Name: Karan Dua
Student Id: 21331391

Course Code: CS7DS2-202223 OPTIMISATION ALGORITHMS FOR DATA ANALYSIS

Question (a)(i) Obtain derivative for equation x^4 using Sympy

To obtain the derivative for equation x^4, I have first written a generic function that take the power as an argument and returns the quadratic equation of x using Sympy library for the power argument provided.

```
def get_quadratic_equation(power_argument) :
    f = sm.symbols('x', real = True) ** power_argument
    return f
```

Input: f = get_quadratic_equation(power_argument = 4)
Output: Quadratic equation for f is: x**4

To obtain the derivative of the equation, I created another function that takes equation as input and used **diff()** function of Sympy library to return the derivative for the equation provided as input

```
def get_quadratic_equation_derivative(equation) :
    derivative = sm.diff(equation, sm.symbols('x', real = True))
    return derivative
```

Input: derivative_df_dx = get_quadratic_equation_derivative(f)
Output: Derivative for f = x**4 is: 4*x**3

Question (a)(ii) Sympy Vs. Finite Distance Estimate for perturbation factor δ = 0.01

To obtain the values from derivative using Sympy library, I have first created a function that take the range of x as input. Then It uses the two functions that I have created above to get the expression for the equation x^4 and it's derivative $4x^3$ using **diff()** function from sympy library. Then, I have used **lambdify()** function of Sympy library that creates the lambda for the derivative expression $4x^3$. Finally, I used this lambda expression to caclulate values for the input range of values of x.

```
def get_quadratic_equation_sympy_points(x_axis_point_range) :
    f = get_quadratic_equation(power_argument = 4)
    derivative_df_dx = get_quadratic_equation_derivative(f)
    derivative_df_dx_lambda = sm.lambdify(sm.symbols('x', real = True), derivative_df_dx)
    return derivative_df_dx_lambda(x_axis_point_range)
```

To obtain the values using finite distance I used the finit distance formula $(f(x + \delta) – f(x))/ \delta$. In our case $f(x) = x^4$, therefore the finite distance formula come out to be $((x + \delta)^4 – (x)^4)/\delta$. I have created a function that uses this formula and take a range of x values and perturbation factor δ as input and return the values calculated using this formula.

```
def get_quadratic_equation_finite_difference_points(x_axis_point_range, perturbation_factor) :
    return (((x_axis_point_range + perturbation_factor) ** 4 - x_axis_point_range ** 4) / perturbation_factor)
```

Now, to test these functions, I used the **linespace()** method of the numpy library, to get **500** evenly distanced points in **the range -5 to 5**. Then, I executed these two functions for this range. Below is the plot generated:

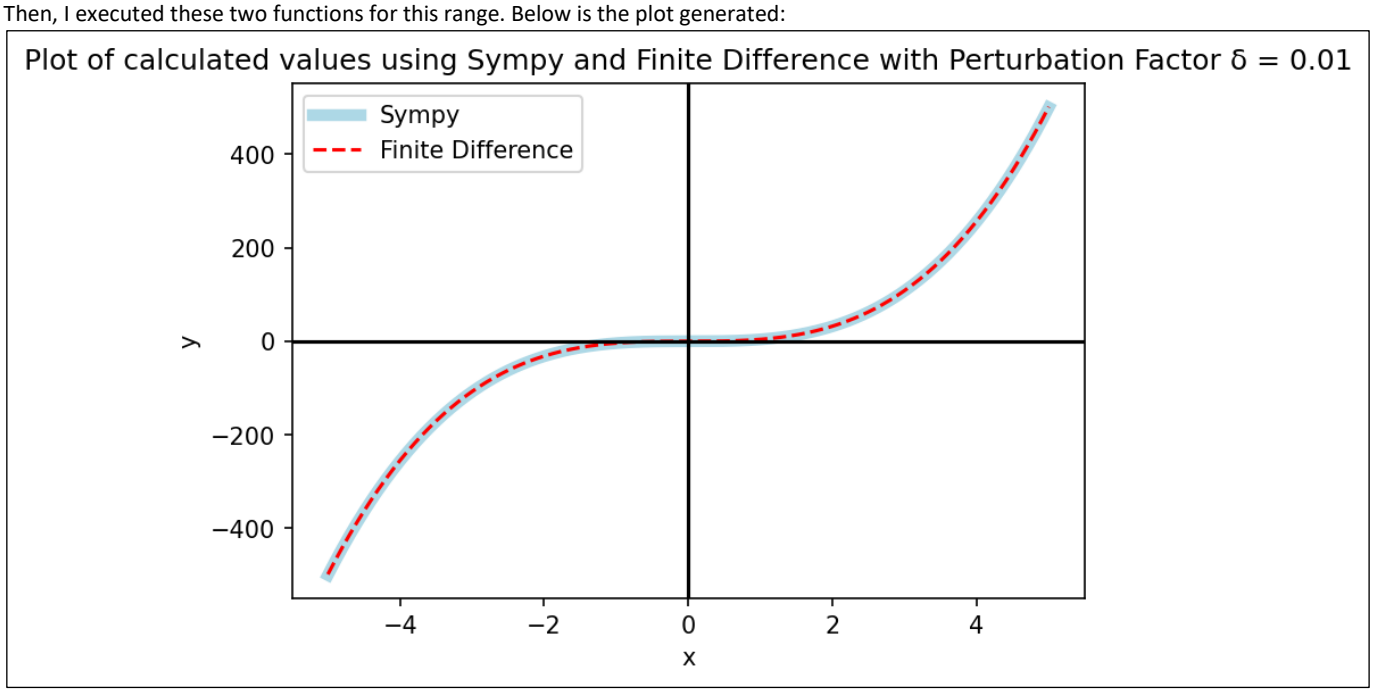

Figure 1: Plot of calculated values using Sympy and Finite Difference

I can clearly analyse from the plot that the two line graphs perfectly overlap with each other for the entire range of x values. Hence, I can verify that the values calculated using finite distance formula exhibit the expected behavior as derivative function for perturbation factor δ = 0.01. Therefore, I can conclude that δ = 0.01 is fit to estimate finite distance points for this equation x^4

Question (a)(iii): Sympy Vs. Finite Distance Estimate for multiple perturbation factors δ

To plot the graph for Finite distance for multiple perturbation factors, I used the same function and passed different perturbation factors in the method. Then, I used the calculated values and generated below graph:

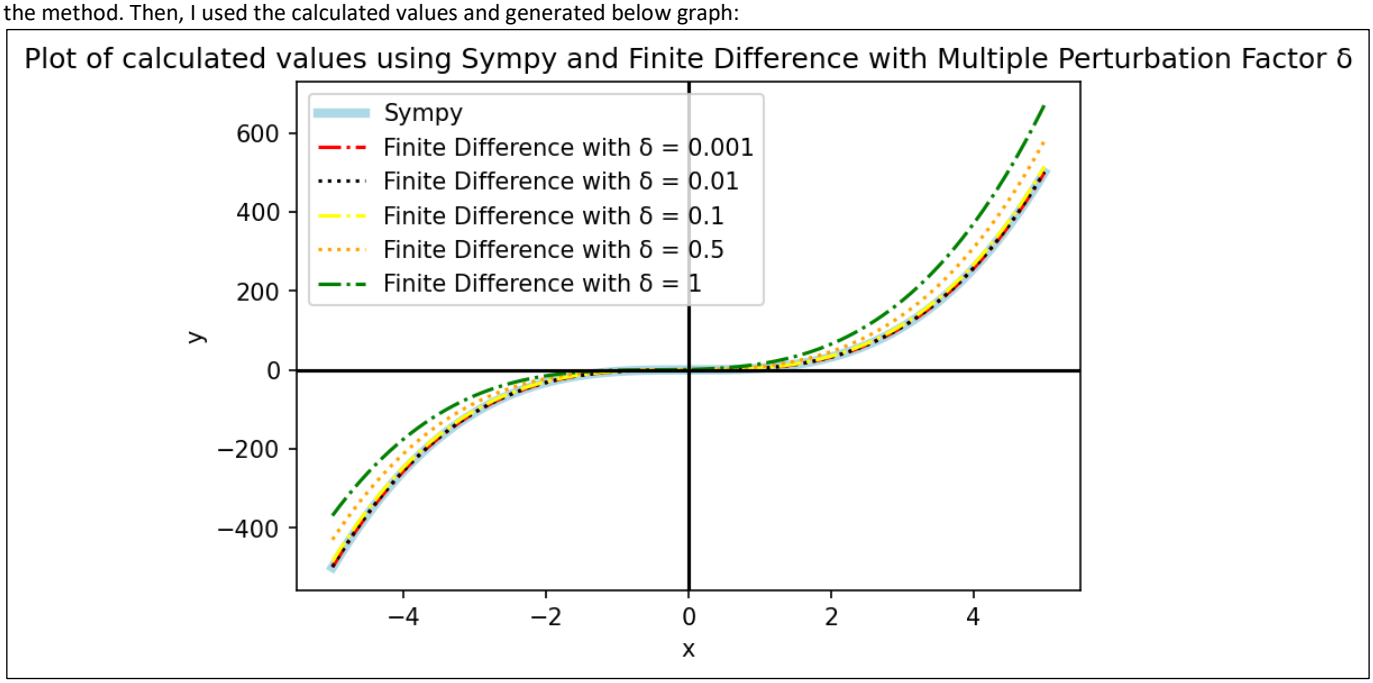

Figure 2: Plot for multiple perturbation factor δ

I can clearly analyse from the plot that as the value of perturbation factor δ increases from 0.001 to 1, the variance of the finite difference plot from the Sympy line plot also increases and reaches the maximum deviation when δ = 1. The primary reason for this behaviour is that the finite distance formula is the calculation of the slope between any two points in the graph and δ decided the distance between these 2 points on the graph, therefore as δ increases, the distance between 2 points increases and there is a larger deviation in the estimated value of slope than the original value of the slope. Therefore, I can conclude that for estimating values using the finite distance formula we should use smaller values of δ as two very close points in the graph will lead to a smaller deviation in slope from the actual slope value calculated using Sympy and line plots will overlap.

Question (b)(i): Gradient Descent with fixed step size

Gradient descent is an algorithm which tries to find the local minima for a given function. It is achieved by calculating the derivative of the function to get slope of the function at point and then updating the current parameters to move towards the steepest descent by a factor called alpha/learning rate/step size. Using this strategy, it moves closer to minima by alpha steps in each iteration.

I have used this strategy and created function that takes the starting point, step size and number of iterations as input and then uses these values to perform the gradient descent. In each iteration, I first store the value of x and f(x) **(in our case x^4)** in two lists which will be used to plot later. Then, I calculate the steepest gradient/slope of the graph **(in our case 4 * x^3)** at that point x using the function **get_quadratic_equation_derivative_value**. Finally, I use this value of steepest slope or tangent at that point to update my starting point parameter for the next iteration by moving further in the direction of the slope by a factor of alpha or step size. This process repeats for n iterations and with each iteration, we move closer towards the local minima of that function by moving in the direction of slope.

```
def get_quadratic_equation_value(x) :
    value = x ** 4
    return value

def get_quadratic_equation_derivative_value(x) :
    value = 4 * (x ** 3)
    return value

def execute_fixed_size_gradient_descent(starting_point = 1, step_size = 0.15, num_iterations = 50) :
    x_point = starting_point
    x_values = []
    y_values = []
    for _ in range(num_iterations) :
        x_values.append(x_point)
```

```
    y_values.append(get_quadratic_equation_value(x_point))
    gradient = get_quadratic_equation_derivative_value(x_point)
    x_point = x_point - (step_size * gradient)

return x_values, y_values
```

Question (b)(ii): Gradient Descent execution with starting point = 1 and alpha = 0.1 for function x^4

I used the above funnctions for generating the below plots to visualise gradient descent with starting point x =1 and step size 0.1.

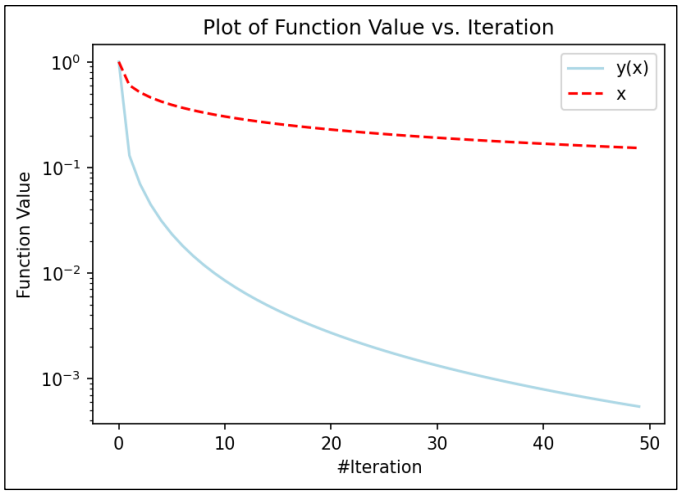

Figure 3a: Plot of function(x^4) value with number of iteration

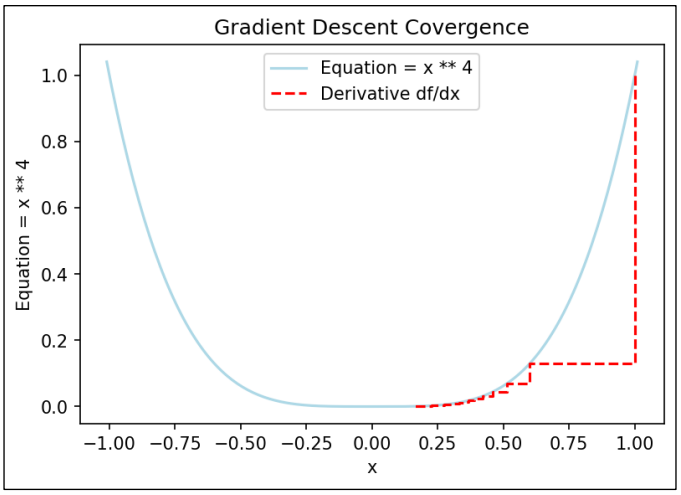

Figure 3b: Plot of Gradient Descent Convergence

From **Figure 3a** I can analyse the that rate of decrease of y(x) (in our case x^4) is faster than that of x. The primary reason for this behaviour is rate of reduction for these graphs. As discussed above, the x value decreases by the rate of step size * gradient($4x^3$) while the function value y decreases at the rate of x^4. I can also analyse that the value of y decreases sharply for the first 10 iterations and then the rate of decrease slows down. The primary reason for this behaviour is that as number of iterations increases the value of gradient decrease which make x smaller and smaller. Therefore, the function value also reduces at a slower rate.

From **Figure 3b** I can analyse that rate of convergence is slowing down with each iteration in gradient descent method. The primary reason is that when curve is steeper, the slope/tangent at any point on the curve is also steeper and so the gradient calculated has a higher value because of the steeper slope. Therefore, the method takes huge steps initially towards the minima. But as the curve flattens out towards minima, the slope of the curve decreases, therefore the value of gradient also decreases and steps become smaller with each iteration.

Question (b)(iii): Gradient Descent execution for variety of alpha and x0

To generate the graphs for a variety of step sizes, I have created 2 functions that are used to plot these graphs. I called these functions for a fixed value of step size at a time and varied the starting position to see its effect. Then I varied the step size and repeated this process for multiple step sizes till convergence is achieved. Below are the plot generated.

For Step Size: 0.001

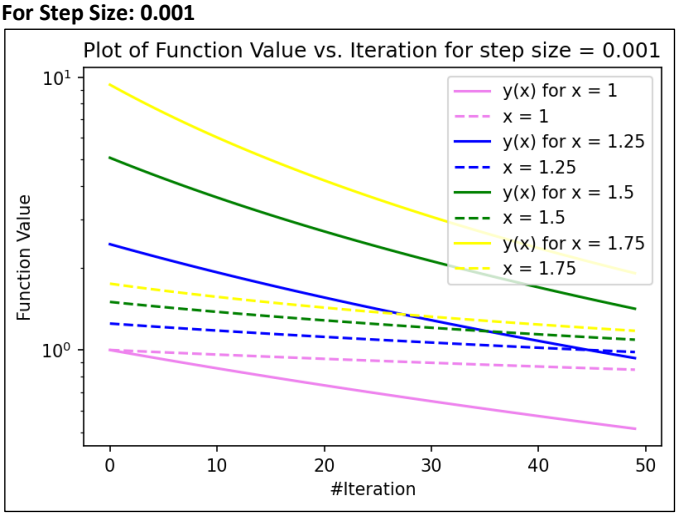

Figure 4a: Plot of function(x^4) value with number of iteration

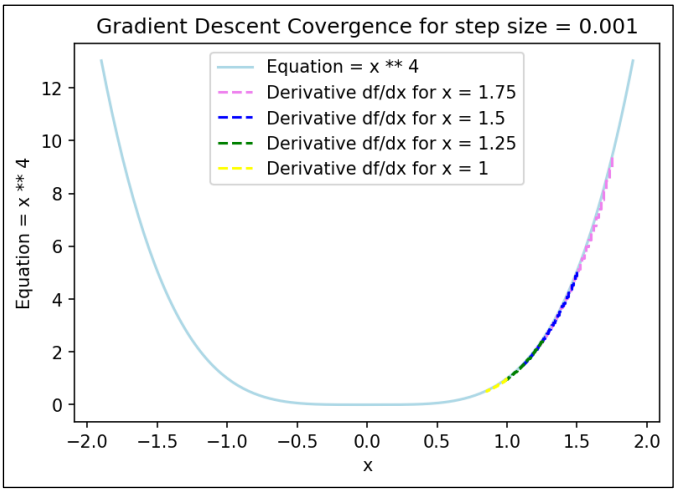

Figure 4b: Plot of Gradient Descent Convergence

Name: Karan Dua
Student Id: 21331391

Course Code: CS7DS2-202223 OPTIMISATION ALGORITHMS FOR DATA ANALYSIS

From **Figure 4a** I can analyse that for all of the starting point x0 from 1 to 1.75 the graphs tend converge towards minima as number of iterations increases. But still for large initial value of x = 1.75 or 1.5 the graphs of y(x) have not fully converged and are still hanging around at approximate value of 1. The primary reason for this behaviour is that function value y has not fully descended as the step size is very small.

From **Figure 4b** I can analyse that for none of the starting points gradient descent was able to converge to minima. The primary reason for this behaviour is that the step size is too low and the gradient descent was not able to converge to minima in the stipulated iterations as initial descent was small when slope was steep. I was able to achieve the convergence up to approximately 0.7 for starting point x0 = 1. This

For Step Size: 0.01

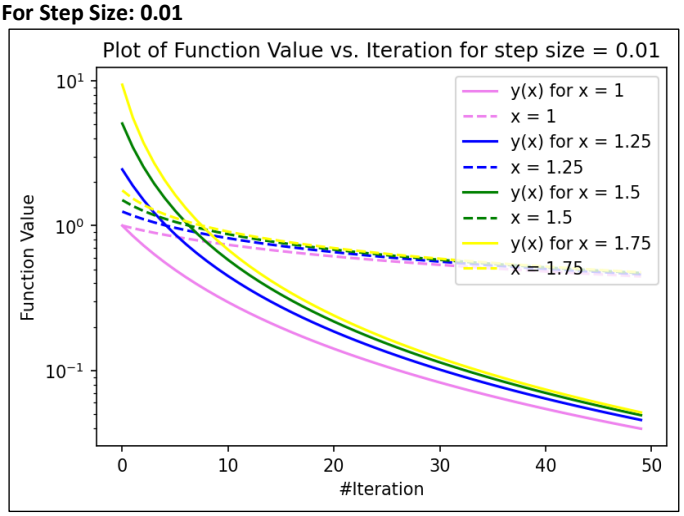

Figure 5a: Plot of function(x^4) value with number of iteration

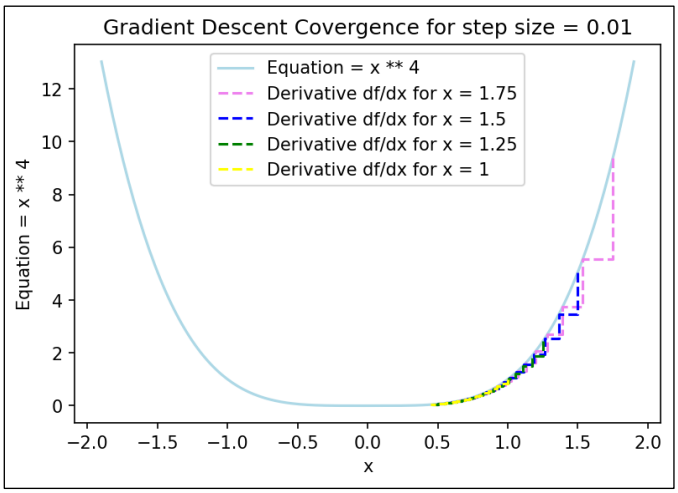

Figure 5b: Plot of Gradient Descent Convergence

From **Figure 5a** I can analyse that graphs all the graphs converge smoothly but still not fully converged. The primary reason is the increase in step size for this plot but still, this step size is low to get full convergence to minima.

From **Figure 5b** I can analyse that increase in step size has positively impacted the convergence towards minima as for step size = 1, I can see that gradient descent was able to achieve convergence value of approximately 0.4. But still, it has not managed to reach minima as step size is not sufficient enough.

For Step Size: 0.15

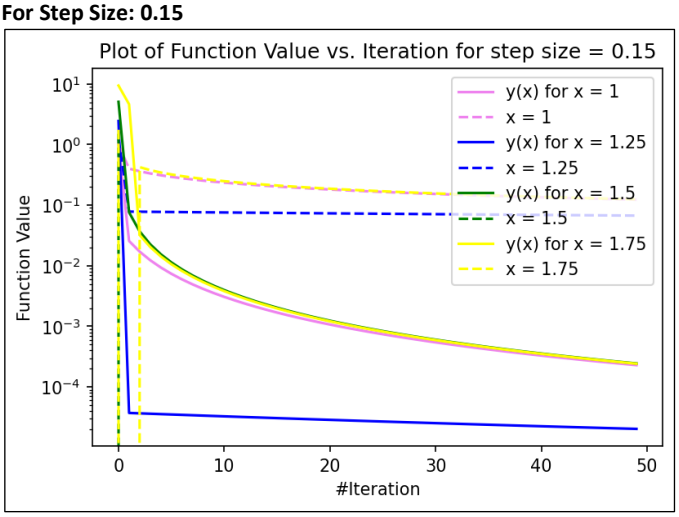

Figure 6a: Plot of function(x^4) value with number of iteration

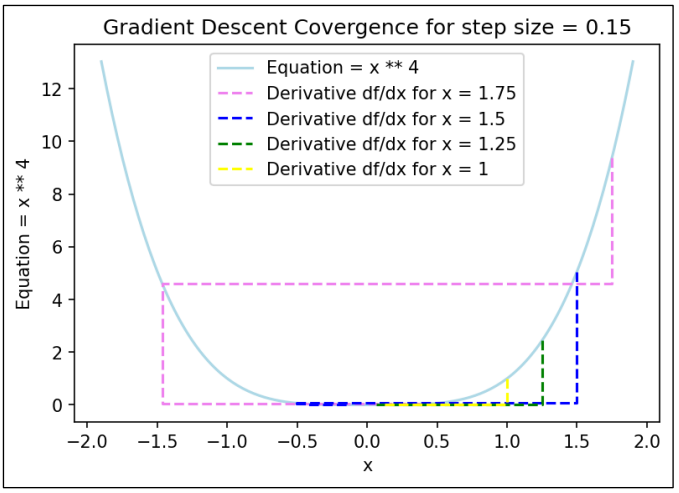

Figure 6b: Plot of Gradient Descent Convergence

From **Figure 6a** I can analyse that all graphs have substantially converged towards minima in the given number of iterations. Thus, this step size is suitable for this equation x^4

From **Figure 6b** I can analyse that all the graphs were able to move very close to minima. This was made possible as increase in step size led to larger descent towards minima in the first few iterations when the curve was steep.

For Step Size: 0.5

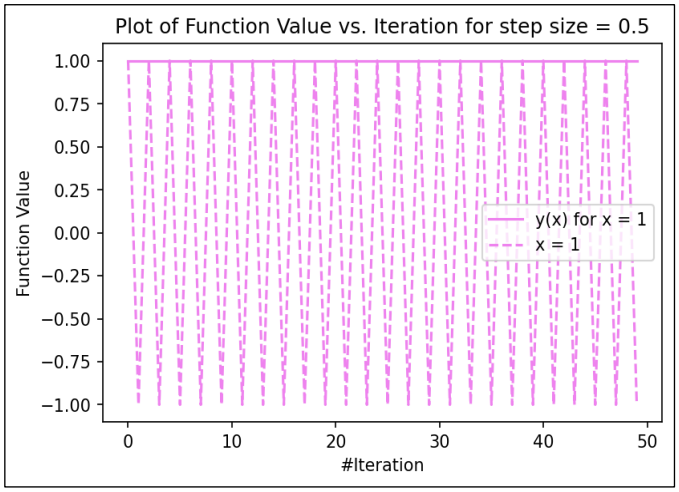

Figure 7a: Plot of function(x^4) value with number of iteration

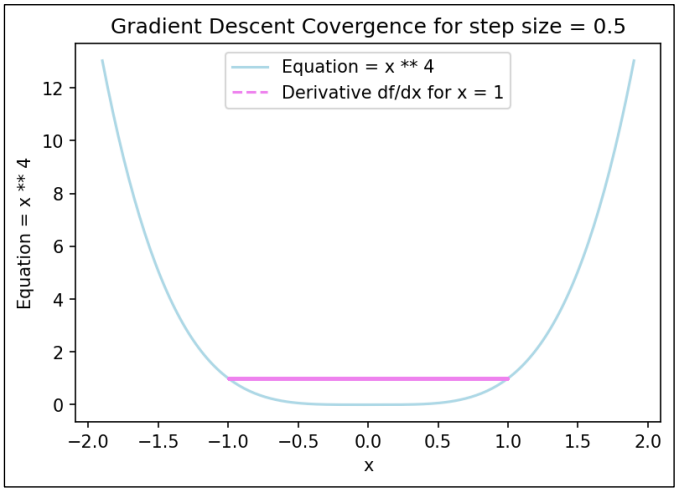

Figure 7b: Plot of Gradient Descent Convergence

From Figures 7a and 7b, I can analyse that as step size becomes sufficiently large the graph never converges towards minima. Rather it keeps on hopping in every iteration between 1 and -1. The primary reason for this behaviour is that for the large step size, the gradient calculated is too big such that instead of converging towards minima it jumps to the other side of minima. Then as slope is reversed and gradient is calculated again misses the minima and slope change again. This happens in every iteration and the graph just hops over minima and never converges towards it. Also, for all other starting points, the value was too large and could not be processed as Python gave error.

Thus I can conclude that step size plays an important role in overall performance of gradient descent and it should be choosen large enough in order to converge to minima smoothly and not very large that it hops over the minima.

Question (c)(i): Gradient Descent execution for y = γx2 for variety of γ

I have modified gradient descent function to accomodate the new equation y = γx2. Then, I created below plots for a variety of γ variables to understand its impact on gradient descent. I have chosen the value of step size and γ variable **based on the lecture notes**.

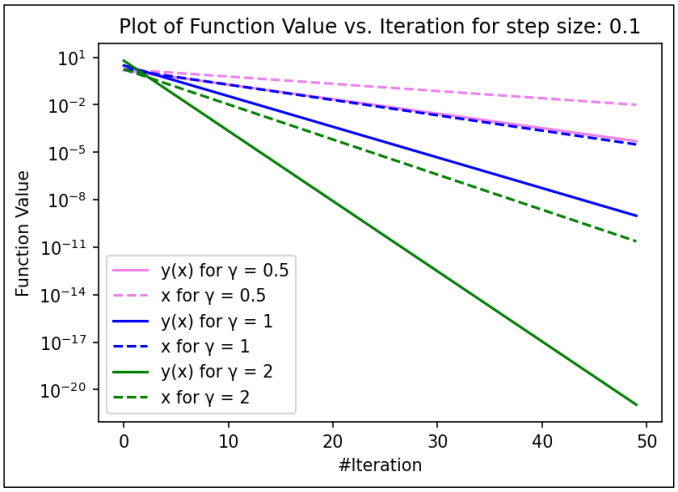

Figure 8a: Plot of function(γx2) value with number of iteration for multiple γ

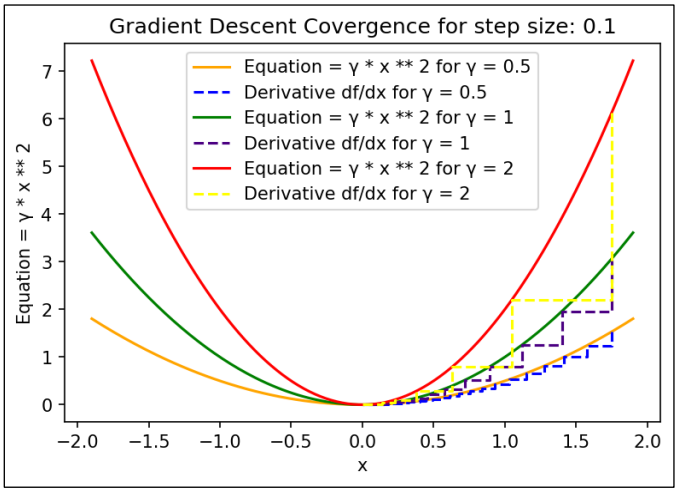

Figure 8b: Plot of Gradient Descent Convergence

From **Figure 8a** I can analyse that as γ increases from 0.5 to 2, the convergence towards minima also increases significantly. I was able to achieve the best convergence for γ = 2. The primary reason for this behaviour is that as γ increases the step between 2 points also increases due to the gradient (2*γ*x) , which is dependent on γ, as step size is common for all the plots and γ will define the magnitude of the step and larger γ will ensure that the graph converges smoothly towards minima because of the sufficiently large gradient.

From **Figure 8b** I can analyse that increasing γ leads to a steeper curve and thus gradient descent to converge faster towards minima. When γ = 2, I can analyse that the curve is steeper than the others the rate of convergence is very high and it leads to larger descend initially. Due to the sufficiently large gradient (2*γ*x), the graph was able to converge to very close to the minima

Thus I can conclude that for function $\gamma x^{2,}$ having larger value of γ helps the graph to descent towards the minima sufficiently fast when step size is constant.

Name: Karan Dua
Student Id: 21331391

Question (c)(ii): Gradient Descent execution for y = γ|x| for variety of γ

I have modified gradient descent function to accomodate the new equation y = γ|x|. Then, I created below plots for a variety of γ variables to understand its impact on gradient descent. I have chosen the value of step size and γ variable **based on the lecture notes**.

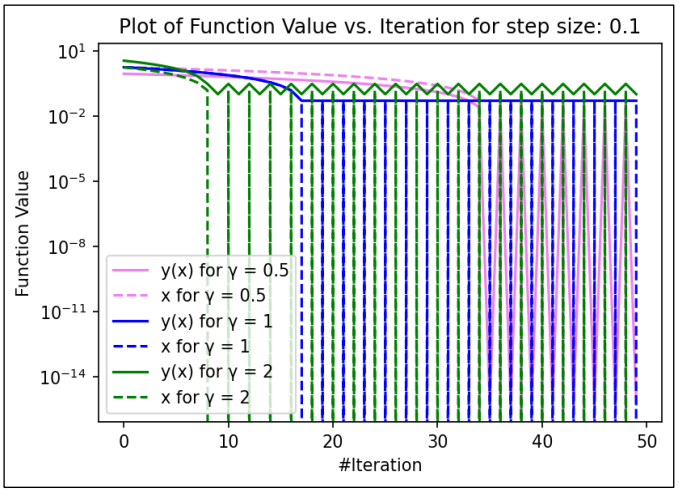

Figure 9a: Plot of function(γx2) value with number of iteration for multiple γ on log scale

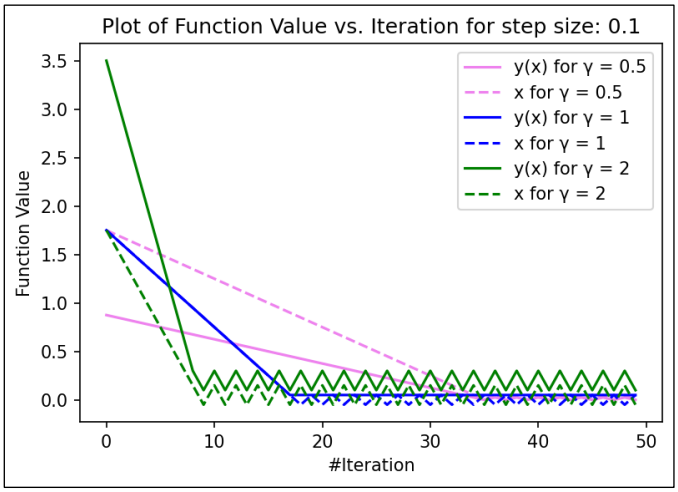

Figure 9b: Plot of function(γx2) value with number of iteration for multiple γ on normal scale

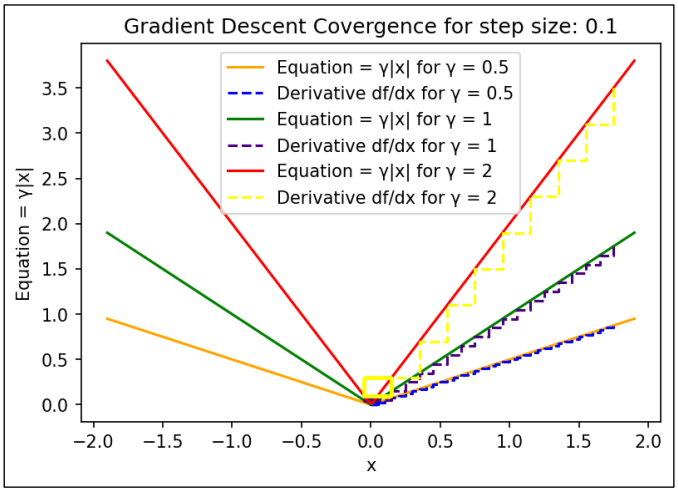

Figure 9c: Plot of Gradient Descent Convergence

From **Figures 9a and 9b,** I can analyse that as γ increases from 0.5 to 2 the earlier it starts to hop over the minima. The reason for hopping is due to presence of a kink in this graph so gradient descent is never able to find the minima even though it has descended towards it but it keeps on hopping over it. But as γ increases the onset of hopping starts earlier. The primary reason for this behaviour is that gradient of this function γ|x| is γx/|x|, therefore step increases as step size is constant and γ will determine the magnitude of step between two points. Due to larger step, the graph with γ = 2 reaches near minima faster than the others, in about 7 iterations, and then it starts hopping over it.

From **Figure 9c** I can analyse that as γ increases from 0.5 to 2 the slope of the curve also increases. This leads to increase in the step in gradient descent. The primary reason for increase in step is that γ determines the slope of the curve γx/|x| and larger γ means larger step in gradient descent. Once the graph reaches near minima it keeps hopping over it.

Thus I can conclude that for function γ|x|, we cannot reach minima for any value of γ. Increasing γ will only lead to a faster descent due to a larger step due to increase in magnitude of gradient but it will keep on hopping over the minima and never find it.

Appendix: Code for Question 1 - All Parts

```
import numpy as np
import matplotlib.pyplot as plot
import sympy as sm
import decimal
```

Question a(i): Obtain derivative for x ** 4

```
def get_quadratic_equation(power_argument) :
    f = sm.symbols('x', real = True) ** power_argument
    return f

f = get_quadratic_equation(power_argument = 4)
print(f"Quadratic equation for f is: {f}")

def get_quadratic_equation_derivative(equation) :
    derivative = sm.diff(equation, sm.symbols('x', real = True))
    return derivative

derivative_df_dx = get_quadratic_equation_derivative(f)
print(f"Derivative for f = {f} is: {derivative_df_dx}")
```

Question a(ii): Sympy Vs. Finite Distance Estimate for perturbation factor δ = 0.01

```
def get_quadratic_equation_sympy_points(x_axis_point_range) :

    f = get_quadratic_equation(power_argument = 4)
    derivative_df_dx = get_quadratic_equation_derivative(f)
    derivative_df_dx_lambda = sm.lambdify(sm.symbols('x', real = True), derivative_df_dx)
    return derivative_df_dx_lambda(x_axis_point_range)

def get_quadratic_equation_finite_difference_points(x_axis_point_range, perturbation_factor) :

    return (((x_axis_point_range + perturbation_factor) ** 4 - x_axis_point_range ** 4) / perturbation_factor)

calculated_sympy_points = get_quadratic_equation_sympy_points(np.linspace(-5, 5, 500))

calculated_finite_difference_points = get_quadratic_equation_finite_difference_points(np.linspace(-5, 5, 500), 0.01)

plot.figure(dpi = 150)
plot.plot(np.linspace(-5, 5, 500), calculated_sympy_points, color = 'lightblue', lw = 5, label = "Sympy")
plot.plot(np.linspace(-5, 5, 500), calculated_finite_difference_points, color = 'red', linestyle='dashed', label='Finite Difference')
plot.axhline(0, color = 'black')
plot.axvline(0, color = 'black')
plot.xlabel('x')
plot.ylabel('y')
plot.title('Plot of calculated values using Sympy and Finite Difference with Perturbation Factor δ = {}'.format(0.01))
plot.legend()
plot.show()
```

Question a(iii): Sympy Vs. Finite Distance Estimate for multiple perturbation factors

```
calculated_sympy_points = get_quadratic_equation_sympy_points(np.linspace(-5, 5, 500))

calculated_finite_difference_points_0_001 = get_quadratic_equation_finite_difference_points(np.linspace(-5, 5, 500), 0.001)
calculated_finite_difference_points_0_01 = get_quadratic_equation_finite_difference_points(np.linspace(-5, 5, 500), 0.01)
calculated_finite_difference_points_0_1 = get_quadratic_equation_finite_difference_points(np.linspace(-5, 5, 500), 0.1)
calculated_finite_difference_points_0_5 = get_quadratic_equation_finite_difference_points(np.linspace(-5, 5, 500), 0.5)
calculated_finite_difference_points_1 = get_quadratic_equation_finite_difference_points(np.linspace(-5, 5, 500), 1)

plot.figure(dpi = 150)
plot.plot(np.linspace(-5, 5, 500), calculated_sympy_points, color = 'lightblue', lw = 4, label = "Sympy")
plot.plot(np.linspace(-5, 5, 500), calculated_finite_difference_points_0_001, color = 'red', linestyle='dashdot', label='Finite Difference with δ
= 0.001')
```

```
plot.plot(np.linspace(-5, 5, 500), calculated_finite_difference_points_0_01, color = 'black', linestyle='dotted', label='Finite Difference with δ = 0.01')
plot.plot(np.linspace(-5, 5, 500), calculated_finite_difference_points_0_1, color = 'yellow', linestyle='dashdot', label='Finite Difference with δ = 0.1')
plot.plot(np.linspace(-5, 5, 500), calculated_finite_difference_points_0_5, color = 'orange', linestyle='dotted', label='Finite Difference with δ = 0.5')
plot.plot(np.linspace(-5, 5, 500), calculated_finite_difference_points_1, color = 'green', linestyle='dashdot', label='Finite Difference with δ = 1')
plot.axhline(0, color = 'black')
plot.axvline(0, color = 'black')
plot.xlabel('x')
plot.ylabel('y')
plot.title('Plot of calculated values using Sympy and Finite Difference with Multiple Perturbation Factor δ')
plot.legend()
plot.show()
```

Name: Karan Dua
Student Id: 21331391

Appendix: Code for Question 2 - All Parts

```
import numpy as np
import matplotlib.pyplot as plot
import sympy as sm
import decimal
```

Question b(i): Gradient Descent with fixed step size

```
def get_quadratic_equation_value(x) :
    value = x ** 4
    return value

def get_quadratic_equation_derivative_value(x) :
    value = 4 * (x ** 3)
    return value

def execute_fixed_size_gradient_descent(starting_point = 1, step_size = 0.15, num_iterations = 50) :
    x_point = starting_point
    x_values = []
    y_values = []
    for _ in range(num_iterations) :
        x_values.append(x_point)
        y_values.append(get_quadratic_equation_value(x_point))
        gradient = get_quadratic_equation_derivative_value(x_point)
        x_point = x_point - (step_size * gradient)

    return x_values, y_values
```

Question b(ii): Gradient Descent execution with x0 = 1 and alpha = 0.1

```
x_values, y_values = execute_fixed_size_gradient_descent(starting_point = 1, step_size = 0.1, num_iterations = 50)

plot.figure(dpi = 150)
plot.plot(range(50), y_values, color = 'lightblue', label = "y(x)")
plot.plot(range(50), x_values, color = 'red', linestyle='dashed', label='x')
plot.xlabel('#Iteration')
plot.ylabel('Function Value')
plot.title('Plot of Function Value vs. Iteration')
plot.yscale("log")
plot.legend()
plot.show()

x_points = np.linspace(-1.01, 1.01, 200)
y_points = get_quadratic_equation_value(x_points)

x_values, y_values = execute_fixed_size_gradient_descent(starting_point = 1, step_size = 0.1, num_iterations = 50)

plot.figure(dpi = 150)
plot.plot(x_points, y_points, color = 'lightblue', label = "Equation = x ** 4")
plot.step(x_values, y_values, color = 'red', linestyle='dashed', label='Derivative df/dx')
plot.xlabel('x')
plot.ylabel('Equation = x ** 4')
plot.title('Gradient Descent Covergence')
plot.legend()
plot.show()
```

Question b(iii): Gradient Descent execution for variety of alpha and x0

```
def get_quadratic_equation_value(x) :
    value = x ** 4
    return value

def get_quadratic_equation_derivative_value(x) :
    value = 4 * (x ** 3)
    return value
```

```
def plot_fixed_size_gradient_descent(plot, color, starting_point = 2, step_size = 0.15, num_iterations = 50) :
    x_point = starting_point
    x_values = []
    y_values = []
    for _ in range(num_iterations) :
        x_values.append(x_point)
        y_values.append(get_quadratic_equation_value(x_point))
        gradient = get_quadratic_equation_derivative_value(x_point)
        x_point = x_point - (step_size * gradient)


    plot.plot(range(num_iterations), y_values, color = color, label = "y(x) for x = {}".format(starting_point))
    plot.plot(range(num_iterations), x_values, color = color, linestyle='dashed', label='x = {}'.format(starting_point))

def plot_fixed_size_gradient_descent_derivative(plot, color, starting_point = 2, step_size = 0.15, num_iterations = 50) :
    x_point = starting_point
    x_values = []
    y_values = []
    for _ in range(num_iterations) :
        x_values.append(x_point)
        y_values.append(get_quadratic_equation_value(x_point))
        gradient = get_quadratic_equation_derivative_value(x_point)
        x_point = x_point - (step_size * gradient)

    plot.step(x_values, y_values, color = color, linestyle='dashed', label="Derivative df/dx for x = {}".format(starting_point))


step_size = 0.001
starting_points = [1, 1.25, 1.5, 1.75]
color = ['violet', 'blue', 'green', 'yellow']
count = 0
plot.figure(dpi = 150)
for starting_point in starting_points :
    try :
        plot_fixed_size_gradient_descent(plot, color = color[count], starting_point = starting_point, step_size = step_size, num_iterations = 50)
        count += 1
    except :
        print('Exception Occured')

plot.xlabel('#Iteration')
plot.ylabel('Function Value')
plot.title('Plot of Function Value vs. Iteration for step size = {}'.format(step_size))
plot.yscale("log")
plot.legend()
plot.show()

x_points = np.linspace(-1.9, 1.9, 200)
y_points = get_quadratic_equation_value(x_points)

step_size = 0.001
starting_points = [1.75, 1.5, 1.25, 1]
color = ['violet', 'blue', 'green', 'yellow']
count = 0
plot.figure(dpi = 150)
plot.plot(x_points, y_points, color = 'lightblue', label = "Equation = x ** 4")
for starting_point in starting_points :
    try :
        plot_fixed_size_gradient_descent_derivative(plot, color = color[count], starting_point = starting_point, step_size = step_size,
num_iterations = 50)
        count += 1
    except :
        print('Exception Occured')

plot.xlabel('x')
plot.ylabel('Equation = x ** 4')
```

```
plot.title('Gradient Descent Covergence for step size = {}'.format(step_size))
plot.legend()
plot.show()

step_size = 0.01
starting_points = [1, 1.25, 1.5, 1.75]
color = ['violet', 'blue', 'green', 'yellow']
count = 0
plot.figure(dpi = 150)
for starting_point in starting_points :
   try :
      plot_fixed_size_gradient_descent(plot, color = color[count], starting_point = starting_point, step_size = step_size, num_iterations = 50)
      count += 1
   except :
      print('Exception Occured')

plot.xlabel('#Iteration')
plot.ylabel('Function Value')
plot.title('Plot of Function Value vs. Iteration for step size = {}'.format(step_size))
plot.yscale("log")
plot.legend()
plot.show()

x_points = np.linspace(-1.9, 1.9, 200)
y_points = get_quadratic_equation_value(x_points)

step_size = 0.01
starting_points = [1.75, 1.5, 1.25, 1]
color = ['violet', 'blue', 'green', 'yellow']
count = 0
plot.figure(dpi = 150)
plot.plot(x_points, y_points, color = 'lightblue', label = "Equation = x ** 4")
for starting_point in starting_points :
   try :
      plot_fixed_size_gradient_descent_derivative(plot, color = color[count], starting_point = starting_point, step_size = step_size,
num_iterations = 50)
      count += 1
   except :
      print('Exception Occured')

plot.xlabel('x')
plot.ylabel('Equation = x ** 4')
plot.title('Gradient Descent Covergence for step size = {}'.format(step_size))
plot.legend()
plot.show()

step_size = 0.15
starting_points = [1, 1.25, 1.5, 1.75]
color = ['violet', 'blue', 'green', 'yellow']
count = 0
plot.figure(dpi = 150)
for starting_point in starting_points :
   try :
      plot_fixed_size_gradient_descent(plot, color = color[count], starting_point = starting_point, step_size = step_size, num_iterations = 50)
      count += 1
   except :
      print('Exception Occured')

plot.xlabel('#Iteration')
plot.ylabel('Function Value')
plot.title('Plot of Function Value vs. Iteration for step size = {}'.format(step_size))
plot.yscale("log")
plot.legend()
plot.show()

x_points = np.linspace(-1.9, 1.9, 200)
```

```
y_points = get_quadratic_equation_value(x_points)

step_size = 0.15
starting_points = [1.75, 1.5, 1.25, 1]
color = ['violet', 'blue', 'green', 'yellow']
count = 0
plot.figure(dpi = 150)
plot.plot(x_points, y_points, color = 'lightblue', label = "Equation = x ** 4")
for starting_point in starting_points :
  try :
    plot_fixed_size_gradient_descent_derivative(plot, color = color[count], starting_point = starting_point, step_size = step_size, num_iterations = 50)
    count += 1
  except :
    print('Exception Occured')

plot.xlabel('x')
plot.ylabel('Equation = x ** 4')
plot.title('Gradient Descent Covergence for step size = {}'.format(step_size))
plot.legend()
plot.show()

step_size = 0.5
starting_points = [1, 1.25, 1.5, 1.75]
color = ['violet', 'blue', 'green', 'yellow']
count = 0
plot.figure(dpi = 150)
for starting_point in starting_points :
  try :
    plot_fixed_size_gradient_descent(plot, color = color[count], starting_point = starting_point, step_size = step_size, num_iterations = 50)
    count += 1
  except :
    print('Exception Occured')

plot.xlabel('#Iteration')
plot.ylabel('Function Value')
plot.title('Plot of Function Value vs. Iteration for step size = {}'.format(step_size))
plot.legend()
plot.show()

x_points = np.linspace(-1.9, 1.9, 200)
y_points = get_quadratic_equation_value(x_points)

step_size = 0.5
starting_points = [1.75, 1.5, 1.25, 1]
color = ['violet', 'blue', 'green', 'yellow']
count = 0
plot.figure(dpi = 150)
plot.plot(x_points, y_points, color = 'lightblue', label = "Equation = x ** 4")
for starting_point in starting_points :
  try :
    plot_fixed_size_gradient_descent_derivative(plot, color = color[count], starting_point = starting_point, step_size = step_size, num_iterations = 50)
    count += 1
  except :
    print('Exception Occured')

plot.xlabel('x')
plot.ylabel('Equation = x ** 4')
plot.title('Gradient Descent Covergence for step size = {}'.format(step_size))
plot.legend()
plot.show()
```

Appendix: Code for Question 3- All Parts

```
import numpy as np
import matplotlib.pyplot as plot
import sympy as sm
import decimal
```

Question c(i): Gradient Descent execution for y = γx2 for variety of γ

```
def get_quadratic_equation_value(x, gamma) :
    value = gamma * (x ** 2)
    return value

def get_quadratic_equation_derivative_value(x, gamma) :
    value = 2 * gamma * x
    return value

def plot_fixed_size_gradient_descent(plot, color, starting_point = 2, step_size = 0.15, num_iterations = 50, gamma = 1) :
    x_point = starting_point
    x_values = []
    y_values = []
    for _ in range(num_iterations) :
        x_values.append(x_point)
        y_values.append(get_quadratic_equation_value(x_point, gamma))
        gradient = get_quadratic_equation_derivative_value(x_point, gamma)
        x_point = x_point - (step_size * gradient)


    plot.plot(range(num_iterations), y_values, color = color, label = "y(x) for γ = {}".format(gamma))
    plot.plot(range(num_iterations), x_values, color = color, linestyle='dashed', label='x for γ = {}'.format(gamma))

def plot_fixed_size_gradient_descent_derivative(plot, color_curve, color_der, starting_point = 2, step_size = 0.15, num_iterations = 50,
gamma = 1) :
    x_points = np.linspace(-1.9, 1.9, 200)
    y_points = get_quadratic_equation_value(x_points, gamma)
    x_point = starting_point
    x_values = []
    y_values = []
    for _ in range(num_iterations) :
        x_values.append(x_point)
        y_values.append(get_quadratic_equation_value(x_point, gamma))
        gradient = get_quadratic_equation_derivative_value(x_point, gamma)
        x_point = x_point - (step_size * gradient)

    plot.plot(x_points, y_points, color = color_curve, label = "Equation = γ * x ** 2 for γ = {}".format(gamma))
    plot.step(x_values, y_values, color = color_der, linestyle='dashed', label="Derivative df/dx for γ = {}".format(gamma))

step_size = 0.1
starting_point = 1.75
gamma_array = [0.5, 1, 2]
color = ['violet', 'blue', 'green']
count = 0
plot.figure(dpi = 150)
for gamma in gamma_array :
    try :
        plot_fixed_size_gradient_descent(plot, color = color[count], starting_point = starting_point, step_size = step_size, num_iterations = 50,
gamma = gamma)
        count += 1
    except :
        print('Exception Occured')

plot.xlabel('#Iteration')
plot.ylabel('Function Value')
plot.title('Plot of Function Value vs. Iteration for step size: {}'.format(step_size))
plot.yscale("log")
```

```
plot.legend()
plot.show()

step_size = 0.1
starting_point = 1.75
gamma_array = [0.5, 1, 2]
color = ['orange', 'blue', 'green', 'indigo', 'red', 'yellow']
count = 0
plot.figure(dpi = 150)

for gamma in gamma_array :
    try :
        plot_fixed_size_gradient_descent_derivative(plot, color_curve = color[count], color_der = color[count+1], starting_point =
starting_point, step_size = step_size, num_iterations = 50, gamma = gamma)
        count += 2
    except :
        print('Exception Occured')

plot.xlabel('x')
plot.ylabel('Equation = γ * x ** 2')
plot.title('Gradient Descent Covergence for step size: {}'.format(step_size))
plot.legend()
plot.show()
```

Question c(ii): Gradient Descent execution for γ = γ|x| for variety of γ

```
def get_quadratic_equation_value(x, gamma) :
    value = gamma * (abs(x))
    return value

def get_quadratic_equation_derivative_value(x, gamma) :
    value = gamma * (x / abs(x))
    return value

def plot_fixed_size_gradient_descent(plot, color, starting_point = 2, step_size = 0.15, num_iterations = 50, gamma = 1) :
    x_point = starting_point
    x_values = []
    y_values = []
    for _ in range(num_iterations) :
        x_values.append(x_point)
        y_values.append(get_quadratic_equation_value(x_point, gamma))
        gradient = get_quadratic_equation_derivative_value(x_point, gamma)
        x_point = x_point - (step_size * gradient)


    plot.plot(range(num_iterations), y_values, color = color, label = "y(x) for γ = {}".format(gamma))
    plot.plot(range(num_iterations), x_values, color = color, linestyle='dashed', label='x for γ = {}'.format(gamma))

def plot_fixed_size_gradient_descent_derivative(plot, color_curve, color_der, starting_point = 2, step_size = 0.15, num_iterations = 50,
gamma = 1) :
    x_points = np.linspace(-1.9, 1.9, 200)
    y_points = get_quadratic_equation_value(x_points, gamma)
    x_point = starting_point
    x_values = []
    y_values = []
    for _ in range(num_iterations) :
        x_values.append(x_point)
        y_values.append(get_quadratic_equation_value(x_point, gamma))
        gradient = get_quadratic_equation_derivative_value(x_point, gamma)
        x_point = x_point - (step_size * gradient)

    plot.plot(x_points, y_points, color = color_curve, label = "Equation = γ|x| for γ = {}".format(gamma))
    plot.step(x_values, y_values, color = color_der, linestyle='dashed', label="Derivative df/dx for γ = {}".format(gamma))

step_size = 0.1
starting_point = 1.75
```

```
gamma_array = [0.5, 1, 2]
color = ['violet', 'blue', 'green']
count = 0
plot.figure(dpi = 150)
for gamma in gamma_array :
  try :
    plot_fixed_size_gradient_descent(plot, color = color[count], starting_point = starting_point, step_size = step_size, num_iterations = 50,
gamma = gamma)
    count += 1
  except :
    print('Exception Occured')

plot.xlabel('#Iteration')
plot.ylabel('Function Value')
plot.title('Plot of Function Value vs. Iteration for step size: {}'.format(step_size))
plot.yscale("log")
plot.legend()
plot.show()

step_size = 0.1
starting_point = 1.75
gamma_array = [0.5, 1, 2]
color = ['violet', 'blue', 'green']
count = 0
plot.figure(dpi = 150)
for gamma in gamma_array :
  try :
    plot_fixed_size_gradient_descent(plot, color = color[count], starting_point = starting_point, step_size = step_size, num_iterations = 50,
gamma = gamma)
    count += 1
  except :
    print('Exception Occured')

plot.xlabel('#Iteration')
plot.ylabel('Function Value')
plot.title('Plot of Function Value vs. Iteration for step size: {}'.format(step_size))
plot.legend()
plot.show()

step_size = 0.1
starting_point = 1.75
gamma_array = [0.5, 1, 2]
color = ['orange', 'blue', 'green', 'indigo', 'red', 'yellow']
count = 0
plot.figure(dpi = 150)

for gamma in gamma_array :
  try :
    plot_fixed_size_gradient_descent_derivative(plot, color_curve = color[count], color_der = color[count+1], starting_point =
starting_point, step_size = step_size, num_iterations = 50, gamma = gamma)
    count += 2
  except :
    print('Exception Occured')

plot.xlabel('x')
plot.ylabel('Equation = γ|x|')
plot.title('Gradient Descent Covergence for step size: {}'.format(step_size))
plot.legend()
plot.show()
```