Lab 3 : Convex Optimisation

Gradient Descent

Write the code following the instructions to obtain the desired results

Import all the required libraries

In [21]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Logic used behind gradient descent-

Initialy i have taken a starting point x_init for starting the algorithm.
I also have set a precision value, **0.000001**,to avoid running algortithm forever.
A variable 'change' has been declared which calculates the difference between the values of x in successive iterations.
If **change <= prev**
algorithm stops as it reaches very very close to its optimal value.
With each iteration-
x_init = x*init -$|lambda |frac{|partial f(x*{init})}{\partial x}$
This idea is extended to function of two variables by changing x_init and y_init simultaneously.

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

1. Find x analytically
2. Write the update equation of gradient descent
3. Find x using gradient descent method

Example 1 : $f(x) = x^2 + x + 2$

Analytical :

$$\frac{d}{dx}f(x) = 2x + 1 = 0$$

$$\frac{d^2}{dx^2}f(x) = 2\ (Minima)$$

$$x = -\frac{1}{2}\ (analytical\ solution)$$

Gradient Descent Update equation :

$$x_{init} = 4$$

$$x_{updt} = x_{old} - \lambda(\frac{d}{dx}f(x)|x = x_{old})$$

$$x_{updt} = x_{old} - \lambda(2x_{old} + 1)$$

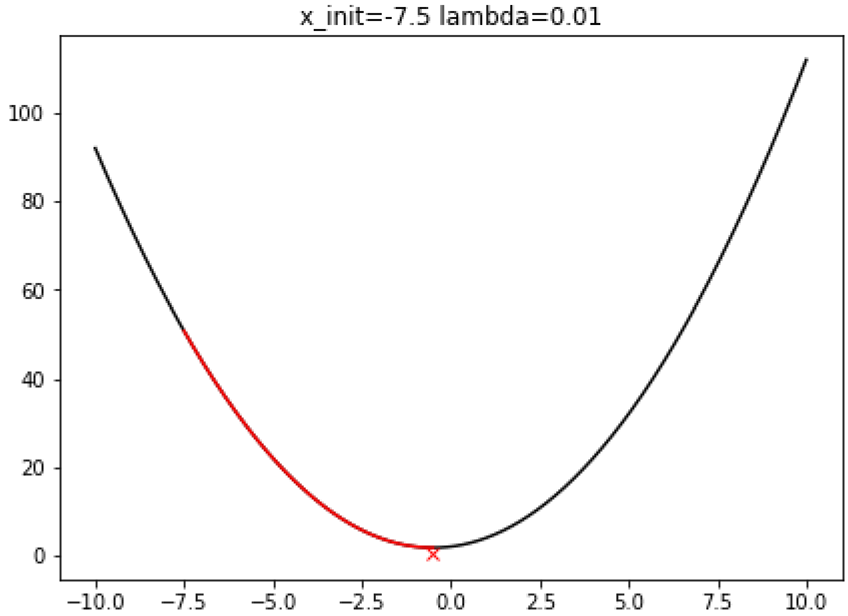
Gradient Descent Method :

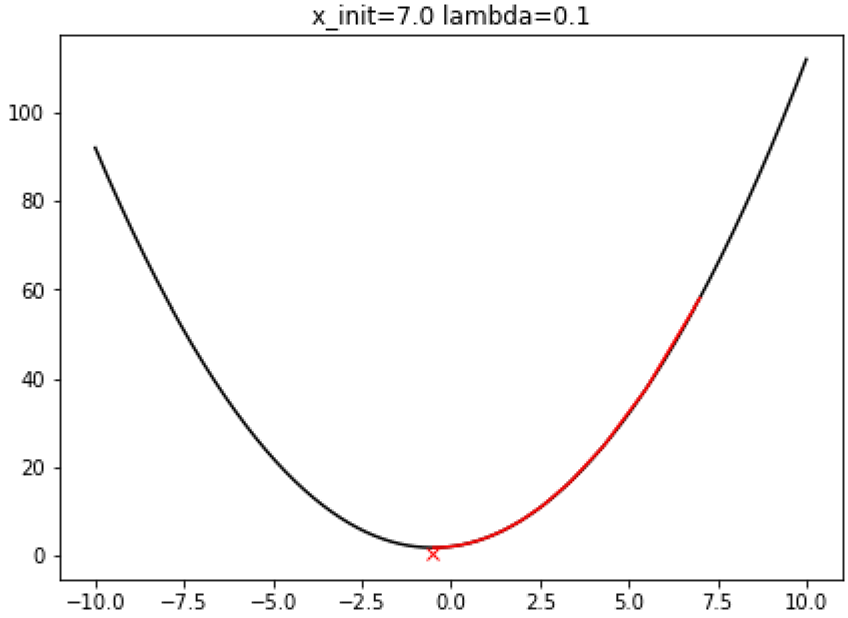
Follow the below steps and write your code in the block below

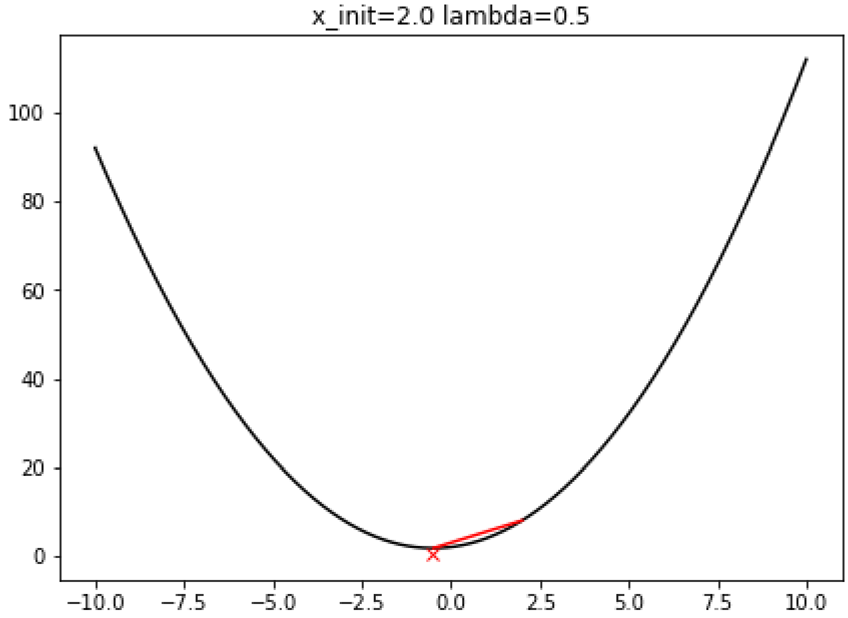
1. Generate x, 1000 data points from -10 to 10
2. Generate and Plot the function $f(x)$ = x^2 + x + 2
3. Initialize the starting point (x_{init}) and learning rate (λ)
4. Use Gradient descent algorithm to compute value of x at which the function $f(x)$ is minimum
5. Also vary the learning rate and initialisation point and plot your observations

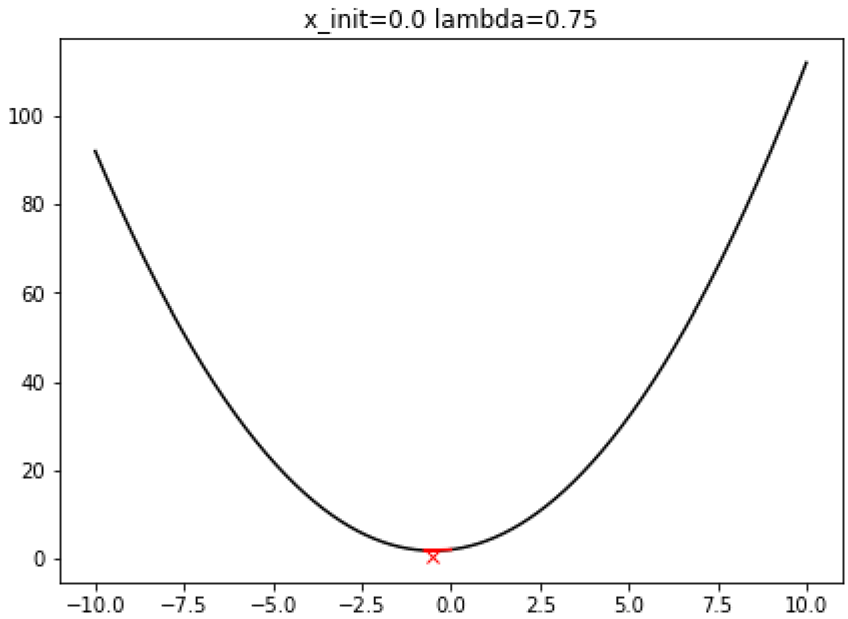
In [22]:

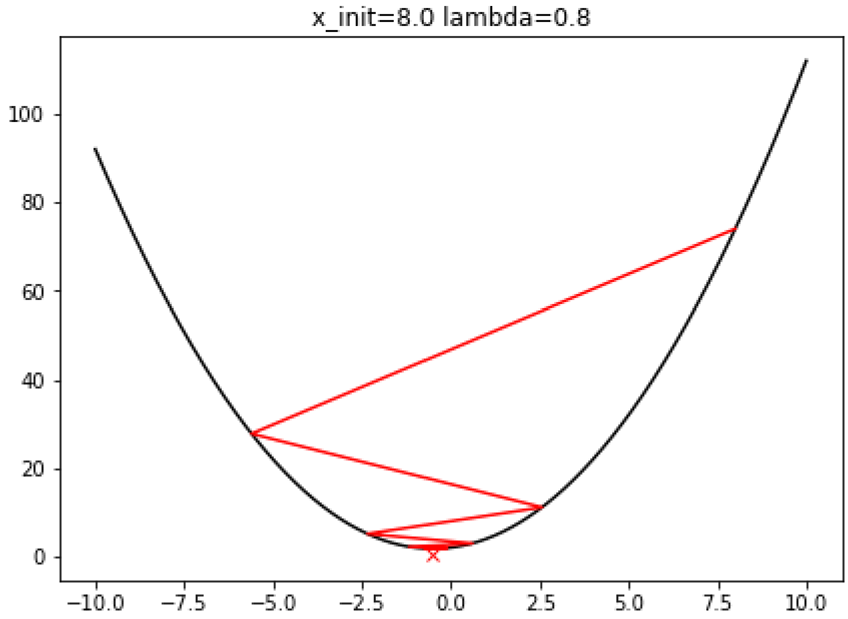
```
## Write your code here
lbda = np.array([0.01,0.1,0.5,0.75,0.8]) #Array of Learning Rate
x_init =np.array([-7.5,7,2,0,8]) #Initializing start value for the algorithm
prec = 0.000001 #To stop the algorthim and saving useless iterations
x = np.linspace(-10,10,1000) #Generating 1000 data points between (-10,10)
f_x = x**2 + x + 2 #Declaring a quadratic function with 1000 entries
for i in range(0,len(lbda)):
    change = 1
    x_d = np.array([]) #decalring array to store x coordinates every iteratio
    vls = np.array([]) #declaring array to stor values of function at corresp
    tmp1 = x_init[i]
    while change > prec: #Condition to avoid algorithm to unnecessarily run f
        x_d = np.append(x_d,x_init[i])
        vls = np.append(vls,x_init[i]**2 + x_init[i] + 2)
        tmp = x_init[i]
        x_init[i] -= lbda[i]*(2*(tmp)+1)
        change = abs(x_init[i]-tmp)
    #Plotting the obtained results
    plt.figure(figsize = [7,5])
    plt.plot(x,f_x,'k')
    plt.plot(x_d,vls,'r')
    plt.plot(x_init[i],x_init[i]*np.sin(x_init[i]),'r',marker = 'x')
    plt.title("x_init="+str(tmp1)+" lambda="+str(lbda[i]))
    plt.show()
print("x=",x_init)
```


x_init=-7.5 lambda=0.01
100
80
60
40
20
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


x_init=7.0 lambda=0.1
100
80
60
40
20
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


x_init=2.0 lambda=0.5
100
80
60
40
20
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0






```
x= [-0.50004853 -0.49999623 -0.5         -0.50000024 -0.49999976]
```

Example 2 : $f(x) = xsinx$

Analytical : Find solution analytically

Gradient Descent Update equation : Write Gradient descent update equations

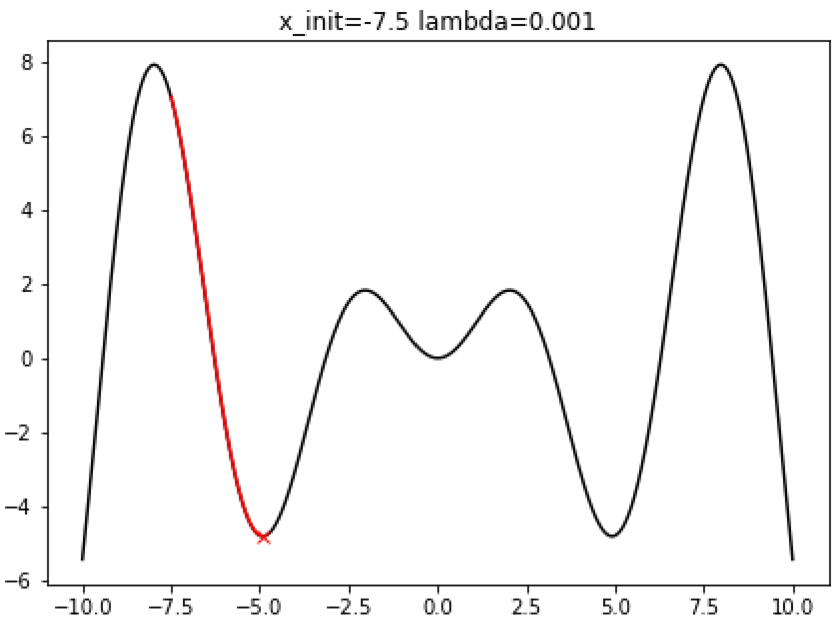
Gradient Descent Method :

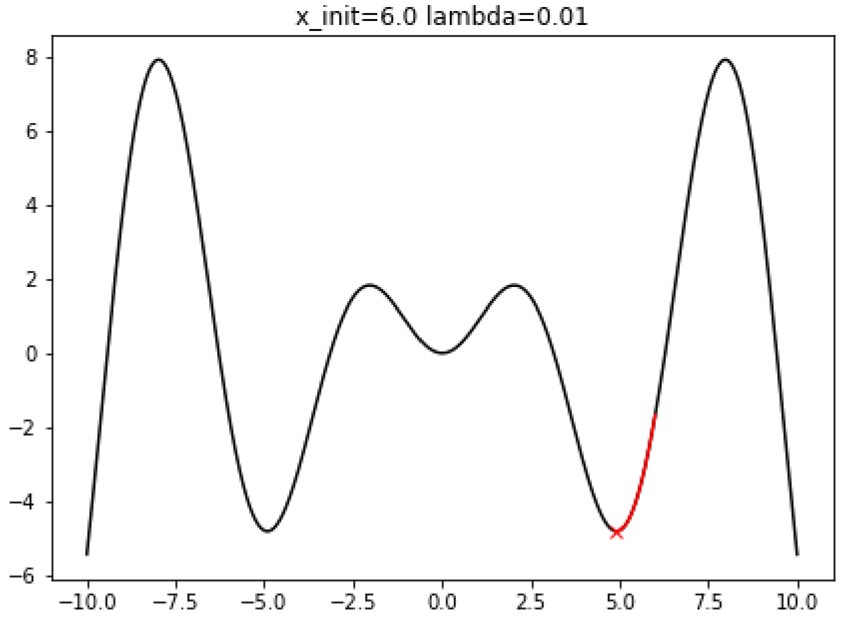
Follow the below steps and write your code in the block below

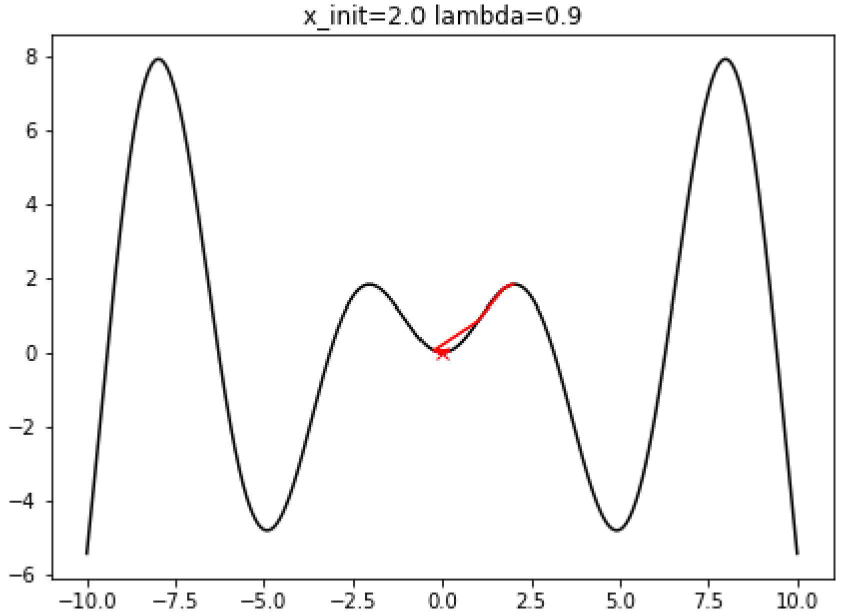
1. Generate x, 1000 data points from -10 to 10
2. Generate and Plot the function $f(x) = x^2 + x + 2$
3. Initialize the starting point (x_{init}) and learning rate (λ)
4. Use Gradient descent algorithm to compute value of x at which the function $f(x)$ is minimum
5. Also vary the learning rate and initialisation point and plot your observations

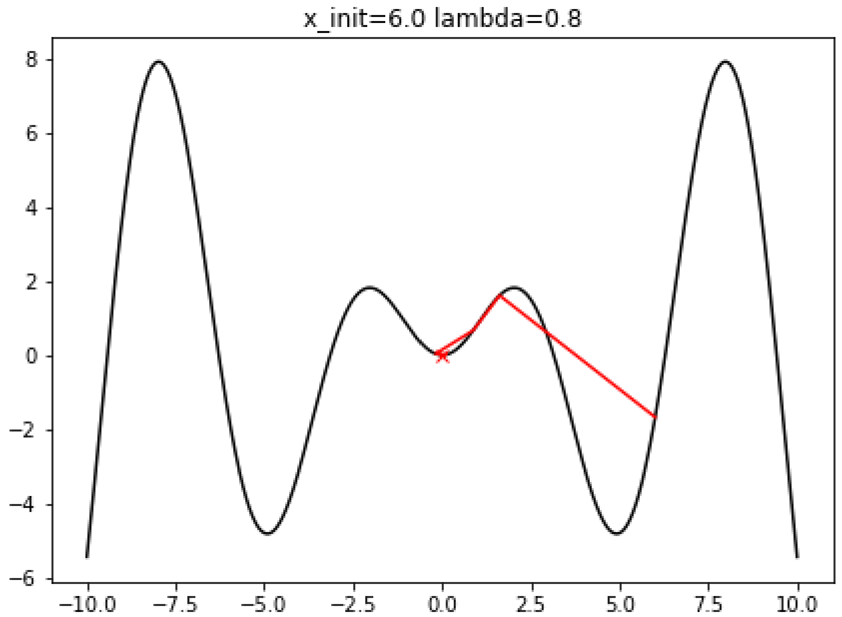
In [23]:
```
## Write your code here
## Write your code here
```

```
lbda = np.array([0.001,0.01,0.9,0.8]) #Array of Learning Rate
x_init = np.array([-7.5,6,2,6])
prec = 0.000001 #To stop the algorthim and saving useless iterations
x = np.linspace(-10,10,1000) #Generating 1000 data points between (-10,10)
f_x = x*np.sin(x) #Declaring a quadratic function with 1000 entries
for i in range(0,len(lbda)):
    change = 1
    x_d = np.array([]) #declaring an array to store values of x_init after ea
    vls = np.array([]) #declaring an array to store values at corresponding x
    tmp1 = x_init[i]
    while change > prec: #condition to stop the algorithm from running foreve
        x_d = np.append(x_d,x_init[i])
        vls = np.append(vls,x_init[i]*np.sin(x_init[i]))
        tmp = x_init[i]
        x_init[i] -= lbda[i]*(x_init[i]*np.cos(x_init[i])+np.sin(x_init[i]))
        change = abs(x_init[i]-tmp)
    #Plotting the results obtained
    plt.figure(figsize = [7,5])
    plt.plot(x,f_x,'k')
    plt.plot(x_d,vls,'r')
    plt.plot(x_init[i],x_init[i]*np.sin(x_init[i]),'r',marker = 'x')
    plt.title("x_init="+str(tmp1)+" lambda="+str(lbda[i]))
    plt.show()
print("x=",x_init)
```










```
x= [-4.91337100e+00  4.91319841e+00  4.10361307e-07 -3.07262149e-07]
```

Find the value of x and y at which $f(x, y)$ is

minimum :

Example 1 : $f(x, y) = x^2 + y^2 + 2x + 2y$

Gradient Descent Method :

Follow the below steps and write your code in the block below

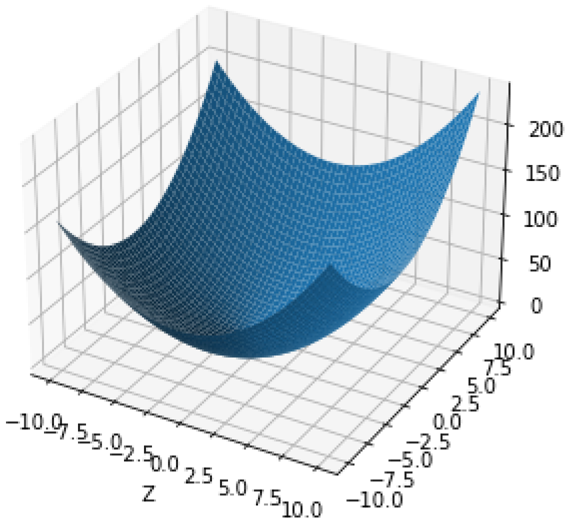
1. Generate x and y, 1000 data points from -10 to 10
2. Generate and Plot the function $f(x, y) = x^2 + y^2 + 2x + 2y$
3. Initialize the starting point (x_{init},y_{init}) and learning rate (λ)
4. Use Gradient descent algorithm to compute value of x and y at which the function $f(x, y)$ is minimum
5. Also vary the learning rate and initialisation point and plot your observations

In [24]:

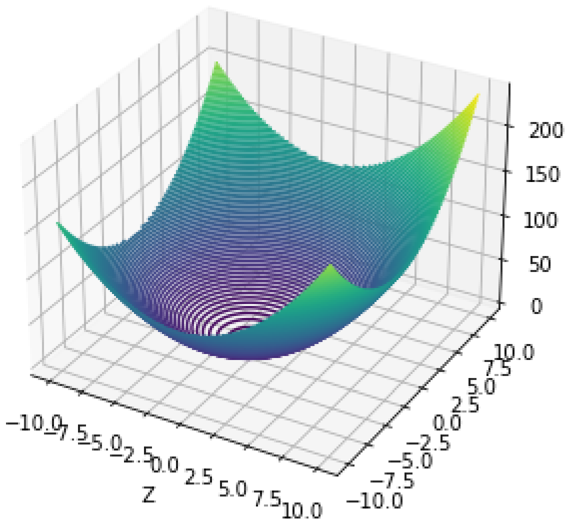
```
## Write your code here (Ignore the warning)
## Write your code here
lbda = np.array([0.001,0.01,0.9,0.8]) #Array of Learning Rate
x_init = np.array([-7.5,6,-1,6]) #starting value of x
y_init = np.array([-3,5.5,4.5,6]) #starting value of y
prec = 0.000001 #To stop the algorthim and saving useless iterations
x = np.linspace(-10,10,1000) #Generating 1000 data points between (-10,10)
y = np.linspace(-10,10,1000) #Generating 1000 data points between (-10,10)
X,Y = np.meshgrid(x,y)
f_xy = X**2 +Y**2 +2*X +2*Y #Declaring a quadratic function with 1000 entries
#Plotting surface plot of the function
fig = plt.figure()
plt.figure(figsize = [7,5])
ax = plt.axes(projection ='3d')
ax.plot_surface(X,Y,f_xy)
ax.set_xlabel("X")
ax.set_xlabel("Y")
ax.set_xlabel("Z")

#Plotting the 3D conour of the function
fig = plt.figure()
plt.figure(figsize = [7,5])
ax = plt.axes(projection ='3d')
ax.contour3D(X, Y, f_xy,100)
ax.set_xlabel("X")
ax.set_xlabel("Y")
ax.set_xlabel("Z")


for i in range(0,len(lbda)):
    change = 1
    x_d = np.array([]) #declaring array to store values of x after each itera
    vls = np.array([]) #declaring array to store values of f(x,y) after each 
    y_d = np.array([]) #declaring array to store values of y after each itera
    x_cor = x_init[i]
    y_cor = y_init[i]
    while change > prec: #Condition to stop algorithm from running forever
        x_d = np.append(x_d,x_init[i])
        y_d = np.append(y_d,y_init[i])
        vls = np.append(vls,x_init[i]**2+y_init[i]**2+2*x_init+2*y_init)
        tmp2 = x_init[i]
        tmp3 = y_init[i]
        tmp2 -= lbda[i]*(2*x_init[i]+2)
        tmp3 -= lbda[i]*(2*y_init[i]+2)
        change = abs(np.sqrt((x_init[i]-tmp2)**2 + (y_init[i]-tmp3)**2))
        x_init[i] = tmp2
```

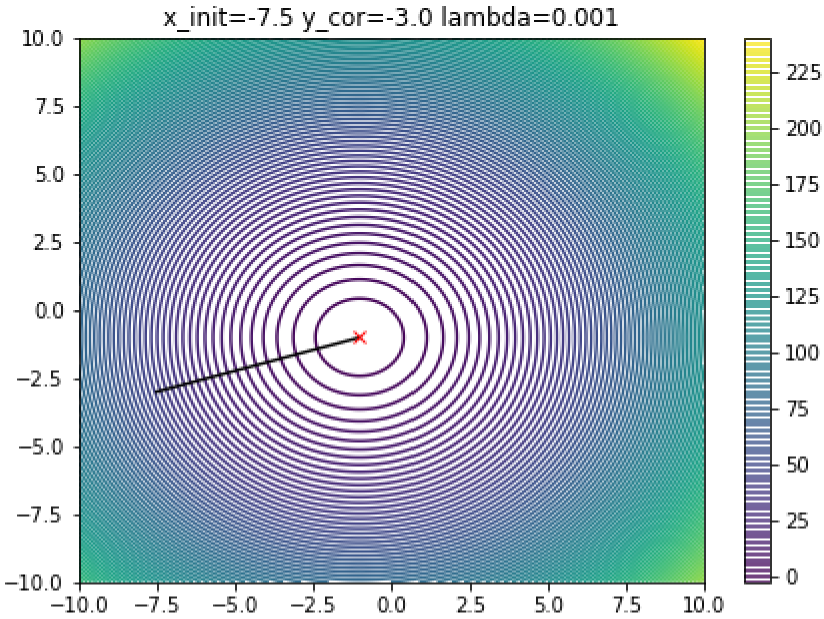
```
        y_init[i] = tmp3
    #Plotting the results obtained
    plt.figure(figsize = [7,5])
    plt.contour(X,Y,f_xy,100)
    plt.colorbar()
    plt.plot(x_d,y_d,'k')
    plt.plot(x_init[i],y_init[i],'r',marker = 'x')
    plt.title("x_init="+str(x_cor)+" y_cor="+str(y_cor)+" lambda="+str(lbda[i
    plt.show()
print("x=",x_init)
print("y=",y_init)
```

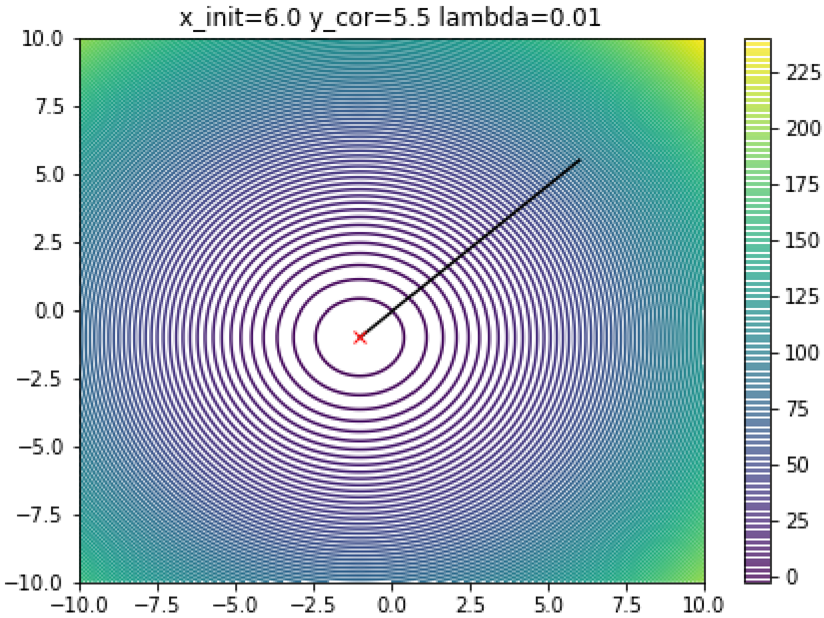
```
<Figure size 432x288 with 0 Axes>
```

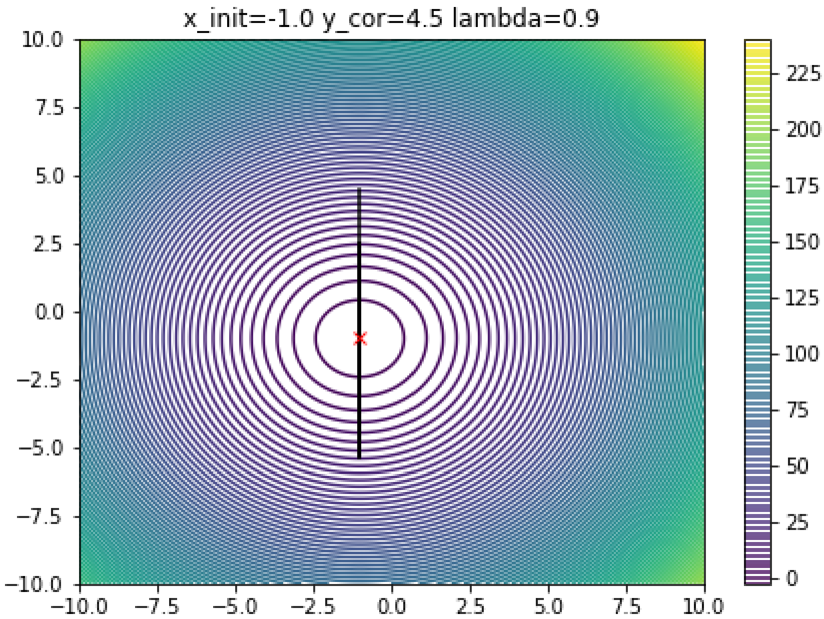


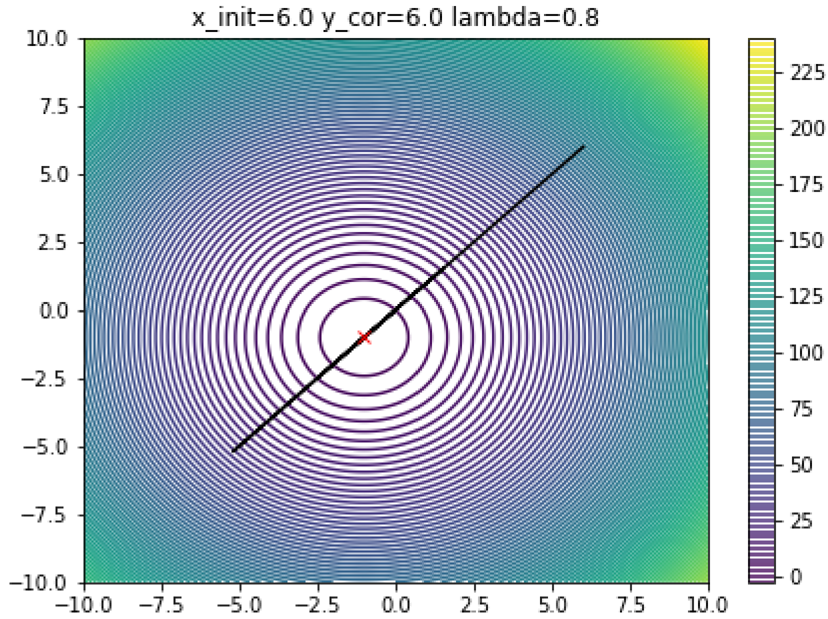

```
<Figure size 432x288 with 0 Axes>
```




x_init=-7.5 y_cor=-3.0 lambda=0.001
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5 10.0
225
200
175
150
125
100
75
50
25
0


x_init=6.0 y_cor=5.5 lambda=0.01
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5 10.0
225
200
175
150
125
100
75
50
25
0


x_init=-1.0 y_cor=4.5 lambda=0.9
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5 10.0
225
200
175
150
125
100
75
50
25
0




```
x= [-1.00047618 -0.99996416 -1.         -0.9999998 ]
y= [-1.00014652 -0.99996672 -0.99999963 -0.9999998 ]
```

Example 2 : $f(x, y) = xsin(x) + ysin(y)$

Gradient Descent Method :

Follow the below steps and write your code in the block below

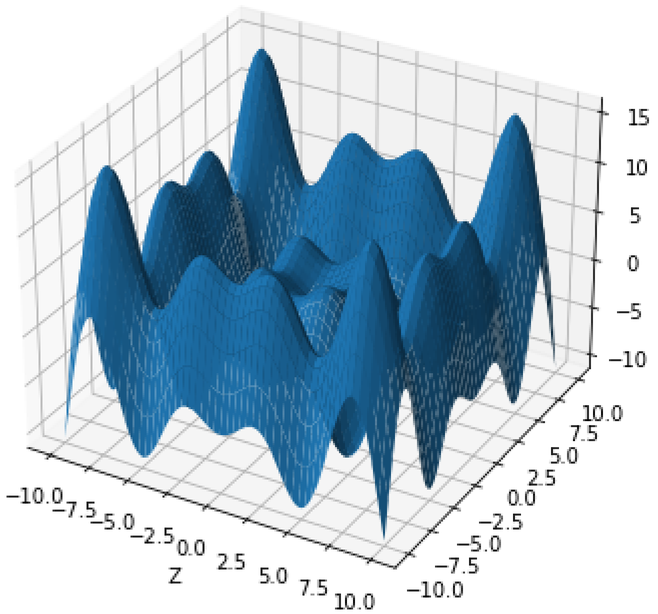
1. Generate x and y, 1000 data points from -10 to 10
2. Generate and Plot the function $f(x, y) = xsin(x) + ysin(y)$
3. Initialize the starting point (x_{init},y_{init}) and learning rate (λ)
4. Use Gradient descent algorithm to compute value of x and y at which the function $f(x, y)$ is minimum
5. Also vary the learning rate and initialisation point and plot your observations

In [25]:

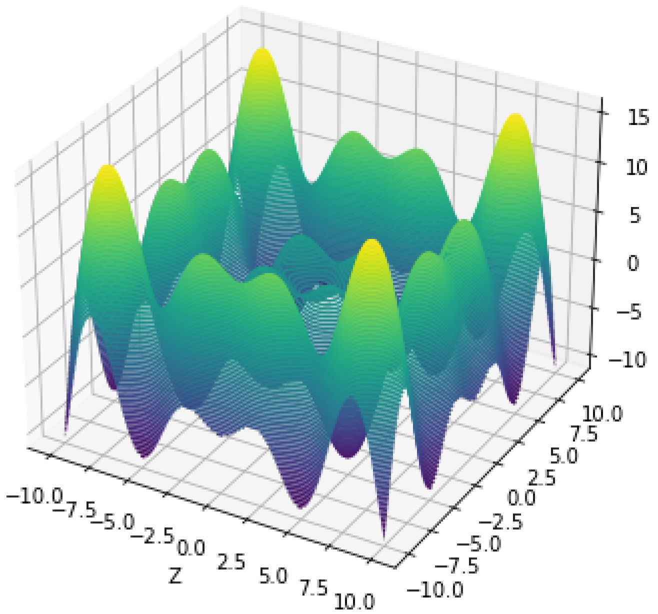
```
## Write your code here (Ignore the warning)
## Write your code here
lbda = np.array([0.001,0.01,0.9,0.8]) #Array of Learning Rate
x_init = np.array([-7.5,6,2,6]) #Starting value for x
y_init = np.array([-7,5.5,3,6]) #Starting value for y
prec = 0.000001 #To stop the algorthim and saving useless iterations
x = np.linspace(-10,10,1000) #Generating 1000 data points between (-10,10)
y = np.linspace(-10,10,1000) #Generating 1000 data points between (-10,10)
X,Y = np.meshgrid(x,y)
f_xy = X*np.sin(X) +Y*np.sin(Y) #Declaring a quadratic function with 1000 ent
#Plotting the surface plot of f_xy
fig = plt.figure()
plt.figure(figsize = [8,6])
ax = plt.axes(projection ='3d')
ax.plot_surface(X,Y,f_xy)
ax.set_xlabel("X")
ax.set_xlabel("Y")
ax.set_xlabel("Z")

#Plotting 3D contours of f_xy
fig = plt.figure()
plt.figure(figsize = [8,6])
ax = plt.axes(projection ='3d')
ax.contour3D(X, Y, f_xy,100)
ax.set_xlabel("X")
ax.set_xlabel("Y")
```

```
ax.set_xlabel("Z")


for i in range(0,len(lbda)):
    change = 1
    x_d = np.array([]) #declaring array to store values of x after each itera
    vls = np.array([]) #declaring array to store values of f_xy after each it
    y_d = np.array([]) #declaring array to store values of y after each itera
    x_cor = x_init[i]
    y_cor = y_init[i]
    while change > prec: #Condition to avoid algorithm from running forever
        x_d = np.append(x_d,x_init[i])
        y_d = np.append(y_d,y_init[i])
        vls = np.append(vls,x_init[i]*np.sin(x_init[i])+y_init*np.sin(y_init)
        tmp2 = x_init[i]
        tmp3 = y_init[i]
        tmp2 -= lbda[i]*(x_init[i]*np.cos(x_init[i])+np.sin(x_init[i]))
        tmp3 -= lbda[i]*(y_init[i]*np.cos(y_init[i])+np.sin(y_init[i]))
        change = abs(np.sqrt((x_init[i]-tmp2)**2 + (y_init[i]-tmp3)**2))
        x_init[i] = tmp2
        y_init[i] = tmp3
    #plotting the results obtained
    plt.figure(figsize = [8,6])
    plt.contour(X,Y,f_xy,100)
    plt.colorbar()
    plt.plot(x_d,y_d,'k')
    plt.plot(x_init[i],y_init[i],'r',marker = 'x')
    plt.title("x_init="+str(x_cor)+" y_cor="+str(y_cor)+" lambda="+str(lbda[i
    plt.show()
print("x=",x_init)
print("y=",y_init)
```

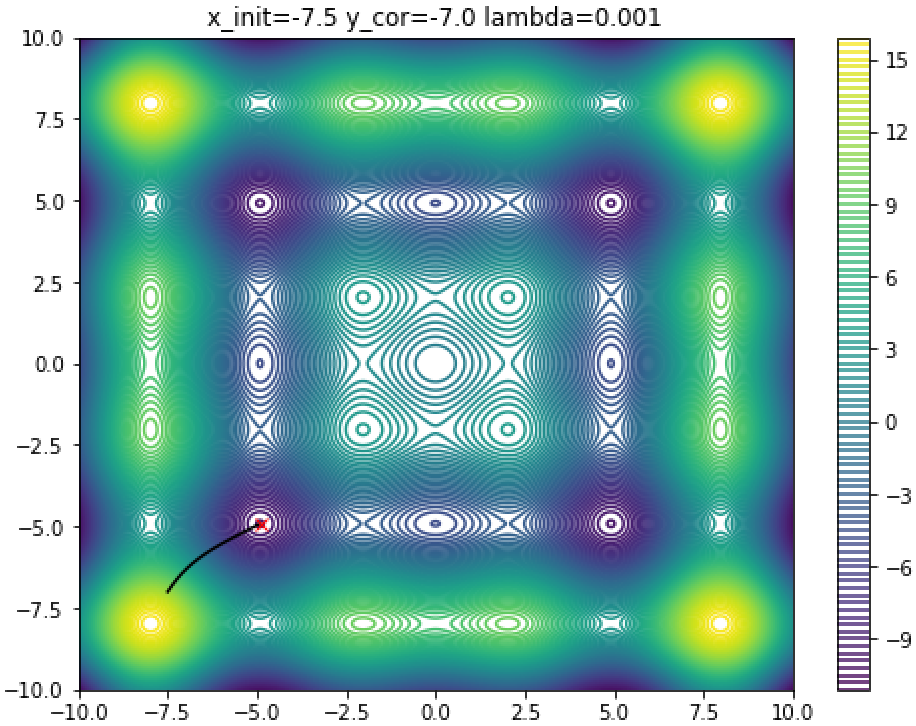
```
<Figure size 432x288 with 0 Axes>
```



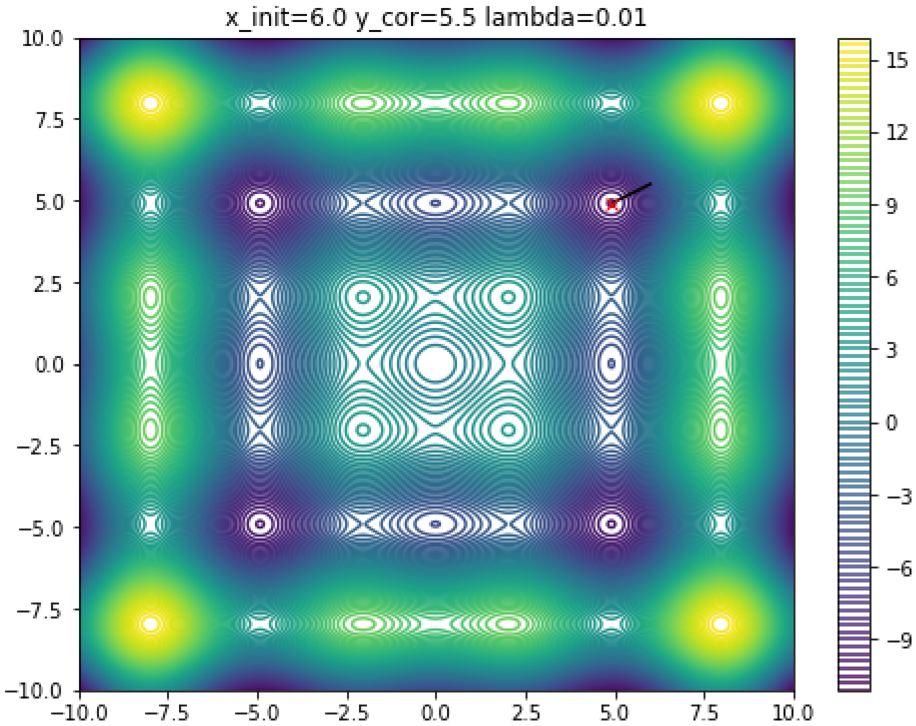

```
<Figure size 432x288 with 0 Axes>
```

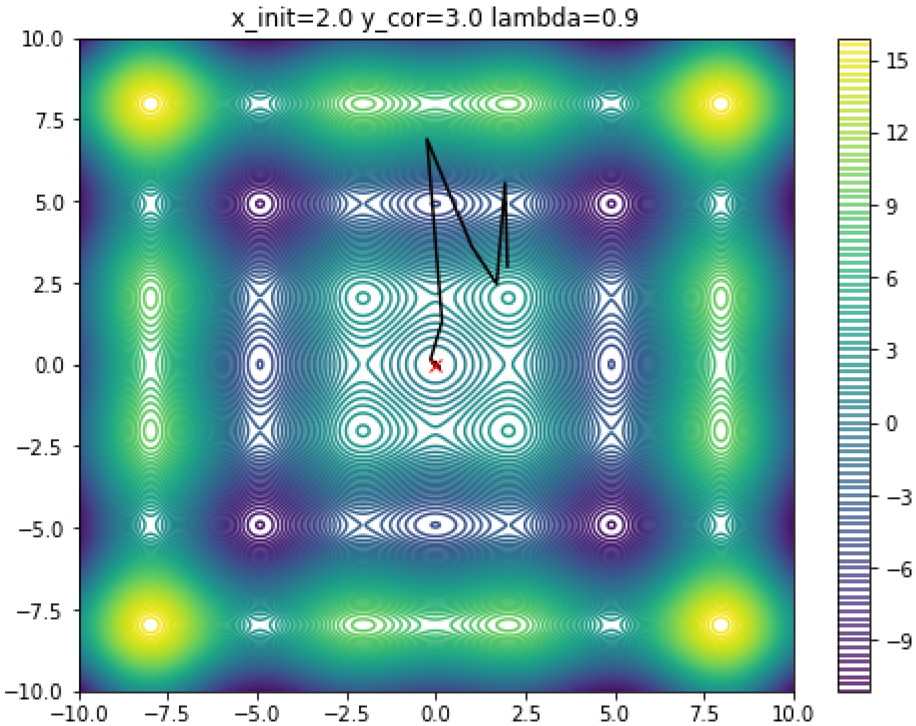


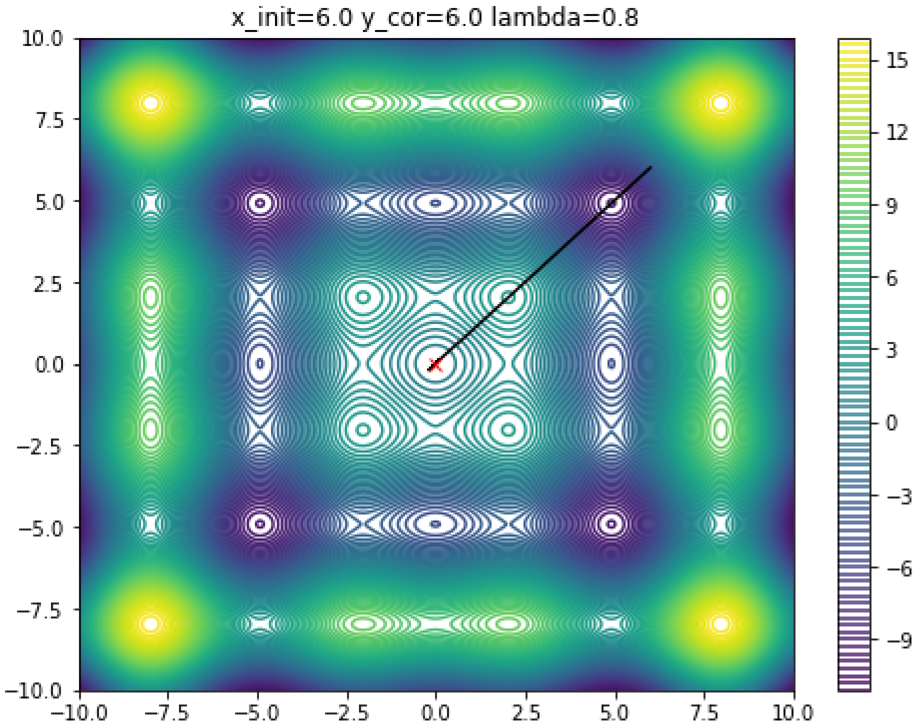
15
10
5
0
−5
−10
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
Z



x_init=-7.5 y_cor=-7.0 lambda=0.001
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
15
12
9
6
3
0
−3
−6
−9


x_init=6.0 y_cor=5.5 lambda=0.01


x_init=2.0 y_cor=3.0 lambda=0.9




```
x= [-4.91334505e+00  4.91319575e+00  2.62631237e-07  1.84357289e-07]
y= [-4.91327545e+00  4.91318878e+00 -2.51015168e-07  1.84357289e-07]
```

In [ ]: