverse  of matrix A here # not
 square so psudoinverse
print('A_inverse=',A_inverse,'\n')

# Solving for X
x_op = # insert your code here (x_op=A^{-1}*b)
print('x=',x_op,'\n')

# validate
error=# insert your code here   error=(b-A*x)
print('error=',error,'\n')

# plotting
# from mpl_toolkits import mplot3d
# %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
def f(x, y,coff):

    return # Write your expression here # ax+by+cz=d, z=(d-ax-by)/c

x = np.linspace(-10, 10, 100)
y = np.linspace(-10, 10, 100)

X, Y = np.meshgrid(x, y)

coff1=np.array([1,1,1,0])
coff2=np.array([3,3,3,0])
coff3=np.array([1, 2, 1,0])

Z1 = f(X, Y,coff1)
Z2= f(X, Y,coff2)
Z3= f(X, Y,coff3)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z1)
ax.plot_surface(X, Y, Z2)
ax.plot_surface(X, Y, Z3)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z');
```

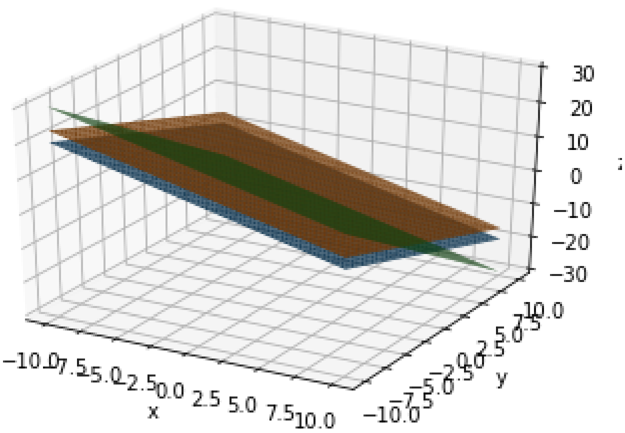
```
A= [[1 1 1]
 [3 3 3]
 [1 2 1]]

b= [[0]
 [0]
 [0]]

Matrix rank= 2

A_inverse= [[ 0.1  0.3 -0.5]
 [-0.1 -0.3  1. ]
 [ 0.1  0.3 -0.5]]

x= [[0.]
 [0.]
 [0.]]

error= [[0.]
 [0.]
 [0.]]
```



In [ ]:

```
#Ex-c:ii) least square solution (b is not in range)

# Defining matrixes
A = np.matrix([[1,1,1], [3,3,3],[1, 2, 1]])
b = np.matrix([[0], [10],[0]])
print('A=',A,'\n')
print('b=',b,'\n')


#or direct matrix rank
rank=# write your code to find the rank of matrix A here
print('Matrix rank=',rank,'\n')

# Taking inverse of A
A_inverse =  # write your code to find the psudoinverse  of matrix A here # not
 square so psudoinverse
print('A_inverse=',A_inverse,'\n')

# Solving for X
x_op = # insert your code here (x_op=A^{-1}*b)
print('x=',x_op,'\n')

# validate
error=# insert your code here   error=(b-A*x)
print('error=',error,'\n')

# plotting
# from mpl_toolkits import mplot3d
# %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
def f(x, y,coff):

    return # Write your expression here # ax+by+cz=d, z=(d-ax-by)/c

x = np.linspace(-10, 10, 100)
y = np.linspace(-10, 10, 100)

X, Y = np.meshgrid(x, y)
coff1=np.array([1,1,1,0])
coff2=np.array([3,3,3,10])
coff3=np.array([1, 2, 1,0])

Z1 = f(X, Y,coff1)
Z2= f(X, Y,coff2)
Z3= f(X, Y,coff3)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z1)
ax.plot_surface(X, Y, Z2)
ax.plot_surface(X, Y, Z3)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z');
```

```
A= [[1 1 1]
 [3 3 3]
 [1 2 1]]

b= [[ 0]
 [10]
 [ 0]]

Matrix rank= 2

A_inverse= [[ 0.1  0.3 -0.5]
 [-0.1 -0.3  1. ]
 [ 0.1  0.3 -0.5]]

x= [[ 3.]
 [-3.]
 [ 3.]]

error= [[-3.00000000e+00]
 [ 1.00000000e+00]
 [-3.55271368e-15]]
```



Excersise question (find the solutions and justify the type):

1.$2x+3y+5z=2, 9x+3y+2z=5, 5x+9y+z=7$

2.$2x+3y=1, 5x+9y=4, x+y=0$

3.$2x+5y+10z=0, 9x+2y+z=1, 4x+10y+20z=5$

4.$2x+3y=0, 5x+9y=2, x+y=-2$

5.$2x+5y+3z=0, 9x+2y+z=0, 4x+10y+6z=0$

In [ ]:

```
import numpy as np
A=np.matrix([[2,3,5],[9,3,2],[5,9,1]])
print(A)
b=np.matrix([[2],[5],[7]])

print('Matrix rank=',np.linalg.matrix_rank(A),'\n')

x=np.linalg.inv(A) * b
print(x.shape)
print(A * x)

error=b-A*x
print('error=',error,'\n')
```

```
[[2 3 5]
 [9 3 2]
 [5 9 1]]
Matrix rank= 3

(3, 1)
[[2.]
 [5.]
 [7.]]
error= [[4.4408921e-16]
 [0.0000000e+00]
 [8.8817842e-16]]
```

2. Gradient descent

Usually used to find the minima of a differantiable function.

a) Find the value of x at which $f(x)$ is minimum.

Let:

$$1)\ f(x) = x^2 + x + 2$$

$$2)\ f(x) = xsinx$$

1)

$$f(x) = x^2 + x + 2$$
$$\frac{d}{dx}f(x) = 2x + 1 = 0$$
$$\frac{d^2}{dx^2}f(x) = 2\ (Minima)$$
$$x = -\frac{1}{2}\ (analytical\ solution)$$
$$x_{init} = 4$$
$$x_{updt} = x_{old} - \lambda(\frac{d}{dx}f(x)|x = x_{old})$$
$$x_{updt} = x_{old} - \lambda(2x_{old} + 1)$$

2)

$$f(x) = xsinx$$
$$\frac{d}{dx}f(x) = sinx + xcosx$$
$$x_{updt} = x_{old} - \lambda(sin(x_{old}) + x_{old}cos(x_{old}))$$

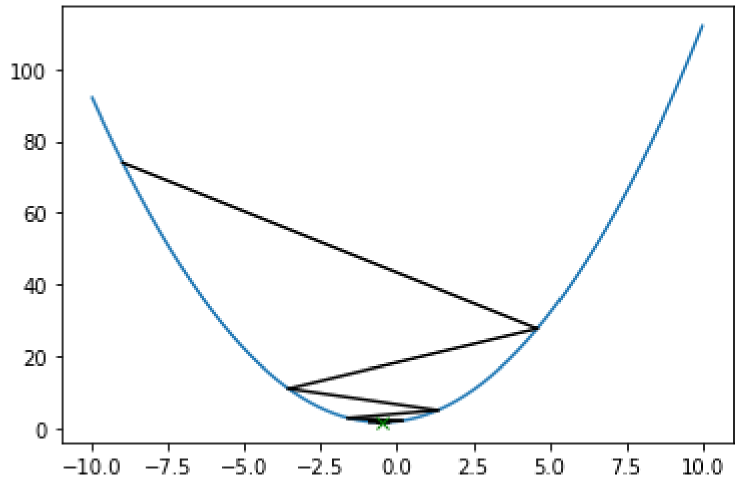
In [ ]:

```
# Example 1
# import time
import numpy as np
import matplotlib.pyplot as plt

x=np.linspace(-10,10,1000)
def f(x):
  return # insert your equation here
y=f(x)
plt.figure()
plt.plot(x,y)
# gradient
x_init=-9  # initialization
lr=0.8  # learning rate

# gradiant computation
def grad_upd(x_old,lr):
  x_new= # write the update equation here.(x_old-(lr X ((2 X x_old)+1)))
  return x_new

for i in list(range(1000)):
  if i==0:
    x=x_init
    x_old=x_init
    x=grad_upd(x,lr)
  else:
    x=grad_upd(x,lr)

  dev=np.abs(x-x_old)
  #print(x)
  plt.plot([x_old,x],[f(x_old),f(x)],color='k')
  x_old=x
  #time.sleep(5)
  if dev<=0.000001:
    break
print(x)
plt.plot(x,f(x),'x',color='g')
```

```
-0.5000002435350299
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7f50b858efd0>]
```



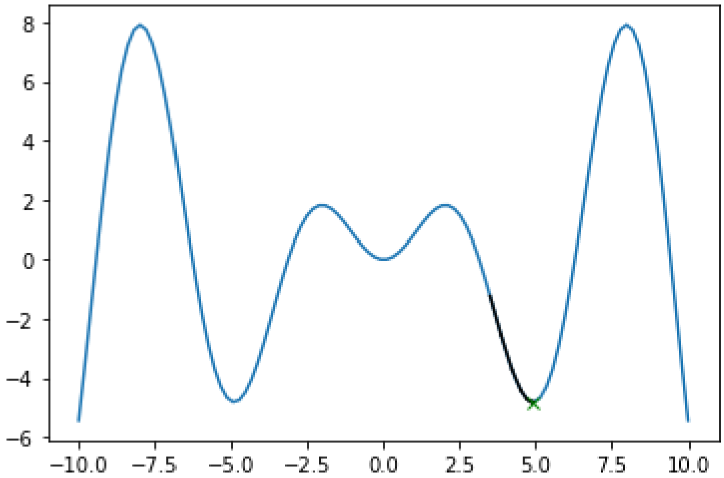
In [ ]:

```
# example 2

x=np.linspace(-10,10,100)
def f(x):
  return np.multiply(x,np.sin(x))
y=f(x)
plt.plot(x,y)
# gradient
x_init=3.5  # initialization
lr=0.1 # learning rate
def grad_upd(x_old,lr):
  x_new=# write the update equation here.
  return x_new

for i in list(range(1000)):
  if i==0:
    x=x_init
    x_old=x_init
    x=grad_upd(x,lr)
  else:
    x=grad_upd(x,lr)

  dev=np.abs(x-x_old)
  plt.plot([x_old,x],[f(x_old),f(x)],color='k')
  x_old=x
  if dev<=0.000001:
    break
print(x)
plt.plot(x,f(x),'x',color='g')
```

```
4.913179571739345
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7f50ba276908>]
```



b) Find the value of x and y at which f(x,y) is minimum.

1. $x^2 + y^2 + 2x + 2y$
2. $xsin(x) + ysin(y)$
3. Exercise problem: $f(x) = x^T Ax + b^T x + c$, where x is a M dimensional vector.

In [ ]:

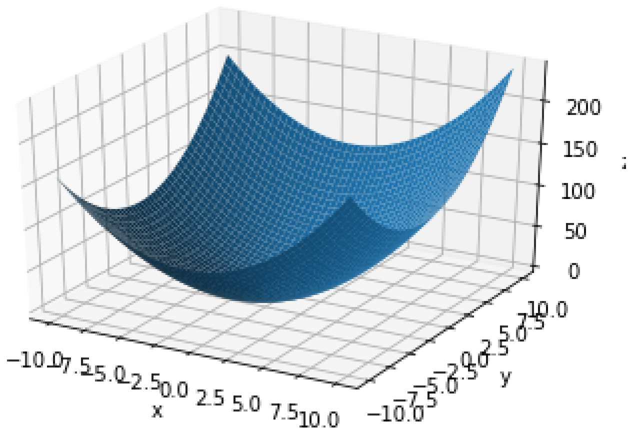
```
# Example b:1
import numpy as np
import matplotlib.pyplot as plt

x=np.linspace(-10,10,1000)
y=np.linspace(-10,10,1000)
def f(x,y):
  return # insert your equation here
X, Y = np.meshgrid(x, y)
Z=f(X,Y)

## only for ploting the surface.
plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z');
plt.figure()
con=plt.axes(projection='3d')
con.contour3D(X, Y, Z, 50)
con.set_xlabel('x')
con.set_ylabel('y')
con.set_zlabel('z');


plt.figure()
plt.contour(X,Y,Z,100)

# Gradient descent
#let [x , y] is a matrix 0f x
def grad(x):
  return 2*x+2 # same grad for both x and y as will be used in gradient descent


def grad_update(x_old,lr):
  x_new=# write the update equation here. (use grad function defined in earlier
 steps)
  return x_new
# initialization
# gradient
x_init=np.array([-9,-9])  # initialization
lr=0.1  # learning rate

for i in list(range(1000)):
  if i==0:
    x=x_init
    x_old=x_init
    x=grad_update(x,lr)
  else:
    x=grad_update(x,lr)

  dev=np.linalg.norm(x-x_old)
  plt.plot([x_old[0],x[0]],[x_old[1],x[1]],color='k')
  x_old=x
  if dev<=0.000001:
    break
```

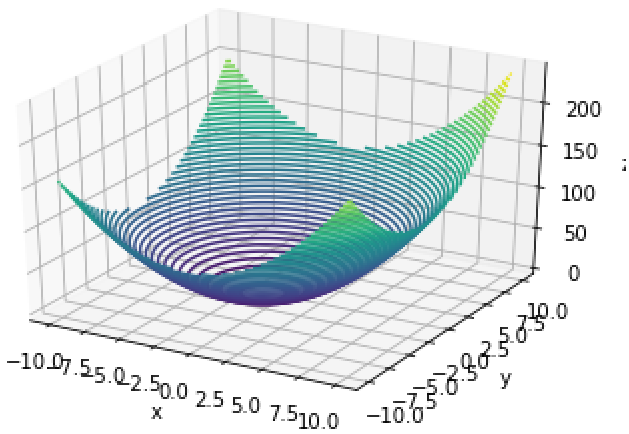
```
print(x)
plt.plot(x[0],x[1],'x',color='g')
```

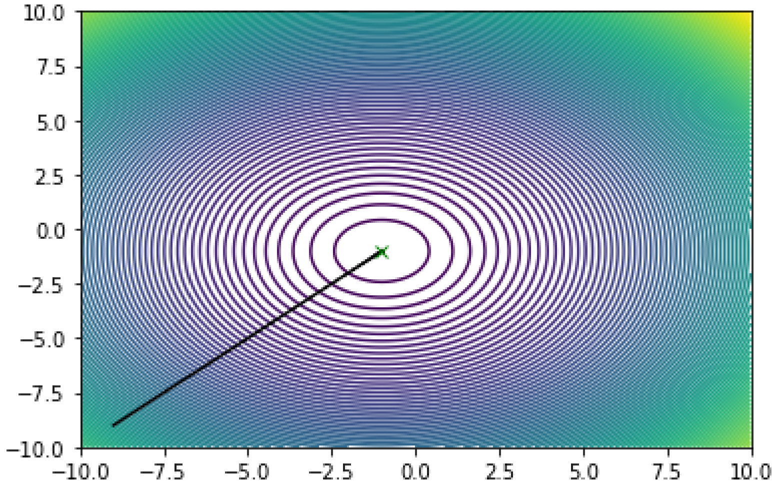
[-1.00000257 -1.00000257]

Out[ ]:

[<matplotlib.lines.Line2D at 0x7f50b85b58d0>]







In [ ]:

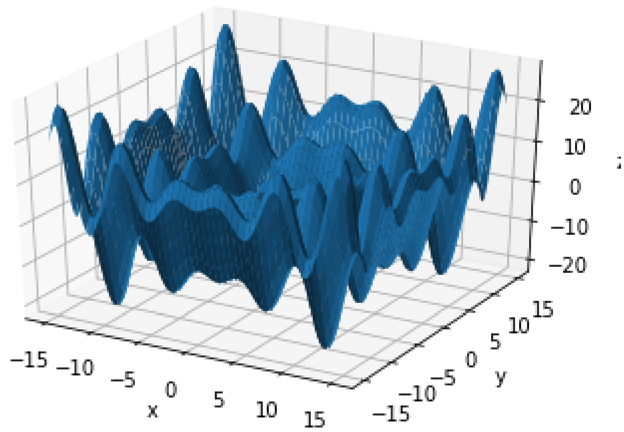
```
# Example b:2

import numpy as np
import matplotlib.pyplot as plt

x=np.linspace(-15,15,1000)
y=np.linspace(-15,15,1000)
# write the equation to be optimized here
def f(x,y):
  return x*np.sin(x)+y*np.sin(y)
X, Y = np.meshgrid(x, y)
Z=f(X,Y)

## only for ploting the surface.
plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z');
plt.figure()
con=plt.axes(projection='3d')
con.contour3D(X, Y, Z, 50)
con.set_xlabel('x')
con.set_ylabel('y')
con.set_zlabel('z');


plt.figure()
plt.contour(X,Y,Z,100)

# Gradient descent
#let [x , y] is a matrix 0f x
def grad(x):
  return np.sin(x)+np.multiply(x,np.cos(x)) # same grad for both x and y as will be used in gradient descent
def grad_update(x_old,lr):
  x_new=# write the update equation here. (use grad function defined in earlier steps)
  return x_new
# initialization
# gradient
x_init=np.array([0,-7.5])  # initialization
lr=0.1  # learning rate

for i in list(range(1000)):
  if i==0:
    x=x_init
    x_old=x_init
    x=grad_update(x,lr)
  else:
    x=grad_update(x,lr)

  dev=np.linalg.norm(x-x_old)
  plt.plot([x_old[0],x[0]],[x_old[1],x[1]],color='k')
  x_old=x
  if dev<=0.000001:
    break
```

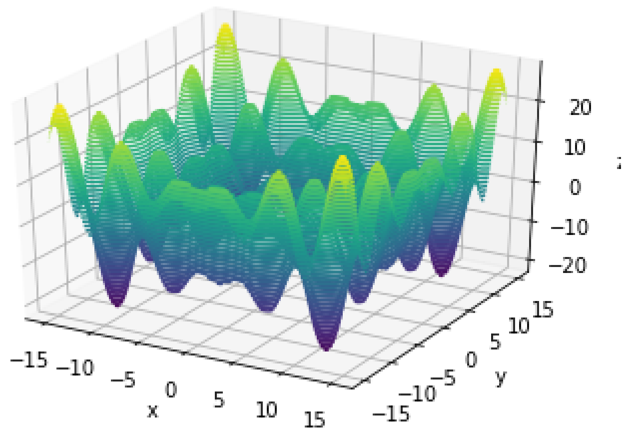
```
print(x)
plt.plot(x[0],x[1],'x',color='g')
```

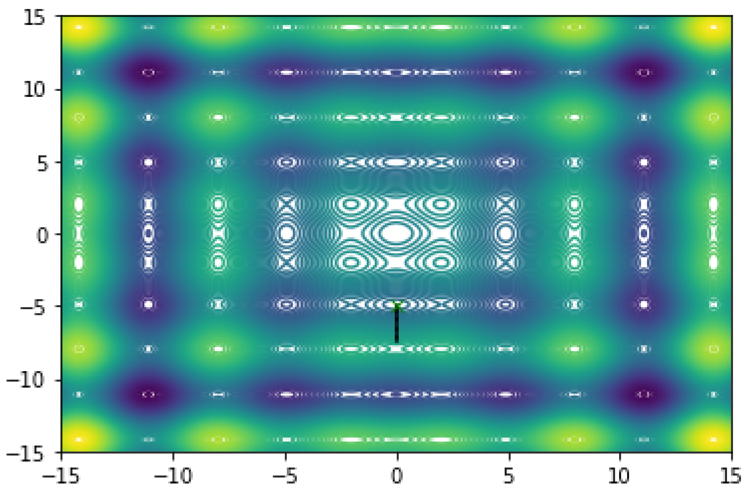
```
[ 0.         -4.91318121]
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7f66a0c5da58>]
```







In [ ]:

```
f(x[0],x[1])
```

Out[ ]:

```
-15.855177905640703
```

3.Sampling

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

1.Sampling from uniform distribution

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

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

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

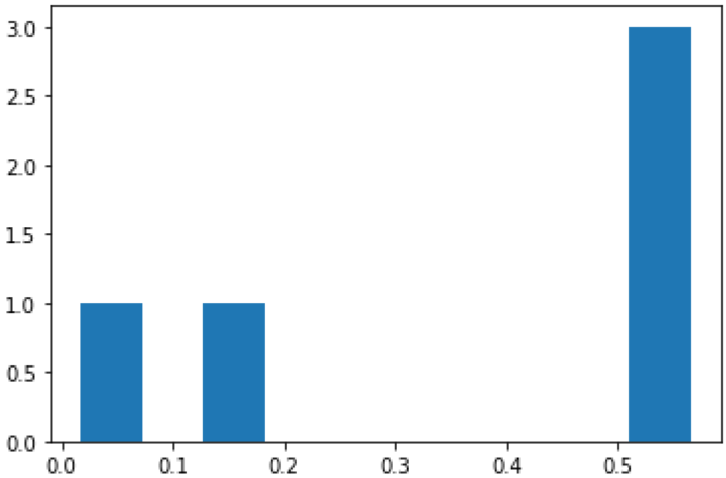
In [ ]:

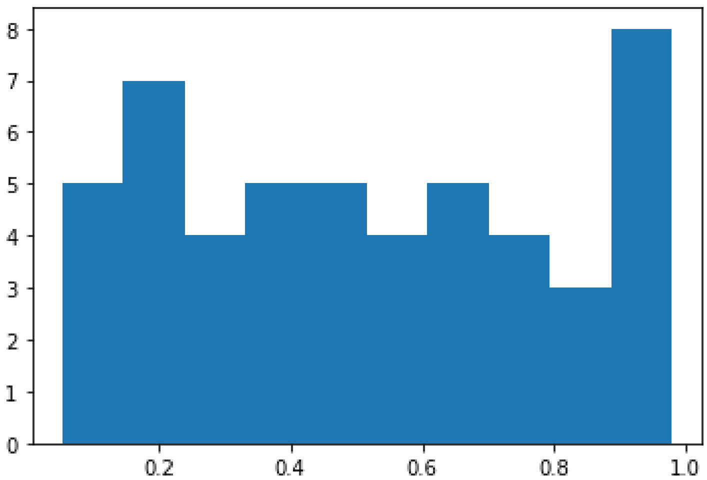
```
import numpy as np
import matplotlib.pyplot as plt

N=10
m=0.5
x=np.random.uniform(0,1,N) # can also use np.random.random(N)
#print(x)

# (b) see the histogram
N=np.array([5,50,500,2000,100000])
print(N[N.shape[0]-1])
for i in N:
  x=np.random.uniform(0,1,i)
  plt.figure()
  plt.hist(x)


# (c) law of large numbers
m_sampled=np.zeros(N[N.shape[0]-1])
x=np.zeros(N[N.shape[0]-1])
print(m_sampled.shape)
plt.figure()
for j in list(range(10)):
  print(N[N.shape[0]-1])
  i=np.arange(1,N[N.shape[0]-1]+1)
  x=np.random.uniform(0,1,(N[N.shape[0]-1]))
  m_sampled=np.cumsum(x)/(i)
  plt.semilogx(m_sampled)

m=np.tile(m,x.shape)
plt.semilogx(m,color='k')
```

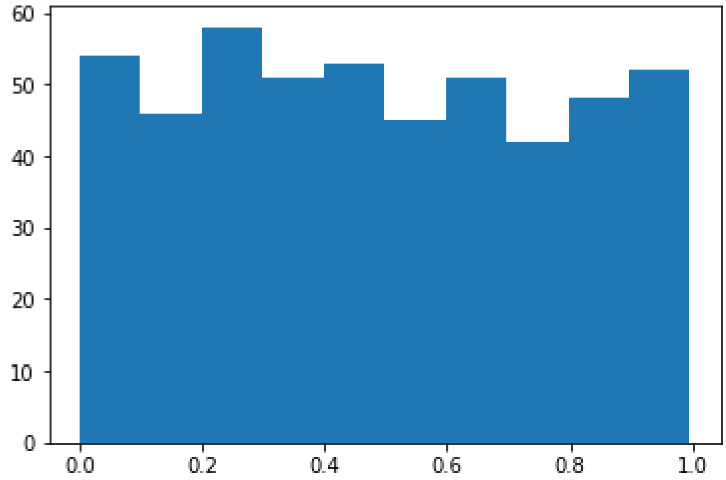
```
100000
(100000,)
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
```

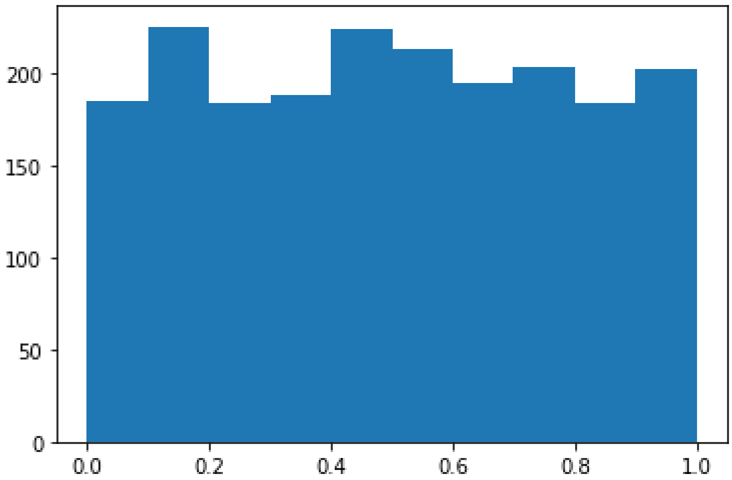
Out[ ]:

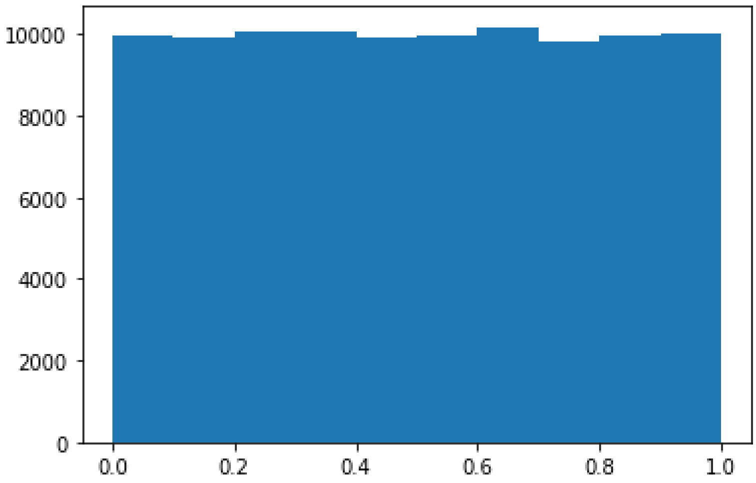
```
[<matplotlib.lines.Line2D at 0x7f50b876ad68>]
```

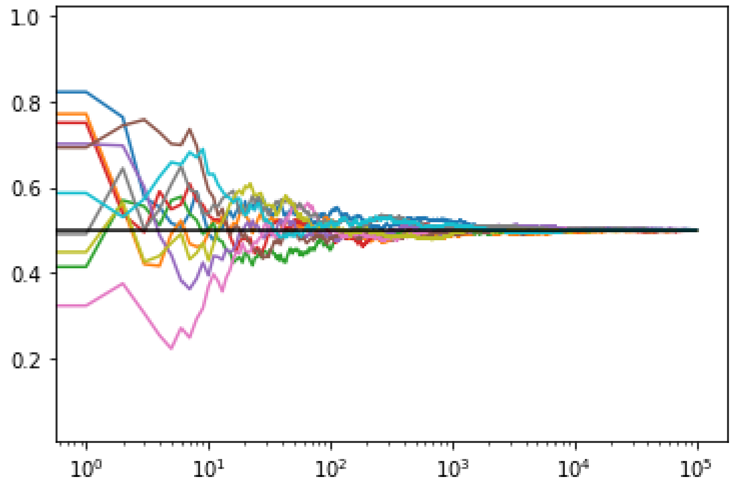





60
50
40
30
20
10
0
0.0
0.2
0.4
0.6
0.8
1.0


200
150
100
50
0
0.0
0.2
0.4
0.6
0.8
1.0


10000
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0



2. Sampling from Gaussian Distribution

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

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

c) Law of large number

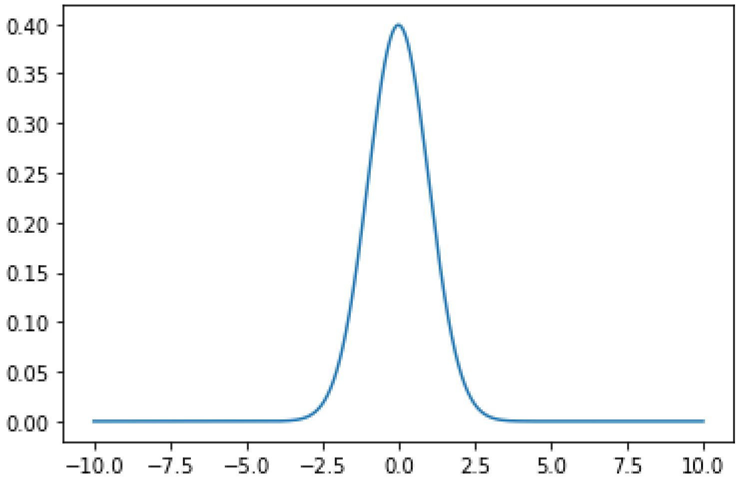
In [1]:

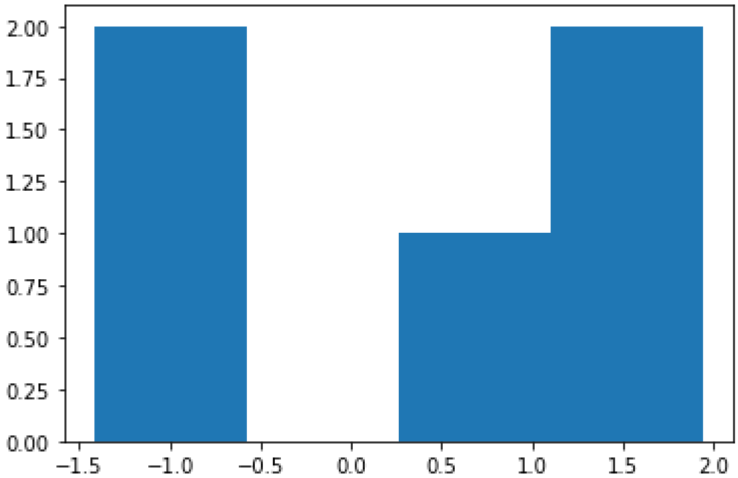
```
# a
import numpy as np
import matplotlib.pyplot as plt

x=np.linspace(-10,10,1000)
mu=0
var=1
px=(1/(np.sqrt(2 * np.pi * var)))*np.exp((-np.power(x-mu,2)/(2*var)))

plt.figure()
plt.plot(x,px)

# b

N=np.array([5,50,500,2000,100000])
m_sampled=np.zeros(N.shape)

for i in N:
  x_sampled=np.random.normal(mu,np.sqrt(var),i)
  plt.figure()
  plt.hist(x_sampled,np.int(np.log(i)*3))



#  law of large numbers
m_sampled=np.zeros(N[N.shape[0]-1])
x=np.zeros(N[N.shape[0]-1])
print(m_sampled.shape)
plt.figure()
for j in list(range(10)):
  print(N[N.shape[0]-1])
  i=np.arange(1,N[N.shape[0]-1]+1)
  x=np.random.normal(mu,np.sqrt(var),(N[N.shape[0]-1]))
  m_sampled=np.cumsum(x)/(i)
  plt.semilogx(m_sampled)

m=np.tile(mu,x.shape)
plt.semilogx(m,color='k')
```

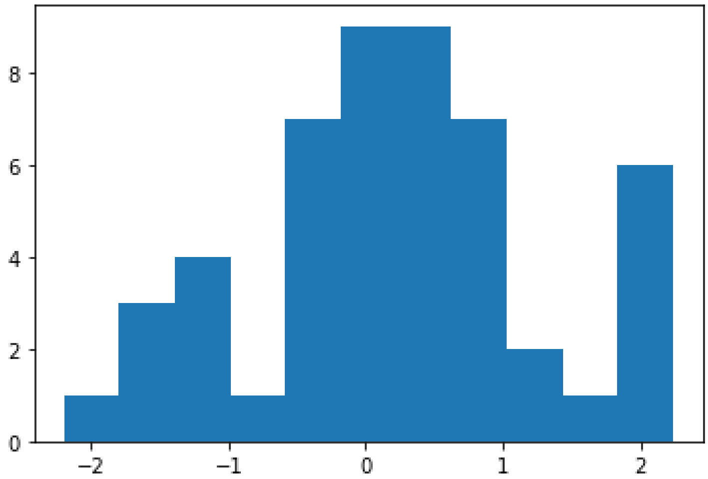
```
(100000,)
100000
100000
100000
100000
100000
100000
100000
100000
100000
100000
```

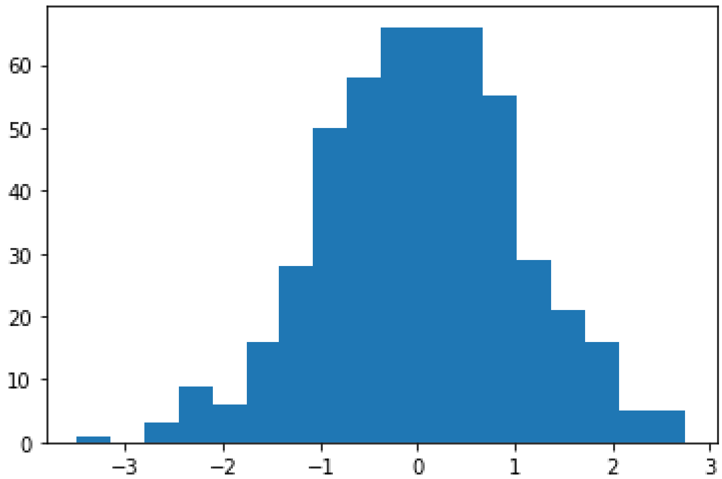
Out[1]:

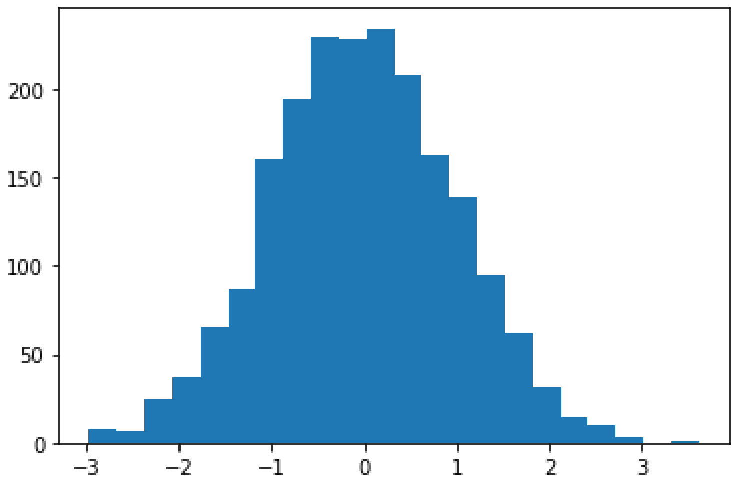
```
[&lt;matplotlib.lines.Line2D at 0x7f61a9955160&gt;]
```

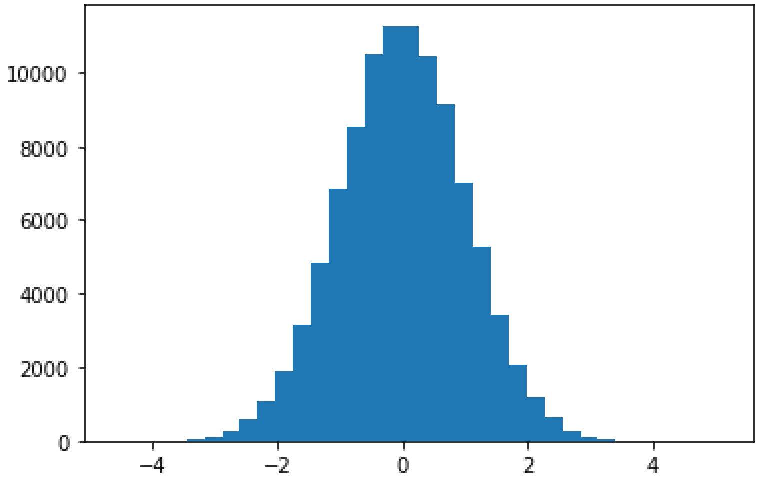


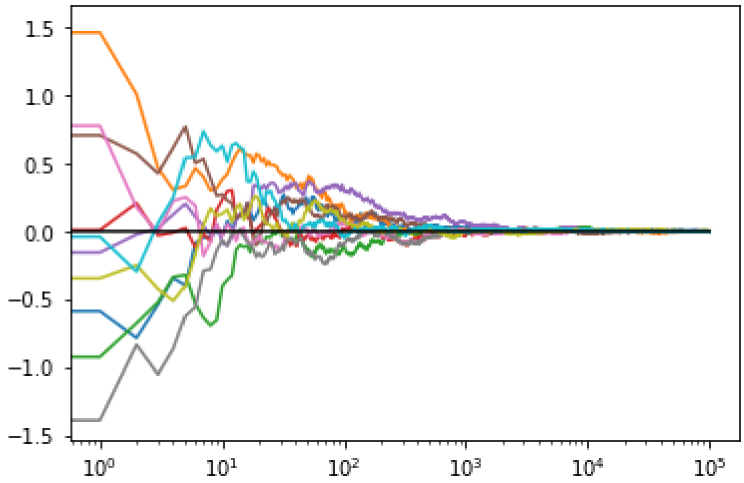



0
2
4
6
8
−2
−1
0
1
2


0
10
20
30
40
50
60
−3
−2
−1
0
1
2
3


0
50
100
150
200
−3
−2
−1
0
1
2
3





3.Sampling of categorical from uniform

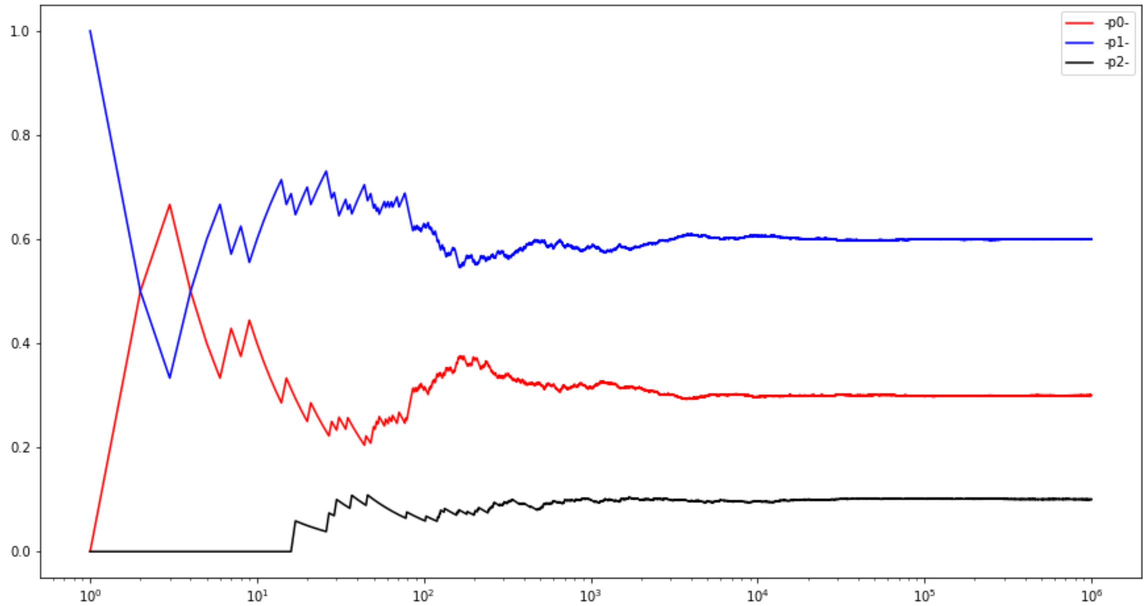
In [4]:

```
# number of samples
n = 1000000
y = # insert your code here
x = np.arange(1, n+1)

print('y=',y)

prob0=0.3
prob1=0.6
prob2=0.1


# count number of occurrences and divide by the number of total draws
p0 = np.cumsum(y < prob0) / x
print('p0=',p0)

p1 = np.cumsum(np.logical_and(y >= prob0,y < prob0+prob1)) / x
p2 = np.cumsum(y >= prob0+prob1) / x

print('p1=',p1)
print('p2=',p2)

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-'])
```

```
y= [0.41164867 0.19421562 0.12852623 ... 0.33572556 0.28619319 0.908
87258]
p0= [0.         0.5        0.66666667 ... 0.3001956  0.3001963  0.30
0196  ]
p1= [1.         0.5        0.33333333 ... 0.5994472  0.5994466  0.59
9446  ]
p2= [0.        0.        0.         ... 0.1003572 0.1003571 0.100358
]
```

Out[4]:

```
<matplotlib.legend.Legend at 0x7f61a91ddd68>
```



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.

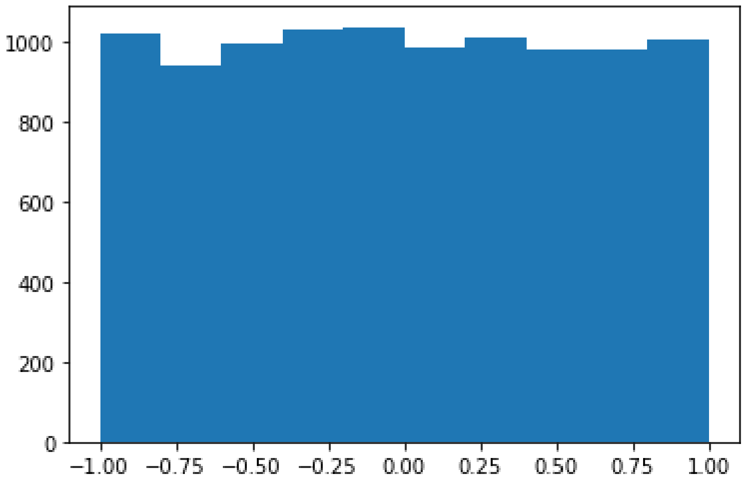
In [ ]:

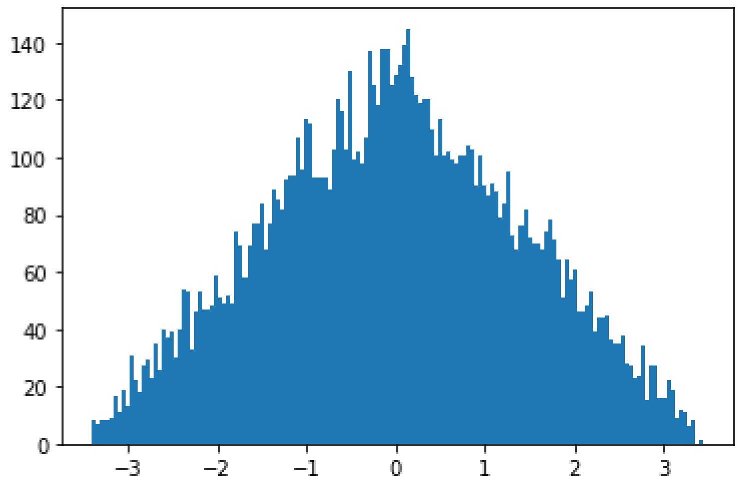
```
x=np.random.uniform(-1,1,[10000,1000])
#print(x.shape)
plt.figure()
plt.hist(x[:,0])
# addition of 2 random variables
tmp2=np.sum(x[:,0:2],axis=1)/(np.std(x[:,0:2]))
plt.figure()
plt.hist(tmp2,150)

# addition of 100 random variables

tmp100=np.sum(x[:,0:100],axis=1)/(np.std(x[:,0:100]))
plt.figure()
plt.hist(tmp100,150)

# addition of 1000 random variables

tmp1000=np.sum(x[:,0:1000],axis=1)/(np.std(x[:,0:1000]))
plt.figure()
plt.hist(tmp1000,150)
```

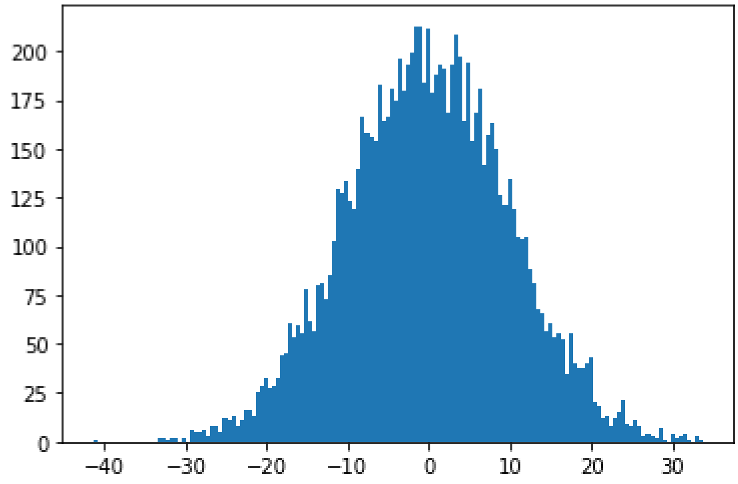
Out[ ]:

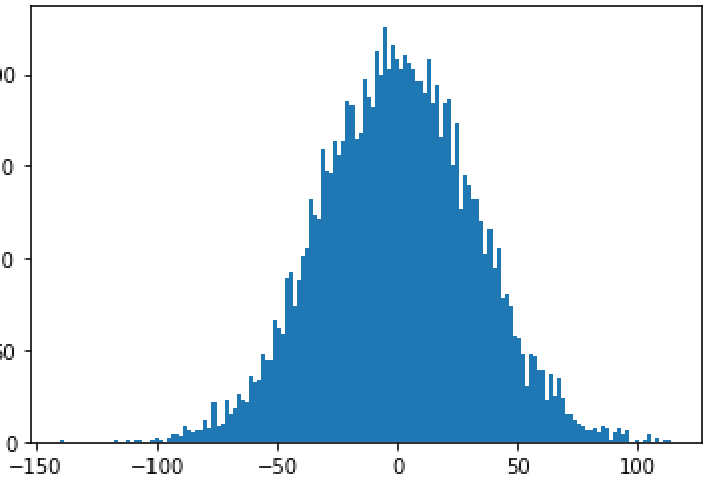
```
(array([  1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
0.,
          0.,   0.,   1.,   0.,   0.,   1.,   0.,   1.,   1.,   0.,
0.,
          1.,   2.,   1.,   0.,   2.,   4.,   4.,   3.,   9.,   6.,
5.,
          6.,   6.,  12.,   7.,  22.,   9.,  10.,  23.,  15.,  18.,
26.,
         23.,  22.,  36.,  32.,  34.,  48.,  45.,  45.,  66.,  62.,
59.,
         89.,  92.,  74.,  88., 101., 106., 132., 123., 121., 159.,
147.,
        146., 163., 156., 163., 185., 183., 165., 168., 197., 187.,
182.,
        213., 200., 226., 203., 216., 208., 203., 210., 206., 203.,
196.,
        196., 190., 208., 184., 194., 166., 184., 186., 150., 173.,
126.,
        145., 140., 132., 132., 120., 102., 116.,  95., 106.,  78.,
81.,
         74.,  58.,  57.,  48.,  30.,  48.,  47.,  39.,  39.,  23.,
37.,
         25.,  35.,  24.,  15.,  15.,  12.,  10.,   9.,   6.,   6.,
7.,
          5.,   8.,   7.,   1.,   5.,   7.,   4.,   6.,   0.,   1.,
0.,
          1.,   4.,   0.,   2.,   0.,   1.,   1.]),
 array([-139.71056536, -138.02150558, -136.33244579, -134.64338601,
        -132.95432623, -131.26526644, -129.57620666, -127.88714688,
        -126.19808709, -124.50902731, -122.81996753, -121.13090774,
        -119.44184796, -117.75278818, -116.06372839, -114.37466861,
        -112.68560883, -110.99654904, -109.30748926, -107.61842948,
        -105.92936969, -104.24030991, -102.55125012, -100.86219034,
         -99.17313056,  -97.48407077,  -95.79501099,  -94.10595121,
         -92.41689142,  -90.72783164,  -89.03877186,  -87.34971207,
         -85.66065229,  -83.97159251,  -82.28253272,  -80.59347294,
         -78.90441316,  -77.21535337,  -75.52629359,  -73.83723381,
         -72.14817402,  -70.45911424,  -68.77005446,  -67.08099467,
         -65.39193489,  -63.70287511,  -62.01381532,  -60.32475554,
         -58.63569576,  -56.94663597,  -55.25757619,  -53.56851641,
         -51.87945662,  -50.19039684,  -48.50133705,  -46.81227727,
         -45.12321749,  -43.4341577 ,  -41.74509792,  -40.05603814,
         -38.36697835,  -36.67791857,  -34.98885879,  -33.299799  ,
         -31.61073922,  -29.92167944,  -28.23261965,  -26.54355987,
         -24.85450009,  -23.1654403 ,  -21.47638052,  -19.78732074,
         -18.09826095,  -16.40920117,  -14.72014139,  -13.0310816 ,
         -11.34202182,   -9.65296204,   -7.96390225,   -6.27484247,
          -4.58578269,   -2.8967229 ,   -1.20766312,    0.48139666,
           2.17045645,    3.85951623,    5.54857601,    7.2376358 ,
           8.92669558,   10.61575537,   12.30481515,   13.99387493,
          15.68293472,   17.3719945 ,   19.06105428,   20.75011407,
          22.43917385,   24.12823363,   25.81729342,   27.5063532 ,
          29.19541298,   30.88447277,   32.57353255,   34.26259233,
          35.95165212,   37.6407119 ,   39.32977168,   41.01883147,
          42.70789125,   44.39695103,   46.08601082,   47.7750706 ,
          49.46413038,   51.15319017,   52.84224995,   54.53130973,
          56.22036952,   57.9094293 ,   59.59848908,   61.28754887,
          62.97660865,   64.66566844,   66.35472822,   68.043788  ,
          69.73284779,   71.42190757,   73.11096735,   74.80002714,
```

```
        76.48908692,   78.1781467 ,   79.86720649,   81.55626627,
        83.24532605,   84.93438584,   86.62344562,   88.3125054 ,
        90.00156519,   91.69062497,   93.37968475,   95.06874454,
        96.75780432,   98.4468641 ,  100.13592389,  101.82498367,
       103.51404345,  105.20310324,  106.89216302,  108.5812228 ,
       110.27028259,  111.95934237,  113.64840215]),
 <a list of 150 Patch objects>)
```









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:

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

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

In [7]:

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

r=1

x= # insert your code here (output will be a (1000 X 2) matrix) (refer: https://
numpy.org/doc/stable/reference/random/generated/numpy.random.uniform.html )

ax.scatter(x[:,0],x[:,1],color='y')

# find the number points present inbetween the circle

x_cr= (np.power(x[:,0],2)+np.power(x[:,1],2)<=1) # indicator where ever the cond
ition is satisfied


circle1=plt.Circle((0, 0), 1,fc='None',ec='k')
ax.add_artist(circle1)




pii=(np.count_nonzero(x_cr)*(np.power((2*r),2)))/x.shape[0]

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

```
computed value of pi= 3.096
```