Question (a)

Part (i)

For the function $y(x) = x^4$, one can obtain an expression for the derivative dx/dy by initializing the symbol x using the *SymPy* package, creating a function $y(x) = x^4$ as y=x**4 and finding the derivative dy/dx of $y(x)$ using `sp.diff()` with y and x as shown:

```
import sympy as sp
def i():
    x = sp.symbols('x')
    y = x ** 4
    derivative = sp.diff(y, x)
    return derivative
```

Part (ii)

The range of x values I chose was $-3 \leq x \leq 3$. I plotted the derivative from part (i) using the plot function from `sympy.plotting`. On the same plot, I also estimated the derivative value from (i) using a finite difference with the equation we learned from lectures $f(x) - f(x^{'})/\delta$, which in this case is $(y(x+\delta) - y(x))/\delta$ using the code below:

```
estimated_derivative = ((x + perturbation)**4 - x**4) / perturbation
```

where I used perturbation $\delta = 0.01$ with the `ii()` function in `part_a.py` to get the resulting plot with the real derivative and estimated derivative for the expression in part (i) which is shown below:

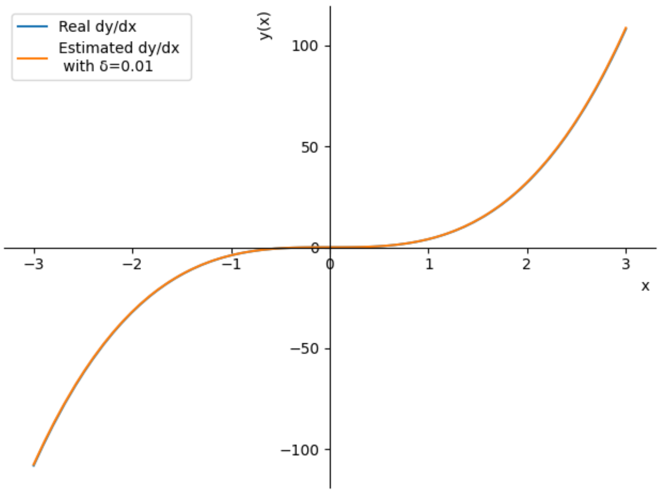

Figure 1: Derivative and estimated derivative (using $\delta = 0.01$) from part (i)

The plot shows that the estimated derivative using finite differences is extremely similar to the real derivative in the range $-3 \leq x \leq 3$. The real derivative in blue is hidden by the almost identical estimated derivative in orange that overlaps it. Therefore, using a perturbation of $\delta = 0.01$ is accurate, and this aligns with the teachings in lectures that "Perturbation δ needs to be small e.g. 0.01 or less".

Part (iii)

I varied the size of the perturbation δ of x using the range [0.001, 0.01, 0.1, 1] and plotted the difference between the derivative estimate and the real derivative values calculated with *SymPy* for $-3 \leq x \leq 3$ to get this plot:

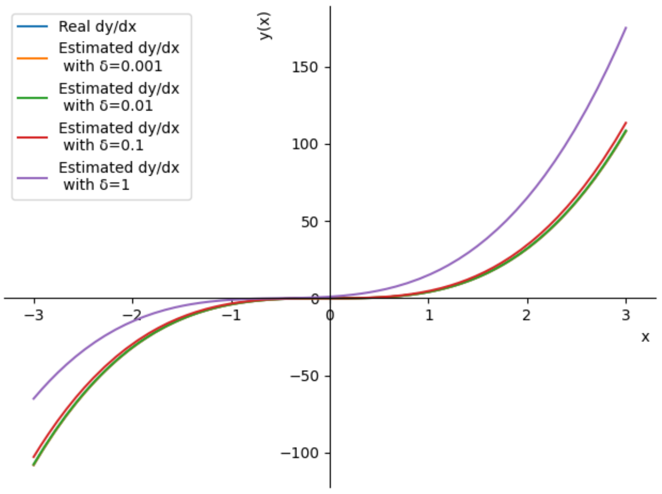

Figure 2: Derivative and estimated derivative (using $\delta = [0.001, 0.01, 0.1, 1]$) from part (i)

From this plot, we can analyze that the estimated derivative is almost identical to the real derivative when $\delta \leq 0.01$ (blue line and orange line are hidden since the green line is on top of the two). The estimated derivative starts to get inaccurate starting from $\delta = 0.1$ (red line). At $\delta = 1$, the estimated derivative is extremely incorrect. Thus, we can conclude that higher values for δ makes the accuracy of the estimated derivative go down because we increase the difference between $y(x + \delta)$ and $y(x)$. Smaller values for δ imply smaller differences and hence more accurate estimations.

Question (b)

Part (i)

In `part_b.py`, the function `gradient_descent()` takes the function $y(x)$, the starting value of x, the number of gradient descent iterations to run, step size α and the derivative.

```
def gradient_descent(function_y, initial_x_value, number_of_iterations,
step_size, derivative):
    y_array = [function_y(initial_x_value)]
    x_array = [initial_x_value]
    for _ in range(number_of_iterations):
        step = step_size * derivative(initial_x_value)
        initial_x_value = initial_x_value - step
        y_array.append(function_y(initial_x_value))
        x_array.append(initial_x_value)
    return x_array, y_array
```

The code above works by iterating through the number of gradient descent iterations specified and decreasing x by a step in each iteration. Each step is calculated by multiplying α (step size) with the derivative of x. The updated values of x and subsequently $y(x)$ are appended to initial array representations of them for plotting. By doing so, gradient descent is implemented.

Part (ii)

Using a starting x value of $x_0 = 1$ and step size of $\alpha = 0.1$, I ran the `gradient_descent()` function with the function $y = x^4$. For 100 gradient descent iterations, I plotted how x and $y(x)$ varied by passing in lambda functions for $y(x) = x^4$ and $dy/dx = 4x^3$.

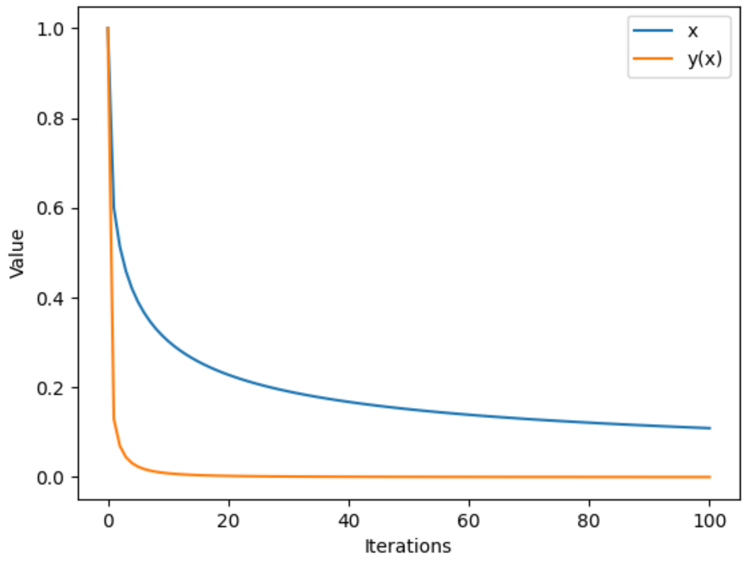

Figure 3: x and $y(x)$ varying with 100 gradient descent iterations

From this plot, we can see that the value of x (blue line) decreases slowly. This is because as x decreases, $\alpha * 4x^3$ decreases at a faster rate exponentially, which is the amount that is subtracted from x at each iteration. This results in x decreasing more slowly over time. When we are not bound by this behavior, we can see that the value of $y(x)$ decreases quickly and converges at a minimum before 10 iterations (low convergence time).

Part (iii)

With the range of initial x_0 values [0.1, 0.5, 1] and step sizes [0.001, 0.01, 0.1], I repeated the `gradient_descent()` function to see how x and $y(x)$ vary over 100 iterations for each value of α. Using $\alpha = 0.001$, each iteration causes very little change in x. For $x_0 = 1$ and $x_0 = 0.5$, 100 iterations is not enough for them to converge. Using even smaller values such as $x_0 = 0.01$ causes $y(x)$ to be so small that when plotted, becomes a line similar to $y = 0$.

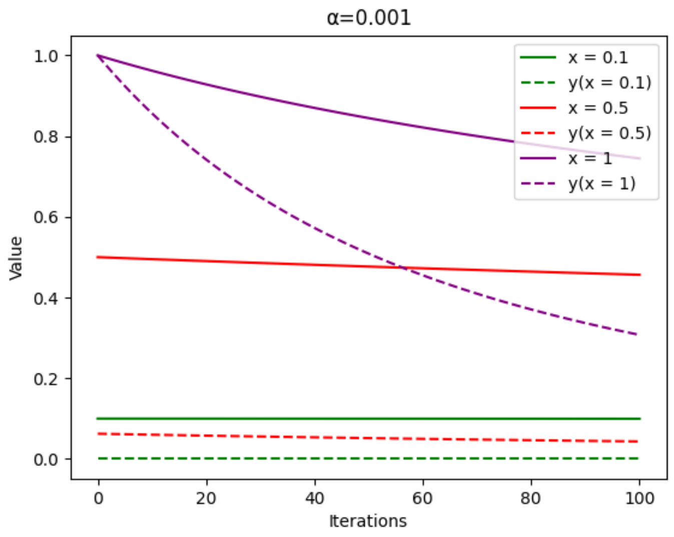

Figure 4: x (solid line) and $y(x)$ (dashed line) varying with 100 gradient descent iterations using x values [0.1, 0.5, 1] and $\alpha = 0.001$

Using $\alpha = 0.01$, each iteration causes very little change in x once again, with $x_0 = 0.1$ and $x_0 = 0.5$ taking around 85 iterations to seemingly converge at a minimum. $x_0 = 0.1$ still causes $y(x)$ to be so small that it becomes close to a line $y = 0$ when plotted. $x_0 = 1$ takes around 100 iterations to begin to converge.

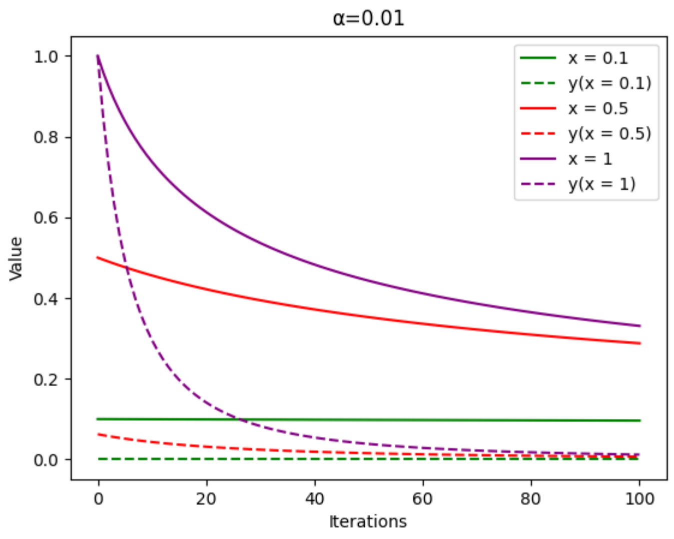

Figure 5: x (solid line) and $y(x)$ (dashed line) varying with 100 gradient descent iterations using x values [0.1, 0.5, 1] and $\alpha = 0.01$

Using $\alpha = 0.1$, each iteration causes fast convergence that are accurate for all values of x_0. From this observation, it is evident that x is decreased too slowly during each iteration if α is too small; the most efficient value of α here is 0.1 since the step size is bigger and we are taking bigger leaps each iteration, hence decreasing the convergence time. However for more complicated examples, larger step sizes may cause us to overshoot the minimum (and thus effecting the accuracy). From these experiments, the initial value of x does not have a large impact on the time taken for convergence or accuracy using this range of x values and step sizes. This minimal impact of x_0, at least with these ranges, can be explained by bigger values of x causing larger derivatives at each iteration.

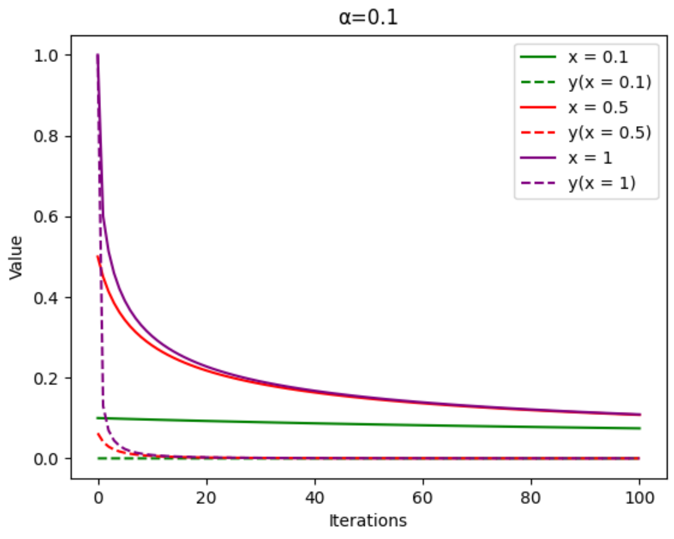

Figure 6: x (solid line) and $y(x)$ (dashed line) varying with 100 gradient descent iterations using x values [0.1, 0.5, 1] and $\alpha = 0.1$

Question (c)

Part (i)

Using function $y(x) = \gamma x^2$, I plotted 100 iterations of gradient descent using a constant step size of $\alpha = 0.1$, starting x value of $x_0 = 1$ and values [0.1, 1, 2] for parameter γ.

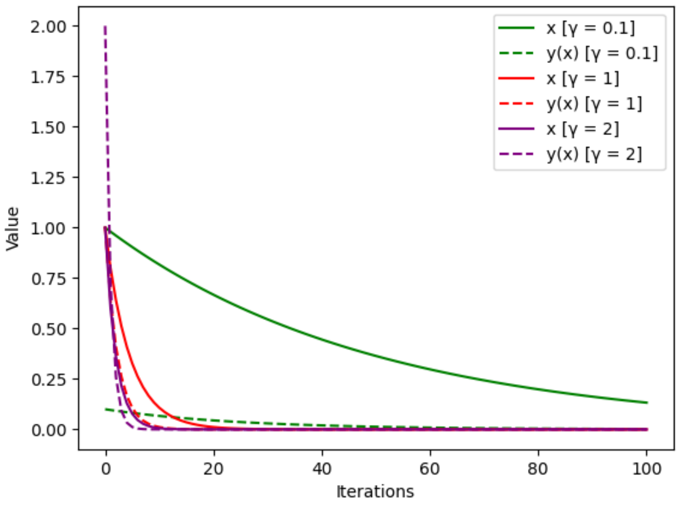

Figure 7: x (solid line) and $y(x)$ (dashed line) varying with 100 gradient descent iterations using $x_0 = 1$ and $\alpha = 0.1$ for $y(x) = \gamma x^2$

From the plot above, we conclude that a bigger γ value converges quicker on the minimum. The derivative of $y(x)$ is $2\gamma x$. This derivative will be bigger when γ is bigger. Since we use a constant initial x value of $x_0 = 1$, x will be decreased by more at each iteration in gradient descent because of γ being larger, which explains the quicker convergence.

Part (ii)

Using function $y(x) = \gamma|x|$, I plotted 100 iterations of gradient descent using a constant step size of $\alpha = 0.1$, starting x value of $x_0 = 1$ and values [0.1, 1, 2] for parameter γ.

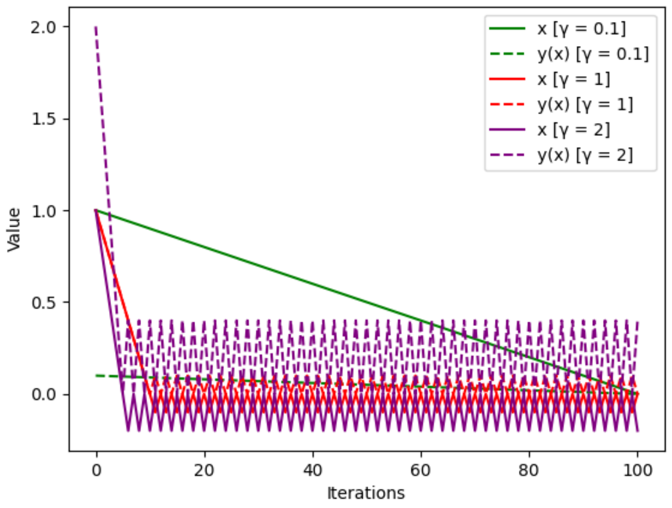

Figure 8: x (solid line) and $y(x)$ (dashed line) varying with 100 gradient descent iterations using $x_0 = 1$ and $\alpha = 0.1$ for $y(x) = \gamma x^2$

The derivative of $y(x)$ is $(\gamma x)/|x|$. From the plot above, larger values of γ decreases the value of $y(x)$ initially since $(x)/|x|$ in the derivative $(\gamma)(x)/|x|$ gives a positive integer when $x \geq 0$, which is multiplied by γ in $(\gamma)(x)/|x|$. This derivative (from multiplying a large value of γ with a positive integer from $(x)/|x|$ when $x \geq 0$) is subtracted from x at each interval in gradient descent. However, this behaviour stops after a certain point when $x < 0$, when it starts to oscillate close to the minimum of 0 without actually reaching it, stopping $y(x)$ from converging on the minimum. This phenomenon can be seen happening for bigger values of γ such as $\gamma = 1$ and $\gamma = 2$. However, $\gamma = 0.1$ didn't reach convergence yet (at 0) since we are multiplying the positive integer $(x)/|x|$ with a very small value of γ. Subsequently, we subtract a small amount from x, and this increases convergence time.

1 Appendix

1.1 Question (a) (i)

```
import sympy as sp

def i():
    # Initialize variable x
    x = sp.symbols('x')
    # Create function y(x)=x^4 using variable x
    y = x ** 4
    # Find the derivative of y(x) using the diff() function
    derivative = sp.diff(y, x)
    return derivative

if __name__ == '__main__':
    dy_over_dx = i()
    print("The derivative of y=x^4 is: " + str(dy_over_dx))
```

1.2 Question (a) (ii)

```
import sympy.plotting as plt

def ii(perturbation, x, derivative):
    # Range of x values
    x_range = (x, -3, 3)
    # First, plot the actual derivative
    plot = plt.plot(derivative, x_range, label='Real dy/dx', xlabel=' ' *
    135 + 'x', ylabel=' ' * 95 + 'y(x)', show=False)
    # Second, calculate the estimated derivative using a finite difference
    with the specified perturbation
    estimated_derivative = ((x + perturbation) ** 4 - x ** 4) /
    perturbation
    # Plot the estimated derivative and append it on top of the first plot
    plot.extend(plt.plot(estimated_derivative, x_range, label=f'Estimated
    dy/dx \n with δ={perturbation}', show=False))
    plot.legend = True
    plot.show()
```

```
if __name__ == '__main__':
    dy_over_dx = i()
    ii(0.01, sp.symbols('x'), dy_over_dx)
```

1.3 Question (a) (iii)

```
import sympy.plotting as plt

def iii(perturbation_values, x, derivative):
    # Range of x values
    x_range = (x, -3, 3)
    # First, plot the actual derivative
    plot = plt.plot(derivative, x_range, label='Real dy/dx', xlabel=' ' *
    135 + 'x', ylabel=' ' * 95 + 'y(x)', show=False)
    for perturbation in perturbation_values:
        # Second, plot the estimated derivative with the specified
        perturbation
        estimated_derivative = ((x + perturbation) ** 4 - x ** 4) /
        perturbation
        plot.extend(plt.plot(estimated_derivative, x_range, show=False,
        label=f'Estimated dy/dx \n with δ={perturbation}'))
    plot.legend = True
    plot.show()

if __name__ == '__main__':
    dy_over_dx = i()
    iii([0.001, 0.01, 0.1, 1], sp.symbols('x'), dy_over_dx)
```

1.4 Question (b) (i)

```
import matplotlib.pyplot as plt

def gradient_descent(function_y, initial_x_value, number_of_iterations,
step_size, derivative):
    y_array = [function_y(initial_x_value)]
    x_array = [initial_x_value]
    for _ in range(number_of_iterations):
        step = step_size * derivative(initial_x_value)
        initial_x_value = initial_x_value - step
        y_array.append(function_y(initial_x_value))
        x_array.append(initial_x_value)
    return x_array, y_array
```

1.5 Question (b) (ii)

```
import matplotlib.pyplot as plt

def ii():
    x_array, y_array = gradient_descent(function_y=lambda x: x ** 4,
    initial_x_value=1, number_of_iterations=100, step_size=0.1,
    derivative=lambda x: 4 * (x ** 3))
    iterations = list(range(101))
    plt.plot(iterations, x_array)
    plt.plot(iterations, y_array)
    plt.ylabel('Value')
    plt.xlabel('Iterations')
    plt.legend(['x', 'y(x)'])
    plt.show()
```

1.6 Question (b) (iii)

```
import matplotlib.pyplot as plt

def iii():
    iterations = list(range(101))
    for step_size in [0.001, 0.01, 0.1]:
        for x, color in zip([0.1, 0.5, 1], ['green', 'red', 'purple']):
            x_array, y_array = gradient_descent(function_y=lambda x: x **
            4, initial_x_value=x, number_of_iterations=100,
            step_size=step_size, derivative=lambda x: 4 * (x ** 3))
            plt.plot(iterations, x_array, color=color)
            plt.plot(iterations, y_array, color=color, linestyle='dashed')
        plt.legend(['x = 0.1', 'y(x = 0.1)', 'x = 0.5', 'y(x = 0.5)', 'x =
        1', 'y(x = 1)'])
        plt.ylabel('Value')
        plt.xlabel(f'Iterations')
        plt.title(f'\u03B1={step_size}')
        plt.show()
```

1.7 Question (c) (i)

```
import matplotlib.pyplot as plt
import sympy as sp

def i():
    # Finding the derivative of γx^2 is 2γx
    x = sp.Symbol('x')
    print(sp.diff(sp.Symbol('gamma') * (x ** 2), x))

    # Subsequently, using γx^2 as function_y and 2γx as the derivative
    iterations = list(range(101))
    for gamma, color in zip([0.1, 1, 2], ['green', 'red', 'purple']):
        x_array, y_array = gradient_descent(function_y=lambda x: gamma *
        (x ** 2), initial_x_value=1, number_of_iterations=100,
        step_size=0.1, derivative=lambda x: 2 * gamma * x)
        plt.plot(iterations, x_array, color=color)
        plt.plot(iterations, y_array, color=color, linestyle='dashed')
    plt.legend(['x [γ = 0.1]', 'y(x) [γ = 0.1]', 'x [γ = 1]', 'y(x) [γ =
    1]', 'x [γ = 2]', 'y(x) [γ = 2]'])
    plt.ylabel('Value')
    plt.xlabel('Iterations')
    plt.show()
```

1.8 Question (c) (ii)

```
import matplotlib.pyplot as plt
import sympy as sp

def ii():
    # Finding the derivative of γ|x| is (gamma*x)/|x|
    x = sp.Symbol('x')
    print(sp.diff(sp.Symbol('gamma') * abs(x), x))

    # Subsequently, using γ|x| as function_y and (gamma*x)/|x| as the
    derivative
    iterations = list(range(101))
    for gamma, color in zip([0.1, 1, 2], ['green', 'red', 'purple']):
        x_array, y_array = gradient_descent(function_y=lambda x: gamma *
        abs(x), initial_x_value=1, number_of_iterations=100,
        step_size=0.1, derivative=lambda x: gamma * x / abs(x))
        plt.plot(iterations, x_array, color=color)
        plt.plot(iterations, y_array, color=color, linestyle='dashed')
    plt.legend(['x [γ = 0.1]', 'y(x) [γ = 0.1]', 'x [γ = 1]', 'y(x) [γ =
    1]', 'x [γ = 2]', 'y(x) [γ = 2]'])
    plt.ylabel('Value')
    plt.xlabel('Iterations')
    plt.show()
```