Lab 3 : Convex Optimisation

Gradient Descent

Write the code following the instructions to obtain the desired results

Import all the required libraries

In [152]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
%matplotlib inline
```

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 :

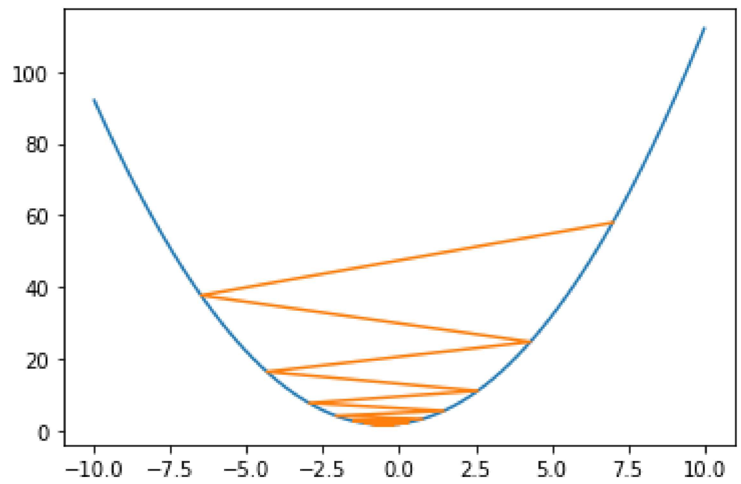
Follow the below steps and write your code in the block below

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 [153]:

```
x = np.linspace(-10,10,1000)
def Quadratic(x):
    return x**2+x+2
Quadratic = np.frompyfunc(Quadratic,1,1)



def GradientDescent(x, x_init, lam):
    """
    x_init is initial value
    x is input variable
    lam is learning rate

    """
    y = Quadratic(x)

    x_val = []

    while abs(2*x_init + 1)> 0.00002:
        x_val.append(x_init)
        x_init = x_init - lam*(2*x_init+1)


    return [x_init,x_val]

x_min, x_val = GradientDescent(x, 7, 0.9)

print("function is minimuum at : ",x_min)

plt.plot(x, Quadratic(x), x_val, Quadratic(x_val))
plt.show()
```

```
function is minimuum at :  -0.5000091949732451
```



Example 2 : $f(x) = xsinx$

Analytical : Find solution analytically

Gradient Descent Update equation : Write Gradient descent update equations

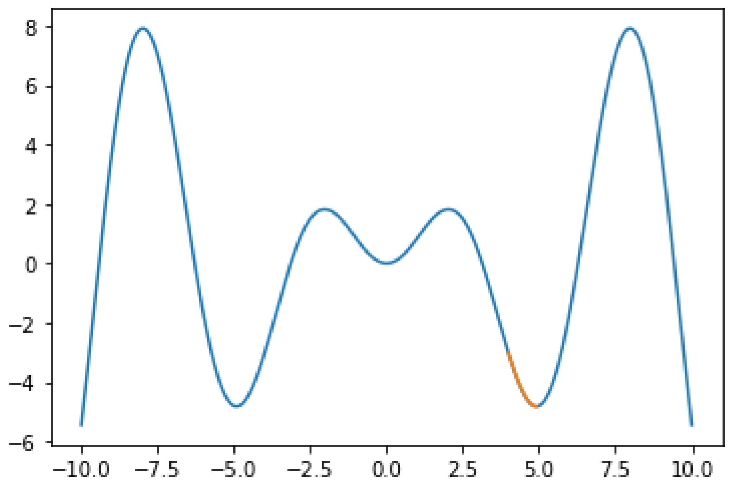
Gradient Descent Method :

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 [154]:
```
x = np.linspace(-10,10,1000)
def xsinx(x):
    return x*np.sin(x)
xsinx = np.frompyfunc(xsinx,1,1)



def GradientDescent(x, x_init, lam):
    """
    x_init is initial value
    x is input variable
    lam is learning rate

    """
    y = xsinx(x)

    x_val = []

    while abs(np.sin(x_init) + x_init*np.cos(x_init))> 0.00002:
        x_val.append(x_init)
        x_init = x_init - lam*(np.sin(x_init) + x_init*np.cos(x_init))


    return [x_init,x_val]

x_min, x_val = GradientDescent(x, 4, 0.1)

print("function is minimum at : ",x_min)

plt.plot(x, xsinx(x))
plt.plot(x_val, xsinx(x_val))
plt.show()
```

```
function is minimum at :  4.91317756297795
```



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

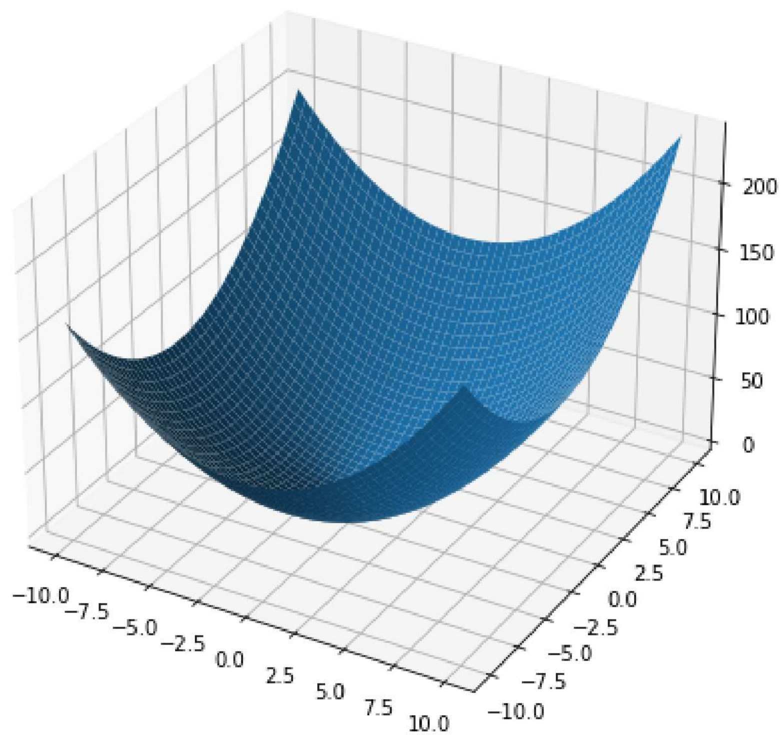
1. Generate x and y, 1000 data points from -10 to 10
2. Generate and Plot the function $f(x, y) = x^2 + y^2 + 2x + 2y$
3. Initialize the starting point (x_{init},y_{init}) and learning rate (λ)
4. Use Gradient descent algorithm to compute value of x and y at which the function $f(x, y)$ is minimum
5.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

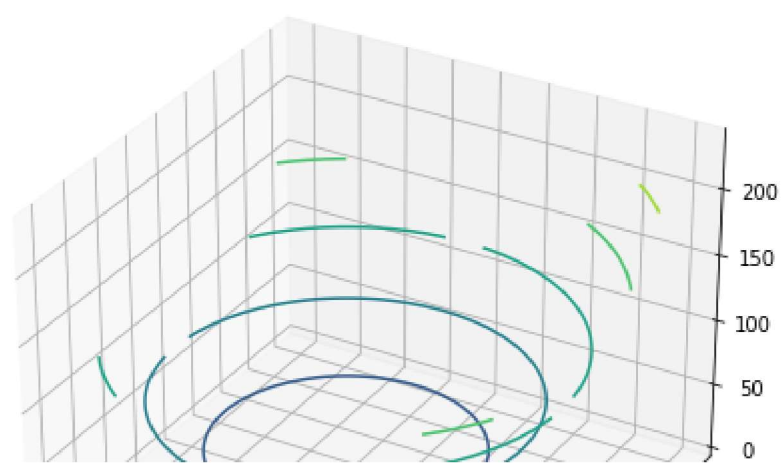
In [155]:
```
from turtle import color


x = np.linspace(-10,10,1000)
y = np.linspace(-10,10,1000)


def f(x,y):
    return np.array(x**2+ y**2+ 2*x+ 2*y)




def GradientDescent(x,y, x_init, y_init, lam):
    """
    x_init is initial value
    x is input variable
    lam is learning rate

    """

    x_val = []
    y_val = []

    while abs(2*x_init + 2) > 0.00002 and abs(2*y_init + 2) > 0.00002 :
        x_val.append(x_init)
        y_val.append(y_init)
        x_init = x_init - lam*(2*x_init + 2)
        y_init = y_init - lam*(2*y_init + 2)


    return [x_init, x_val, y_init, y_val]

x_min, x_val, y_min , y_val = GradientDescent(x, y, 4, 4, 0.3)

print("function is minimum at : {}, {}".format(x_min, y_min))


fig = plt.figure(figsize=(16,16))
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
# syntax for 3-D projection
```

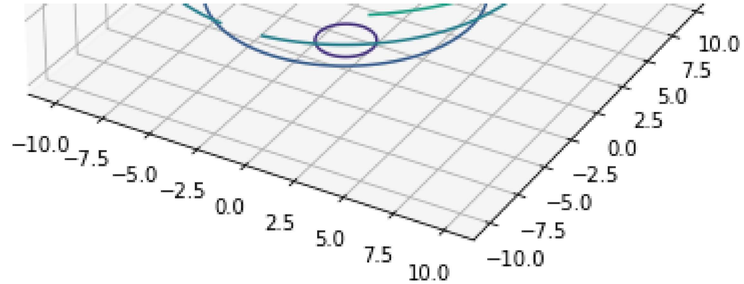
```
ax = fig.add_subplot(2, 1, 1, projection='3d')
ax.plot_surface(X, Y, Z )

ax = fig.add_subplot(2, 1, 2, projection='3d')
ax.contour3D(X, Y, Z )

plt.figure()
plt.contour(X,Y,Z)
plt.plot(x_val,y_val,color="000000")
```

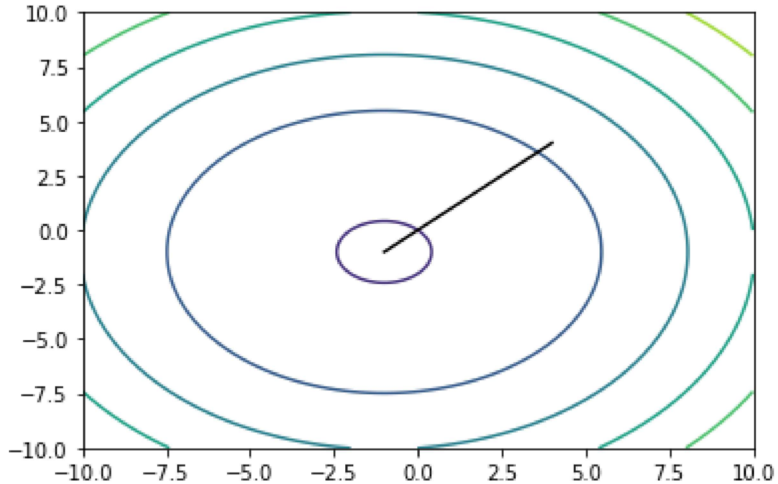
```
function is minimum at : -0.99999463129088, -0.99999463129088
```

Out[155]:
```
[<matplotlib.lines.Line2D at 0x1d259478160>]
```


200
150
100
50
0
−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


200
150
100
50
0





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 [156]:

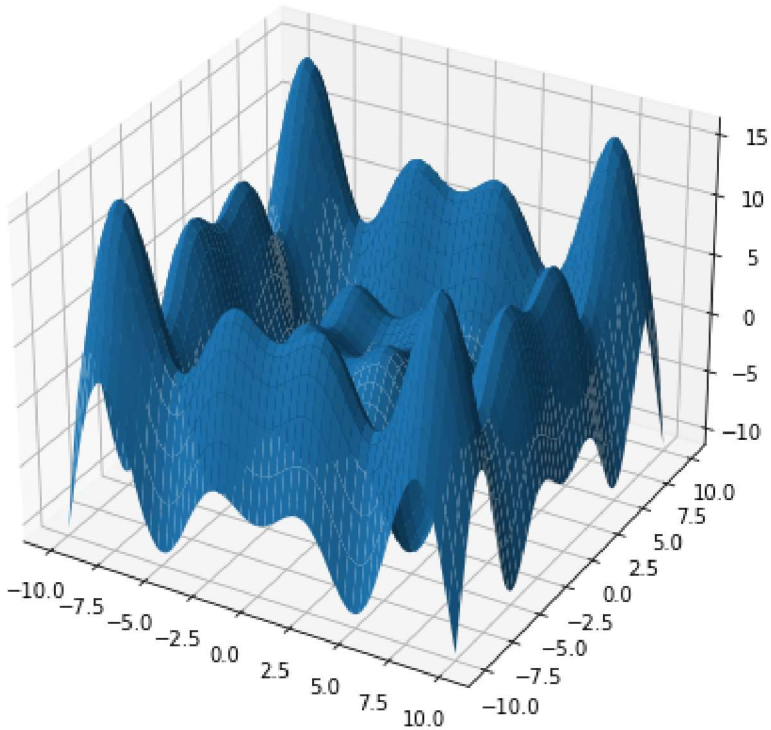
```
x = np.linspace(-10,10,1000)
y = np.linspace(-10,10,1000)


def f(x,y):
    return np.array(x*np.sin(x)+ y*np.sin(y))




def GradientDescent(x,y, x_init, y_init, lam):
    """
    x_init is initial value
    x is input variable
    lam is learning rate

    """

    x_val = []
    y_val = []

    while abs(np.sin(x_init) + x_init*np.cos(x_init)) > 0.00002 and abs(np.sin(y_init) + y_init*np.cos(y_init
)) > 0.00002 :
        x_val.append(x_init)
        y_val.append(y_init)
        x_init = x_init - lam*(np.sin(x_init) + x_init*np.cos(x_init))
        y_init = y_init - lam*(np.sin(y_init) + y_init*np.cos(y_init))


    return [x_init, x_val, y_init, y_val]

x_min, x_val, y_min , y_val = GradientDescent(x, y, 4, 4, 0.3)

print("function is minimum at : {}, {}".format(x_min, y_min))


fig = plt.figure(figsize=(16,16))
X, Y = np.meshgrid(x, y)
Z = f(X, Y)
# syntax for 3-D projection
ax = fig.add_subplot(2, 1, 1, projection='3d')
ax.plot_surface(X, Y, Z )
```

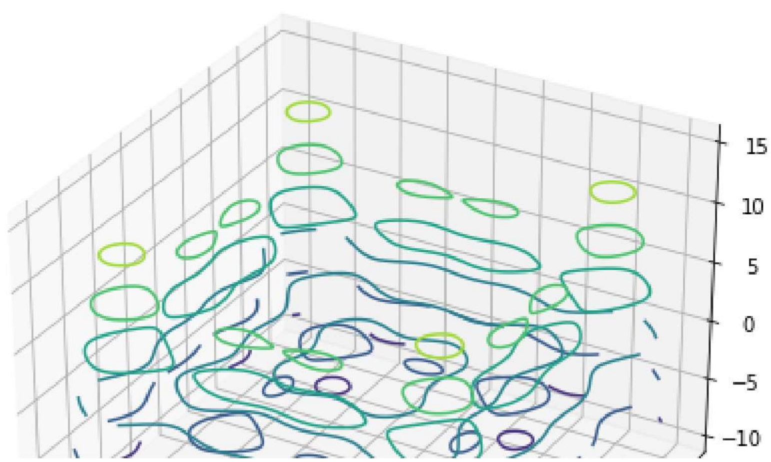
```
ax = fig.add_subplot(2, 1, 2, projection='3d')
ax.contour3D(X, Y, Z )

plt.figure()
plt.contour(X,Y,Z)
plt.plot(x_val,y_val, color="000000")
```

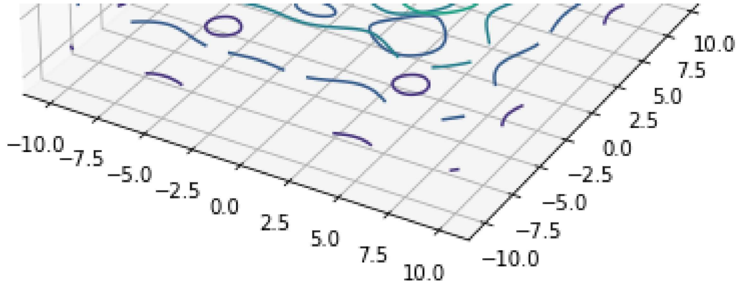
```
function is minimum at : 4.91318378987333375, 4.91318378987333375
```

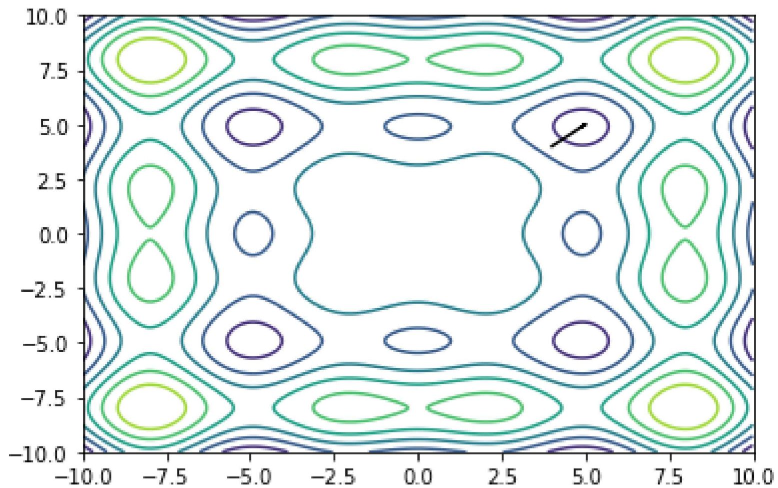
Out[156]:

```
[<matplotlib.lines.Line2D at 0x1d2363f5910>]
```


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


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


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