Lab 3 : Convex Optimisation

Gradient Descent

Write the code following the instructions to obtain the desired results

Import all the required libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%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 :

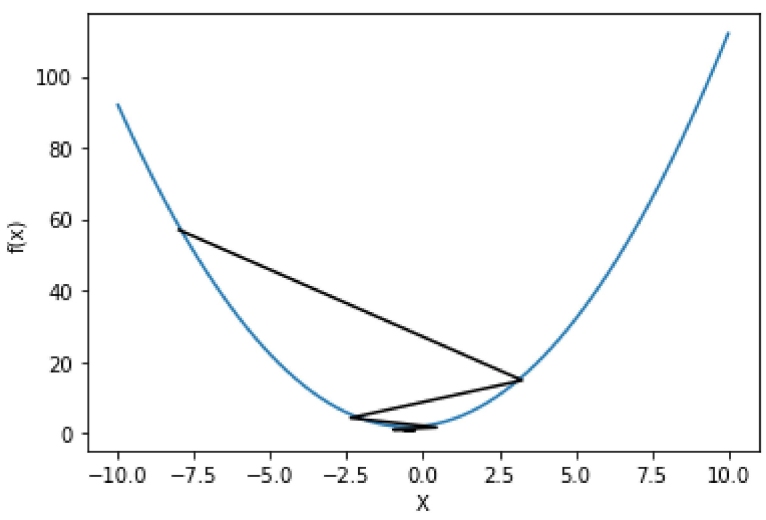
Follow the below steps and write your code in the block below

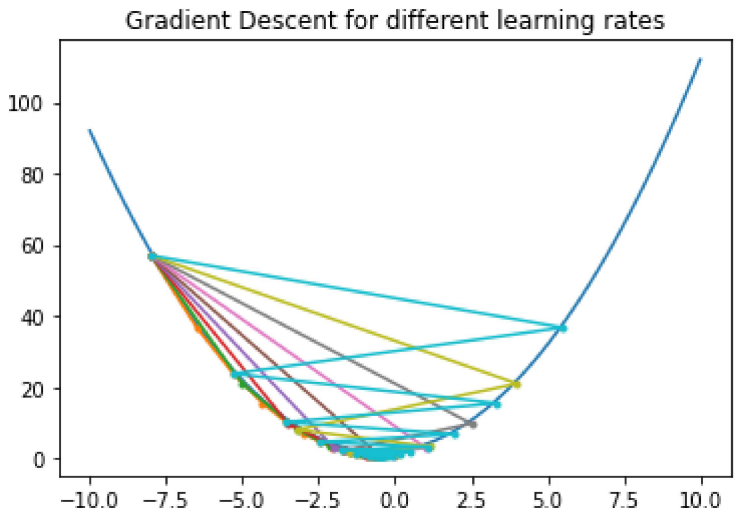
1. Generate x, 1000 data points from -10 to 10
2. Generate and Plot the function $f(x) = x^2 + x + 2$
3. Initialize the starting point (x_{init}) and learning rate (λ)
4. Use Gradient descent algorithm to compute value of x at which the function $f(x)$ is minimum
5. Also vary the learning rate and initialisation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

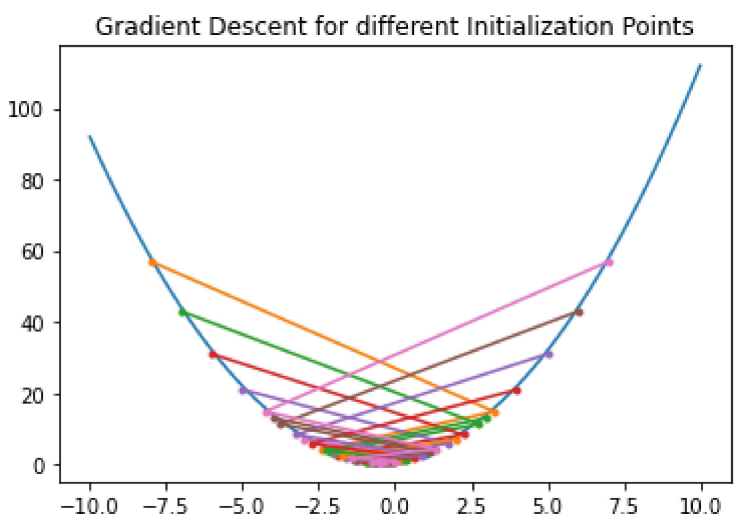
In [2]:
```
#GRADIENT DESCENT ALOGRITHM

#Generate  x , 1000 data points from -10 to 10
x = np.linspace(-10, 10, 1000)

#Generate and Plot the function f(x) = x2 + x +  2
y = x**2 + x + 2

fig1 = plt.figure("Figure 1")
plt.plot(x, y)
plt.xlabel('X')
plt.ylabel('f(x)')

#Initialize the starting point ( xinit ) and learning rate ( λ )
xInit = -8
learningRate = 0.75
```

```
iterations = 50

#Use Gradient descent algorithm to compute value of  x  at which the function  f(x)  is minimum
def GradientDescent(rate, init, iterations):
    solutions = [init]

    for i in range(1, iterations):
        init = init - rate*(2*init + 1)
        solutions.append(init)

    return init, solutions

#Use Gradient descent algorithm to compute value of  x  at which the function  f(x)  is minimum
print('The minimum value of x is', GradientDescent(learningRate, xInit, iterations)[0])

x_val = np.array(GradientDescent(learningRate, xInit, iterations)[1])
y_val = x_val**2 + x_val + 1

plt.plot(x_val, y_val, '-', color='black')
plt.show()

#Also vary the learning rate and initialisation point and plot your observations

rateVector = np.arange(0.1, 1, 0.1)
#replotting the function
fig2 = plt.figure("Figure 2")
plt.plot(x, y)
plt.title('Gradient Descent for different learning rates')

for i in range(rateVector.shape[0]):
    x_vec = np.array(GradientDescent(rateVector[i], xInit, iterations)[1])
    y_vec = x_vec**2 + x_vec + 1

    plt.plot(x_vec, y_vec, '.-')
    plt.show

InitializationPoints = np.arange(-8, 8, 1)
#replotting the function
fig3 = plt.figure("Figure 3")
plt.plot(x, y)
plt.title('Gradient Descent for different Initialization Points')

for i in range(InitializationPoints.shape[0]):
    x_vec = np.array(GradientDescent(learningRate, InitializationPoints[i], iterations)[1])
    y_vec = x_vec**2 + x_vec + 1

    plt.plot(x_vec, y_vec, '.-')
    plt.show
```

```
The minimum value of x is -0.4999999999999867
```







Example 2 : $f(x) = xsinx$

Analytical :

$$\frac{d}{dx} f(x) = xsinx = 0$$

$$x = -tanx \ (analytical\ solution)$$

Gradient Descent Update equation : Write Gradient descent update equations

Gradient Descent Method :

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

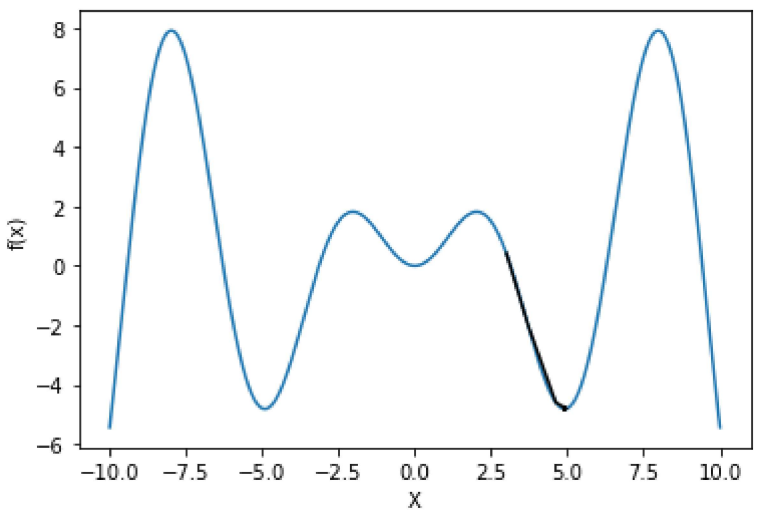
1. Generate x, 1000 data points from -10 to 10
2. Generate and Plot the function $f(x) = x^2 + x + 2$
3. Initialize the starting point (x_{init}) and learning rate (λ)
4. Use Gradient descent algorithm to compute value of x at which the function $f(x)$ is minimum
5. Also vary the learning rate and initialisation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

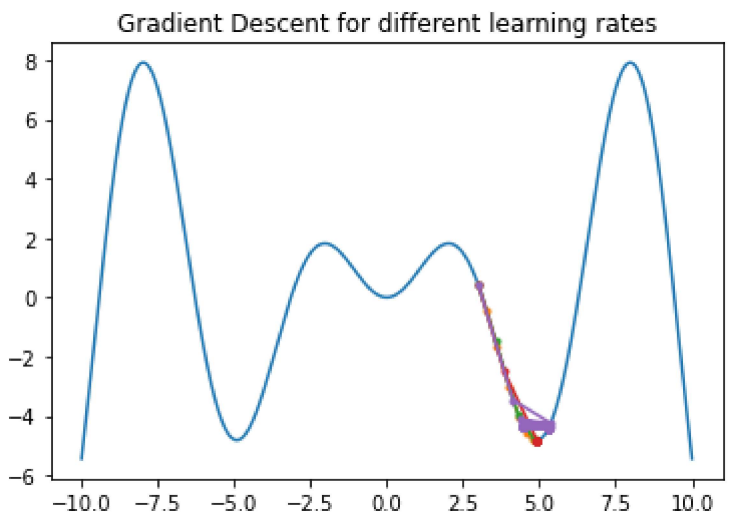
In [3]:

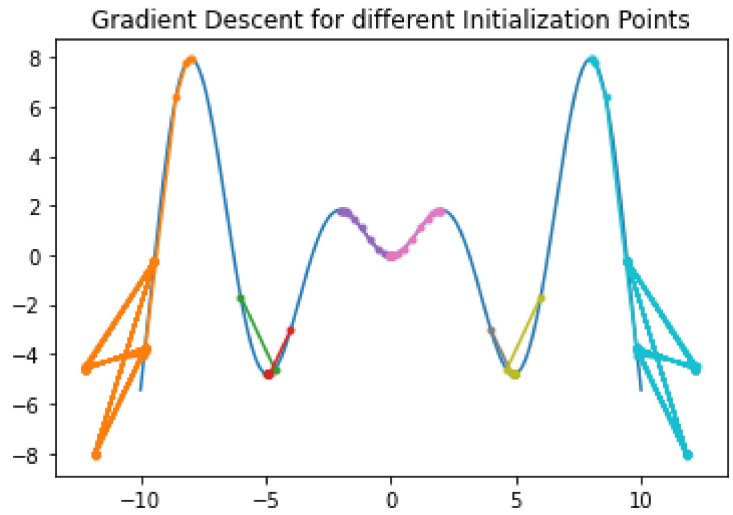
```
#GRADIENT DESCENT ALOGRITHM

#Generate  x , 1000 data points from -10 to 10
x = np.linspace(-10, 10, 1000)

#Generate and Plot the function f(x) = x2 + x +  2
y = (np.sin(x))*x

fig1 = plt.figure("Figure 1")
plt.plot(x, y)
plt.xlabel('X')
plt.ylabel('f(x)')

#Initialize the starting point ( xinit ) and learning rate ( λ )
xInit = 3
learningRate = 0.25
iterations = 50

#Use Gradient descent algorithm to compute value of  x  at which the function  f(x)  is minimum
def GradientDescent(rate, init, iterations):
    solutions = [init]
```

```
    for i in range(1, iterations):
        init = init - rate*(init*np.cos(init) + np.sin(init))
        solutions.append(init)

    return init, solutions

#Use Gradient descent algorithm to compute value of  x  at which the function  f(x)  is minimum
print('The minimum value of x is', GradientDescent(learningRate, xInit, iterations)[0])

x_val = np.array(GradientDescent(learningRate, xInit, iterations)[1])
y_val = x_val*np.sin(x_val)

plt.plot(x_val, y_val, '-', color='black')
plt.show()

#Also vary the learning rate and initialisation point and plot your observations

rateVector = np.arange(0.1, 0.5, 0.1)
#replotting the function
fig2 = plt.figure("Figure 2")
plt.plot(x, y)
plt.title('Gradient Descent for different learning rates')

for i in range(rateVector.shape[0]):
    x_vec = np.array(GradientDescent(rateVector[i], xInit, iterations)[1])
    y_vec = x_vec*np.sin(x_vec)

    plt.plot(x_vec, y_vec, '.-')
    plt.show

InitializationPoints = np.arange(-8, 10, 2)
#replotting the function
fig3 = plt.figure("Figure 3")
plt.plot(x, y)
plt.title('Gradient Descent for different Initialization Points')

for i in range(InitializationPoints.shape[0]):
    x_vec = np.array(GradientDescent(learningRate, InitializationPoints[i], iterations)[1])
    y_vec = x_vec*np.sin(x_vec)

    plt.plot(x_vec, y_vec, '.-')
    plt.show
```

```
The minimum value of x is 4.913180439434884
```







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

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

Analytical :

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

$$x = -1, y = -1 \; (analytical \; solution)$$

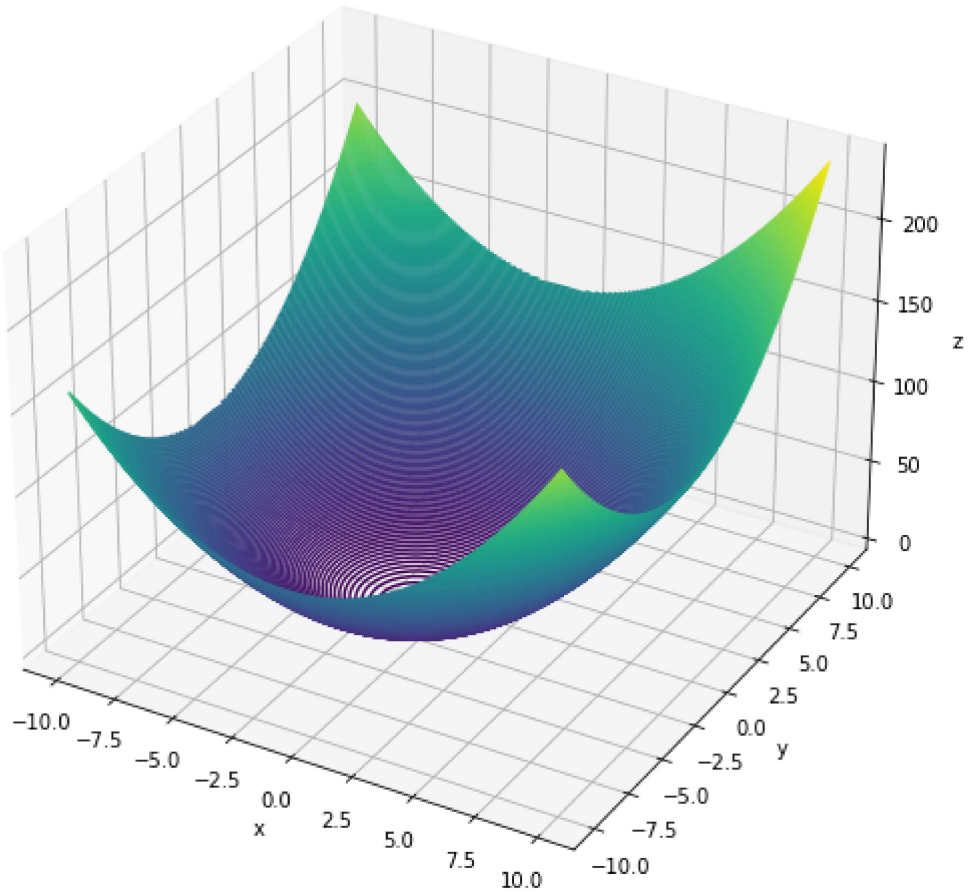
Gradient Descent Method :

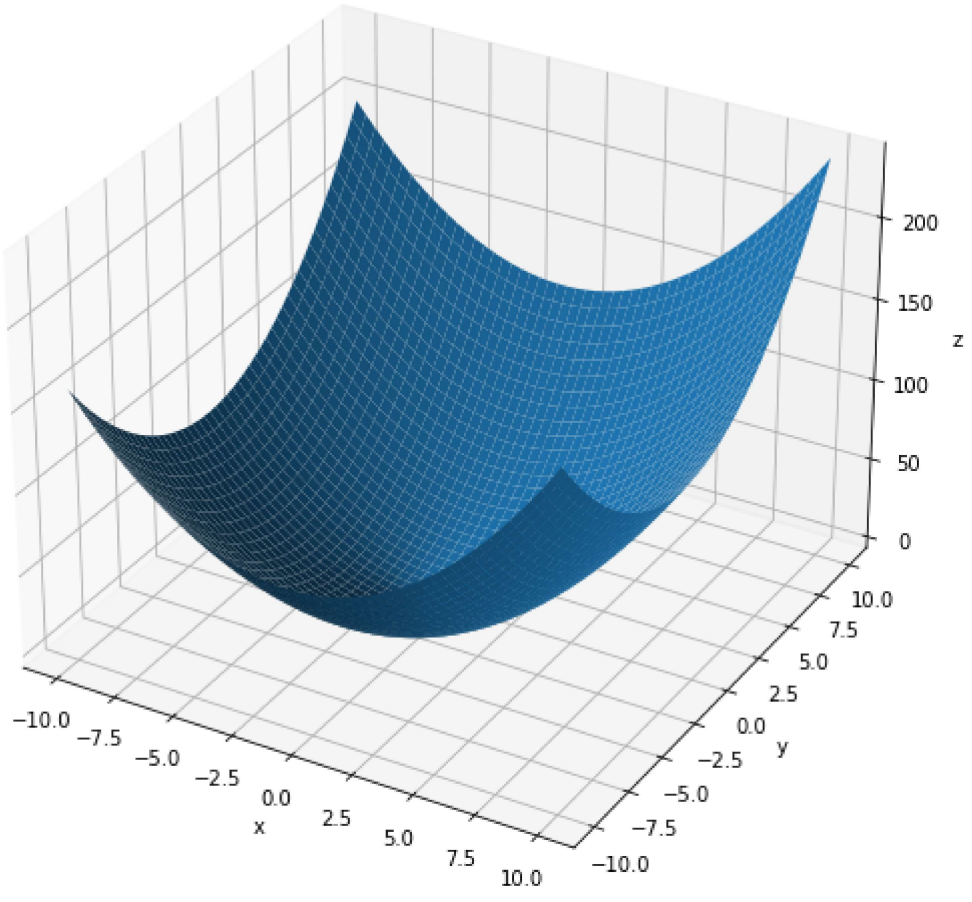
Follow the below steps and write your code in the block below

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 [4]:

```
#Gradient desent for finding minima for 2 variables functions

#Generate  x  and  y , 1000 data points from -10 to 10
x = np.linspace(-10, 10, 1000)
y = np.linspace(-10, 10, 1000)

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

#Generate and Plot the function f(x,y) =  x2+y2+2x+2y
def f(x, y):
    return x**2 + y**2 + 2*x + 2*y

Z = f(X, Y)

#contour plot
fig = plt.figure(figsize=(16,9))
ax = plt.axes(projection='3d')
ax.contour3D(X, Y, Z, 250)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z');

#surface plot
fig = plt.figure(figsize=(16,9))
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z');
```





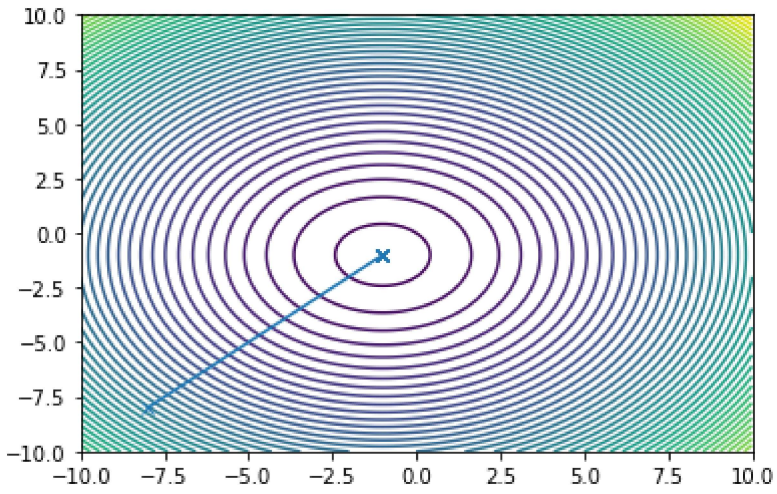
In [5]:

```
#Initialize the starting point (xinit , yinit) and learning rate ( λ )

xInit = -8
yInit = -8
learningRate = 0.5
iterations = 10

def GradientDescentMultiVariable(rate, x_init, y_init, iterations):
    Xsolutions = [x_init]
    Ysolutions = [y_init]
    for i in range(1, iterations):
        x_init = x_init - rate*(2*x_init + 2)
        y_init = y_init - rate*(2*y_init + 2)
        Xsolutions.append(x_init)
        Ysolutions.append(y_init)
```

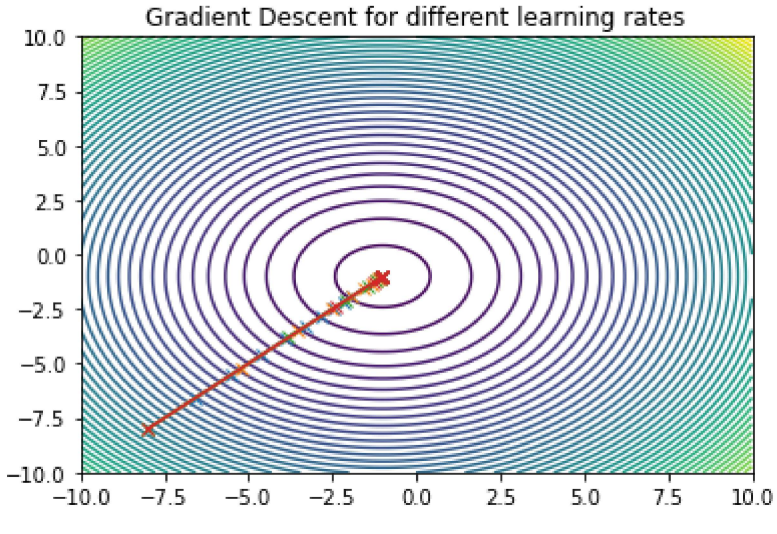
```
    return x_init, y_init, Xsolutions, Ysolutions

x_optimal, y_optimal, X_vector, Y_vector = GradientDescentMultiVariable(learningRate, xInit, yInit, iteration

#Use Gradient descent algorithm to compute value of  x  and  y  at which the function  f(x,y)  is minimum
print('The minima of this function calculated through Gradient descent is [', x_optimal, y_optimal, ']')

plt.contour(X, Y, Z, 50)
plt.plot(X_vector, Y_vector, '-x')
```

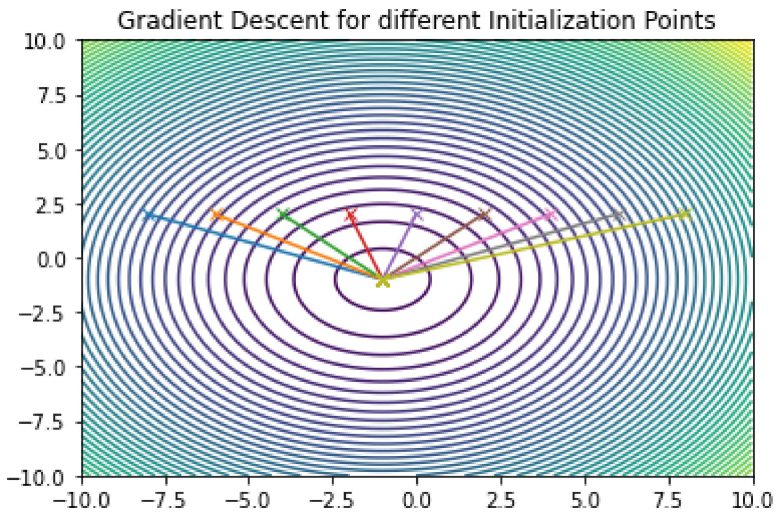
```
The minima of this function calculated through Gradient descent is [ -1.0 -1.0 ]
```

Out[5]:
```
[<matplotlib.lines.Line2D at 0x22272b93c10>]
```



In [6]:
```
#Also vary the learning rate and initialisation point and plot your observations

rateVector = np.arange(0.1, 0.5, 0.1)

#replotting the function
fig2 = plt.figure("Figure 2")
plt.contour(X, Y, Z, 50)
plt.title('Gradient Descent for different learning rates')

for i in range(rateVector.shape[0]):
    x_vec = np.array(GradientDescentMultiVariable(rateVector[i], xInit, yInit, iterations)[2])
    y_vec = np.array(GradientDescentMultiVariable(rateVector[i], xInit, yInit, iterations)[3])
    z_vec = x_vec**2 + y_vec**2 + 2*x_vec + 2*y_vec

    plt.plot(x_vec, y_vec, '-x')
    plt.show

InitializationPoints_x = np.arange(-8, 10, 2)
InitializationPoints_y = np.repeat(2, InitializationPoints_x.shape)

#replotting the function
fig3 = plt.figure("Figure 3")
plt.contour(X, Y, Z, 50)
plt.title('Gradient Descent for different Initialization Points')

for i in range(InitializationPoints.shape[0]):
    x_vec = np.array(GradientDescentMultiVariable(learningRate, InitializationPoints_x[i], InitializationPoin
    y_vec = np.array(GradientDescentMultiVariable(learningRate, InitializationPoints_x[i], InitializationPoin
    z_vec = x_vec**2 + y_vec**2 + 2*x_vec + 2*y_vec

    plt.plot(x_vec, y_vec, '-x')
    plt.show
```





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

Analytical :

$$\frac{d}{dx}f(x) = xsinx = 0$$

$$x = -tanx, y = -tanx\ (analytical\ solution)$$

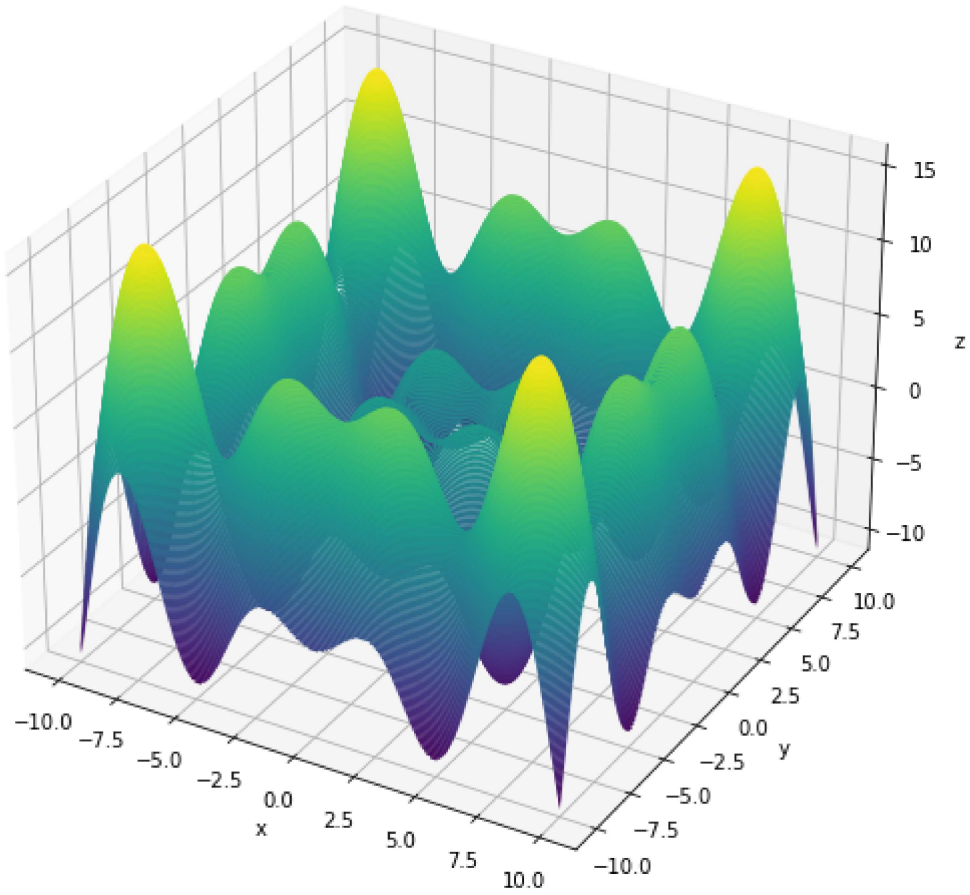
Gradient Descent Method :

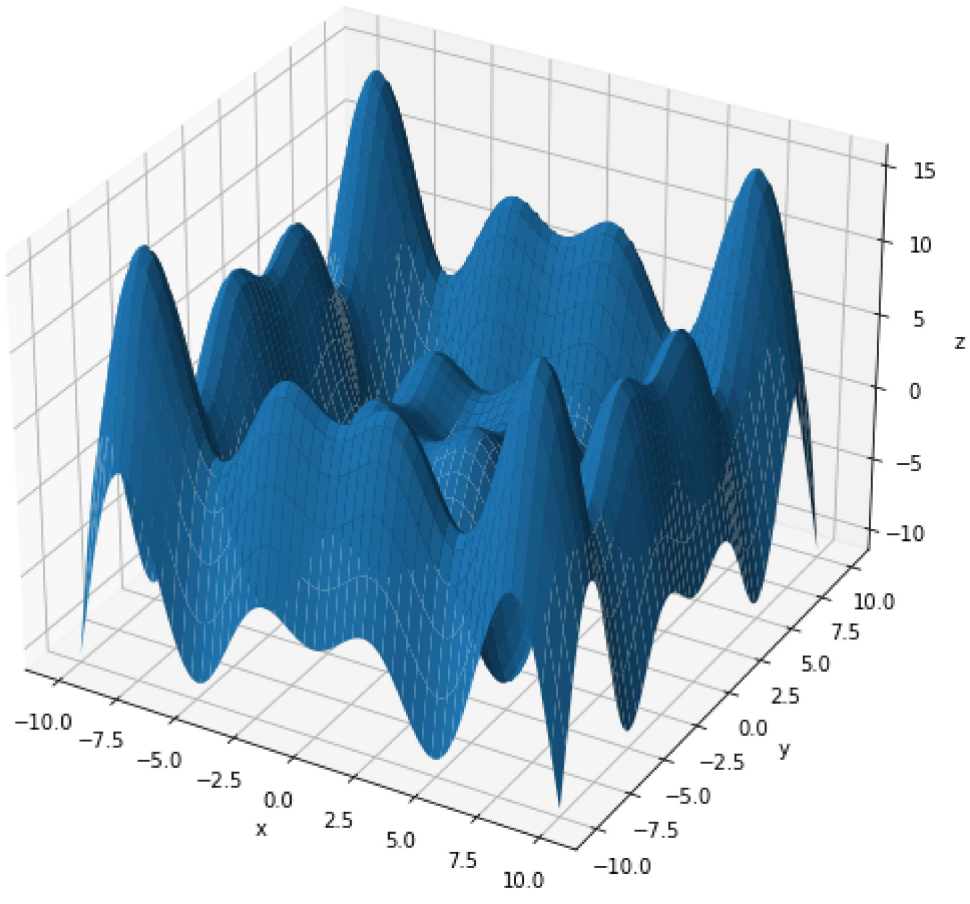
Follow the below steps and write your code in the block below

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 [7]:

```
#Gradient desent for finding minima for 2 variables functions

#Generate  x  and  y , 1000 data points from -10 to 10
x = np.linspace(-10, 10, 1000)
y = np.linspace(-10, 10, 1000)

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

#Generate and Plot the function f(x,y) =  x2+y2+2x+2y
def f(x, y):
    return x*np.sin(x) + y*np.sin(y)

Z = f(X, Y)

#contour plot
fig = plt.figure(figsize=(16,9))
ax = plt.axes(projection='3d')
ax.contour3D(X, Y, Z, 250)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z');

#surface plot
fig = plt.figure(figsize=(16,9))
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z');
```





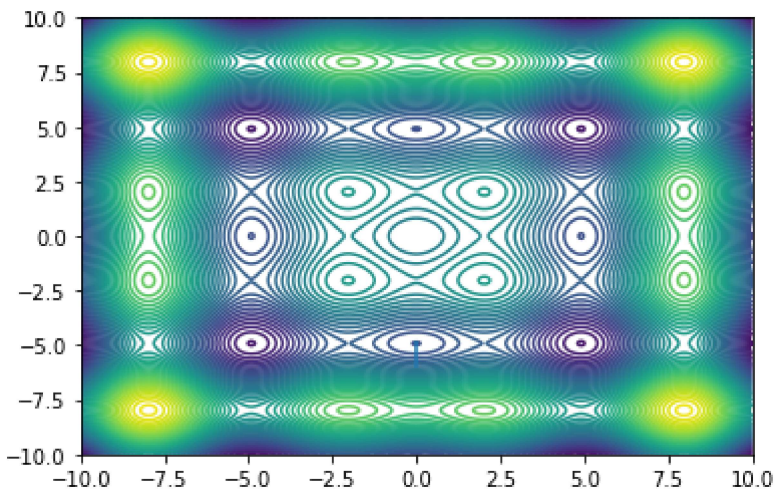
In [8]:

```
#Initialize the starting point (xinit , yinit) and learning rate ( λ )

xInit = 0
yInit = -6
learningRate = 0.1
iterations = 20

def GradientDescentMultiVariable(rate, x_init, y_init, iterations):
    Xsolutions = [x_init]
    Ysolutions = [y_init]
    for i in range(1, iterations):
        x_init = x_init - rate*(x_init*np.cos(x_init) + np.sin(x_init))
        y_init = y_init - rate*(y_init*np.cos(y_init) + np.sin(y_init))
        Xsolutions.append(x_init)
        Ysolutions.append(y_init)
```

```
    return x_init, y_init, Xsolutions, Ysolutions

x_optimal, y_optimal, X_vector, Y_vector = GradientDescentMultiVariable(learningRate, xInit, yInit, iteration

#Use Gradient descent algorithm to compute value of  x  and  y  at which the function  f(x,y)  is minimum
print('The minima of this function calculated through Gradient descent is [', x_optimal, y_optimal, ']')

plt.contour(X, Y, Z, 50)
plt.plot(X_vector, Y_vector, '-')
```

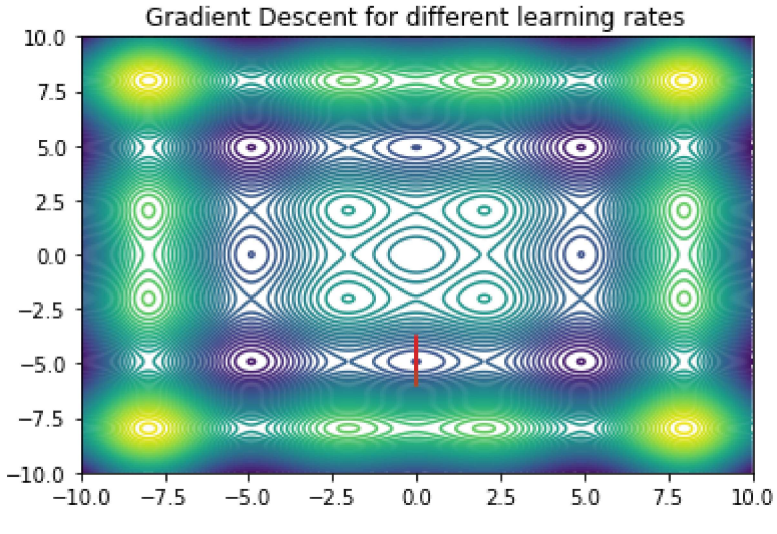
```
The minima of this function calculated through Gradient descent is [ 0.0 -4.913181262489657 ]
```

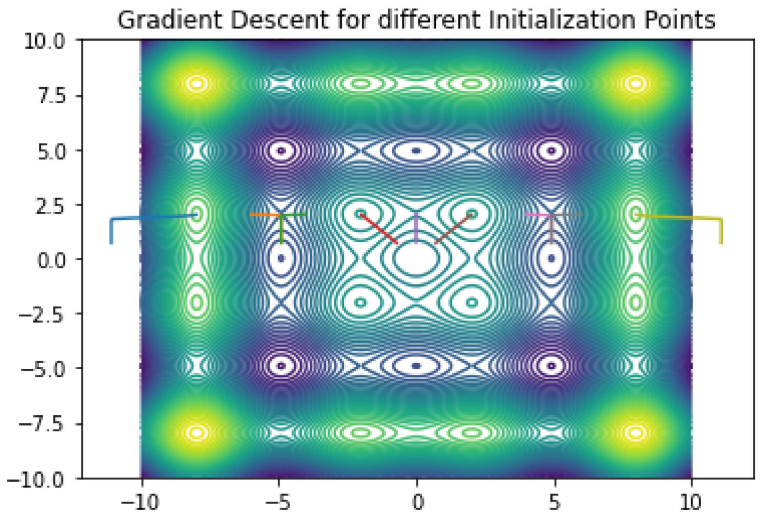
Out[8]: [<matplotlib.lines.Line2D at 0x22200cc6160>]



In [9]:

```
#Also vary the learning rate and initialisation point and plot your observations

rateVector = np.arange(0.1, 0.5, 0.1)

#replotting the function
fig2 = plt.figure("Figure 2")
plt.contour(X, Y, Z, 50)
plt.title('Gradient Descent for different learning rates')

for i in range(rateVector.shape[0]):
    x_vec = np.array(GradientDescentMultiVariable(rateVector[i], xInit, yInit, iterations)[2])
    y_vec = np.array(GradientDescentMultiVariable(rateVector[i], xInit, yInit, iterations)[3])
    z_vec = x_vec**2 + y_vec**2 + 2*x_vec + 2*y_vec

    plt.plot(x_vec, y_vec, '-')
    plt.show

InitializationPoints_x = np.arange(-8, 10, 2)
InitializationPoints_y = np.repeat(2, InitializationPoints_x.shape)

#replotting the function
fig3 = plt.figure("Figure 3")
plt.contour(X, Y, Z, 50)
plt.title('Gradient Descent for different Initialization Points')

for i in range(InitializationPoints.shape[0]):
    x_vec = np.array(GradientDescentMultiVariable(learningRate, InitializationPoints_x[i], InitializationPoin
    y_vec = np.array(GradientDescentMultiVariable(learningRate, InitializationPoints_x[i], InitializationPoin
    z_vec = x_vec**2 + y_vec**2 + 2*x_vec + 2*y_vec

    plt.plot(x_vec, y_vec, '-')
    plt.show
```




Gradient Descent for different Initialization Points
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10
−5
0
5
10

In [ ]: