Project

Important:

- Write your name as well as your NU ID on your assignment. Please number your problems.
- Submit both results and your code.
- Give complete answers. Do not just give the final answer; instead show steps you went through to get there and explain what you are doing. Do not leave out critical intermediate steps.
- This assignment must be submitted electronically through Gradescope by June 23rd 2025 (Monday) by 11:59 PM.
- All of your codes must be commented.
- All of your plots must be labeled, titled and have a legend (when applicable).

The project consists of developing a (simplistic) machine learning model that allows you to predict the diastolic blood pressure and systolic blood pressure of a patient given the total cholesterol level in their blood. In this part of the project, you will explore the gradient descent algorithm and start implementing the framework needed to build the predictive model.

The main goals of this project are the following:

- Learn how one optimization algorithm (gradient descent) works.
- Gain a a deeper understanding of the limitations of gradient descent works by analyzing its performance on different functions.
- Learn how to develop code to be able to implement the mathematical concepts presented in this project. The project will guide you to divide your code into several smaller functions. You should always avoid having functions with a large number of code lines. It is much easier to read code that is split into several functions rather than having to read one large block of code. In this part of the project, I am walking you through how to divide your code into several functions that only include a small number of lines. In the next part, you will divide your code into several functions by yourselves.
- Implement derivative approximation to gradient descent to any function.
- Generalize gradient descent to minimize functions of two variables.
- Derive cost function for model optimization.
- Implement gradient function to the cost function.
- Evaluate the performance of your model on different data sets.

1 Gradient Descent

Suppose you want to minimize a function $f(x)$. A minimum of $f(x)$ occurs at a point x where the derivative $f'(x)$, i.e slope of the tangent line to $f(x)$, is 0.

For example, consider $f(x) = x^2$. If you plot the function, you can see that its minimum value is achieved at $x = 0$. We can also find the location of the minimum by computing the derivative $f'(x) = 2x$ and solving $f'(x) = 2x = 0$ resulting in the critical point $x = 0$. For $f(x) = x^2$, we can conclude that the minimum of $f(x)$ is achieved at $x = 0$ and has a value of $f(0) = 0$.

At times, it might be challenging to solve the equation $f'(x) = 0$, depending on how complex $f'(x)$ is. For example, if we consider $f(x) = \frac{e^x}{(1+x^2)^2}$, the equation $f'(x) = 0$ cannot be solved exactly. We need some numerical algorithms that would allow us to estimate the minimum of a function. One possible optimization algorithm is known as the **gradient descent algorithm**.

We motivate the idea behind gradient descent as follows: suppose after hiking up a mountain, you finally reached the top and seek to go down the mountain to get to the ground. How can you do that in the fastest way possible? We define $f(x)$ to be the function modeling the mountain. You can notice that the fastest direction to go down the mountain is to always follow the direction of the derivative of that function at your current location. You can see that in figure 1.

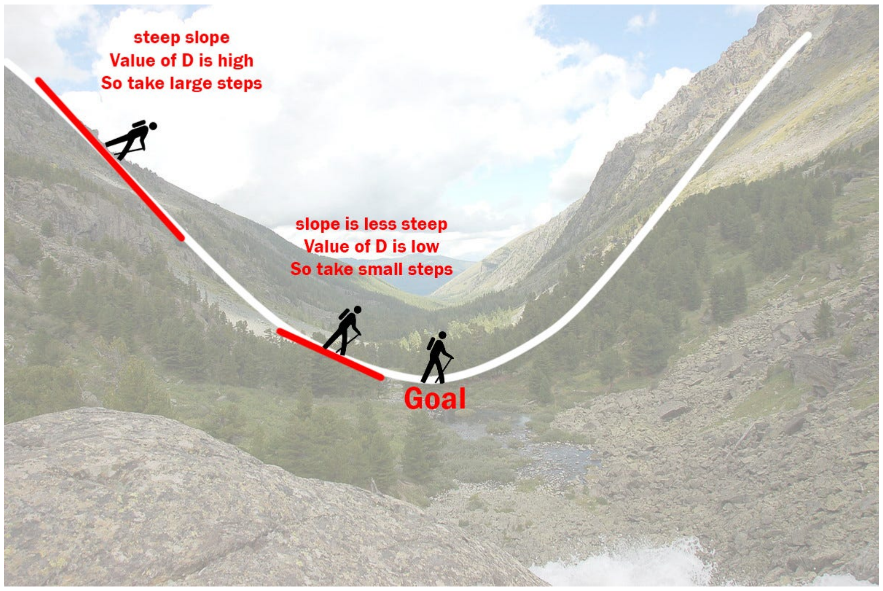

Figure 1: Gradient Descent Algorithm. D is the value of the magnitude of the derivative.

You can take a step in the direction of the derivative to go down as fast as possible from the mountain. The next question you might ask yourself though is how big should your step be? Usually, a smaller step size is preferred over a larger step size in order not to miss the location of minimum and climb up the mountain further. Let α be the step size.

The gradient descent algorithm would consist of the following steps:

- You start at some initial point x_0 on the mountain modeled by $f(x)$.
- You keep on taking steps of size α in the direction of the derivative. That is, you update your position by applying,

$$x_1 = x_0 - \alpha f'(x_0), \tag{1}$$

where x_1 is the new position. The equation above can be generalized to the kth step as,

$$x_{k+1} = x_k - \alpha f'(x_k). \tag{2}$$

- You keep on updating the position according to the gradient descent update rule up until the value of x_k barely changes. That is, for some number of steps k, you would notice that x_{k+1} and x_k are very close to each other up to some tolerance ϵ. For instance, say you chose the tolerance $\epsilon = 0.001$, a relatively small number. You would keep on applying the gradient descent update rule up until x_{k+1} and x_k are within $\epsilon = 0.001$ from each other, i.e.,

$$|x_{k+1} - x_k| < \epsilon. \tag{3}$$

1.1 Quadratic Functions

We consider the following two quadratic functions,

$$f_1(x) = x^2 \text{ and } f_2(x) = x^2 - 2x + 3. \tag{4}$$

1. Start by writing a function that implements the gradient descent algorithm. In the algorithm, define a variable *iter* that count the number of gradient descent iterations performed. In the while loop, add a second condition that checks whether *iter* is not greater than some maximum number of iterations allowed, given by *iter_max*. This will allow you to avoid infinite loops. In all parts of this problem, you can fix *iter_max* = 1000. Test your algorithm for $x_0 = 3$, $\alpha = 0.1$ and $\epsilon = 0.001$ for both $f_1(x)$ and $f_2(x)$. Plot the location of the optimal x you find using gradient descent along with $f_1(x)$ and $f_2(x)$ (each function and its optimizer on one separate plot).

By the end of this part, you should have implemented, including writing comments, the following functions:

```
def gradient_descent():

def f1():

def deriv_f1():

def f2():

def deriv_f2():

def plot_opt():
```

The inputs of the functions are omitted above as I want you to figure out the inputs of the functions.

In the next sub-parts, we will be conducting a perturbation analysis of the parameters, also known as parameter hyper tuning, to understand the effect of varying the values of x_0, $\alpha$and ϵ on the output of the gradient descent algorithm.

2. We vary the value of x_0, and fix $\alpha = 0.1$ and $\epsilon = 0.001$. We consider $x_0 = 3$ and $x_0 = -3$. For each value of x_0, compute the optimal value of x_k obtained by the gradient descent algorithm for both $f_1(x)$ and $f_2(x)$. Compare the optimal values computed for both values of x_0. Could you have expected how the optimal result varies for different x_0 before even running the algorithm?

3. Now, we fix $x_0 = 3$ and $\epsilon = 0.001$ and consider different values of $\alpha \in \{1, 0.001, 0.0001\}$. What is the output of the gradient algorithm for every value of α? Are you observing any variations in the output? If so, why do you think this is happening?
Hint: For $\alpha = 1$, check the value of *iter*.

4. Finally, we fix $x_0 = 3$ and $\alpha = 0.1$, and consider the following values of $\epsilon \in \{0.1, 0.01, 0.0001\}$. Once again, what is the output of the gradient algorithm for every value of ϵ? Are you observing any variations in the output? If so, why do you think this is happening?

When testing the gradient descent implementation with different values of x_0, α and ϵ, include all of these tests within one main function:

```
def main():
```

1.2 More Complex Functions

We now consider a more complex function given by,

$$f_3(x) = \sin(x) + \cos(\sqrt{2}x), \text{ for } 0 \leq x \leq 10. \quad (5)$$

1. Start by plotting $f_3(x)$. Notice how many local minima $f_3(x)$ has. For this part, you will need to write one function:

```
def plot_f3():
```

2. We fix $\alpha = 0.1$ and $\epsilon = 0.0001$. The derivative of $f_3(x)$ is given by $f_3'(x) = \cos(x) - \sqrt{2}\sin(\sqrt{2}x)$. Implement the gradient descent algorithm for $x_0 = 1$, $x_0 = 4$, $x_0 = 5$ and $x_0 = 7$. What is the output of the gradient descent algorithm for each value of x_0? Plot the function along with the local minima you obtained for the different values of x_0.

You can update the function *plot_ opt* you implemented in part 1 of section 1.1 to plot more than one optimal points. After completing this part, you should have coded (or modified) the following additional functions:

```
def plot_opt():

def f3():

def deriv_f3():
```

2 Derivative Approximation for Functions of One Variable

For complex functions $f(x)$, computing the derivatives can be challenging. In many algorithms, computer scientists come up with ways to approximate derivatives. Notice that the quantity $\lim_{h\to 0} \frac{f(x+h)-f(x)}{h}$ will be very close to the derivative $f'(x)$ for a small value of h.

1. Write a Python function which approximates the derivative of any function $f(x)$ at some point x. You will need to choose a small value of h. Test your Python function on $f_1(x) = x^2$ and $f_1(x) = x^2 - 2x + 3$. Check that your function is returning similar values to the exact derivatives for different values of x. You can choose whichever values of x you want to test.

2. Modify the *gradient_ descent()* function that you implemented in the previous assignment so that it uses the derivative approximation function instead of the actual derivative. Test your algorithm on $f_1(x) = x^2$ and $f_1(x) = x^2 - 2x + 3$ for $x_0 = 3$, $\alpha = 0.1$ and $\epsilon = 0.001$. Do you still obtain the same results as in your original implementation?

3 Gradient Descent for Functions of Two Variables

We extend the gradient descent framework to functions of two variables, for instance function $f(a, b)$ where a and b are its inputs. Multi-variable functions occur very often in a variety of applications. For instance, suppose you are driving on some highway and your car is currently at position x. You are interested in computing the car's future position after some time t when driving with speed v. You can see that the car's future position depends on several variables: x, t and v.

Suppose you are interested in minimizing a two-variable function $f(x, y)$. The gradient descent algorithm can still be applied and its logic remains the same:

- You start at some initial points x_0 and y_0.
- You keep on taking steps of size α in the direction of the derivative with respect to each variable. That is, you update the values of x_0 and y_0 by applying,

  $$x_1 = x_0 - \alpha \frac{df}{dx}(x_0, y_0) \text{ and } y_1 = y_0 - \alpha \frac{df}{dy}(x_0, y_0) \tag{6}$$

  where x_1, y_1 are the updated values of the parameters, $\frac{df}{dx}(x_0, y_0)$ is the derivative of f with respect to x computed at (x_0, y_0) and $\frac{df}{dy}(x_0, y_0)$ is the derivative of f with respect to y computed at (x_0, y_0).

  The intuition behind derivatives of multi-variable functions is the same as with single variable functions. For example, suppose you want to compute the derivative of f with respect to x at some point (x_0, y_0), you can compute the slope of the "line" joining the points (x, y) and $(x + h, y)$ and take $h \to 0$. You can write,

  $$\frac{df}{dx}(x_0, y_0) = \lim_{h \to 0} \frac{f(x_0 + h, y_0) - f(x_0, y_0)}{h} \text{ and } \frac{df}{dy}(x_0, y_0) = \lim_{h \to 0} \frac{f(x_0, y_0 + h) - f(x_0, y_0)}{h}. \tag{7}$$

  The derivatives can be approximated by extending the approximation function you developed in part 1 of section 1. The gradient descent update rule can be generalized to the kth step as,

  $$x_{k+1} = x_k - \alpha \frac{df}{dx}(x_k, y_k) \text{ and } y_{k+1} = y_k - \alpha \frac{df}{dy}(x_k, y_k) \tag{8}$$

- You keep on updating the values x_k and y_k according to the gradient descent update rule up until the value of both x_k and y_k barely change. That is, for some number of steps k, you would notice that x_{k+1} and x_k, as well as y_{k+1} and y_k, are very close to each other up to some tolerance ϵ. For instance, say you chose the tolerance $\epsilon = 0.001$, a relatively small number. You would keep on applying the gradient descent update rule up until,

  $$|y_{k+1} - y_k| < \epsilon \textbf{ and } |y_{k+1} - y_k| < \epsilon. \tag{9}$$

1. Extend the function written in part 1 of section 1 to approximate derivatives of functions of two variables.

2. Implement the gradient descent algorithm for functions of two variables. Approximate the derivatives in the update steps using the function you wrote in the previous part. Make sure your code is structured into several functions as we did in the previous assignment. Test your code on $f(x, y) = x^2 + y^2$ with $x_0 = 3$, $y_0 = 3$, $\alpha = 0.1$ and $\epsilon = 0.001$. Plot the function with the minimum obtained. Note that this will be a $3D$ plot.

4 Cost Function to Evaluate the Performance of a Predictive Model

Suppose you are given n data points of the form $\{(x^{(i)}, y^{(i)}), i = 1, 2, ..., n\}$, where $x^{(i)}$ is the total cholesterol level in patient i's blood and $y^{(i)}$ is the diastolic blood pressure of patient i. Your current goal is to come up with a model that predicts "well" the diastolic blood pressure of any patient given their total blood cholesterol level. The data considered is shown in figure 2 and uploaded on Canvas and uploaded as *data_ chol_ dias_ pressure.txt*. Note that the current data set is a synthetic data set that I created and not an actual medical data set.

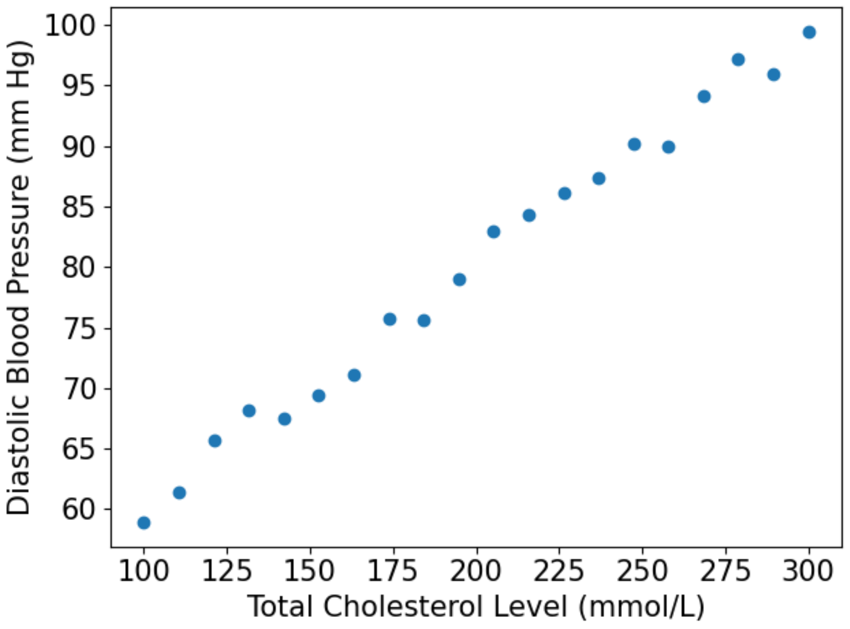

Figure 2: Plot illustrating the diastolic blood pressure of different patients given their total blood cholesterol level. Every patient is represented by a blue dot in the plot.

The data shown in figure 2 looks generally linear. One possible option would be to find a linear function $f(x) = ax + b$ for which the line would be as close as possible to blue dots.
That is, our goal is to find some optimal values of the slope a and intercept b for which,

$$f(x^{(i)}) = ax^{(i)} + b \approx y^{(i)}, \tag{10}$$

for all n patients. We need to compute the magnitude of the difference of $ax^{(i)} + b$ and $y^{(i)}$ for every patient i and try to minimize these quantities. The magnitude has to be a non negative number. One option would be to consider the square of the difference of $ax^{(i)} + b$ and $y^{(i)}$ for every patient i to result in non-negative values. This expression would be,

$$\left(ax^{(i)} + b - y^{(i)}\right)^2. \tag{11}$$

Additionally, we seek to consider the above expression for every patient i to find the optimal line that is as close as possible to all the blue dots shown in figure 2. This is why, we will analyze the cost function,

$$g(a,b) = \left(ax^{(1)} + b - y^{(1)}\right)^2 + \left(ax^{(2)} + b - y^{(2)}\right)^2 + ... + \left(ax^{(n)} + b - y^{(n)}\right)^2, \tag{12}$$

to take into account all patients.

Our goal is to find the optimal a and b which minimize $g(a,b)$ so that the optimal line given by $y = ax + b$ is as close as possible to all the data points.

1. Read the data given on Canvas and split into two arrays x and y. The first column of the data should be stored in x and the second column of the data should be stored in y. Each array x and y will have 20 elements has the data set given to you consists of 20 patients. You can do that using the following code:

```
import numpy as np

def load_data(filename):
    '''
    Load the data into two arrays

    Args:
        filename: string representing the name of the file
            containing the data (x,y)
    Return:
        array containing the values of x
        array containing the values of y

    '''
    data = np.loadtxt(filename)
    return data[:, 0], data[:, 1]

x,y = load_data('data_chol_dias_pressure.txt')
```

2. Write a function which computes $g(a,b)$ for any a and b. This function should take as input a, b, x and y.

3. Notice that $g(a,b)$ is a function of two variables. Apply the gradient descent algorithm developed in the second part of the project, for functions of two variables, to compute the optimal values of a and b, denoted by, a^* and b^*, respectively.

4. Plot the line $y = a^*x + b^*$ along with the data points shown in figure 2. How does the plotted line perform?

5. Was it easy to find the correct values for the initial parameters for gradient descent to converge? Why do you think this is happening?
Hint: Scaling your dataset, i.e. subtracting from every data the mean of the dataset and dividing by the standard deviation of the dataset will help.

5 Non-linear Data

You were just informed that the blood pressure monitor machine used in the previous section had manufacturing errors and resulted in incorrect data. You now used a new blood pressure monitor machine that is less prone to errors and obtained the data shown in figure 3. This data is stored in *data_ chol_ dias_ pressure_ non_ lin.txt* and uploaded on Canvas.

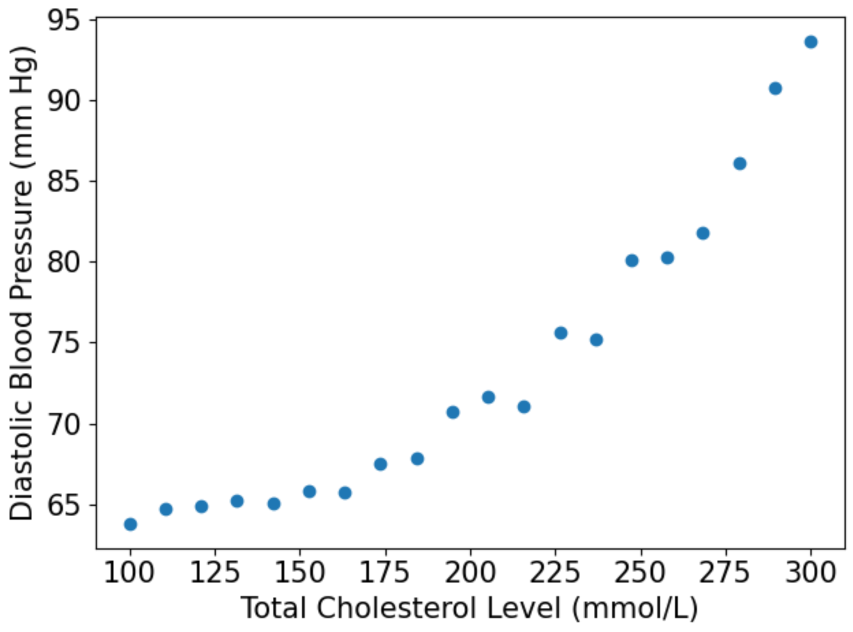

Figure 3: Plot illustrating the diastolic blood pressure of different patients given their total blood cholesterol level, using the new blood pressure monitor machine. Every patient is represented by a blue dot in the plot.

1. Read the data given on Canvas and split into two arrays x and y. The first column of the data should be stored in x and the second column of the data should be stored in y. Each array x and y will have 20 elements has the data set given to you consists of 20 patients.

2. Apply the gradient descent algorithm developed in the second part of the project, for functions of two variables, on $g(a, b)$, computed according to the new data, to compute the optimal values of a and b, denoted by, a^* and b^*, respectively.

3. Plot the line $y = a^*x + b^*$ along with the data points shown in figure 2. How does the plotted line perform now? Why is the optimal line performing different for the updated data set?

4. What possible modifications can you implement to your model to improve its performance? There isn't one correct answer here, what matters to me is your reasoning. Think whether the function $f(x)$ adopted is suitable for the new data set. Implement your proposed change(s).

6 Report

Write a report that explains how you designed your code to implement gradient descent for one variable first and how you generalized it to several variables. Explain how you tested your code with different cases. Summarize what you learnt when applying a perturbation analysis of the parameters, also known as parameter hyper tuning, and how different values of x_0, α and ϵ can affect the performance of gradient descent. Explain how you designed your code to implement gradient descent for functions of several variables on the cost function $g(a, b)$. Summarize what you learnt about how the choice of $f(x)$ can affect the performance on the model depending on the data set.