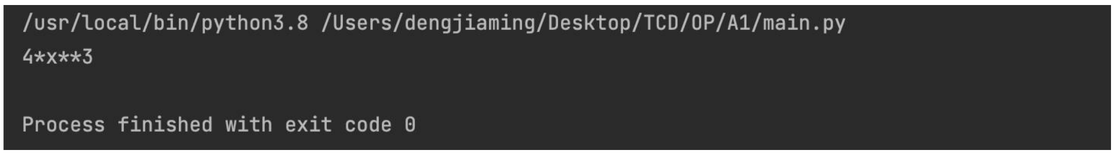
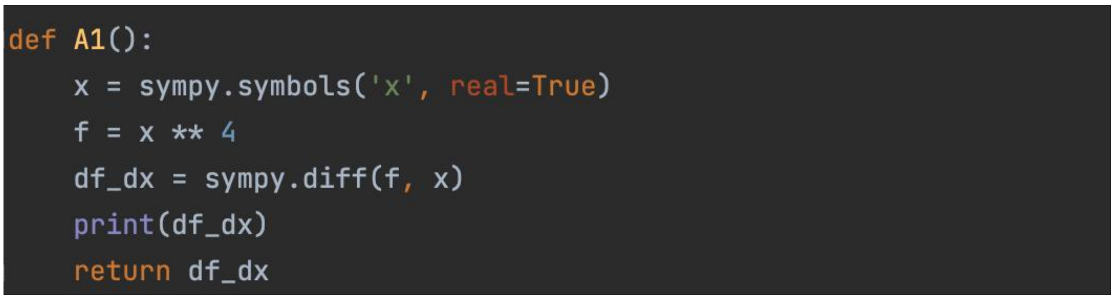
Optimisation Algorithms for Data Analysis Week 2 Assignment

Jiaming Deng   22302794

a (i) Firstly I used sympy to define the function required in the question, and next I used sympy's function to find the derivative.

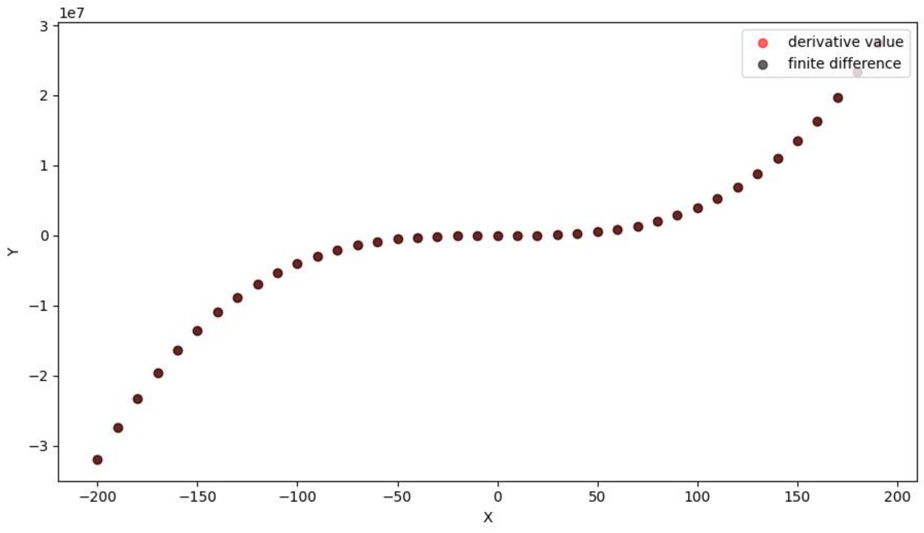


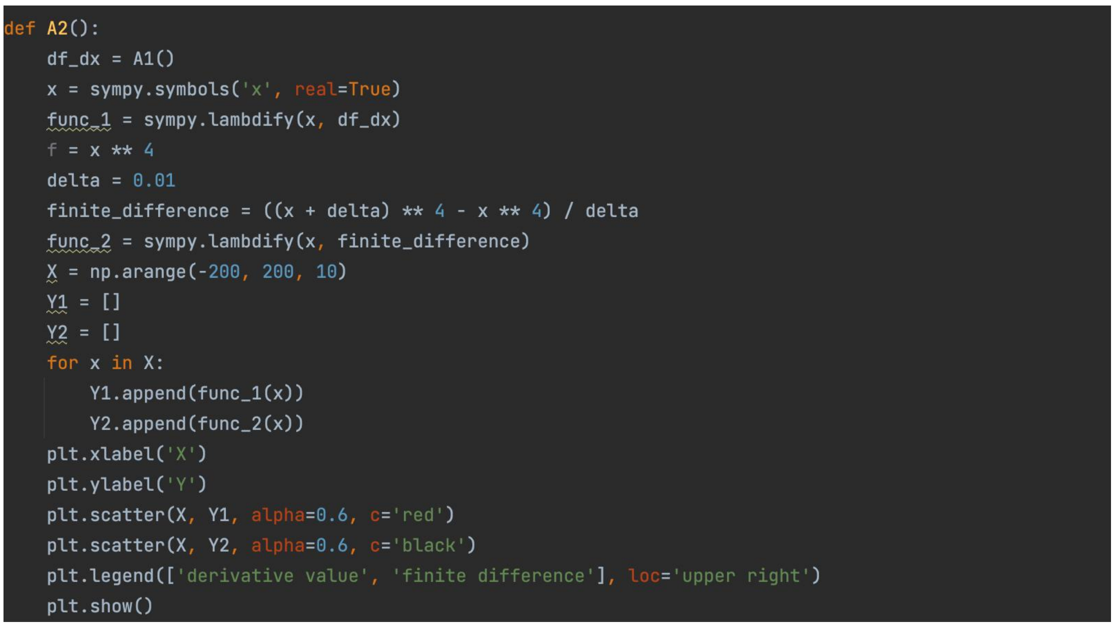
```
/usr/local/bin/python3.8 /Users/dengjiaming/Desktop/TCD/OP/A1/main.py
4*x**3

Process finished with exit code 0
```



```
def A1():
    x = sympy.symbols('x', real=True)
    f = x ** 4
    df_dx = sympy.diff(f, x)
    print(df_dx)
    return df_dx
```

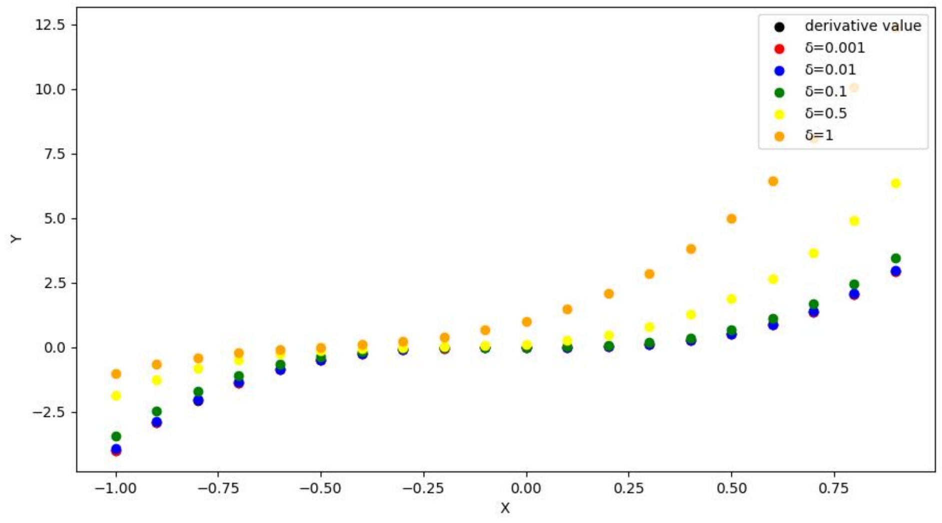
a (ii) Firstly I call the previous method to get derivative expression of the function. Next I get the finite difference by $\frac{f(x+\delta) - f(x)}{\delta}$. Finally I use numpy to generate a large range of points and plot them out. I find that they are basically equal.



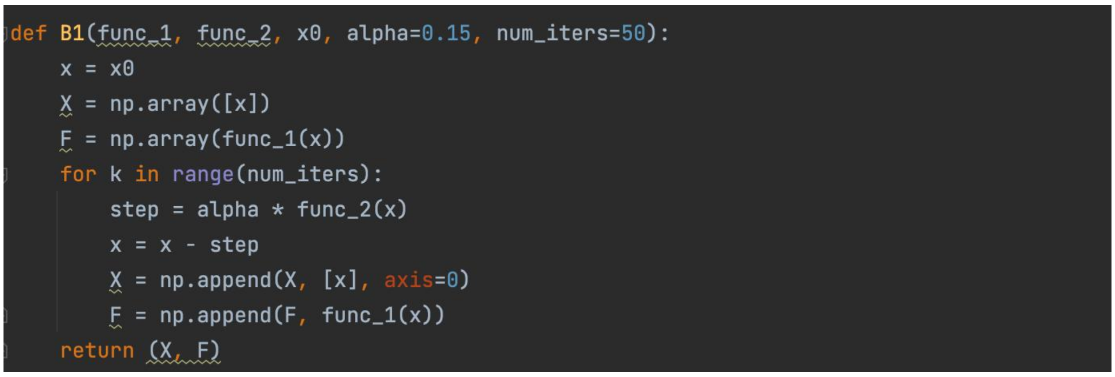


```python
def A2():
    df_dx = A1()
    x = sympy.symbols('x', real=True)
    func_1 = sympy.lambdify(x, df_dx)
    f = x ** 4
    delta = 0.01
    finite_difference = ((x + delta) ** 4 - x ** 4) / delta
    func_2 = sympy.lambdify(x, finite_difference)
    X = np.arange(-200, 200, 10)
    Y1 = []
    Y2 = []
    for x in X:
        Y1.append(func_1(x))
        Y2.append(func_2(x))
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.scatter(X, Y1, alpha=0.6, c='red')
    plt.scatter(X, Y2, alpha=0.6, c='black')
    plt.legend(['derivative value', 'finite difference'], loc='upper right')
    plt.show()
```

a (iii) The image is obtained by modifying the value of delta in the previous problem and plotting the points. I find that the larger the delta, the greater the difference between the finite difference estimate and the data obtained from the derivative. The smaller the delta, the closer the finite difference estimate is to the true result. This is because the finite difference estimate estimates the derivative of a curve at a point on the curve by taking the point on the curve and the point that differs by delta from the horizontal coordinate of that point and calculating the slope of the line through those two points. The derivative is defined as the slope of the line tangent to the curve at a point on the curve. Therefore the greater the delta, the greater the difference between the two points taken and the less accurate the derivative is.

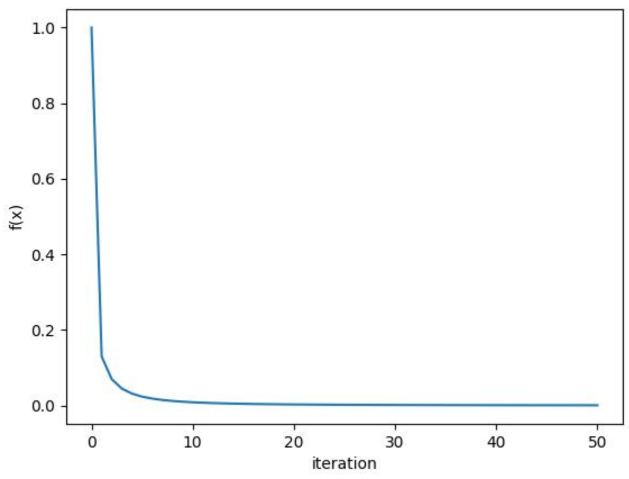


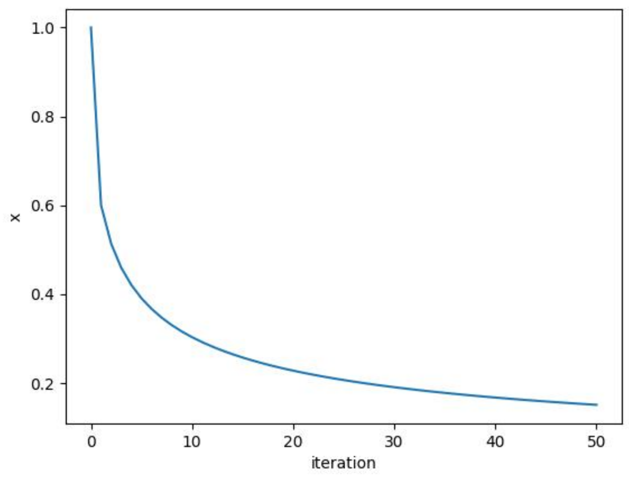
b(i) Firstly the inputs to the function are the expression, the derivative of the expression, the point at which gradient descent begins, a fixed step size alpha, and the number of iterations. Next, two arrays are created, X records the position of each x and F records the value of f(x) at the time. Finally X and F are returned.

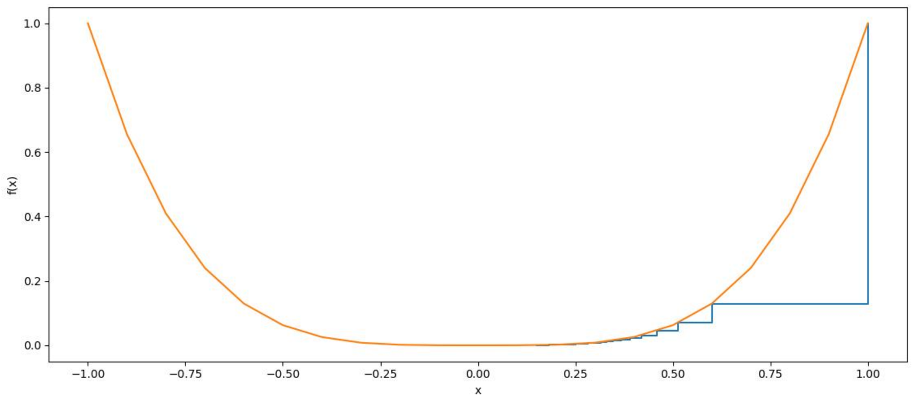


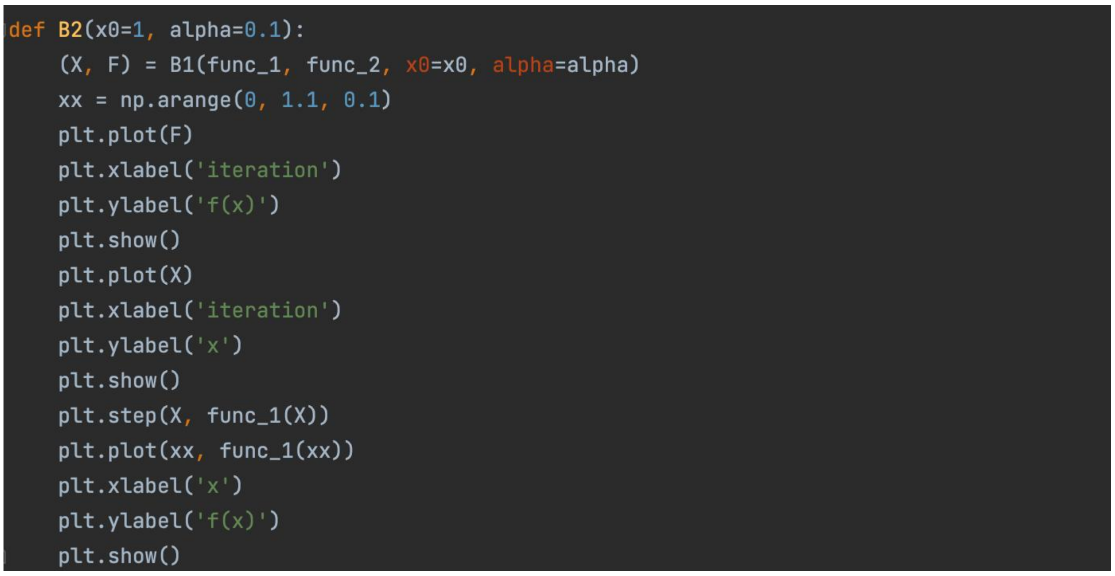
```python
def B1(func_1, func_2, x0, alpha=0.15, num_iters=50):
    x = x0
    X = np.array([x])
    F = np.array(func_1(x))
    for k in range(num_iters):
        step = alpha * func_2(x)
        x = x - step
        X = np.append(X, [x], axis=0)
        F = np.append(F, func_1(x))
    return (X, F)
```

b (ii) I initial value x = 0.1 and step size alpha = 0.1 and run my gradient descent function with the function y = x^4. The first and second graphs show the change in the values of f(x) and x as the number of iterations increases, respectively. The third graph plots the expressions corresponding to and plots the gradient descent on the same graph for easy comparison. The drop in f(x) is very large within the first 5 iterations of the gradient descent and is almost constant from about the 7th iteration. During the first 2 iterations, x decreases by half, and then the decrease gradually slows down. Thus the gradient decreases the most in the first 2 iterations and remains almost constant from about the 7th.









```
def B2(x0=1, alpha=0.1):
    (X, F) = B1(func_1, func_2, x0=x0, alpha=alpha)
    xx = np.arange(0, 1.1, 0.1)
    plt.plot(F)
    plt.xlabel('iteration')
    plt.ylabel('f(x)')
    plt.show()
    plt.plot(X)
    plt.xlabel('iteration')
    plt.ylabel('x')
    plt.show()
    plt.step(X, func_1(X))
    plt.plot(xx, func_1(xx))
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.show()
```

b (iii) Firstly I choose 0.01, 0.1, 0.2 and 0.5 as the values of alpha and obtain four plots as follows. I find that as alpha increases, the rate of gradient descent increases, because as alpha increases, x changes more with each iteration, as the expression $f(x) = x^4$, x decreases more significantly as it's larger. When alpha >= 0.5, the plot behaviors non-convergence because it's too large for the alpha.

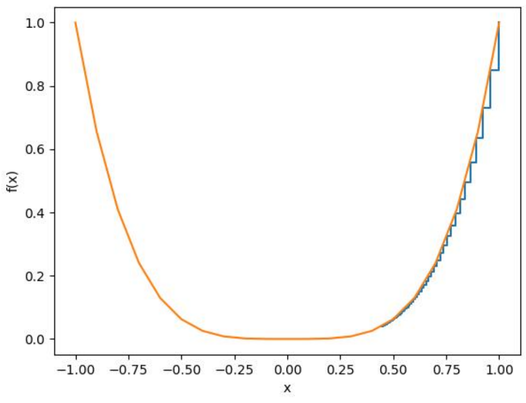


figure 1

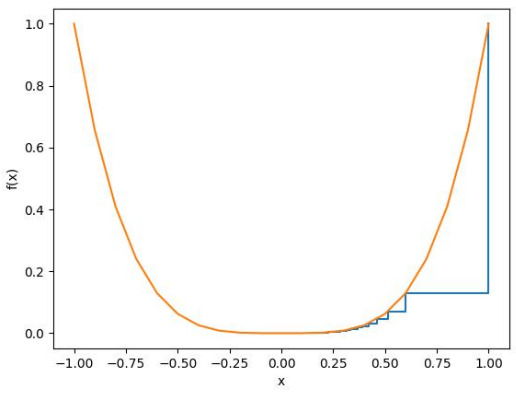


figure 2

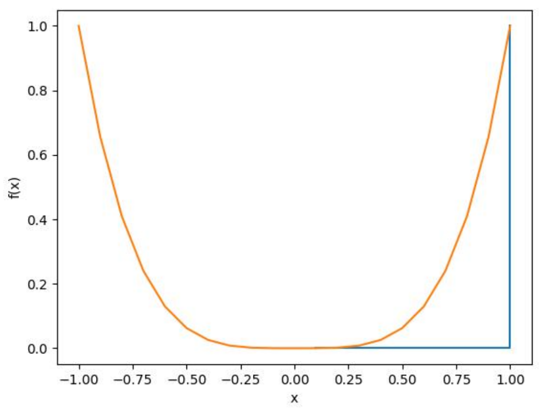


figure 3

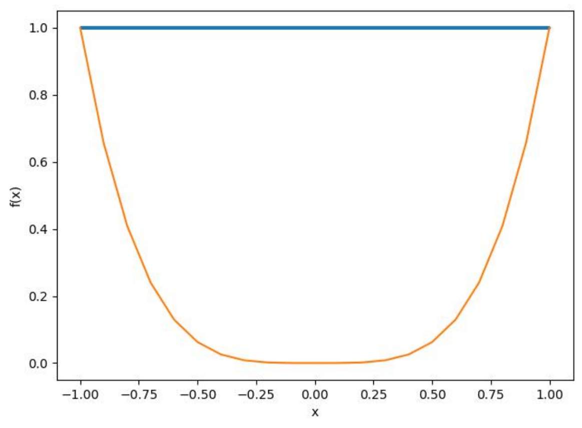


figure 4

Next, I choose 0.5, 0.6, 0.7, 0.8, 0.9 as the values of x and obtain five plots as below. As the x increase, the gradient has dropped faster. Because the derivative of an expression increases as x increases, the derivative is a parameter that reflects the rate of change of the function. Therefore the larger x is, the faster the gradient falls.

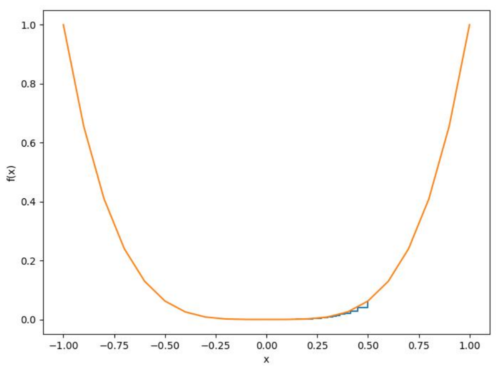


figure 1

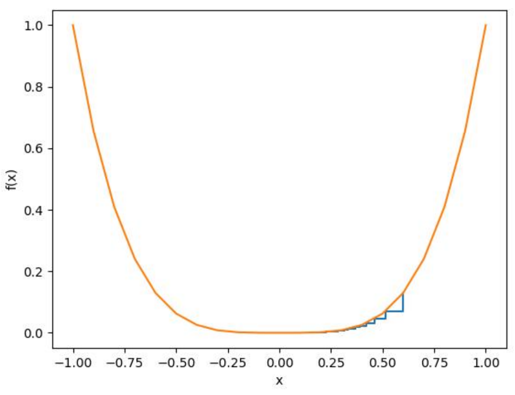


figure 2

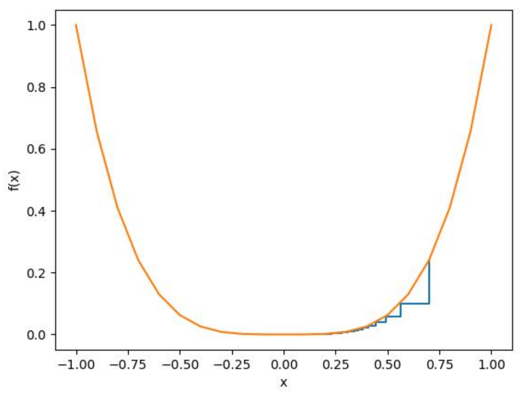


figure 3

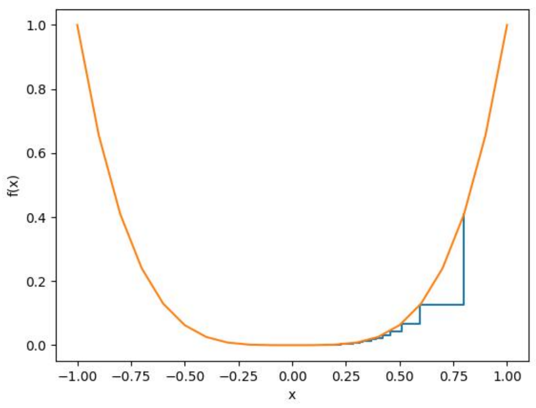


figure 4

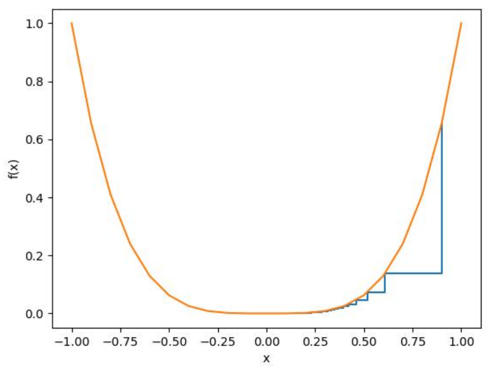
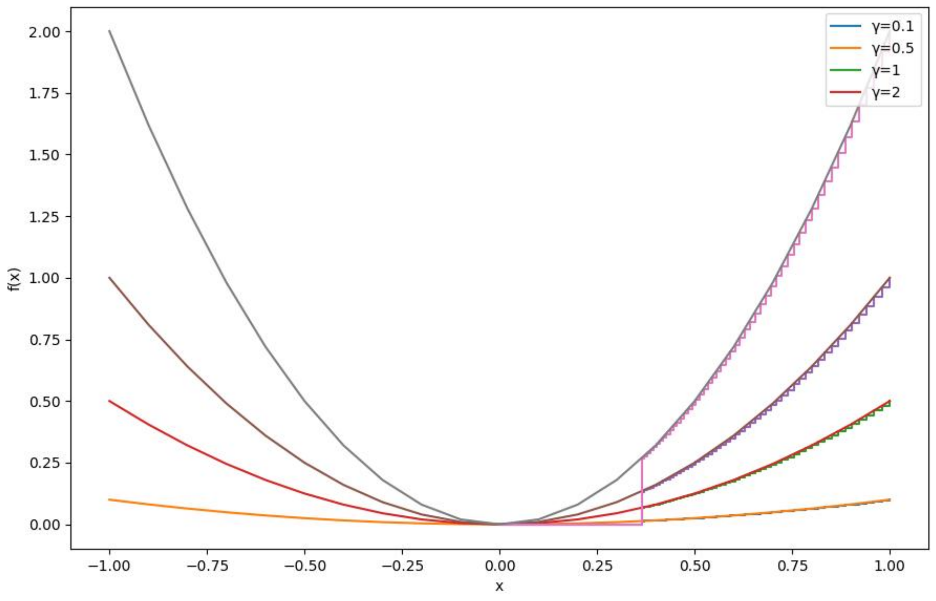
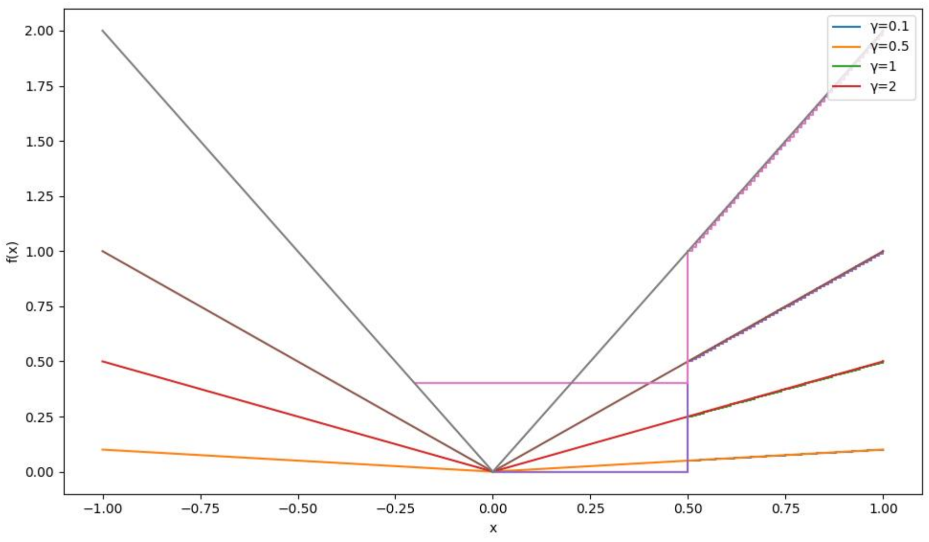


figure 5

c (i) Firstly I fix alpha = 0.1 and choose gamma to be 0.1, 0.5, 1, 2, respectively, to obtain the expression curve with gradient descent as follows. The larger gamma is, the convergence of the gradient descent more like to perform non-convergence because larger gamma means larger derivative.



c (ii) Firstly I fix alpha = 0.1 and choose gamma to be 0.1, 0.5, 1, 2, respectively, to obtain the expression curve with gradient descent as follows. The derivative of the express is purely depend by gamma, so the larger the gamma is, the convergence of the gradient descent more like to perform non-convergence.


γ=0.1
γ=0.5
γ=1
γ=2
f(x)
x
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

Appendix

```python
import sympy
import numpy as np
import matplotlib

matplotlib.use('TkAgg')
import matplotlib.pyplot as plt


def A1():
    x = sympy.symbols('x', real=True)
    f = x ** 4
    df_dx = sympy.diff(f, x)
    print(df_dx)
    return df_dx


def A2():
    df_dx = A1()
    x = sympy.symbols('x', real=True)
    func_1 = sympy.lambdify(x, df_dx)
    f = x ** 4
    delta = 0.01
    finite_difference = ((x + delta) ** 4 - x ** 4) / delta
    func_2 = sympy.lambdify(x, finite_difference)
    X = np.arange(-200, 200, 10)
    Y1 = []
    Y2 = []
    for x in X:
        Y1.append(func_1(x))
        Y2.append(func_2(x))
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.scatter(X, Y1, alpha=0.6, c='red')
    plt.scatter(X, Y2, alpha=0.6, c='black')
    plt.legend(['derivative value', 'finite difference'], loc='upper right')
    plt.show()


def A3():
    df_dx = A1()
    x = sympy.symbols('x', real=True)
    func_1 = sympy.lambdify(x, df_dx)
    f = x ** 4
    X = np.arange(-1, 1, 0.1)

    delta = 0.001
    finite_difference = ((x + delta) ** 4 - x ** 4) / delta
    func_2 = sympy.lambdify(x, finite_difference)

    delta = 0.01
    finite_difference = ((x + delta) ** 4 - x ** 4) / delta
    func_3 = sympy.lambdify(x, finite_difference)

    delta = 0.1
    finite_difference = ((x + delta) ** 4 - x ** 4) / delta
    func_4 = sympy.lambdify(x, finite_difference)

    delta = 0.5
    finite_difference = ((x + delta) ** 4 - x ** 4) / delta
    func_5 = sympy.lambdify(x, finite_difference)

    delta = 1
    finite_difference = ((x + delta) ** 4 - x ** 4) / delta
    func_6 = sympy.lambdify(x, finite_difference)

    Y1 = []
    Y2 = []
    Y3 = []
    Y4 = []
    Y5 = []
    Y6 = []
    for x in X:
        Y1.append(func_1(x))
        Y2.append(func_2(x))
        Y3.append(func_3(x))
        Y4.append(func_4(x))
        Y5.append(func_5(x))
        Y6.append(func_6(x))
```

```python
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.scatter(X, Y1, c='black')
    plt.scatter(X, Y2, c='red')
    plt.scatter(X, Y3, c='blue')
    plt.scatter(X, Y4, c='green')
    plt.scatter(X, Y5, c='yellow')
    plt.scatter(X, Y6, c='orange')
    plt.legend(['derivative value', 'δ=0.001', 'δ=0.01', 'δ=0.1', 'δ=0.5', 'δ=1'], loc='upper right')
    plt.show()


def func_1(x):
    return x ** 4


def func_2(x):
    return 4*x**3


def func_3(x, gamma):
    return gamma*x**2


def func_4(x, gamma):
    return gamma*2*x


def func_5(x, gamma):
    return gamma*abs(x)


def func_6(x, gamma):
    if x < 0:
        return -gamma
    return gamma


def B1(func_1, func_2, x0, alpha=0.15, num_iters=50):
    x = x0
    X = np.array([x])
    F = np.array(func_1(x))
    for k in range(num_iters):
        step = alpha * func_2(x)
        x = x - step
        X = np.append(X, [x], axis=0)
        F = np.append(F, func_1(x))
    return (X, F)


def B2(x0=1, alpha=0.1):
     (X, F) = B1(func_1, func_2, x0=x0, alpha=alpha)
    xx = np.arange(-1, 1.1, 0.1)
    # plt.plot(F)
    # plt.xlabel('iteration')
    # plt.ylabel('f(x)')
    # plt.show()
    # plt.plot(X)
    # plt.xlabel('iteration')
    # plt.ylabel('x')
    # plt.show()
    plt.step(X, func_1(X))
    plt.plot(xx, func_1(xx))
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.show()


def B3():
    for x in np.arange(0.5, 1, 0.1):
        B2(x, alpha=0.1)
    # for alpha in [0.01, 0.1, 0.2, 0.5]:
    #     B2(1, alpha=alpha)


def C1():
    gamma = 0.1
    x = 1
    X_1 = np.array([x])
    F_1 = np.array(func_3(x, gamma))
    for k in range(50):
        step = 0.1 * func_4(x, gamma)
        x = x - step
        X_1 = np.append(X_1, [x], axis=0)
        F_1 = np.append(F_1, func_3(x, gamma))

    gamma = 0.5
    x = 1
    X_2 = np.array([x])
    F_2 = np.array(func_3(x, gamma))
    for k in range(50):
        step = 0.1 * func_4(x, gamma)
        x = x - step
        X_2 = np.append(X_1, [x], axis=0)
        F_2 = np.append(F_1, func_3(x, gamma))

    gamma = 1
    x = 1
    X_3 = np.array([x])
    F_3 = np.array(func_3(x, gamma))
    for k in range(50):
        step = 0.1 * func_4(x, gamma)
        x = x - step
        X_3 = np.append(X_1, [x], axis=0)
        F_3 = np.append(F_1, func_3(x, gamma))

    gamma = 2
    x = 1
    X_4 = np.array([x])
    F_4 = np.array(func_3(x, gamma))
    for k in range(50):
        step = 0.1 * func_4(x, gamma)
        x = x - step
        X_4 = np.append(X_1, [x], axis=0)
        F_4 = np.append(F_1, func_3(x, gamma))

    xx = np.arange(-1, 1.1, 0.1)

    plt.plot(F_1)
    plt.plot(F_2)
    plt.plot(F_3)
    plt.plot(F_4)
    plt.xlabel('iteration')
    plt.ylabel('f(x)')
    plt.show()

    plt.plot(X_1)
    plt.plot(X_2)
    plt.plot(X_3)
    plt.plot(X_4)
    plt.xlabel('iteration')
    plt.ylabel('x')
    plt.show()

    plt.step(X_1, func_3(X_1, 0.1))
    plt.plot(xx, func_3(xx, 0.1))
    plt.step(X_2, func_3(X_2, 0.5))
    plt.plot(xx, func_3(xx, 0.5))
    plt.step(X_3, func_3(X_3, 1))
    plt.plot(xx, func_3(xx, 1))
    plt.step(X_4, func_3(X_4, 2))
    plt.plot(xx, func_3(xx, 2))
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.legend(['γ=0.1', 'γ=0.5', 'γ=1', 'γ=2'], loc='upper right')
    plt.show()


def C2():
    gamma = 0.1
    x = 1
    X_1 = np.array([x])
    F_1 = np.array(func_5(x, gamma))
    for k in range(50):
        step = 0.1 * func_6(x, gamma)
        x = x - step
        X_1 = np.append(X_1, [x], axis=0)
        F_1 = np.append(F_1, func_5(x, gamma))

    gamma = 0.5
    x = 1
    X_2 = np.array([x])
    F_2 = np.array(func_5(x, gamma))
    for k in range(50):
        step = 0.1 * func_6(x, gamma)
        x = x - step
        X_2 = np.append(X_1, [x], axis=0)
        F_2 = np.append(F_1, func_5(x, gamma))

    gamma = 1
    x = 1
    X_3 = np.array([x])
    F_3 = np.array(func_5(x, gamma))
    for k in range(50):
        step = 0.1 * func_6(x, gamma)
        x = x - step
        X_3 = np.append(X_1, [x], axis=0)
        F_3 = np.append(F_1, func_5(x, gamma))

    gamma = 2
    x = 1
    X_4 = np.array([x])
    F_4 = np.array(func_5(x, gamma))
    for k in range(50):
        step = 0.1 * func_6(x, gamma)
        x = x - step
        X_4 = np.append(X_1, [x], axis=0)
        F_4 = np.append(F_1, func_5(x, gamma))

    xx = np.arange(-1, 1.1, 0.1)
    plt.plot(F_1)
    plt.plot(F_2)
    plt.plot(F_3)
    plt.plot(F_4)
    plt.xlabel('iteration')
    plt.ylabel('f(x)')
    plt.show()

    plt.plot(X_1)
    plt.plot(X_2)
    plt.plot(X_3)
    plt.plot(X_4)
    plt.xlabel('iteration')
    plt.ylabel('x')
    plt.show()

    plt.step(X_1, func_5(X_1, 0.1))
    plt.plot(xx, func_5(xx, 0.1))
    plt.step(X_2, func_5(X_2, 0.5))
    plt.plot(xx, func_5(xx, 0.5))
    plt.step(X_3, func_5(X_3, 1))
    plt.plot(xx, func_5(xx, 1))
    plt.step(X_4, func_5(X_4, 2))
    plt.plot(xx, func_5(xx, 2))
    plt.xlabel('x')
    plt.ylabel('f(x)')
```

```python
    plt.legend(['γ=0.1', 'γ=0.5', 'γ=1', 'γ=2'], loc='upper right')
    plt.show()


if __name__ == '__main__':
    C1()
```