#Lab 3 : Convex Optimisation

Gradient Descent

Write the code following the instructions to obtain the desired results

#Import all the required libraries

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

Solver for One Variable gradient descent

```
def solver(_lambda, x_init, max_iter, epsilon, max_val, sols, grad, func):
    figure, axis = plt.subplots(len(_lambda), len(x_init), figsize=(7*len(x_init), 7*len(_lambda)))
    min_err = float('inf')
    for i, lr in enumerate(_lambda):
        for j, x in enumerate(x_init):
            x_test = [x]
            for t in range(max_iter):
                x = x - lr * grad(x)
                x_test.append(x)
                if abs(lr * grad(x))<epsilon: break
                if abs(lr * grad(x))>max_val: break
            x_test = np.array(x_test)
            y_test = func(x_test)
            delta = abs(np.max(x_test)-np.min(x_test))
            X = np.linspace(start=np.min(x_test)-delta, stop=np.max(x_test)+delta, num=1000)
            Y = func(X)
            axis[i, j].plot(X, Y)
            for sol in sols:
                axis[i, j].plot([sol], [func(sol)], marker='x', color='green')
            err = round(abs(x-sols[0]), 7)
            axis[i, j].plot(x_test, y_test, color='black')
            axis[i, j].set_title(f'LR: {lr}, x_init: {x_init[j]}, iter: {t}, err: {err}')
            axis[i, j].set_xlabel(f'x_sol: {round(x, 7)}')
            axis[i, j].set_ylabel('f(x)')
            if err<min_err:
                min_err = err
                min_sol = x
    print("-"*100)
    print(f'The value at which f(x) is minimum is: x = {min_sol}')
    print("-"*100)
```

Plotter for One Variable function

```
def plotter(x, func, sols):
    y = func(x)
    plt.plot(x, y)
    for sol in sols:
        plt.plot([sol], [func(sol)], marker='x', color='green')
    plt.title(f'Plot of x vs f(x), Min at x={sol}')
    plt.xlabel('x')
    plt.ylabel('f(x)')
```

Solver for Two Variable gradient Descent

```
def solver_2D(_lambda, p_init, max_iter, epsilon, max_val, sols,
grad_x, grad_y, func):
    figure, axis = plt.subplots(len(_lambda), len(p_init),
figsize=(7*len(p_init), 7*len(_lambda)))
    min_err = float('inf')
    for i, lr in enumerate(_lambda):
        for j, p in enumerate(p_init):
            x, y = p
            x_test = [x]
            y_test = [y]
            for t in range(max_iter):
                x = x - lr * grad_x(x, y)
                y = x - lr * grad_y(x, y)
                x_test.append(x)
                y_test.append(y)
                if abs(lr * grad_x(x, y)) + abs(lr * grad_y(x,
y))<epsilon: break
                if abs(lr * grad_x(x, y)) + abs(lr * grad_y(x,
y))>max_val: break
            x_test = np.array(x_test)
            y_test = np.array(y_test)
            z_test = func(x_test, y_test)
            delta_x = abs(np.max(x_test)-np.min(x_test))
            delta_y = abs(np.max(y_test)-np.min(y_test))
            X = np.linspace(start=min(np.min(x_test)-delta_x, -10),
stop=max(np.max(x_test)+delta_x, 10), num=1000)
            Y = np.linspace(start=min(np.min(y_test)-delta_y, -10),
stop=max(np.max(y_test)+delta_y, 10), num=1000)
            X, Y = np.meshgrid(X, Y)
            Z = func(X, Y)
            axis[i, j].contour(X, Y, Z, 70)
            err = round(abs(x-sols[0][0]), 6) + round(abs(y-sols[0]
[1]), 6)
            axis[i, j].plot(x_test, y_test, color='black')
            axis[i, j].plot([sols[0][0]], [sols[0][1]], color='green',
marker='x')
            axis[i, j].set_title(f'LR: {lr}, p_init: {p_init[j]},
iter: {t}, err: {err}')
            axis[i, j].set_xlabel(f'x_sol: {round(x, 7)}')
```

```
            axis[i, j].set_ylabel(f'y_sol: {round(y, 7)}')
            if err<min_err:
                min_err = err
                min_sol = (x, y)
    print("-"*100)
    print(f'The value at which f(x, y) is minimum is: x = 
{min_sol[0]}, y = {min_sol[1]}')
    print("-"*100)
```

Plotter for Two Variable Function

```
def plotter_3D(x, y, func):
    X, Y = np.meshgrid(x, y)
    z = func(X, Y)
    fig1 = plt.figure(figsize=(7, 7))
    ax1 = plt.axes(projection='3d')
    ax1.plot_surface(x, y, z)
    ax1.set_title(f'3D Surface plot of x, y vs f(x, y)')
    ax1.set_xlabel('x')
    ax1.set_ylabel('y')
    ax1.autoscale_view(8)

    fig2 = plt.figure(figsize=(7, 7))
    ax2 = plt.axes(projection='3d')
    ax2.contour3D(x, y, z, 150)
    ax2.set_title(f'3D Contour plot of x, y vs f(x, y)')
    ax2.set_xlabel('x')
    ax2.set_ylabel('y')
```

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 :

$$x_{init} = 4$$

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

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

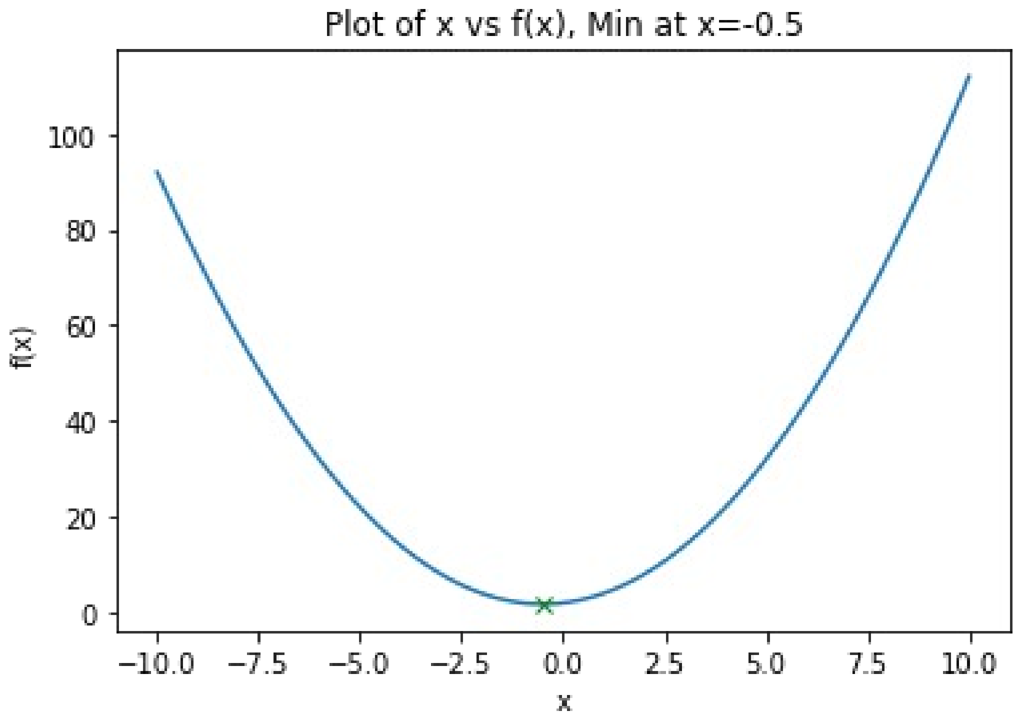
Gradient Descent Method :

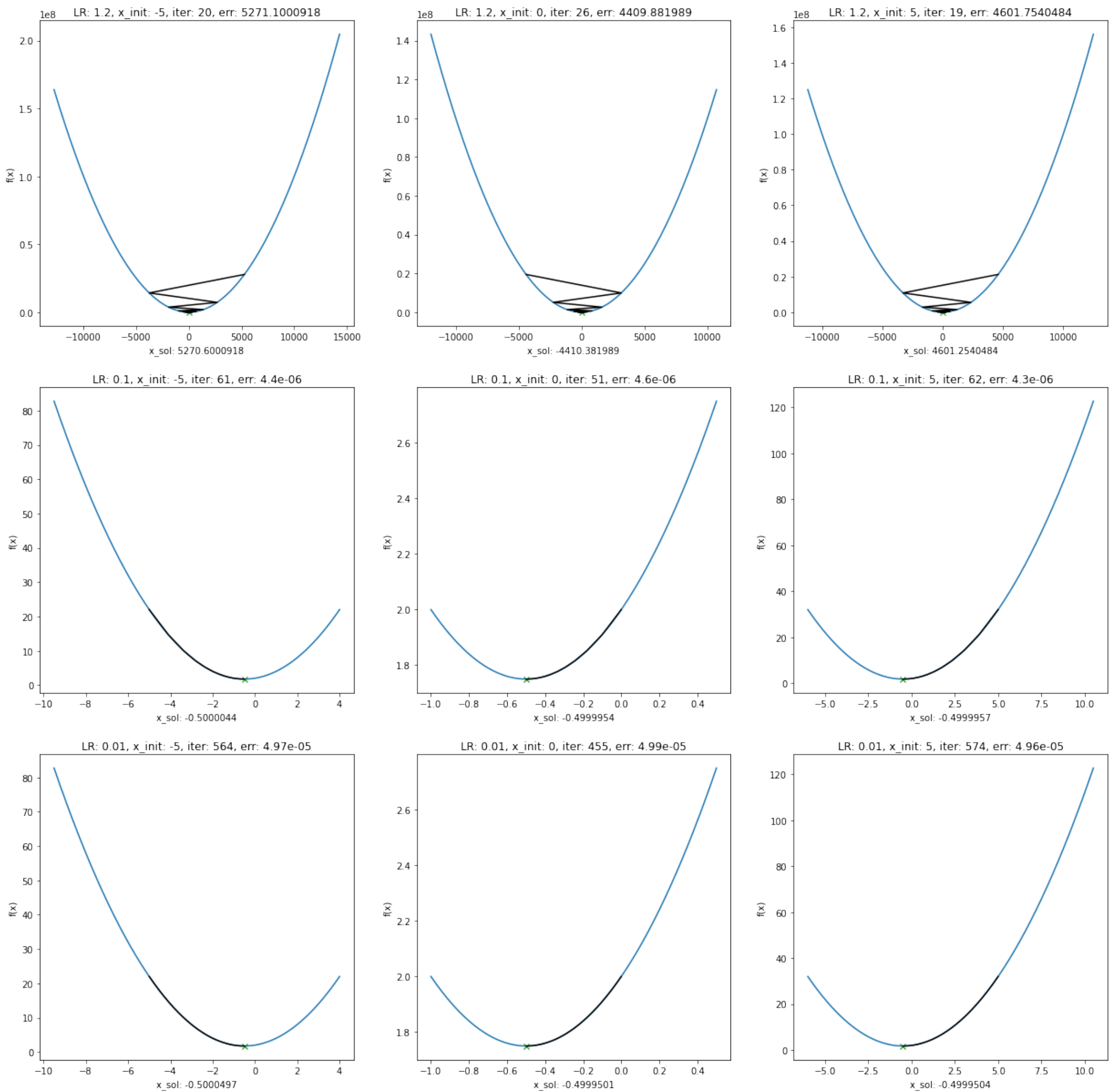
Follow the below steps and write your code in the block below

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

```
x = np.linspace(start=-10, stop=10, num=1000)

_lambda = [1.2, 0.1, 0.01]
x_init = [-5, 0, 5]
max_iter = int(1e5)
epsilon = 1e-6
max_val = int(1e4)
func = lambda X: np.power(X, 2) + X + 2
grad = lambda x: 2*x+1
sols = [-0.5] # Analytical Solution

plotter(x, func, sols)

solver(_lambda, x_init, max_iter, epsilon, max_val, sols, grad, func)
```

```
-----------------------------------------------------------------------
------------------------------
The value at which f(x) is minimum is: x = -0.49999568449255694
-----------------------------------------------------------------------
------------------------------
```


Plot of x vs f(x), Min at x=-0.5
f(x)
x
0
20
40
60
80
100
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0



Example 2 : $f(x) = x \sin x$

Analytical : Find solution analytically

$$\frac{d}{dx} f(x) = \sin x + x \cos x = 0$$

$$\frac{d^2}{dx^2} f(x) = 2\cos x - x \sin x \, (Minima/Maxima)$$

$$x = -\tan x$$

$$x \approx 4.193 \, (analytical solution)$$

Gradient Descent Update equation : Write Gradient descent update equations

$$x_{init} = 3$$

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

$$x_{updt} = x_{old} - \lambda (\sin x_{old} + x_{old} \cos x_{old} = 0)$$

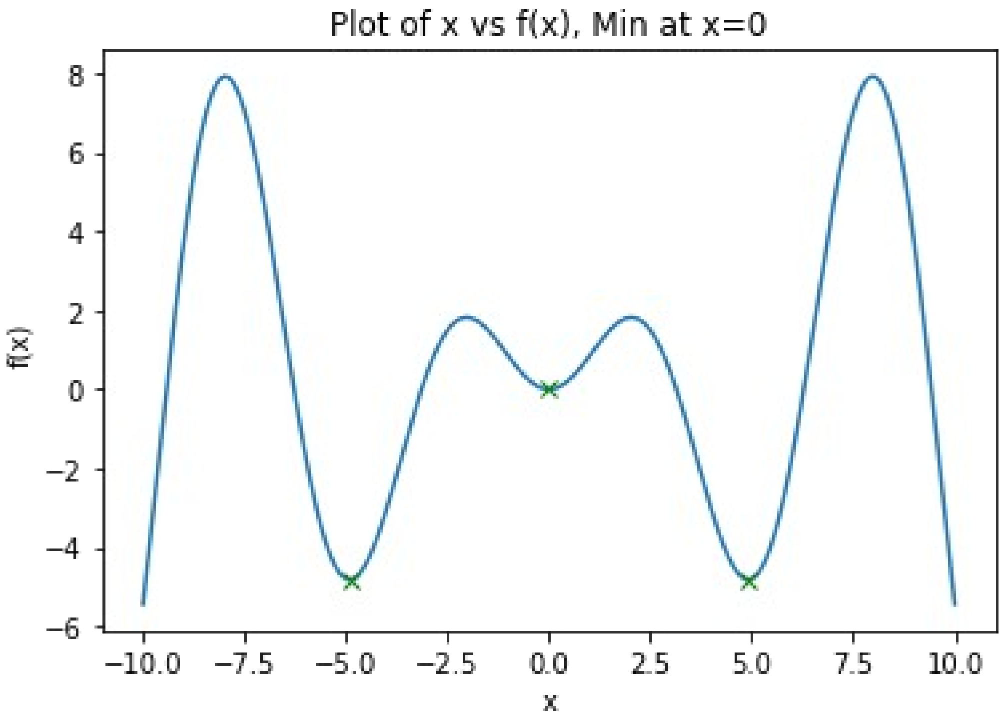
Gradient Descent Method :

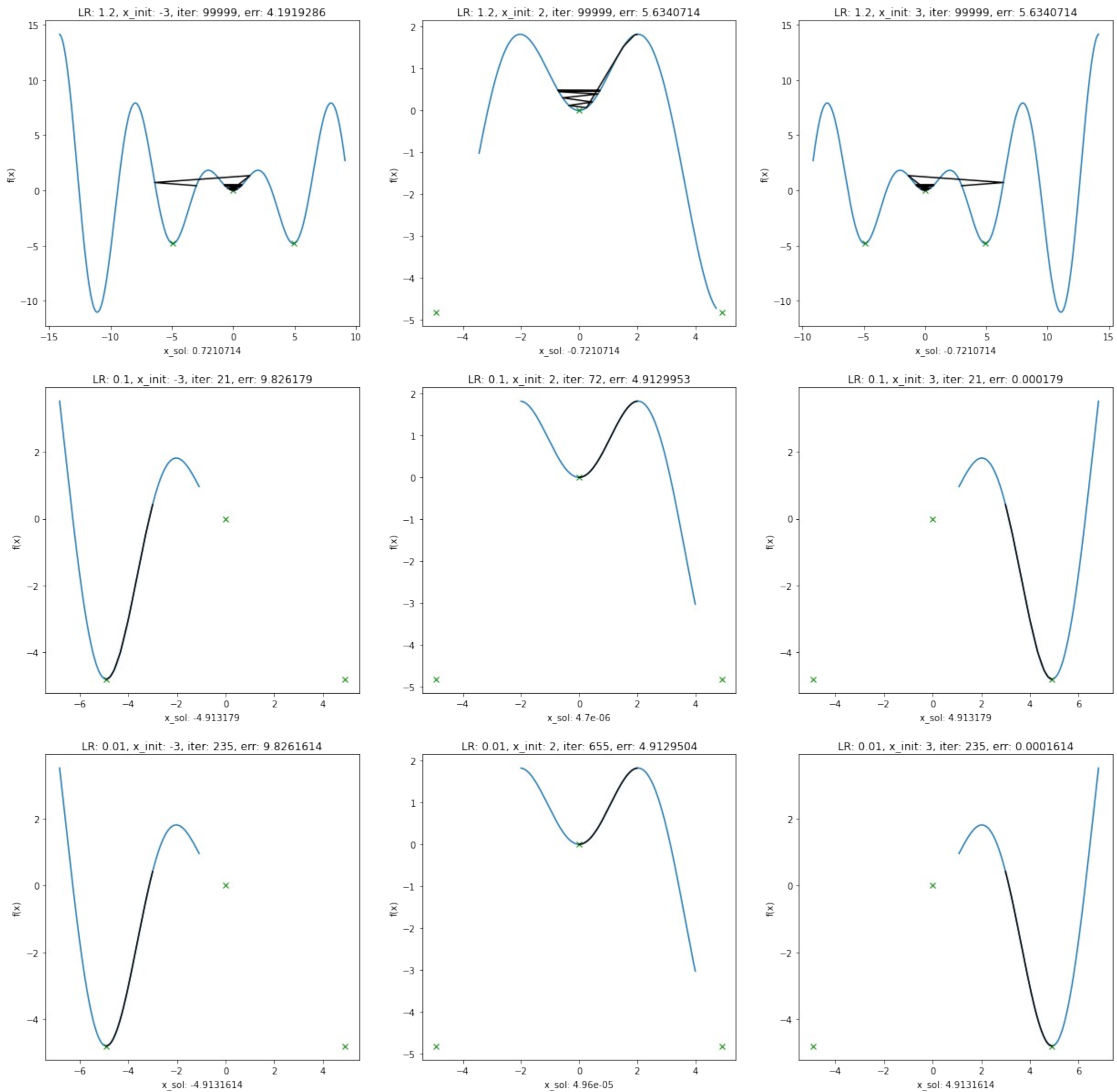
Follow the below steps and write your code in the block below

1. Generate x, 1000 data points from -10 to 10
2. Generate and Plot the function $f(x) = x \sin x$
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

```
x = np.linspace(start=-10, stop=10, num=1000)

_lambda = [1.2, 0.1, 0.01]
x_init = [-3, 2, 3]
max_iter = int(1e5)
epsilon = 1e-6
max_val = int(1e4)
func = lambda X: X*np.sin(X)
grad = lambda x: np.sin(x) + x*np.cos(x)
sols = [4.913, -4.913, 0] # Analytical Solutions

plotter(x, func, sols)

solver(_lambda, x_init, max_iter, epsilon, max_val, sols, grad, func)
```

```
----------------------------------------------------------------------
------------------------------
The value at which f(x) is minimum is: x = 4.913161356155126
----------------------------------------------------------------------
------------------------------
```


Plot of x vs f(x), Min at x=0
f(x)
x
8
6
4
2
0
−2
−4
−6
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0



#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 :

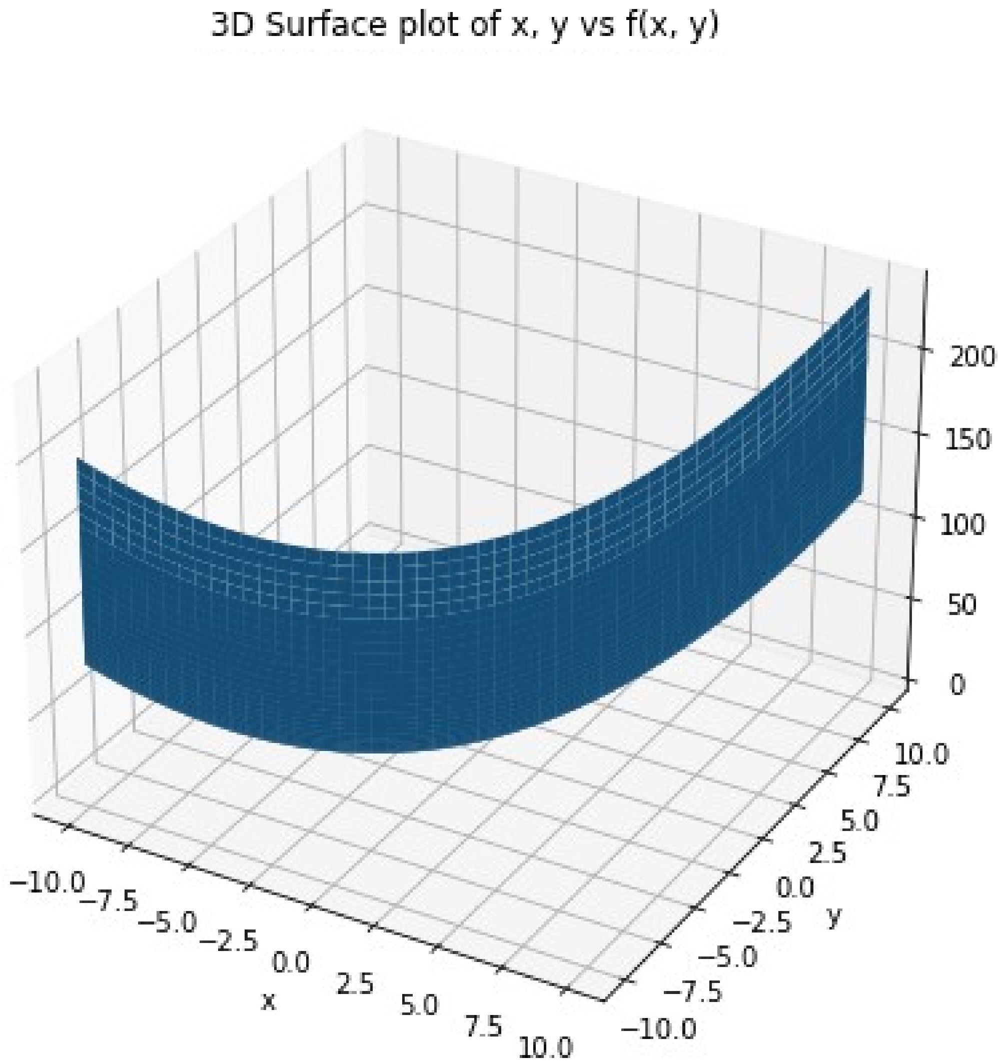
Follow the below steps and write your code in the block below

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

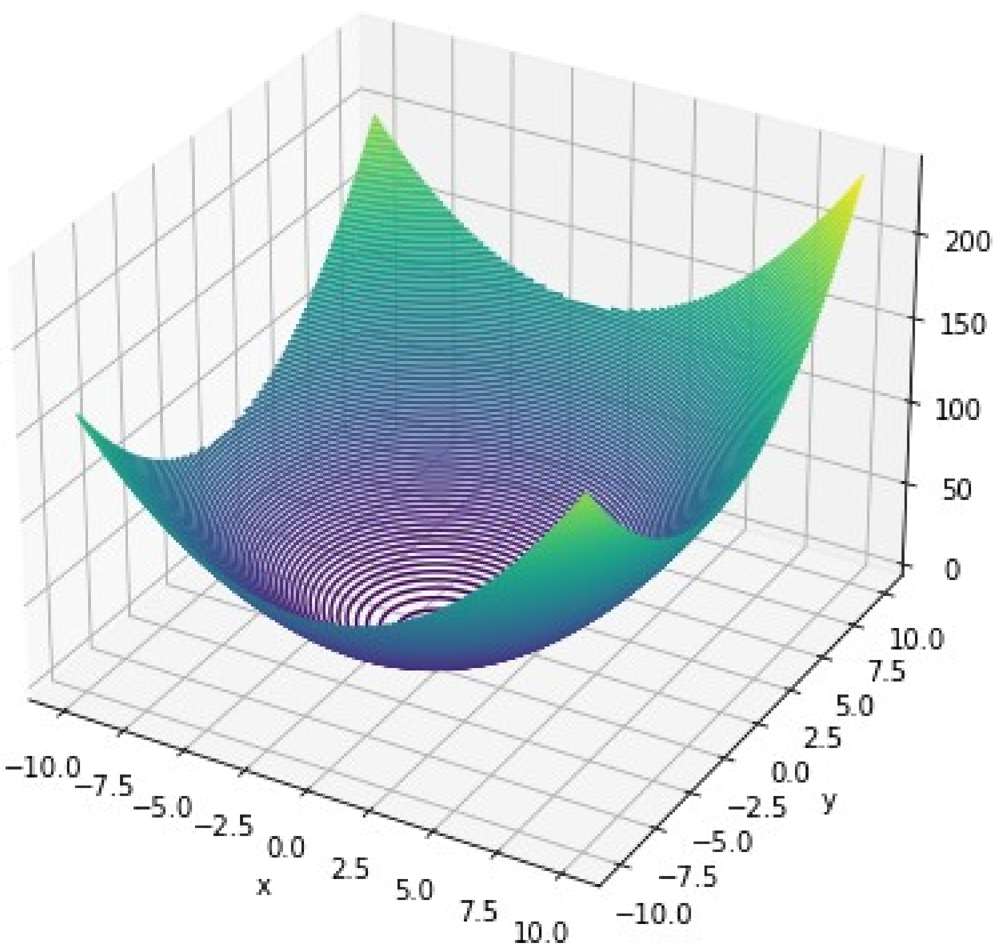
```
## Write your code here (Ignore the warning)
x = np.linspace(start=-10, stop=10, num=1000)
y = np.linspace(start=-10, stop=10, num=1000)
```

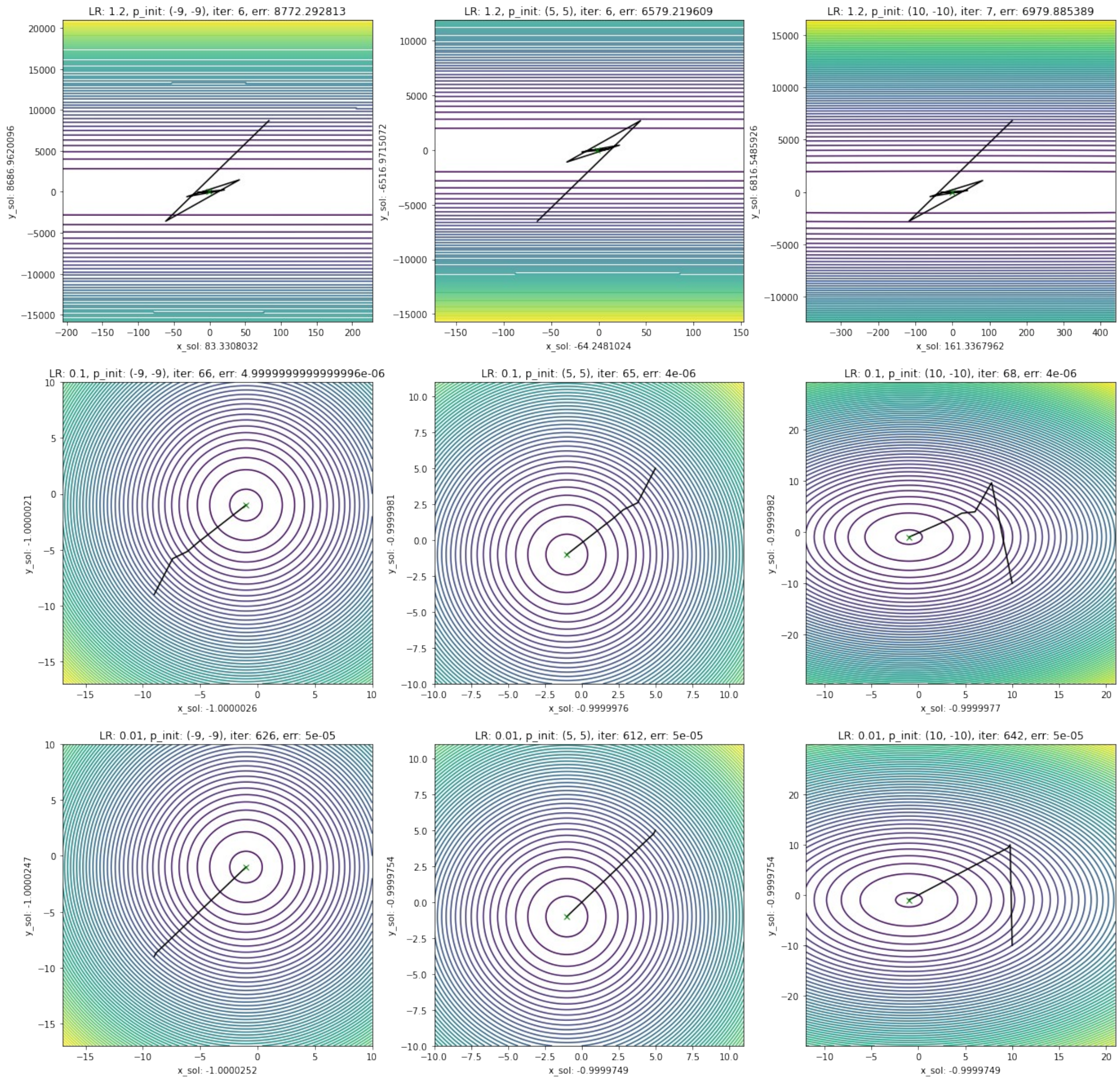
```python
_lambda = [1.2, 0.1, 0.01]
p_init = [(-9, -9), (5, 5), (10, -10)]
max_iter = int(1e5)
epsilon = 1e-6
max_val = int(1e4)
func = lambda X, Y: np.power(X, 2) + np.power(Y, 2) + 2*X + 2*Y
grad_x = lambda x, y: 2*x + 2
grad_y = lambda x, y: 2*y + 2
sols = [(-1, -1)] # Analytical Solution

plotter_3D(x, y, func)

solver_2D(_lambda, p_init, max_iter, epsilon, max_val, sols, grad_x,
grad_y, func)
```

```
----------------------------------------------------------------------
------------------------------
The value at which f(x, y) is minimum is: x = -0.9999975895929336, y =
-0.9999980716743468
----------------------------------------------------------------------
------------------------------
```


3D Surface plot of x, y vs f(x, y)
x
y
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
0
50
100
150
200

3D Contour plot of x, y vs f(x, y)





Example 2 : $f(x, y) = x\sin(x) + y\sin(y)$

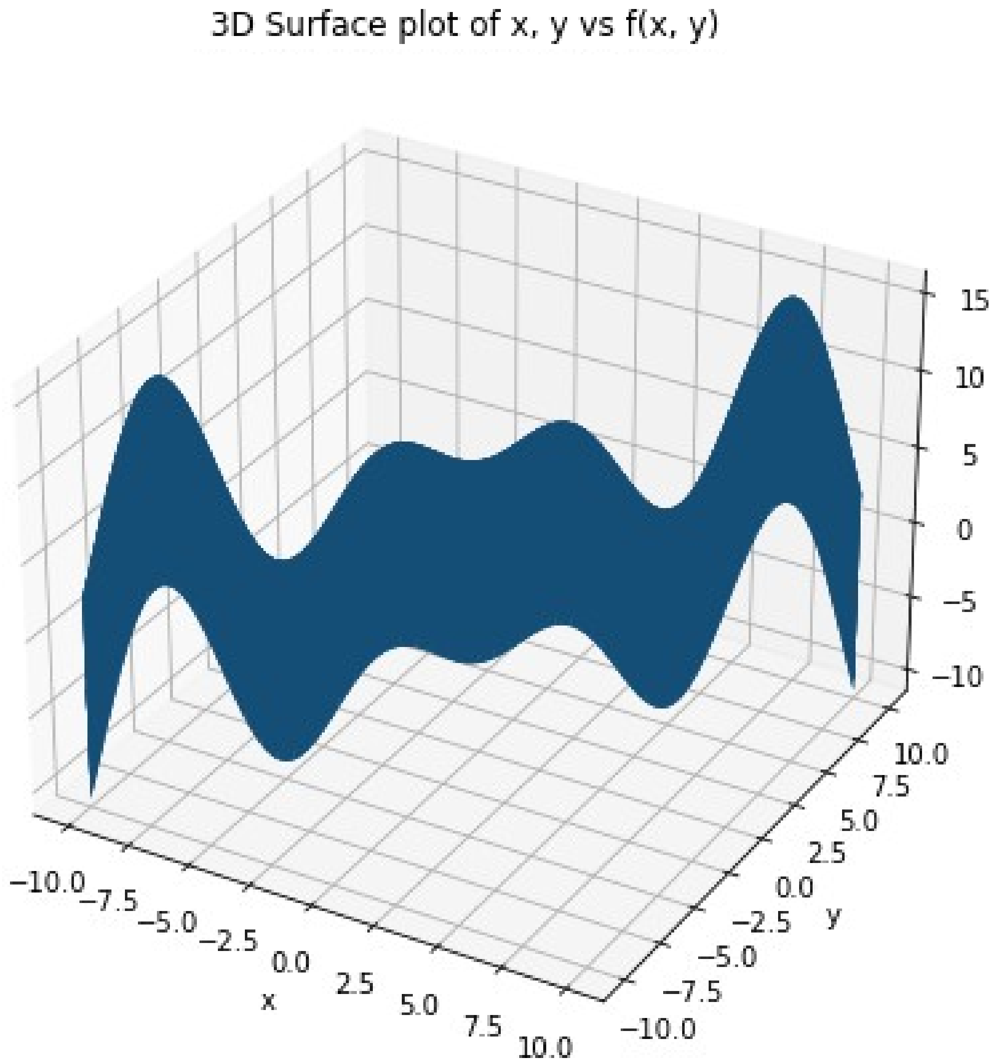
Gradient Descent Method :

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

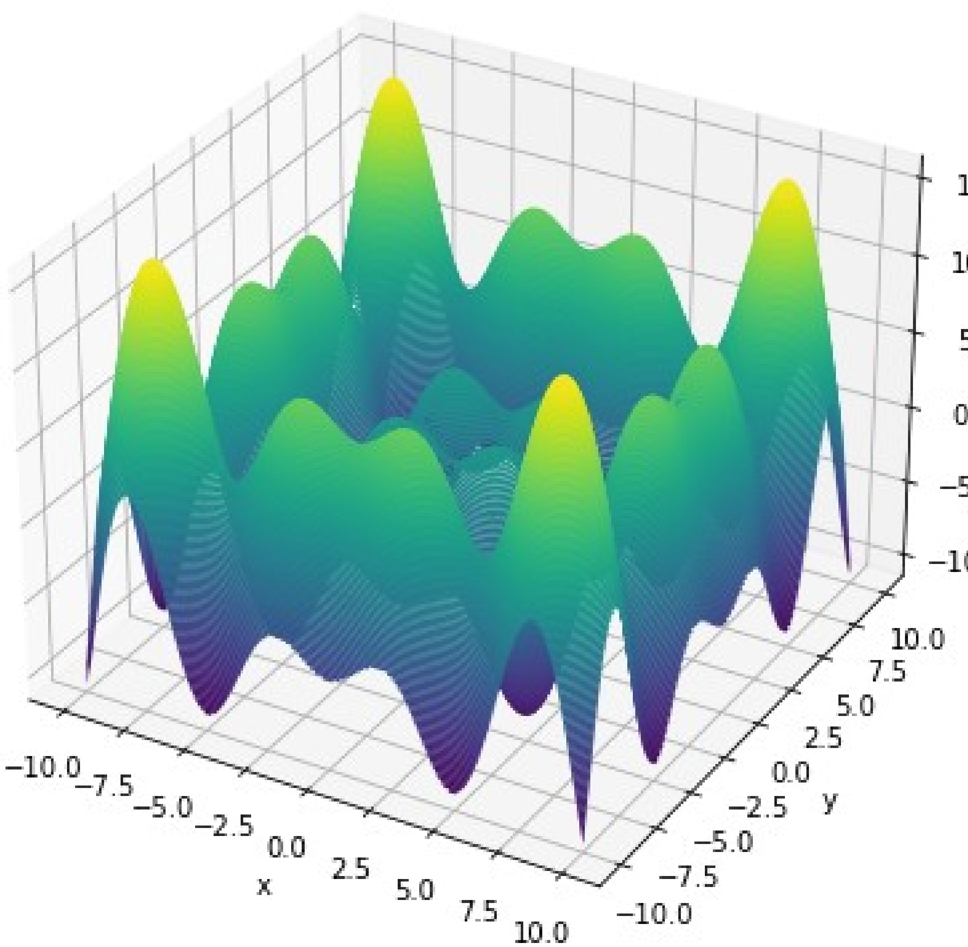
1. Generate x and y, 1000 data points from -10 to 10
2. Generate and Plot the function $f(x, y) = x\sin(x) + y\sin(y)$
3. Initialize the starting point (x_{init},y_{init}) and learning rate (λ)
4. Use Gradient descent algorithm to compute value of x and y at which the function $f(x, y)$ is minimum
5.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

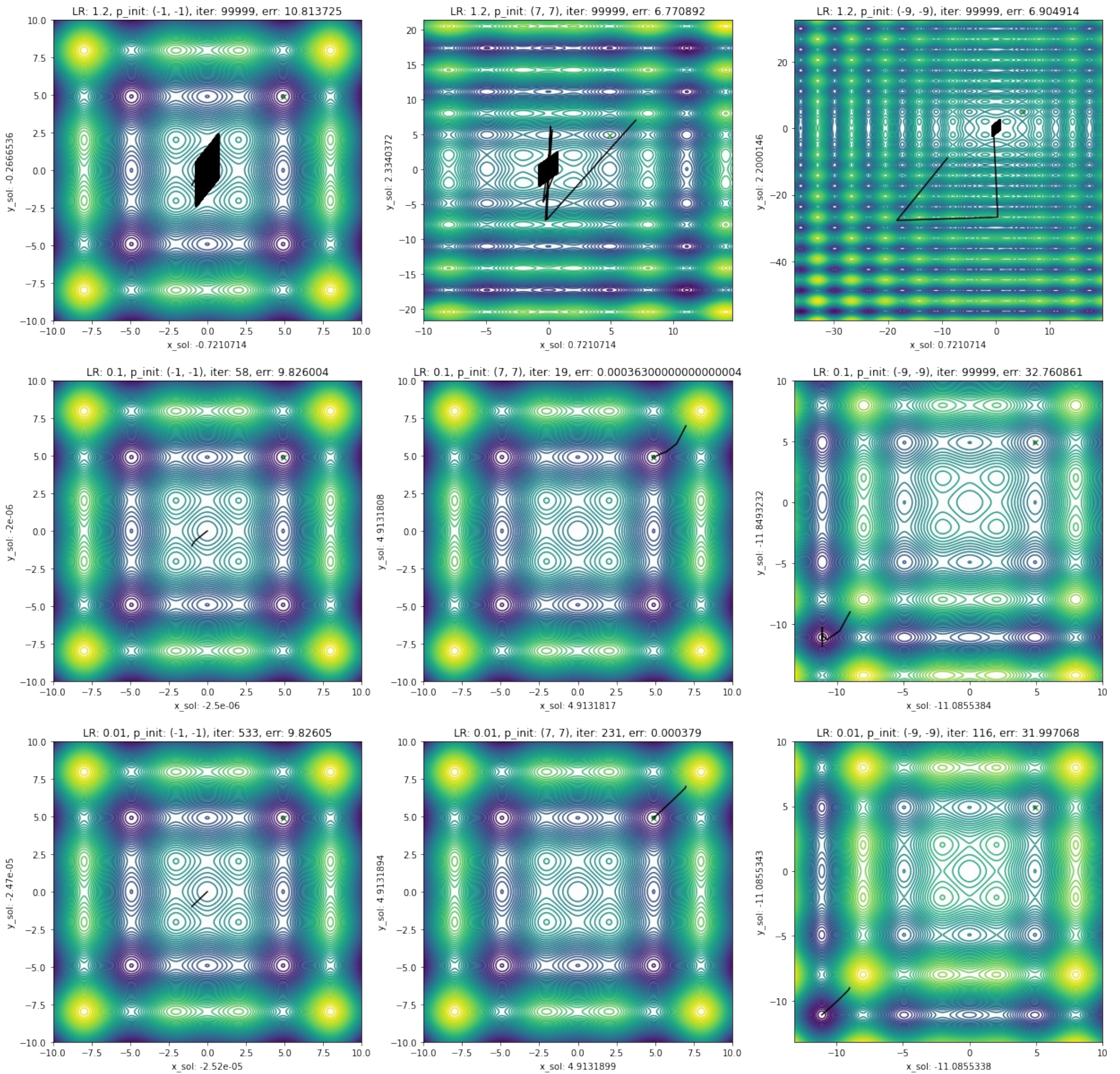
```python
## Write your code here (Ignore the warning)
x = np.linspace(start=-10, stop=10, num=1000)
y = np.linspace(start=-10, stop=10, num=1000)

_lambda = [1.2, 0.1, 0.01]
```

```
p_init = [(-1, -1), (7, 7), (-9, -9)]
max_iter = int(1e5)
epsilon = 1e-6
max_val = int(1e4)
func = lambda X, Y: X*np.sin(X) + Y*np.sin(Y)
grad_x = lambda x, y: np.sin(x) + x*np.cos(x)
grad_y = lambda x, y: np.sin(y) + y*np.cos(y)
sols = [(4.913, 4.913)] # Analytical Solution

plotter_3D(x, y, func)

solver_2D(_lambda, p_init, max_iter, epsilon, max_val, sols, grad_x,
grad_y, func)

----------------------------------------------------------------------
------------------------------
The value at which f(x, y) is minimum is: x = 4.913181730058289, y =
4.913180822534731
----------------------------------------------------------------------
------------------------------
```


3D Surface plot of x, y vs f(x, y)
15
10
5
0
−5
−10
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
y
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
x

3D Contour plot of x, y vs f(x, y)




LR: 1.2, p_init: (-1, -1), iter: 99999, err: 10.813725
y_sol: -0.2666536
x_sol: -0.7210714
LR: 1.2, p_init: (7, 7), iter: 99999, err: 6.770892
y_sol: 2.3340372
x_sol: 0.7210714
LR: 1.2, p_init: (-9, -9), iter: 99999, err: 6.904914
y_sol: 2.2000146
x_sol: 0.7210714
LR: 0.1, p_init: (-1, -1), iter: 58, err: 9.826004
y_sol: -2e-06
x_sol: -2.5e-06
LR: 0.1, p_init: (7, 7), iter: 19, err: 0.00036300000000000004
y_sol: 4.9131808
x_sol: 4.9131817
LR: 0.1, p_init: (-9, -9), iter: 99999, err: 32.760861
y_sol: -11.8493232
x_sol: -11.0855384
LR: 0.01, p_init: (-1, -1), iter: 533, err: 9.82605
y_sol: -2.47e-05
x_sol: -2.52e-05
LR: 0.01, p_init: (7, 7), iter: 231, err: 0.000379
y_sol: 4.9131894
x_sol: 4.9131899
LR: 0.01, p_init: (-9, -9), iter: 116, err: 31.997068
y_sol: -11.0855343
x_sol: -11.0855338