200010021_lab3

August 27, 2022

1 Lab 3 : Convex Optimisation

1.0.1 Convex function:

1. A function $f(x)$ is called convex if the line segment between any two points on the graph of the function lies above the graph between the two points.

1.0.2 Gradient Descent Method:

1. Gradient Descent is an iterative algorithm to find local minima of differentiable functions.
2. The method repeatedly steps in the direction of steepest descent.
3. The direction of steepest descent is opposite direction of gradient.

Drawbacks:

1. Gets stuck on saddle points.
2. Gets stuck on local minima.

1.0.3 Convex Optimisation:

1. Convex optimisation is the process of minimising convex functions.
2. Here we use gradient descent to do convex optimisation

2 Question 1) Single variable gradient descent

2.0.1 A) $f(x) = x^2 + x + 2$

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

2.0.2 B) $f(x) = xSin(x)$

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

Gradient Descent Method:

1. Generate x, 1000 points from -10 to 10
2. Generate and plot $f(x)$
3. Initialize starting point (x_{init}) and learning rate (λ)
4. Use gradient descent algorithm to compute value of x where, $f(x)$ is minimum
5. Vary learning rate and initial point and plot observations.

2.0.3 Part A)

1) Analytical Solution

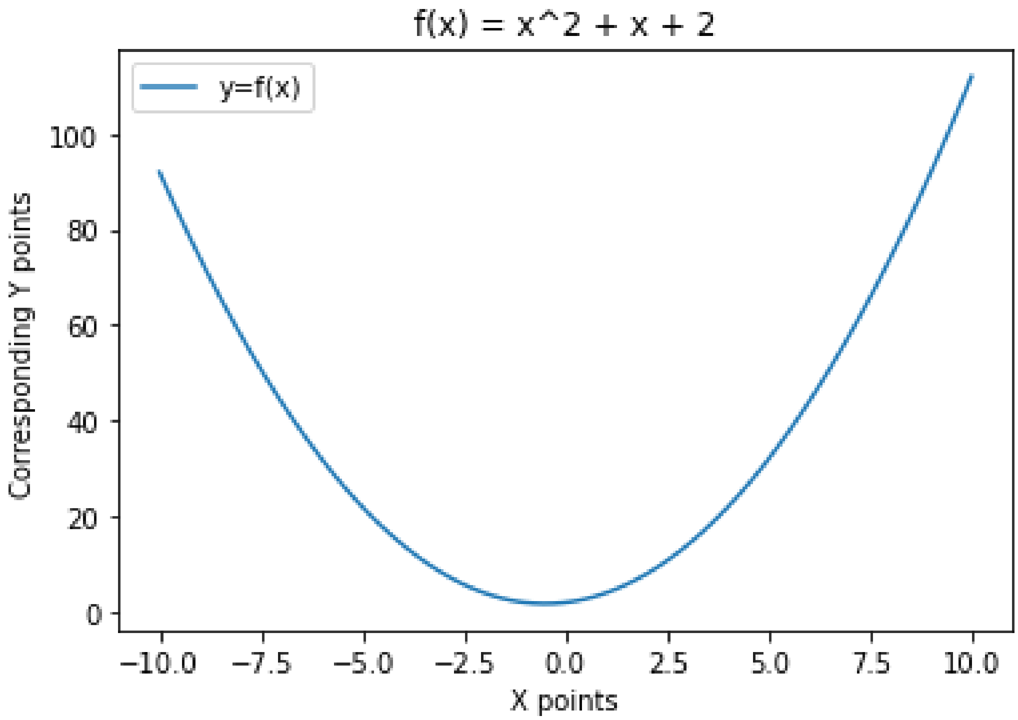
1. Given $f(x) = x^2 + x + 2$
2. We find minima of $f(x)$ at critical points (where $f'(x) = 0 $ and $f''(x) > 0$)
3. $f'(x) =$ $ 2x + 1 $ found after differentiating $f(x)$ similarly $f''(x) = 2 > 0$
4. So, minima will be found at $f'(x) = 0 => x = -1/2$
5. So the anylytical solution is $x = - 0.5 $

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

2) Now we generate and plot f(x)

1. First we generate 1000 X_points from -10 to 10 and corresponding Y_points
2. Then we plot Y_points vs X_points

[2]:
```
def plot_fx(coefficients):
    X_points = np.linspace(-10,10,10000)
    Y_points = coefficients[0]*(X_points**2) + coefficients[1] * X_points +␣
 ↪coefficients[2]

    plt.plot(X_points, Y_points)
    plt.xlabel('X points')
    plt.ylabel('Corresponding Y points')
    plt.title('f(x) = x^2 + x + 2')
    plt.legend(['y=f(x)'])

coeff = [1, 1, 2]

plot_fx(coeff)
```



3) Initialization

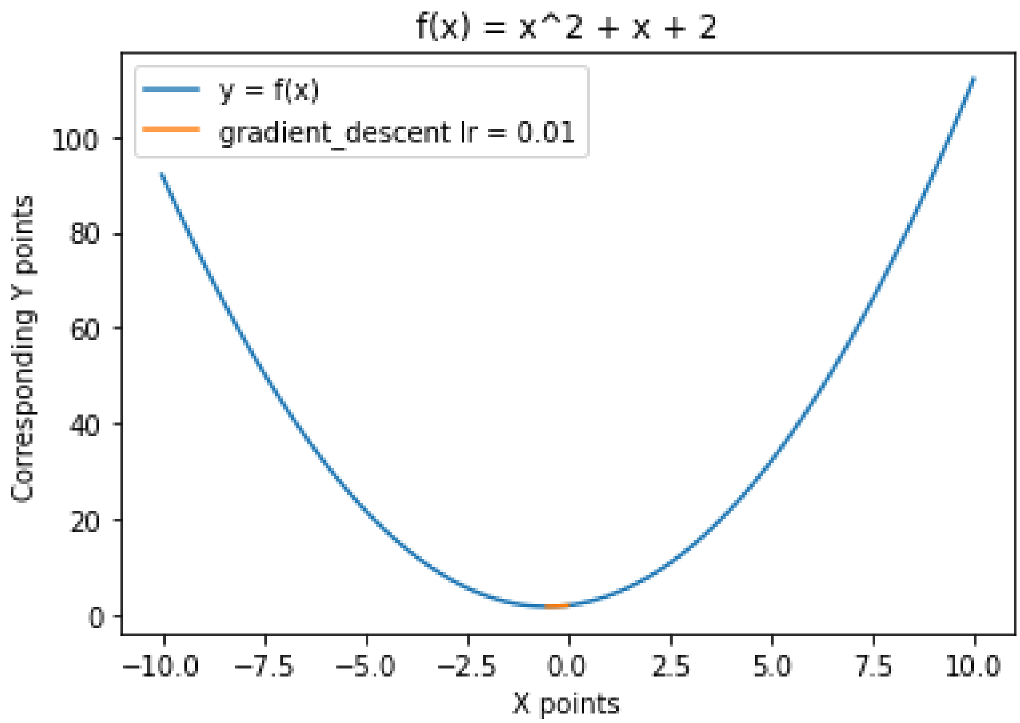
1. From plot we see that x_{min} is around 0 so $x_{init} = 0$
2. We take learning rate = 0.01
3. Now we write the algorithms

[3]:
```
x_init = 0

lr = 0.01

def derivative_fx(coeff,x):
  return 2*coeff[0]*x + coeff[1]

def gradient_update(coeff, x, learning_rate):
  x -= learning_rate * derivative_fx(coeff, x)
  return x

def gradient_plot(coeff, x_hist, y_hist, learning_rate):
    ### Plotting the gradient descent
  X_points = np.linspace(-10,10,10000)
  Y_points = coeff[0]*(X_points**2) + coeff[1] * X_points + coeff[2]

  plt.plot(X_points, Y_points)
```

```
  plt.xlabel('X points')
  plt.ylabel('Corresponding Y points')
  plt.title('f(x) = x^2 + x + 2')

  x_hist = np.array(x_hist)
  y_hist = coeff[0]*(x_hist**2) + coeff[1]*x_hist+coeff[2]

  plt.plot(x_hist,y_hist)

  plt.legend(['y = f(x)', f'gradient_descent lr = {learning_rate}'])



def gradient_descent(num_iter, x_guess, precision, learning_rate, show_plot =
 ↪True):
  x_now = x_guess
  x_prev = 9999999999

  x_hist = []
  y_hist = []

  iterations = []
  iterations.append(num_iter)
  t = False
  for i in range(num_iter):
    if abs(x_now - x_prev) < precision:
      t = i
      break

    x_hist.append(x_now)
    x_prev = x_now

    x_now = gradient_update(coeff, x_now, learning_rate)

  if t:
    iterations.append(t)
  if show_plot:
    gradient_plot(coeff, x_hist, y_hist, learning_rate)
    return x_now

  x_hist = np.array(x_hist)
  y_hist = coeff[0]*(x_hist**2) + coeff[1]*x_hist+coeff[2]

  return [x_hist, y_hist, iterations[-1]]


x_min = gradient_descent(1000, x_init, 1e-10, lr)
```

```
print(f"X_min is {x_min}")
```

```
X_min is -0.49999999512048
```

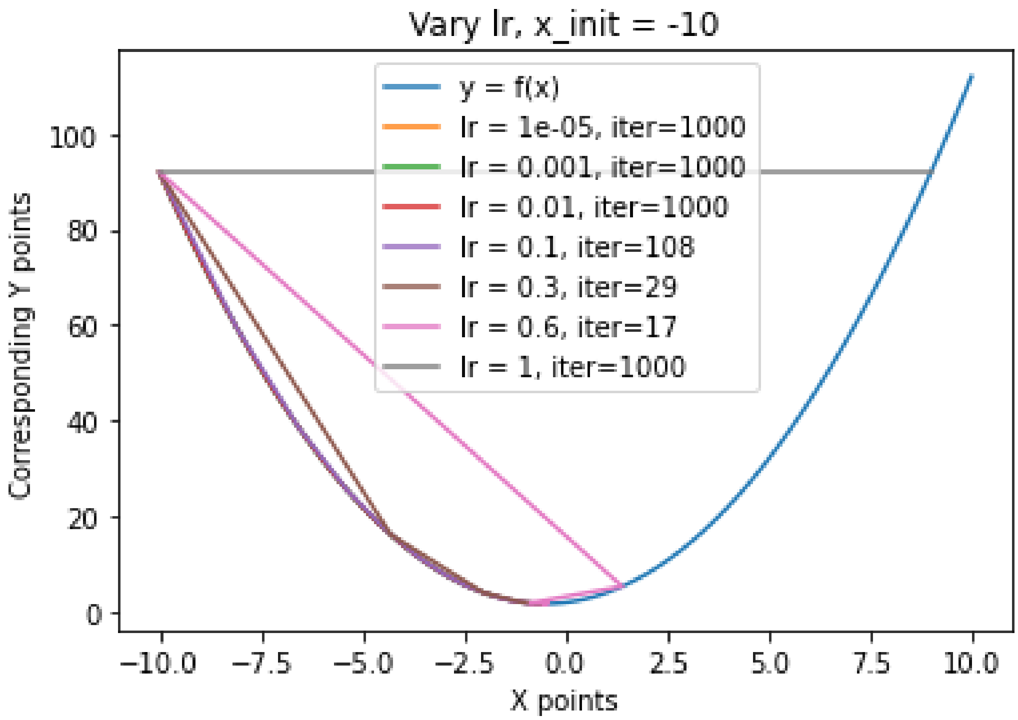


4) Now we vary learning rate and initial points and plot the graphs

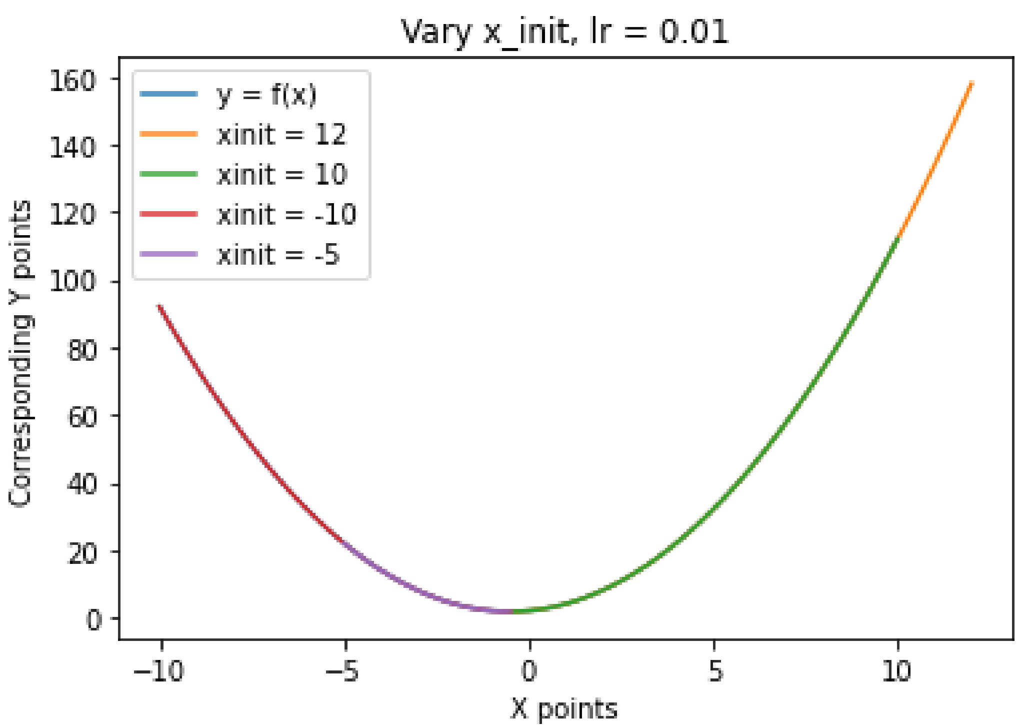
1. lr = [0.00001, 0.001, 0.01, 0.1, 0.3, 0.6, 1, 10]
2. x_init = [12, 10, 3, 5, -1, -10, -5, -8]

[4]:
```
lr = [0.00001, 0.001, 0.01, 0.1, 0.3, 0.6, 1]

x_init = [12, 10, -10, -5]


def Vary_Learning_rate(xinit=10):
    History = []
    legend = ['y = f(x)']

    for rate in lr:
        History.append(gradient_descent(1000, xinit, 1e-10, rate, False))

    X_points = np.linspace(-10,10,10000)
    Y_points = coeff[0]*(X_points**2) + coeff[1] * X_points + coeff[2]
```

```
    plt.figure(0)
    plt.plot(X_points, Y_points)
    plt.xlabel('X points')
    plt.ylabel('Corresponding Y points')
    plt.title(f'Vary lr, x_init = {xinit}')

    for i in range(len(lr)):
        x_hist  = History[i][0]
        y_hist = History[i][1]
        plt.plot(x_hist,y_hist)

        legend.append(f"lr = {lr[i]}, iter={History[i][2]}")

    plt.legend(legend)

Vary_Learning_rate(xinit=-10)



def Vary_xinit(lr=0.01):
    History = []
    legend = ['y = f(x)']

    for init in x_init:
        History.append(gradient_descent(1000, init, 1e-10, lr, False))

    X_points = np.linspace(-10,10,10000)
    Y_points = coeff[0]*(X_points**2) + coeff[1] * X_points + coeff[2]
    plt.figure(1)
    plt.plot(X_points, Y_points)
    plt.xlabel('X points')
    plt.ylabel('Corresponding Y points')
    plt.title(f'Vary x_init, lr = {lr}')

    for i in range(len(x_init)):
        x_hist  = History[i][0]
        y_hist = History[i][1]
        plt.plot(x_hist,y_hist)

        legend.append(f"xinit = {x_init[i]}")

    plt.legend(legend)

Vary_xinit()
```



Vary lr, x_init = -10
y = f(x)
lr = 1e-05, iter=1000
lr = 0.001, iter=1000
lr = 0.01, iter=1000
lr = 0.1, iter=108
lr = 0.3, iter=29
lr = 0.6, iter=17
lr = 1, iter=1000
Corresponding Y points
X points



Vary x_init, lr = 0.01
y = f(x)
xinit = 12
xinit = 10
xinit = -10
xinit = -5
Corresponding Y points
X points

2.0.4 Part B)

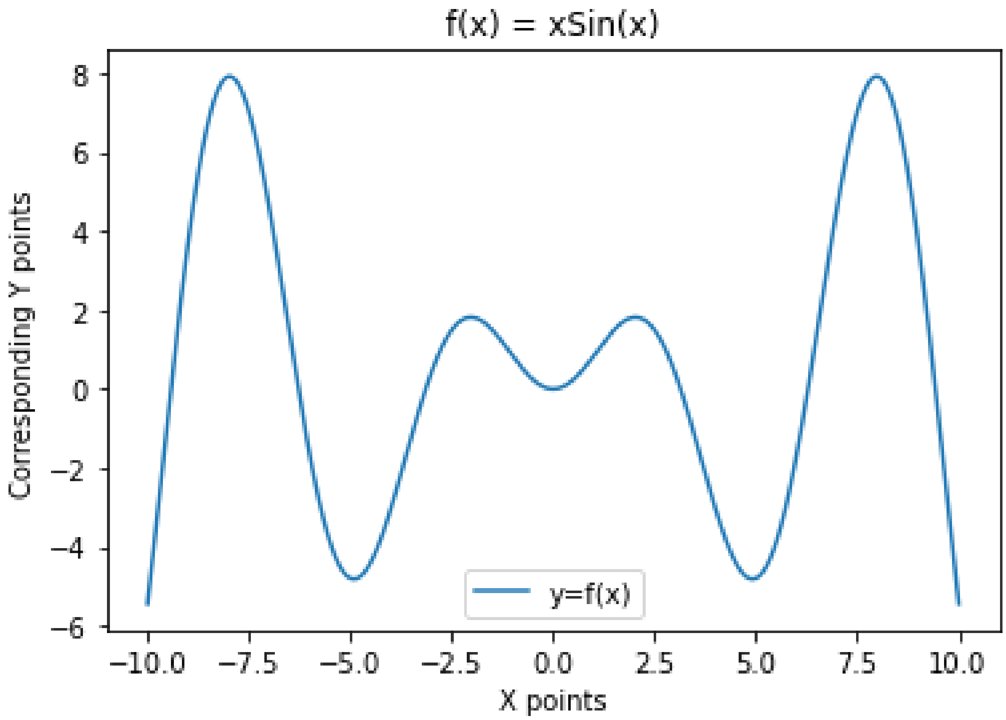
1) Analytical Solution

1. Given $f(x) = xSin(x)$
2. We find minima of $f(x)$ at critical points (where $f'(x) = 0 $ and $f''(x) > 0$)
3. $f'(x)$ = $ Sin(x) + x Cos(x) $ found after differentiating $f(x)$ similarly $f''(x) =Sin(x) + Cos(x) - xSin(x) $
4. So, minima will be found at $f'(x) = 0 => x = -Tan(x)$
5. So the anylytical solution is $x = 0$ etc…
6. There are infinite local minima.

2) Now we generate and plot f(x)

1. First we generate 1000 X_points from -10 to 10 and corresponding Y_points
2. Then we plot Y_points vs X_points

[5]:
```
def plot_fx():
    X_points = np.linspace(-10,10,10000)
    Y_points = X_points * np.sin(X_points)

    plt.plot(X_points, Y_points)
    plt.xlabel('X points')
    plt.ylabel('Corresponding Y points')
    plt.title('f(x) = xSin(x)')
    plt.legend(['y=f(x)'])


plot_fx()
```



3) Initialization

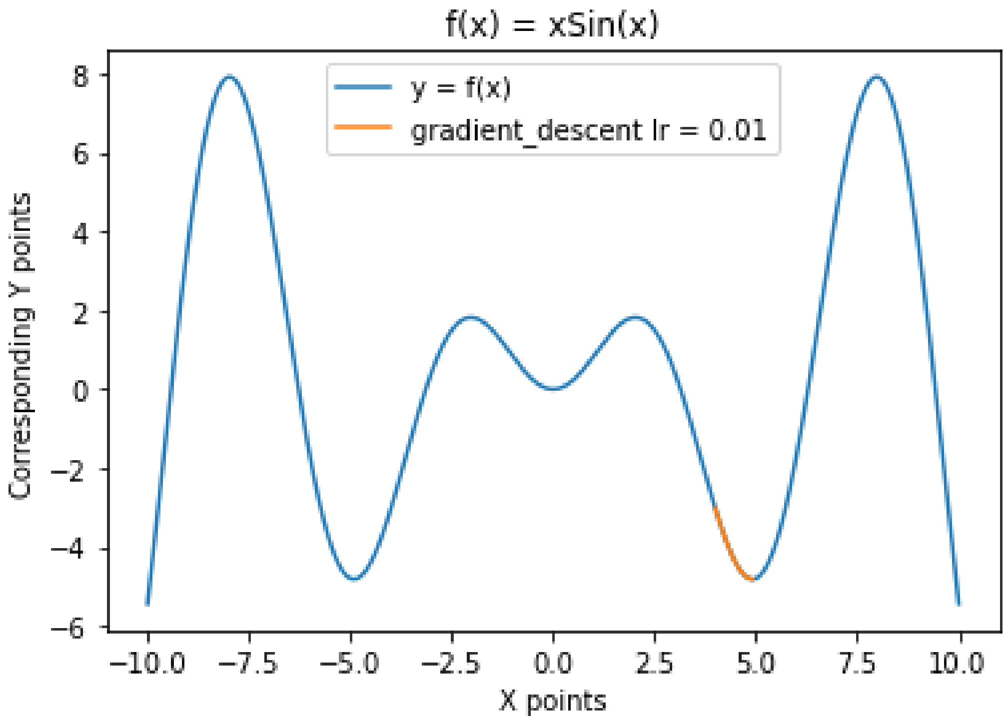
1. From plot we see that x_{min} is around 0 , +/- 2.5, 7.5 so $x_{init} = 1,4,7 $
2. We take learning rate = 0.01
3. Now we write the algorithms

[6]:
```
from math import cos, sin


x_init = 4

lr = 0.01

def derivative_fx(x):
  return x*cos(x) + sin(x)

def gradient_update(x, learning_rate):
  x -= learning_rate * derivative_fx(x)
  return x

def gradient_plot(x_hist, y_hist, learning_rate):
    ### Plotting the gradient descent
  X_points = np.linspace(-10,10,10000)
```

```
  Y_points = X_points * np.sin(X_points)

  plt.plot(X_points, Y_points)
  plt.xlabel('X points')
  plt.ylabel('Corresponding Y points')
  plt.title('f(x) = xSin(x)')

  x_hist = np.array(x_hist)
  y_hist = x_hist * np.sin(x_hist)

  plt.plot(x_hist,y_hist)

  plt.legend(['y = f(x)', f'gradient_descent lr = {learning_rate}'])



def gradient_descent(num_iter, x_guess, precision, learning_rate, show_plot =
 ↪True):
  x_now = x_guess
  x_prev = 9999999999

  x_hist = []
  y_hist = []

  iterations = []
  iterations.append(num_iter)
  t = False
  for i in range(num_iter):
    if abs(x_now - x_prev) < precision:
      t = i
      break

    x_hist.append(x_now)
    x_prev = x_now

    x_now = gradient_update(x_now, learning_rate)

  if t:
    iterations.append(t)
  if show_plot:
    gradient_plot(x_hist, y_hist, learning_rate)
    return x_now

  x_hist = np.array(x_hist)
  y_hist = x_hist * np.sin(x_hist)

  return [x_hist, y_hist, iterations[-1]]
```

```
x_min = gradient_descent(1000, x_init, 1e-10, lr)

print(f"X_min is {x_min}")
```

```
X_min is 4.913180437644452
```

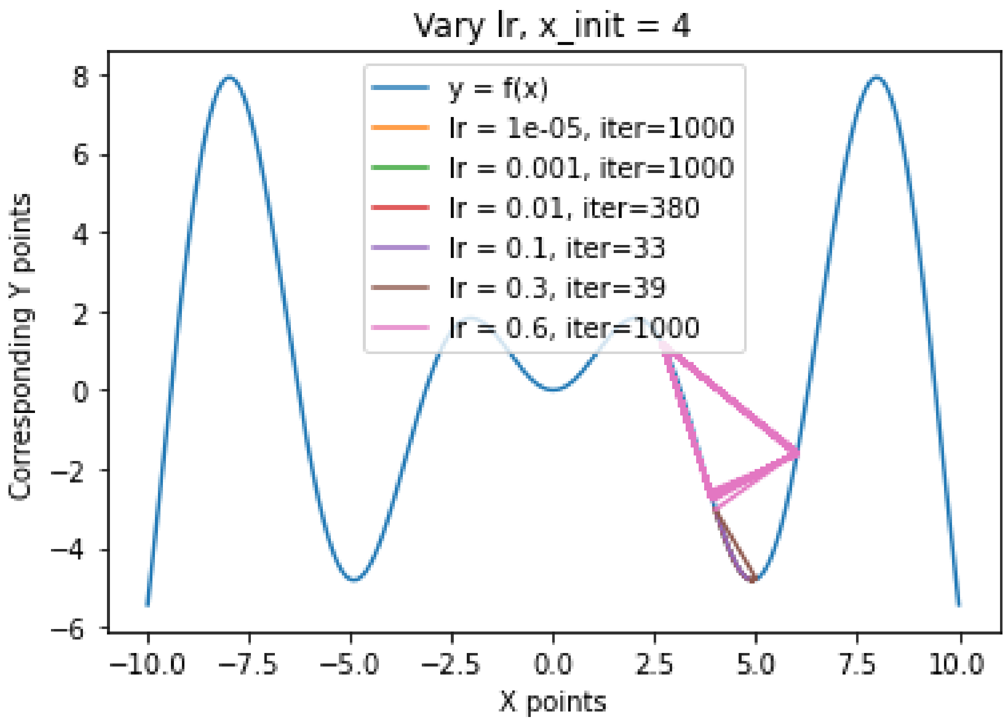


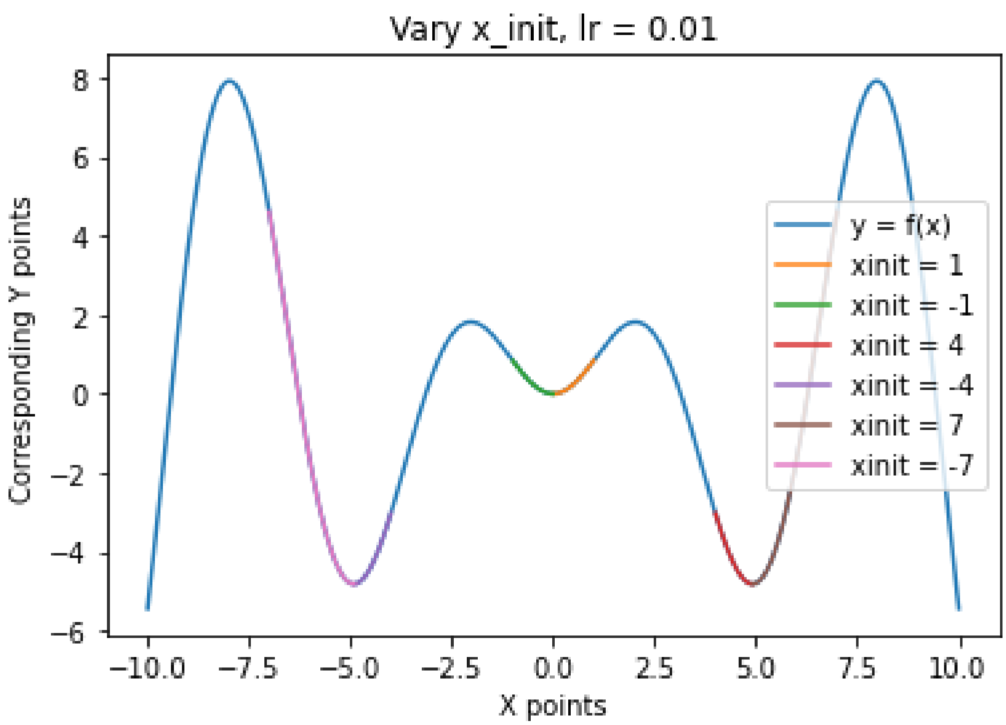
4) Now we vary learning rate and initial points and plot the graphs

1. lr = [0.00001, 0.001, 0.01, 0.1, 0.3, 0.6, 1, 10]
2. x_init = [12, 10, 3, 5, -1, -10, -5, -8]

[7]:
```
lr = [0.00001, 0.001, 0.01, 0.1, 0.3, 0.6]

x_init = [1, -1, 4, -4, 7, -7]


def Vary_Learning_rate(xinit=4):
    History = []
    legend = ['y = f(x)']

    for rate in lr:
        History.append(gradient_descent(1000, xinit, 1e-10, rate, False))
```

```
    X_points = np.linspace(-10,10,10000)
    Y_points = X_points * np.sin(X_points)
    plt.figure(0)
    plt.plot(X_points, Y_points)
    plt.xlabel('X points')
    plt.ylabel('Corresponding Y points')
    plt.title(f'Vary lr, x_init = {xinit}')

    for i in range(len(lr)):
        x_hist  = History[i][0]
        y_hist = History[i][1]
        plt.plot(x_hist,y_hist)

        legend.append(f"lr = {lr[i]}, iter={History[i][2]}")

    plt.legend(legend)

Vary_Learning_rate(xinit=4)



def Vary_xinit(lr=0.01):
    History = []
    legend = ['y = f(x)']

    for init in x_init:
        History.append(gradient_descent(1000, init, 1e-10, lr, False))

    X_points = np.linspace(-10,10,10000)
    Y_points = X_points * np.sin(X_points)
    plt.figure(1)
    plt.plot(X_points, Y_points)
    plt.xlabel('X points')
    plt.ylabel('Corresponding Y points')
    plt.title(f'Vary x_init, lr = {lr}')

    for i in range(len(x_init)):
        x_hist  = History[i][0]
        y_hist = History[i][1]
        plt.plot(x_hist,y_hist)

        legend.append(f"xinit = {x_init[i]}")

    plt.legend(legend)

Vary_xinit()
```


Vary lr, x_init = 4
y = f(x)
lr = 1e-05, iter=1000
lr = 0.001, iter=1000
lr = 0.01, iter=380
lr = 0.1, iter=33
lr = 0.3, iter=39
lr = 0.6, iter=1000
Corresponding Y points
X points


Vary x_init, lr = 0.01
y = f(x)
xinit = 1
xinit = -1
xinit = 4
xinit = -4
xinit = 7
xinit = -7
Corresponding Y points
X points

3 Question 2) Two variable gradient descent

3.0.1 A) $f(x, y) = x^2 + y^2 + 2x + 2y$

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

3.0.2 B) A) $f(x) = xSin(x) + ySin(y)$

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

Gradient Descent Method:

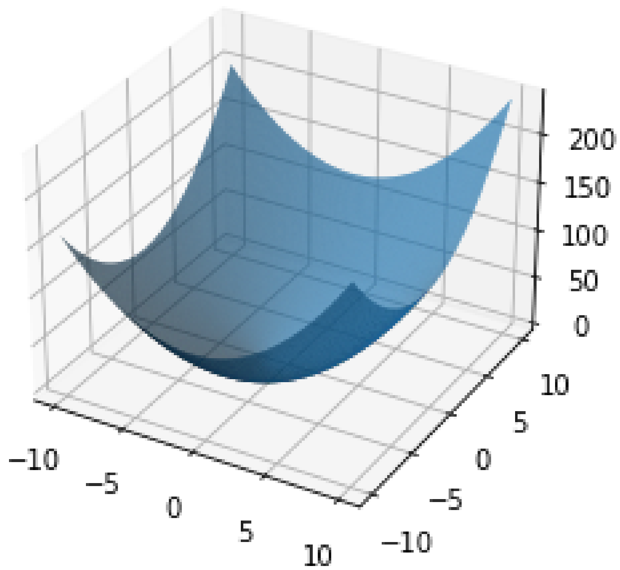
1. Generate x and y, 1000 points from -10 to 10
2. Generate and plot $f(x, y)$
3. Initialize starting point (x_{init}, y_{init}) and learning rate (λ)
4. Use gradient descent algorithm to compute value of x where, $f(x, y)$ is minimum
5. Vary learning rate and initial point and plot observations.

4 Part A)

1) Now we generate and plot f(x,y)

1. First we generate 1000 X_points, 1000 Y_points from -10 to 10 and corresponding Y_points
2. Then we plot Z_points vs X_points, Y_points

[8]:
```
from mpl_toolkits import mplot3d

def plot_fxy():
  X_points = np.linspace(-10,10,200)
  Y_points = np.linspace(-10,10,200)

  X_points, Y_points = np.meshgrid(X_points, Y_points)

  Z_points = X_points**2 + Y_points**2 + 2*X_points + 2*Y_points
  fig = plt.figure()
  ax = plt.axes(projection='3d')
  ax.plot_trisurf(X_points.flatten(), Y_points.flatten(), Z_points.flatten())


plot_fxy()
```



2) Initialization

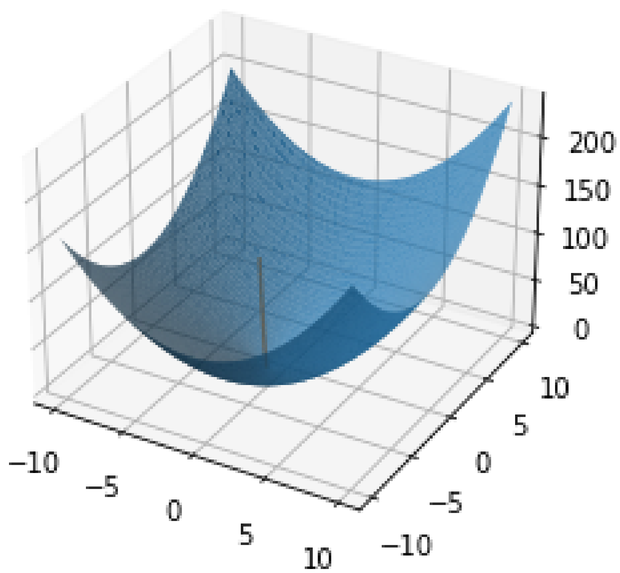
1. From plot we see that x_{min} is around 0,0 so $x_{init}, y_{init} = -5,5 $
2. We take learning rate = 0.01
3. Now we write the algorithms

[9]:
```
from math import cos, sin



x_init = -5
y_init = 5

lr = 0.01

def derivative_fx(x):
    return 2*x + 2

def derivative_fy(y):
    return 2*y + 2

def gradient_update(x, y, learning_rate):
  x -= learning_rate * derivative_fx(x)
  y -= learning_rate * derivative_fy(y)
  return [x,y]
```

```
def gradient_plot(x_hist, y_hist, learning_rate):
    ### Plotting the gradient descent
  X_points = np.linspace(-10,10,100)
  Y_points = np.linspace(-10,10,100)

  X_points, Y_points = np.meshgrid(X_points, Y_points)

  Z_points = X_points**2 + Y_points**2 + 2*X_points + 2*Y_points
  ax = plt.axes(projection='3d')
  ax.plot_trisurf(X_points.flatten(), Y_points.flatten(), Z_points.flatten())

  x_hist = np.array(x_hist)
  y_hist = np.array(y_hist)
  z_hist = x_hist**2 + y_hist**2 + 2*x_hist + 2*y_hist

  ax.plot3D(x_hist,y_hist,z_hist)





def gradient_descent(num_iter, x_guess, y_guess, precision, learning_rate,␣
 ↪show_plot = True):
  x_now = x_guess
  y_now = y_guess
  x_prev = 9999999999
  y_prev = 9999999999

  x_hist = []
  y_hist = []
  z_hist = []

  iterations = []
  iterations.append(num_iter)
  t = False
  for i in range(num_iter):
    if abs(x_now - x_prev) < precision and abs(y_now - y_prev) < precision:
      t = i
      break

    x_hist.append(x_now)
    y_hist.append(y_now)
    x_prev = x_now
    y_prev = y_now
    tmp = gradient_update(x_now,y_now, learning_rate)
    x_now = tmp[0]
    y_now = tmp[1]
```

```
    if t:
      iterations.append(t)
    if show_plot:
      gradient_plot(x_hist, y_hist, learning_rate)
      return [x_now, y_now]

    x_hist = np.array(x_hist)
    y_hist = np.array(y_hist)
    z_hist = x_hist**2 + y_hist**2 + 2*x_hist + 2*y_hist
    return [x_hist, y_hist, z_hist]


x_min = gradient_descent(1000, x_init, y_init, 1e-10, lr)

print(f"X_min is {x_min}")
```

```
X_min is [-1.000000006731869, -0.9999999899021961]
```

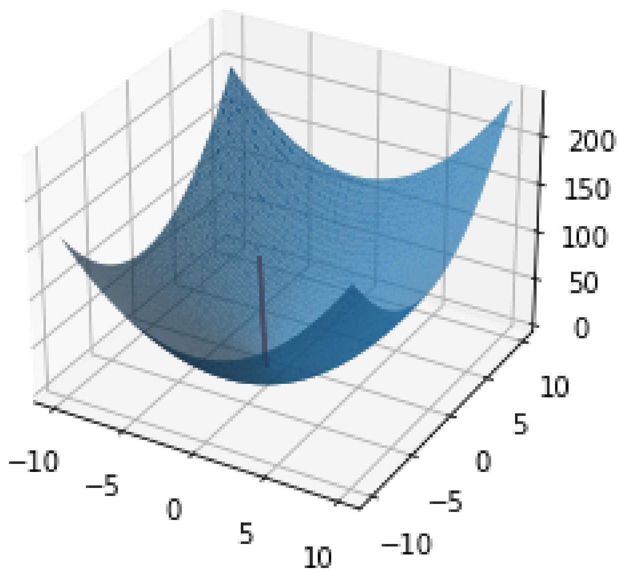


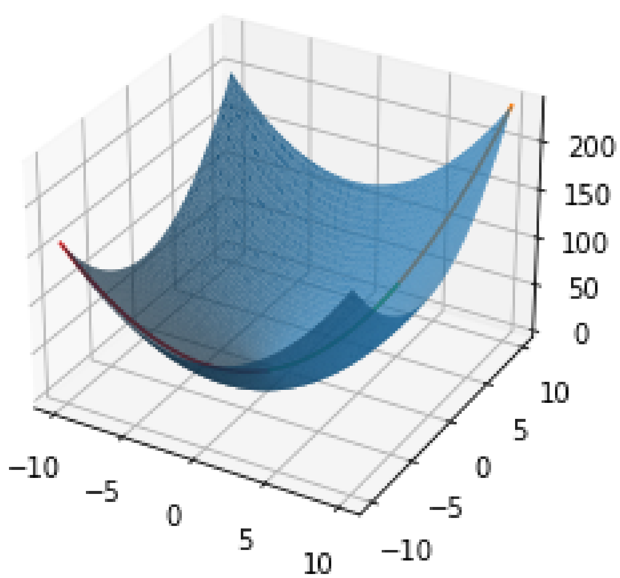
3) Now we vary learning rate and initial points and plot the graphs

1. lr = [0.01, 0.1, 0.3]
2. x_init = [10, 5, -10]

[10]:
```
lr = [0.01, 0.1, 0.3]
X_init = [ 10,  5,  -10]
```

```
def Varylr():
    History = []

    for rate in lr:
        History.append(gradient_descent(1000, x_init, y_init, 1e-10, rate,␣
↪False))

    X_points = np.linspace(-10,10,100)
    Y_points = np.linspace(-10,10,100)

    X_points, Y_points = np.meshgrid(X_points, Y_points)

    Z_points = X_points**2 + Y_points**2 + 2*X_points + 2*Y_points
    ax = plt.axes(projection='3d')
    ax.plot_trisurf(X_points.flatten(), Y_points.flatten(), Z_points.flatten())

    for i in range(len(lr)):
        x_hist  = History[i][0]
        y_hist = History[i][1]
        z_hist = History[i][2]
        ax.plot3D(x_hist,y_hist,z_hist)

Varylr()

def Vary_xinit():
    History = []

    for init in X_init:
        History.append(gradient_descent(1000, init, init, 1e-10, 0.01, False))

    X_points = np.linspace(-10,10,100)
    Y_points = np.linspace(-10,10,100)
    X_points, Y_points = np.meshgrid(X_points, Y_points)

    Z_points = X_points**2 + Y_points**2 + 2*X_points + 2*Y_points
    plt.figure()
    ax = plt.axes(projection='3d')
    ax.plot_trisurf(X_points.flatten(), Y_points.flatten(), Z_points.flatten())

    for i in range(len(lr)):
        x_hist  = History[i][0]
        y_hist = History[i][1]
        z_hist = History[i][2]
        ax.plot3D(x_hist,y_hist,z_hist)

Vary_xinit()
```



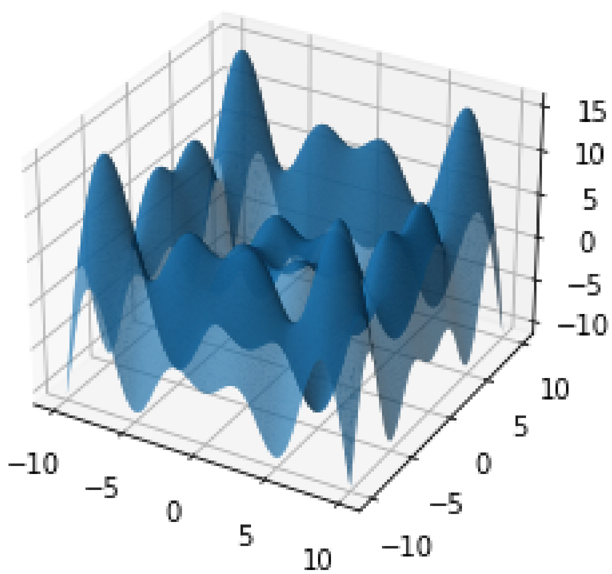


5 Part B)

1) Now we generate and plot f(x,y)

1. First we generate 1000 X__points, 1000 Y__points from -10 to 10 and corresponding Y__points
2. Then we plot Z__points vs X__points, Y__points

[11]:
```
from mpl_toolkits import mplot3d

def plot_fxy():
  X_points = np.linspace(-10,10,200)
  Y_points = np.linspace(-10,10,200)

  X_points, Y_points = np.meshgrid(X_points, Y_points)

  Z_points = X_points*np.sin(X_points) + Y_points*np.sin(Y_points)
  fig = plt.figure()
  ax = plt.axes(projection='3d')
  ax.plot_trisurf(X_points.flatten(), Y_points.flatten(), Z_points.flatten())


plot_fxy()
```



2) Initialization

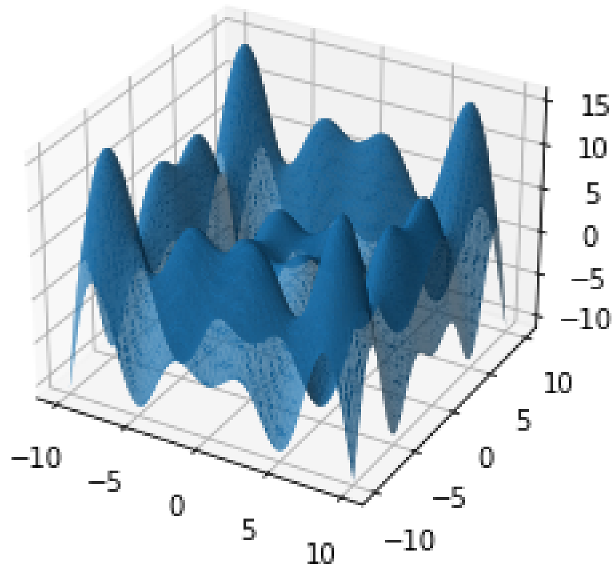
1. From plot we see that x_{min} is around 0,0 so $x__{init}, y__{init} = -5,5 $
2. We take learning rate = 0.01
3. Now we write the algorithms

[12]:

```
from math import cos, sin



x_init = -5
y_init = 5

lr = 0.01

def derivative_fx(x):
    return x*cos(x) + sin(x)

def derivative_fy(y):
    return y*cos(y) + sin(y)

def gradient_update(x, y, learning_rate):
  x -= learning_rate * derivative_fx(x)
  y -= learning_rate * derivative_fy(y)
  return [x,y]

def gradient_plot(x_hist, y_hist, learning_rate):
    ### Plotting the gradient descent
  X_points = np.linspace(-10,10,100)
  Y_points = np.linspace(-10,10,100)

  X_points, Y_points = np.meshgrid(X_points, Y_points)

  Z_points = X_points*np.sin(X_points) + Y_points*np.sin(Y_points)
  ax = plt.axes(projection='3d')
  ax.plot_trisurf(X_points.flatten(), Y_points.flatten(), Z_points.flatten())

  x_hist = np.array(x_hist)
  y_hist = np.array(y_hist)
  z_hist = x_hist*np.sin(x_hist) + y_hist*np.sin(y_hist)

  ax.plot3D(x_hist,y_hist,z_hist)





def gradient_descent(num_iter, x_guess, y_guess, precision, learning_rate,␣
 ↪show_plot = True):
  x_now = x_guess
  y_now = y_guess
  x_prev = 9999999999
  y_prev = 9999999999
```

```python
  x_hist = []
  y_hist = []
  z_hist = []

  iterations = []
  iterations.append(num_iter)
  t = False
  for i in range(num_iter):
    if abs(x_now - x_prev) < precision and abs(y_now - y_prev) < precision:
      t = i
      break

    x_hist.append(x_now)
    y_hist.append(y_now)
    x_prev = x_now
    y_prev = y_now
    tmp = gradient_update(x_now,y_now, learning_rate)
    x_now = tmp[0]
    y_now = tmp[1]

  if t:
    iterations.append(t)
  if show_plot:
    gradient_plot(x_hist, y_hist, learning_rate)
    return [x_now, y_now]

  x_hist = np.array(x_hist)
  y_hist = np.array(y_hist)
  z_hist = x_hist*np.sin(x_hist) + y_hist*np.sin(y_hist)
  return [x_hist, y_hist, z_hist]


x_min = gradient_descent(1000, x_init, y_init, 1e-10, lr)

print(f"X_min is {x_min}")
```

```
X_min is [-4.913180441246508, 4.913180441246508]
```

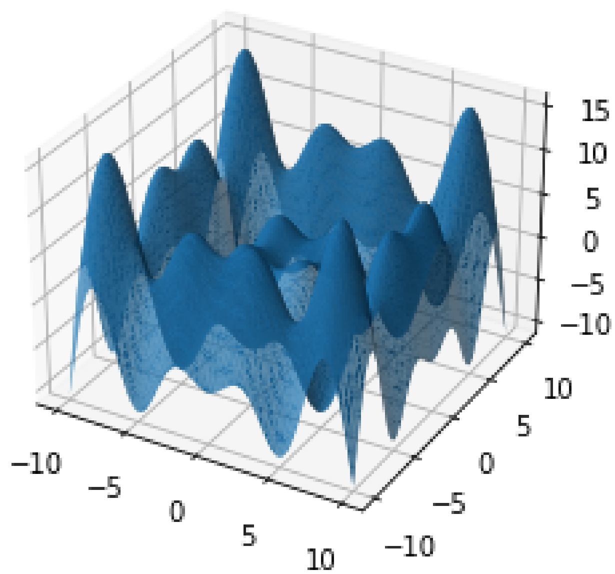


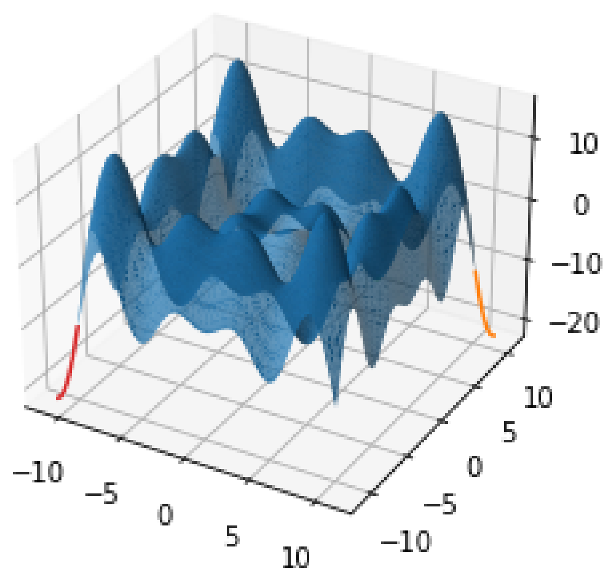
3) Now we vary learning rate and initial points and plot the graphs

1. lr = [0.01, 0.1, 0.3]
2. x_init = [10, 5, -10]

[13]:
```
lr = [0.01, 0.1, 0.3]
X_init = [ 10,  5,  -10]

def Varylr():
    History = []

    for rate in lr:
        History.append(gradient_descent(1000, x_init, y_init, 1e-10, rate,␣
 ↪False))

    X_points = np.linspace(-10,10,100)
    Y_points = np.linspace(-10,10,100)

    X_points, Y_points = np.meshgrid(X_points, Y_points)

    Z_points = X_points*np.sin(X_points) + Y_points*np.sin(Y_points)
    ax = plt.axes(projection='3d')
    ax.plot_trisurf(X_points.flatten(), Y_points.flatten(), Z_points.flatten())

    for i in range(len(lr)):
        x_hist  = History[i][0]
```

```
        y_hist = History[i][1]
        z_hist = History[i][2]
        ax.plot3D(x_hist,y_hist,z_hist)

Varylr()

def Vary_xinit():
    History = []

    for init in X_init:
        History.append(gradient_descent(1000, init, init, 1e-10, 0.01, False))

    X_points = np.linspace(-10,10,100)
    Y_points = np.linspace(-10,10,100)
    X_points, Y_points = np.meshgrid(X_points, Y_points)

    Z_points = X_points*np.sin(X_points) + Y_points*np.sin(Y_points)
    plt.figure()
    ax = plt.axes(projection='3d')
    ax.plot_trisurf(X_points.flatten(), Y_points.flatten(), Z_points.flatten())

    for i in range(len(lr)):
        x_hist  = History[i][0]
        y_hist = History[i][1]
        z_hist = History[i][2]
        ax.plot3D(x_hist,y_hist,z_hist)

Vary_xinit()
```


15
10
5
0
−5
−10
10
5
0
−5
−10
−10
−5
0
5
10


10
0
−10
−20
10
5
0
−5
−10
−10
−5
0
5
10