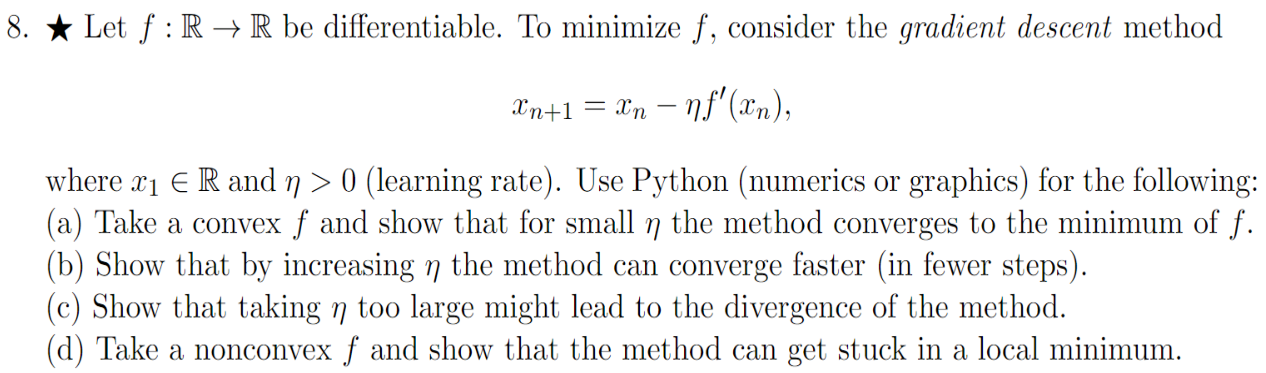
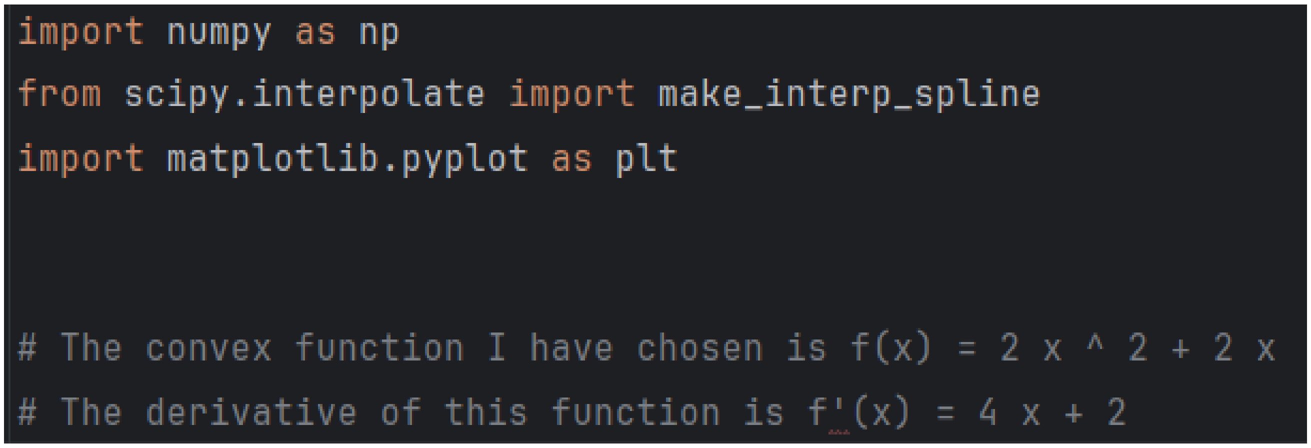
Homework S05

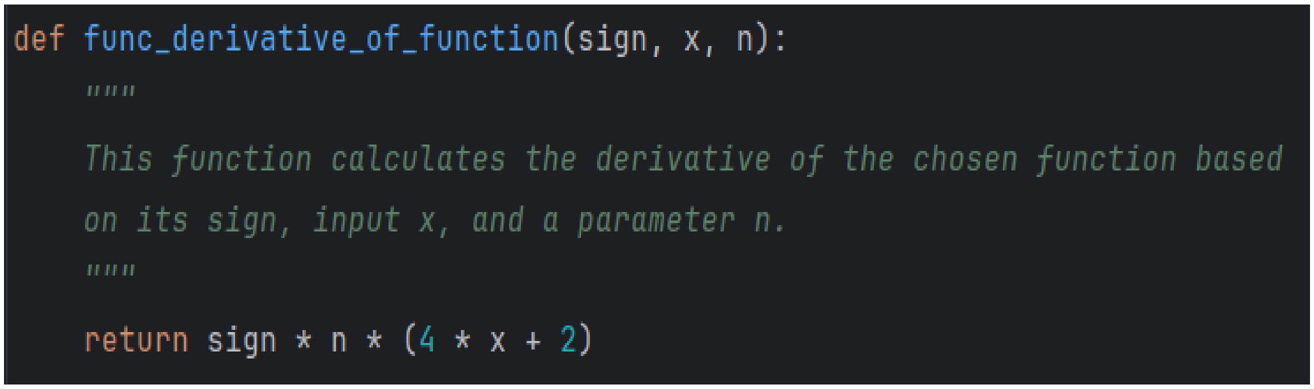
Jitareanu Eduard-David


8. ★ Let $f : \mathbb{R} \to \mathbb{R}$ be differentiable. To minimize f, consider the *gradient descent* method

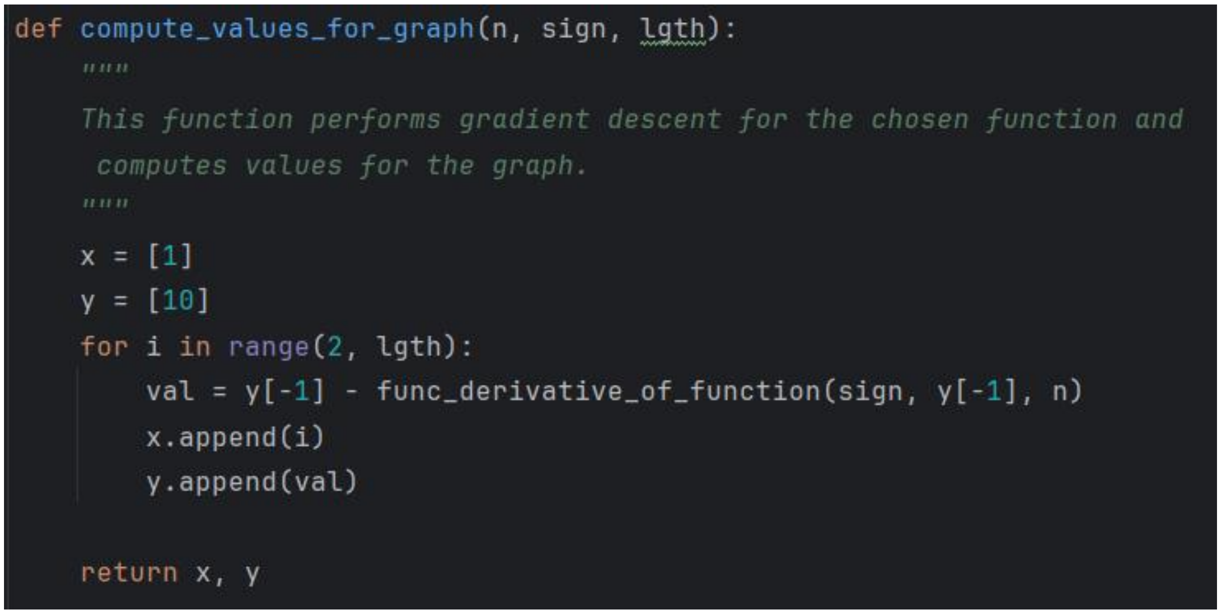
$$x_{n+1} = x_n - \eta f'(x_n),$$

where $x_1 \in \mathbb{R}$ and $\eta > 0$ (learning rate). Use Python (numerics or graphics) for the following:
(a) Take a convex f and show that for small η the method converges to the minimum of f.
(b) Show that by increasing η the method can converge faster (in fewer steps).
(c) Show that taking η too large might lead to the divergence of the method.
(d) Take a nonconvex f and show that the method can get stuck in a local minimum.


```python
import numpy as np
from scipy.interpolate import make_interp_spline
import matplotlib.pyplot as plt


# The convex function I have chosen is f(x) = 2 x ^ 2 + 2 x
# The derivative of this function is f'(x) = 4 x + 2
```

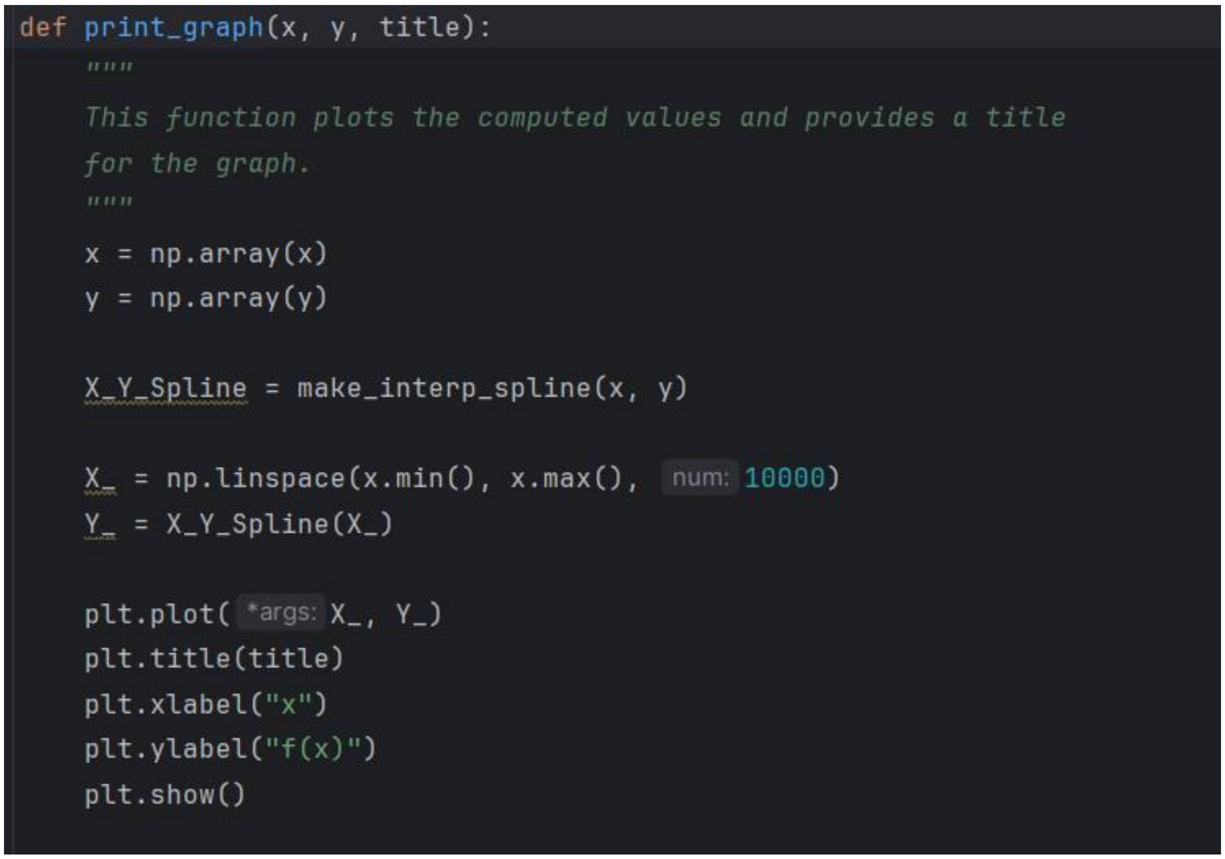

```python
def func_derivative_of_function(sign, x, n):
    """
    This function calculates the derivative of the chosen function based
    on its sign, input x, and a parameter n.
    """
    return sign * n * (4 * x + 2)
```

If n=1, then we have a convex function, otherwise if n=-1, a non-convex function is represented.

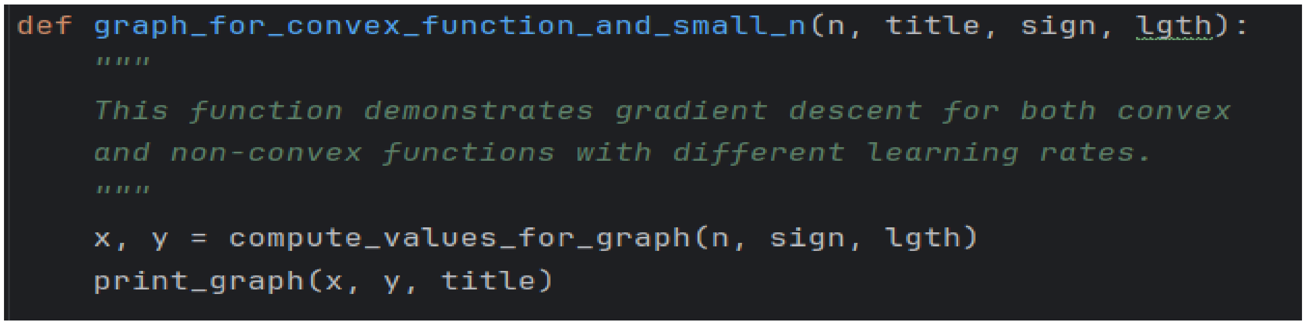


```python
def compute_values_for_graph(n, sign, lgth):
    """
    This function performs gradient descent for the chosen function and
     computes values for the graph.
    """
    x = [1]
    y = [10]
    for i in range(2, lgth):
        val = y[-1] - func_derivative_of_function(sign, y[-1], n)
        x.append(i)
        y.append(val)

    return x, y
```

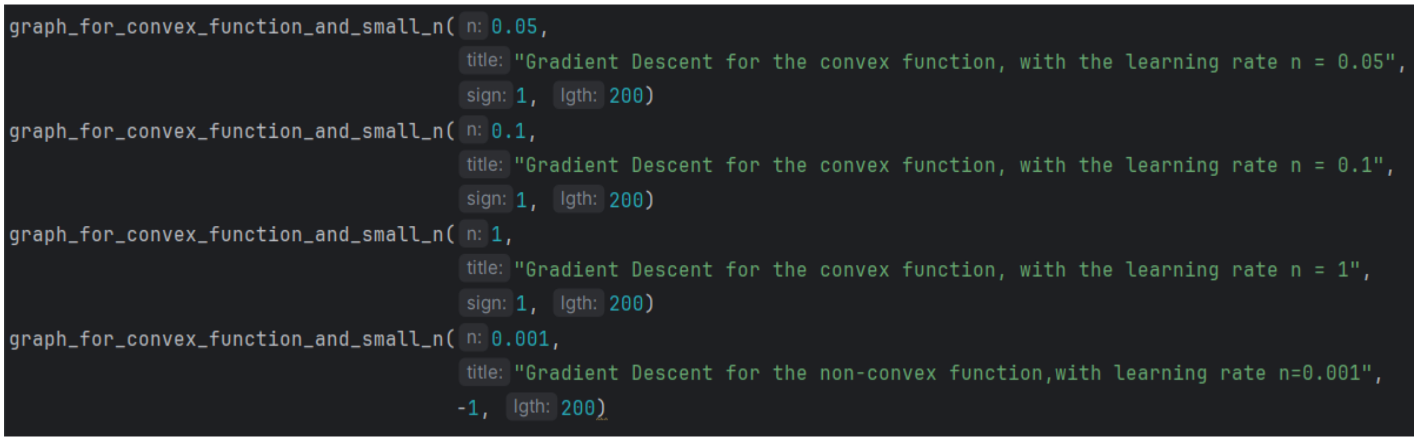
This function above: “compute_values_for_graph” simulates the process of gradient descent for a specified number of iterations, calculates the points along the curve, and returns these points as lists x and y for plotting the curve using Matplotlib



```python
def print_graph(x, y, title):
    """
    This function plots the computed values and provides a title
    for the graph.
    """
    x = np.array(x)
    y = np.array(y)

    X_Y_Spline = make_interp_spline(x, y)

    X_ = np.linspace(x.min(), x.max(),  num: 10000)
    Y_ = X_Y_Spline(X_)

    plt.plot( *args: X_, Y_)
    plt.title(title)
    plt.xlabel("x")
    plt.ylabel("f(x)")
    plt.show()
```



```
def graph_for_convex_function_and_small_n(n, title, sign, lgth):
    """
    This function demonstrates gradient descent for both convex
    and non-convex functions with different learning rates.
    """
    x, y = compute_values_for_graph(n, sign, lgth)
    print_graph(x, y, title)
```

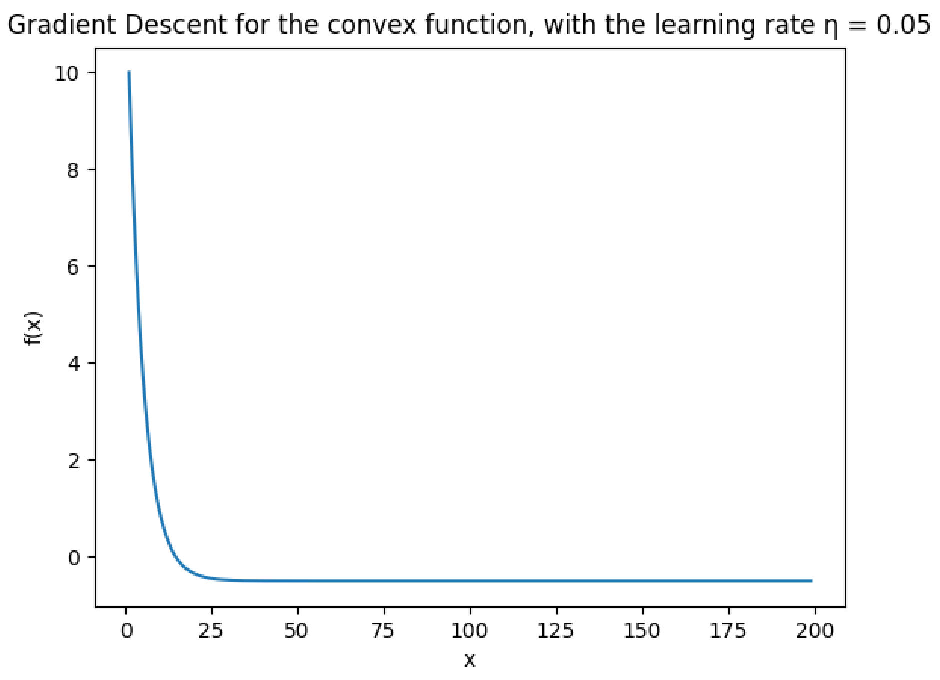


```
graph_for_convex_function_and_small_n( n: 0.05,
                                       title: "Gradient Descent for the convex function, with the learning rate n = 0.05",
                                       sign: 1,  lgth: 200)
graph_for_convex_function_and_small_n( n: 0.1,
                                       title: "Gradient Descent for the convex function, with the learning rate n = 0.1",
                                       sign: 1,  lgth: 200)
graph_for_convex_function_and_small_n( n: 1,
                                       title: "Gradient Descent for the convex function, with the learning rate n = 1",
                                       sign: 1,  lgth: 200)
graph_for_convex_function_and_small_n( n: 0.001,
                                       title: "Gradient Descent for the non-convex function,with learning rate n=0.001",
                                       -1,  lgth: 200)
```

When “graph_for_convex_function_and_small_n” is called with different parameters, it will generate and display the gradient descent plot for the chosen function, learning rate, and title. The smooth curve represents how the gradient descent algorithm converges to the minimum of the function.

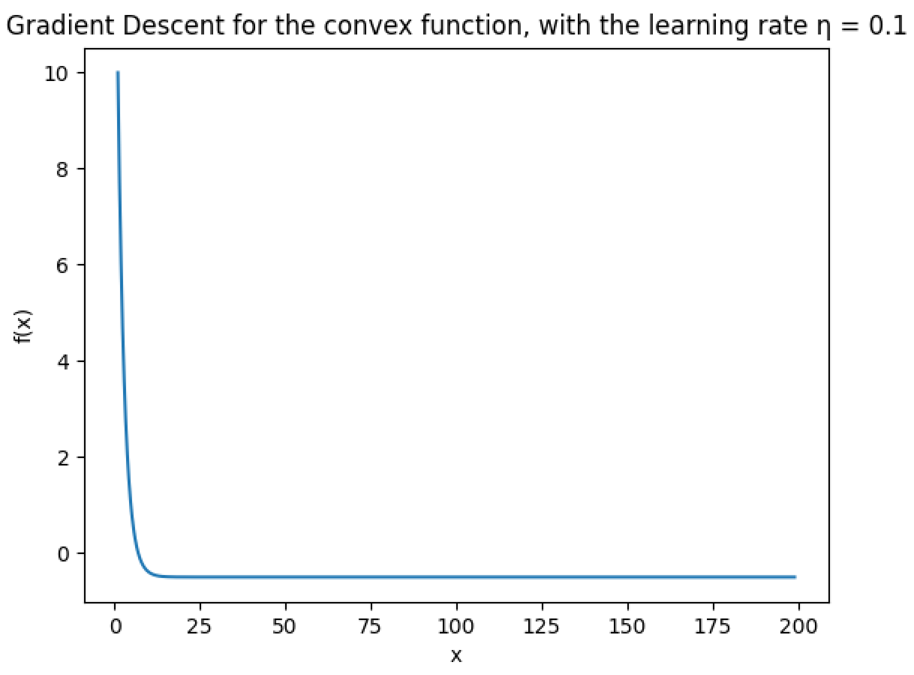
Results!

(a) Take a convex f and show that for small η the method converges to the minimum off.



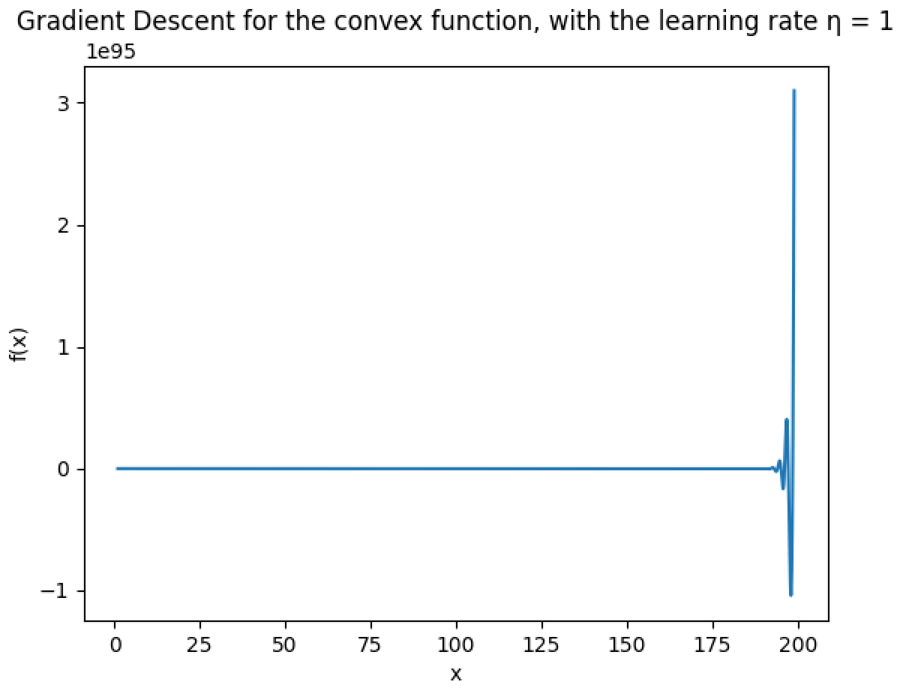
The resulting graph displays a smooth, descending curve indicating the convergence of the algorithm to the minimum point of the convex function with the specified parameters.

(b) Show that by increasing η the method can converge faster (in fewer steps)



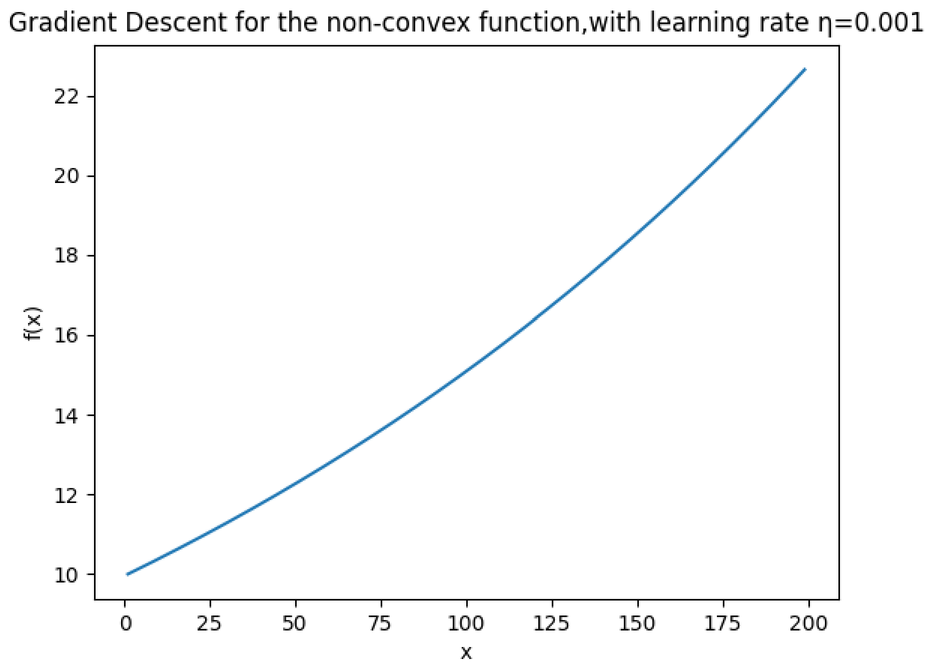
The resulting graph displays a steeper descending curve, reflecting the faster convergence of the algorithm to the minimum point of the convex function due to the larger learning rate.

c) Show that taking η too large might lead to the divergence of the method.



The resulting graph displays an extremely steep descending curve, reflecting the very rapid convergence of the algorithm to the minimum point of the convex function due to the very large learning rate. In this case, it exhibits oscillations or instability due to the high learning rate.

(d) Take a nonconvex f and show that the method can get stuck in a local minimum



The resulting graph displays the path taken by the gradient descent algorithm. It shows how the algorithm explores the nonconvex function and tries to escape from local minima. The small learning rate can allow for a more careful exploration of the landscape.