Introduction:

The optimization of algorithms allows improving both accuracy and operational efficiency of the models. The report analyzes optimization techniques using calculus and numerical methods while conducting implementation through the Python programming language.

Part A (i) – Derivative calculation using sympy library

To compute the derivative of the function $y(x) = x^4$

Steps involved to compute the derivative:

- Importing the python library sympy for symbolic mathematics.
- Defining the symbolic variable, declare x as a symbolic variable using the sympy symbol's function.
- Define the function $y(x) = x^4$ using the sympy expressions
- Compute the derivative using sympy's diff function to differentiate $y(x)$ with respect to x, then the function returns the derivative $4x^3$.
- Output of the function and the derivative verify the correctness of the differentiation.

Part A (ii) – Derivative estimation & Comparison

With previously computed $4x^3$, to calculate the value over a range of x using the NumPy array and then converting the derivative into the numerical values. This transformation from a symbol expression to numerical value is necessary for visual comparisons

Finite Difference Method

$$\frac{dy}{dx} \sim \frac{f(x + \delta) - f(x - \delta)}{2\delta}$$

Here δ is the small perturbation which is added and subtracted from x and $f(x)$ and it is evaluated at the perturbed points, difference in these function values is then divided by twice the perturbation eventually giving approximate value required.

Choice of δ:

- $\delta = 0.01$ is chosen as it can balance between the accuracy and the numerical stability, smaller δ can then reduce the approximation error but it still can increase the numerical errors due to the floating-point limitations.
- The finite difference method enables numerical derivative estimation when symbolic forms prove complicated to obtain along with experimental data analysis that contains only numerical values.

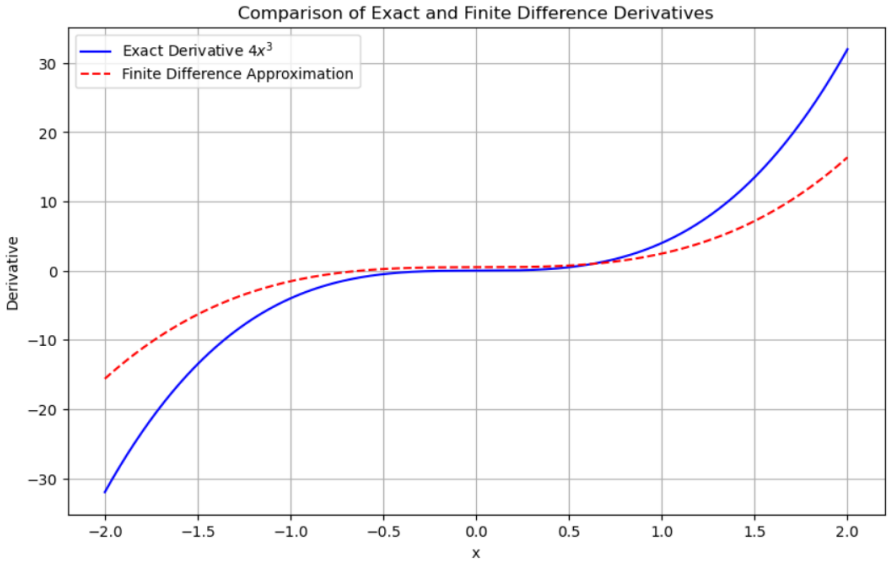

Figure 1, Comparison of Exact & Finite Difference Derivatives

From figure 1, the exact derivative is the blue line and the finite difference method which is represented in red dotted line shows the comparison that these lines align well, validating the effectiveness of the finite difference method with the selected parameters. This is used to understand the limitations and the errors that occur in numerical approximations eventually assists in choosing the appropriate methods and parameters for the derivative calculations.

Analysis:

- The calculated derivative with a step size of δ=0.01 offers satisfactory accuracy across the selected range so it proves suitable for derivative estimation of this function.
- The proximity of the curves shows minor deviations which occur when the function has steep curvature points ($x = -2\ and\ x = 2$). The finite difference method produces an approximate slope across multiple δ points because it calculates an averaged slope smoothing during the time interval.
- The selection of δ value determines how accurately the finite difference approximation calculates its results. Approximations become more accurate as δ decreases even further yet these advantages need evaluation of computational performance and numerical stability.

Part A (iii) - Impact of Perturbation Size on Derivative Accuracy

Examine how varying the perturbation δ in the finite difference approximation affects the accuracy of the estimated derivative of $y(x) = x^4$

Steps:

- Setting Up Parameters: First choose a uniform selection of x values which will serve as the basis for derivative computation. Test the method using multiple δ values starting from tiny (0.001) and extending to bigger ones (1) to observe the relationship between δ and accuracy.
- Finite difference Calculation: For each δ, compute the finite difference approximation of the derivative

$$\frac{dy}{dx} \sim \frac{f(x + \delta) - f(x - \delta)}{2\delta}$$

- Calculate the exact derivative $y(x) = 4x^3$ for each x in the chosen range.
- For each δ, compute the absolute difference between the exact derivative and the finite difference estimate at each x.

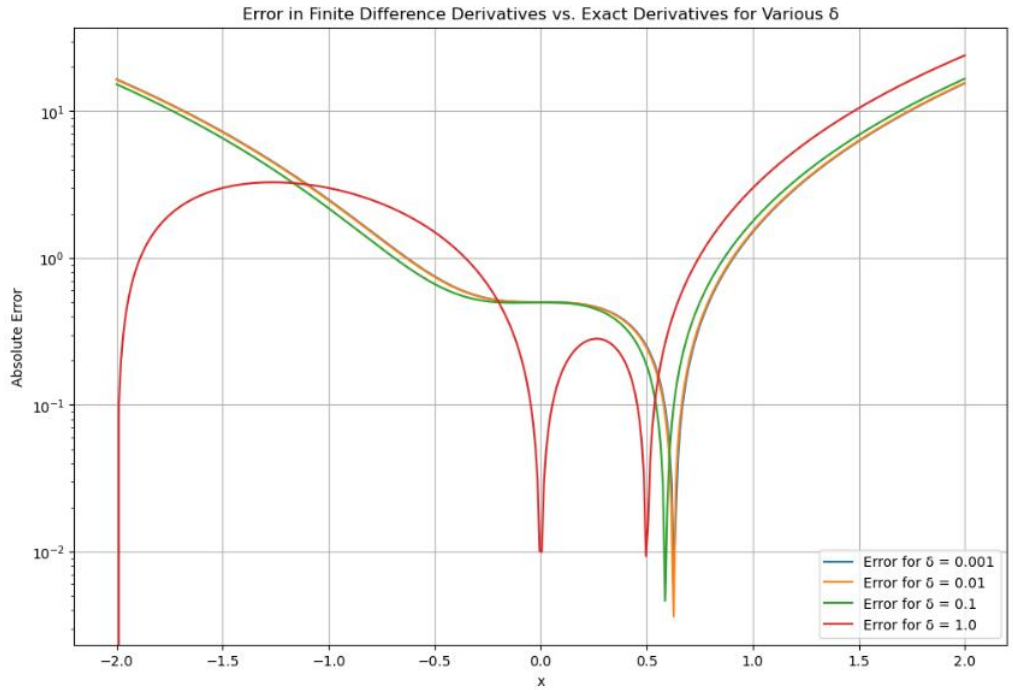

Figure 2, Error in finite difference Vs Exact derivatives for various δ

Analysis:

- From the figure 2, the absolute error will decrease when a smaller value of δ is used. The selection of δ = 0.001 generates the most accurate results over all tested xxx values because small perturbations produce better derivative approximations.
- The highest error produced by δ=1.0 becomes most apparent at the point of x=0x = 0x=0 where the derivative undergoes its greatest variation.
- All δ values produce error spikes at x=0 because this area displays intense curvature in the function $y(x) = x^4$. The finite difference method experiences difficulties in this region because its built-in linear approximation loses its validity.
- When x values distance from zero the computational error becomes stable with reduced sensitivity to the δ parameter when the x values are near -2 and 2. The points behave in this manner because the function's curvature demonstrates reduced severity which produces better accuracy for derivative approximations.

Part B (i) - Gradient Descent

To implement the program for gradient descendent on the function $y(x) = x^4$ with a fixed step size of α.

Steps:

- The defined function $y(x) = x^4$ has its derivative $\frac{dy}{dx} = 4x^3$. The function represents the minimize target while its derivative shows the direction of maximum ascent magnitude at every point x.
- Set the initial value of x to beginning at a location that is not optimum (x=1) for the function. The algorithm takes fixed steps determined by α to move against the slope direction of the gradient. Step size determines both the stability and convergence pace of the method.

Gradient Descent Algorithm:

- The iteration process will execute both pre-defined steps and small x change measurements to detect convergence.
- Each iteration requires an update of x based on the following rule: $x_{new} = x_{old} - \alpha \times$gradient
- The gradient value is $\frac{dy}{dx}$ from the point where the algorithm currently stands. The algorithm implements a transition of x in the direction where y diminishes rapidly.

Output:

[(0.96, 1.0),
(0.92461056, 0.8493465599999999),
(0.8929924039197126, 0.7308620249890496),
(0.8645082525319249, 0.6359032705348381),
(0.838663794966141, 0.5585686711957822),
(0.8150685942414523, 0.4947110145688134),
(0.7934093913400703, 0.4413434015304947),
(0.7734313916682115, 0.3962683139960402),
(0.7549248454386245, 0.3578385951330404),
(0.7377152307243756, 0.3247991432053158)]

- From the output, the gradient descent algorithm reaches its maximum effectiveness through matching choices of α parameter along with setting the right initial value for x. A large value of α in the algorithm can lead to minimum overshooting that creates an unstable search direction toward or away from the minimum. A small α value results in slower convergence because it needs additional iteration steps to reach a suitable proximity around the minimum.

- The value chosen for initial x impacts both the moving direction of the algorithm along with its converging patterns. The convergence process of $y(x) = x^4$ functions will be fast right after initiation near the global minimum because the derivative approaches zero near zero which decreases overshooting risk.

Part B (ii) - Behaviour of Gradient Descent with $\alpha = 0.1$ and $x_{initial} = 1.0$

Steps:

- The gradient descent algorithm was initialized with $x_{initial} = 1.0$ and a fixed step size α=0.1
- The function $y(x) = x^4$ and its derivative $\frac{dy}{dx} = 4x^3$ were used to calculate the gradient.
- In each iteration, x was updated using the formula:

$$x_{new} = x_{old} - \alpha \cdot (4x^3{}_{old})$$

- The updated x was used to calculate $y(x) = x^4$, which tracked the function value at each step.

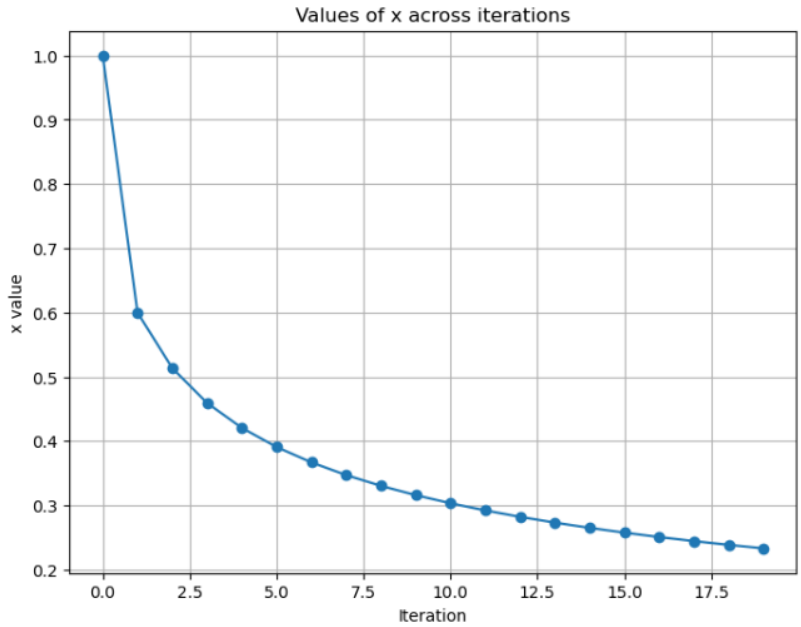

Figure 3, Values of x across the iteration of gradient descent

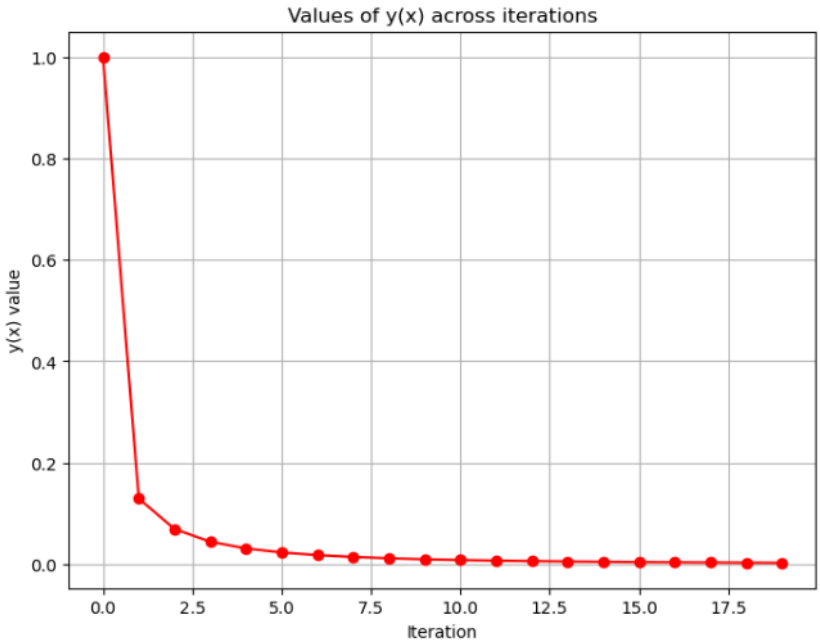

Figure 4, Values of $y(x)$ across iterations of gradient descent

- From the figure 3 & 4, it shows the success of the gradient descent algorithm in minimizing the function $y(x) = x^4$. Following is observed from the figure:
- Under the provided conditions $x_{initial} = 1.0$, $\alpha = 0.1$) the gradient descent algorithm performs efficiently for minimizing $y(x) = x^4$.
- The smooth and efficient convergence of the algorithm leads to the x=0 global minimum as both x and $y(x)$ behave according to their expected patterns during the iteration process.
- Using a step size value of 0.1 the optimization maintains fast convergence rates while preventing the occurrence of both overshooting conditions and divergence.

Part B (iii) - Parameter Variability and Convergence Analysis

- A range of initial values for x was chosen $\{-1.5, -1.0, 0.0, 0.5, 1.0, 1.5\}$.
- The initial values investigate the link between starting position and both gradient descent trajectory and its convergence duration.
- The testing procedure included assessing step sizes ranging from 0.01 to 0.2 in the following sequence (0.01,0.05,0.1,0.2)
- The stepped iterations display diverse outcomes when they converge as they can do so quickly or slowly or exemplify unstable behaviour patterns.
- Beginning from each pair of $x_{initial}$ and α the gradient descent algorithm was executed for identical numbers of iterations such as 50.
- The procedure finished by documenting the end values of x for evaluating convergence results.

Insights:

- The step size α determines the effectiveness of gradient descent with the initial value x. smaller step sizes promote stability while increasing the step size through appropriate selection can speed up the convergence.
- The value of α selection must reflect gradient magnitude and function curvature behaviour for preventing excessive movement or becoming unstable.

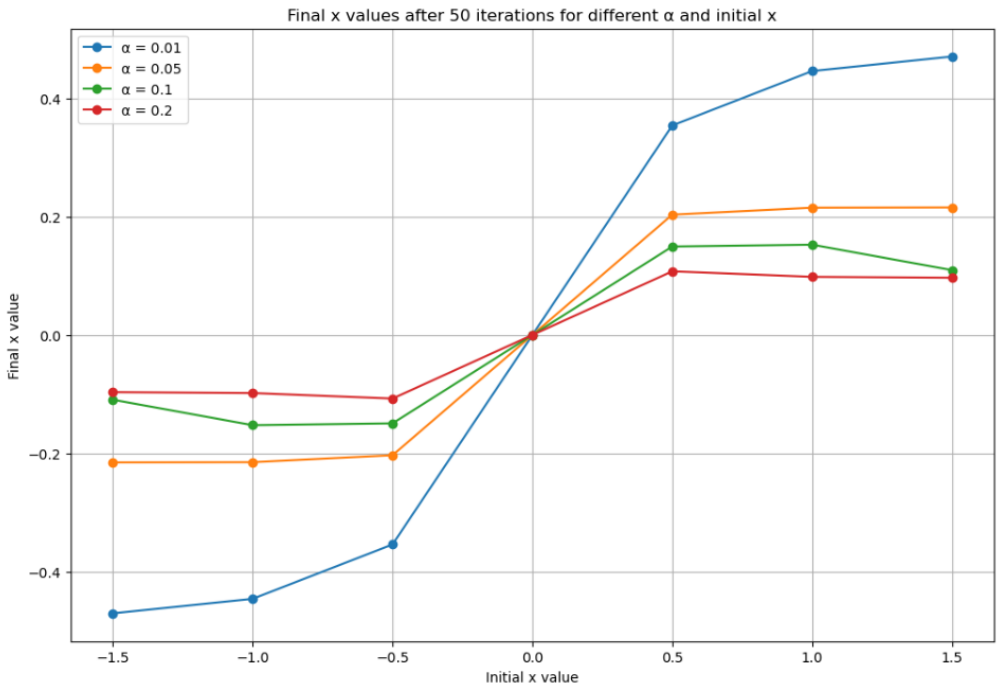

Figure 5, Final x values after 50 iterations for different α and initial x

Analysis:

Form the figure 5, the blue line is the small step size (0.01), orange line is the medium small step size (0.05), green line is the moderate step size (0.1), red line is large step size (0.2).

- The small step size leads to slow but steady progress toward the minimum point at x=0.When the initial x is far from zero, the final value after 50 iterations is still relatively far from the minimum, showing slow convergence.
- Medium step size improves the rate of convergence. For most starting points, the final x values are closer to zero compared to the blue line. The updates are still stable and do not overshoot the minimum.
- In moderate step size, the green line shows faster convergence. For most initial values of x, the final point is close to the minimum. This step size balances convergence speed and stability, working well across a range of starting points.
- The large step size results in instability for initial values farther from zero. The algorithm overshoots the minimum or oscillates around it, leading to less accurate final values. For starting points close to zero, the convergence is fast and accurate, but this step size struggles with stability for larger initial values.

Part C (i) - Convergence Analysis for $y(x) = \gamma x^2$

To analyse the changing parameter that affects the convergence behaviour of gradient descent for the function while keeping the step size constant.

Steps:

- This function $y(x) = \gamma x^2$ serves as the objective while its gradient operates as $2\gamma x$. The function maintains a quadratic structure with symmetrical properties which reaches its global minimum at point $x = 0$. The examination of convergence behaviour required multiple γ values ($\gamma = 0.1,1,10$) to understand how function scaling operates. The numerical process utilized a step size of ($\alpha = 0.1$) along with an initial condition of $x = 1.0$.
- Each trial of gradient descent operated for 50 iterations across the different γ values. Each gradient update for x used the term $2\gamma x$ that points directly to the scaling factor γ.

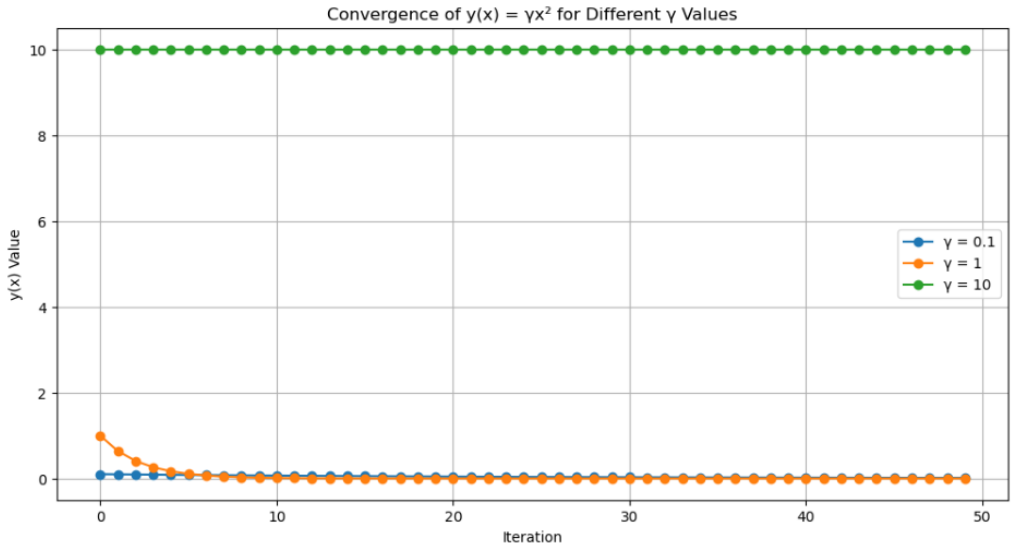

Figure 6, Convergence $of\ y(x) = \gamma x^2$ for different values

From the figure 6, the following are the observations made:

- For low γ (0.1), slow convergence occurred due to the small gradient which generated minimal changes to x values. The process of reaching the minimum occurred through gradual movement.

- For moderate γ (1), the algorithm converged more rapidly as it steadily moved toward the minimum. The efficient updating was made possible through the correct balance of gradient magnitude and step size.
- For high γ (10), the fastest convergence occurred during testing because the large gradient value produced bigger updates. Precaution needed to exist when determining the step size because excessive adjustments could produce overshooting.

Part C (ii) - Convergence Analysis for $y(x) = \gamma \mid x \mid$

To analyse the changing parameter that affects the convergence behaviour of gradient descent for the function while keeping the step size constant. The following are the steps in involved:

- The function $y(x) = \gamma \mid x \mid$ is non-differentiable at $x = 0$, so the subgradient method was used. The subgradient was defined as $\gamma \cdot sign(x)$ which is $\gamma \; for \; x > 0 \; and \; -\gamma \; for \; x < 0$.
- Different values of γ (0.1,1,10) were tested. A fixed step size (α=0.1) and initial x value 1.0 were used.
- Gradient descent was run for 50 iterations for each γ. At each iteration, x was updated based on the sub gradient which scales the γ

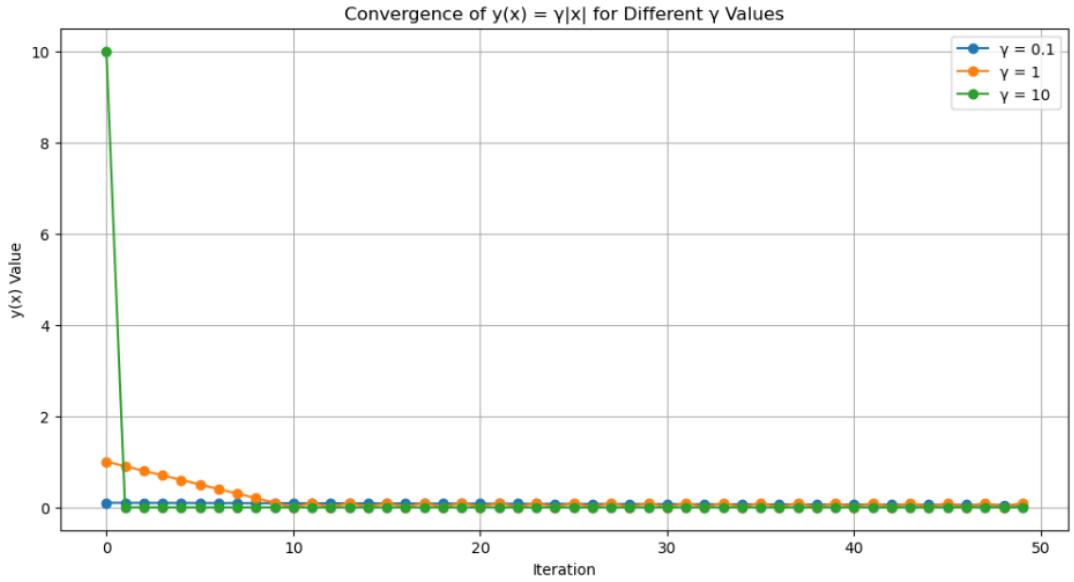

Figure 7, Convergence $y(x) = \gamma \mid x \mid$ for different values

- For low γ (0.1), the slow convergence rate came from small sub gradient values which produces minimal updates to x. Many iterations were needed by the algorithm before it arrived at zero values.
- For moderate γ (1), convergence was steady and efficient. The sub gradient generated useful updates that prevented the algorithm from overpassing the minimum value**.**
- For high γ (10), a quick convergence occurred due to the substantial sub gradient that produced substantial adjustments to the x variable. Large steps to adjust the initial low values are effective but could produce unstable results from x values away from zero.

From both Figure 6 & 7, it infers the following observations:

- The gradient (or sub gradient) becomes more powerful when γ increases thus resulting in enhanced performance of each iteration.
- Large γ values produce instability problems and overshooting effects when step size adjustments are inadequate.
- Small values of γ maintain stability however they lead to slower movement toward the minimum.
- The value of γ determines the speed-stability ratio for optimization progress while also taking into account problem requirements for selection.

APPENDIX

```
# ## Part A (i)
# %%
import sympy as sp

# Define the symbolic variable
x = sp.symbols('x')

# Define the function y(x) = x^4
y = x**4
# Compute the derivative dy/dx
dy_dx = sp.diff(y, x)

# Display the function and its derivative
print("Function: y =", y)
print("Derivative: dy/dx =", dy_dx)

# %% [markdown]
# ## Part A (ii)

# %%
import numpy as np
import matplotlib.pyplot as plt

# Define the range of x values
x_values = np.linspace(-2, 2, 400)

# Calculate exact derivatives using the expression derived
exact_derivatives = 4 * x_values**3

# Calculate finite difference derivatives
delta = 0.01
finite_diff_derivatives = (x_values**4 + delta - (x_values - delta)**4) / (2 * delta)

# Plotting both sets of derivatives
plt.figure(figsize=(10, 6))
plt.plot(x_values, exact_derivatives, label='Exact Derivative $4x^3$', color='blue')
plt.plot(x_values, finite_diff_derivatives, label='Finite Difference Approximation', color='red', linestyle='--')
plt.title('Comparison of Exact and Finite Difference Derivatives')
plt.xlabel('x')
plt.ylabel('Derivative')
plt.legend()
plt.grid(True)
plt.show()

# ## Part A (iii)

# %%
# Define a range of delta values
delta_values = np.array([0.001, 0.01, 0.1, 1])

# Prepare the plot
plt.figure(figsize=(12, 8))

# Compute and plot the errors for each delta
for delta in delta_values:
```

```
        finite_diff_derivatives_delta = (x_values**4 + delta - (x_values - delta)**4) / (2 * delta)
        error = np.abs(finite_diff_derivatives_delta - exact_derivatives)
        plt.plot(x_values, error, label=f'Error for δ = {delta}')

    plt.title('Error in Finite Difference Derivatives vs. Exact Derivatives for Various δ')
    plt.xlabel('x')
    plt.ylabel('Absolute Error')
    plt.legend()
    plt.grid(True)
    plt.yscale('log')  # Log scale for better visibility of errors across scales
    plt.show()

    # ### part B (i)

    # %%
    def gradient_descent(x_initial, alpha, num_iterations):
        x = x_initial
        history = []  # To store the history of x and y values

        # Function y = x^4 and its derivative
        y = lambda x: x**4
        dy_dx = lambda x: 4 * x**3

        for _ in range(num_iterations):
            current_y = y(x)
            gradient = dy_dx(x)
            x = x - alpha * gradient  # Update x by taking a step in the direction of the steepest descent
            history.append((x, current_y))
        return history
    # Parameters
    x_initial = 1.0
    alpha = 0.01  # Step size
    num_iterations = 50
    # Run gradient descent
    history = gradient_descent(x_initial, alpha, num_iterations)
    # Display the first few steps
    history[:10]  # Show only the first 10 iterations for brevity
    # ## Part B (ii)
def gradient_descent_plot(x_initial, alpha, num_iterations):
        x = x_initial
        x_history = []  # To store the history of x values
        y_history = []  # To store the history of y values
        # Function y = x^4 and its derivative
        y = lambda x: x**4
        dy_dx = lambda x: 4 * x**3
        for _ in range(num_iterations):
            current_y = y(x)
            x_history.append(x)
            y_history.append(current_y)
            gradient = dy_dx(x)
            x = x - alpha * gradient  # Update x by taking a step in the direction of the steepest descent
        return x_history, y_history
    # Parameters for this part
    alpha = 0.1  # Step size
    num_iterations = 20  # Using fewer iterations for clarity in visualization

    # Run gradient descent
```

```
x_history, y_history = gradient_descent_plot(x_initial, alpha, num_iterations)

# Plot for the values of x across iterations
plt.figure(figsize=(8, 6))
plt.plot(x_history, marker='o')
plt.title('Values of x across iterations')
plt.xlabel('Iteration')
plt.ylabel('x value')
plt.grid(True)
plt.show()
# Plot for the values of y(x) across iterations
plt.figure(figsize=(8, 6))
plt.plot(y_history, marker='o', color='r')
plt.title('Values of y(x) across iterations')
plt.xlabel('Iteration')
plt.ylabel('y(x) value')
plt.grid(True)
plt.show()
# ### part b (iii)
# Define a range of initial x values and step sizes
initial_x_values = np.linspace(-1.5, 1.5, 7)  # A range of initial values around zero
step_sizes = [0.01, 0.05, 0.1, 0.2]  # Different step sizes
num_iterations = 50  # Consistent iteration count for comparison
# Prepare to store results for plotting
results = []
# Run gradient descent for each combination of initial x and alpha
for x_initial in initial_x_values:
    for alpha in step_sizes:
        x_history, _ = gradient_descent_plot(x_initial, alpha, num_iterations)
        results.append((x_initial, alpha, x_history[-1]))
# Plotting the results
plt.figure(figsize=(12, 8))
for i, alpha in enumerate(step_sizes):
    x_final_values = [result[2] for result in results if result[1] == alpha]
    plt.plot(initial_x_values, x_final_values, marker='o', label=f'α = {alpha}')
plt.title('Final x values after 50 iterations for different α and initial x')
plt.xlabel('Initial x value')
plt.ylabel('Final x value')
plt.legend()
plt.grid(True)
plt.show()

# ### part c (i)
def gradient_descent_gamma(x_initial, alpha, gamma, num_iterations):
    x = x_initial
    x_history = []  # To store the history of x values
    y_history = []  # To store the history of y values
    # Function y = γ * x^2 and its derivative
    y = lambda x: gamma * x**2
    dy_dx = lambda x: 2 * gamma * x
    for _ in range(num_iterations):
        current_y = y(x)
        x_history.append(x)
        y_history.append(current_y)
        gradient = dy_dx(x)
        x = x - alpha * gradient  # Update x by taking a step in the direction of the steepest descent
    return x_history, y_history
```

```
# Parameters for this experiment
gamma_values = [0.1, 1, 10]  # Different gamma values
alpha = 0.1  # Fixed step size
num_iterations = 50
x_initial = 1.0  # Fixed initial x value
# Run gradient descent for different gamma values and plot the results
plt.figure(figsize=(12, 6))
for gamma in gamma_values:
    _, y_history = gradient_descent_gamma(x_initial, alpha, gamma, num_iterations)
    plt.plot(y_history, marker='o', label=f'γ = {gamma}')
plt.title('Convergence of y(x) = γx² for Different γ Values')
plt.xlabel('Iteration')
plt.ylabel('y(x) Value')
plt.legend()
plt.grid(True)
plt.show()
# ### part c (ii)
def gradient_descent_abs_gamma(x_initial, alpha, gamma, num_iterations):
    x = x_initial
    x_history = []  # To store the history of x values
    y_history = []  # To store the history of y values
    # Function y = γ * |x| and its subgradient
    y = lambda x: gamma * abs(x)
    dy_dx = lambda x: gamma * np.sign(x) if x != 0 else 0

    for _ in range(num_iterations):
        current_y = y(x)
        x_history.append(x)
        y_history.append(current_y)
        gradient = dy_dx(x)
        x = x - alpha * gradient  # Update x by taking a step in the direction of the steepest descent
    return x_history, y_history
# Parameters for this experiment
gamma_values = [0.1, 1, 10]  # Different gamma values
alpha = 0.1  # Fixed step size
num_iterations = 50
x_initial = 1.0  # Fixed initial x value
# Run gradient descent for different gamma values and plot the results
plt.figure(figsize=(12, 6))
for gamma in gamma_values:
    _, y_history = gradient_descent_abs_gamma(x_initial, alpha, gamma, num_iterations)
    plt.plot(y_history, marker='o', label=f'γ = {gamma}')

plt.title('Convergence of y(x) = γ|x| for Different γ Values')
plt.xlabel('Iteration')
plt.ylabel('y(x) Value')
plt.legend()
plt.grid(True)
plt.show()
```