ASSIGNMENT 5 - GRADIENT DESCENT BY EE22B025

Overall Approach

Understanding the Problem:

- The main goal of the problem is to optimise the given functions and to find the **optimum values** of each function.
- We need to use *Gradient Descent* to find the optimum values of the functions.
- Two of the given functions are **2** variable functions, thus we also need to extend Gradient Descent for 2 variables.
- We also need to use **Matplotlib** and **FuncAnimation** to plot the graph and display animation of the Gradient Descent.

Implementing Gradient Descent:

- I have created 2 seperate *Grad_Desc* functions :
  1) **Grad_Desc_1D()** - For one variable functions
  2) **Grad_Desc_2D()** - For two variable functions
- The overall structure of both these functions is same.
- For *Grad_Desc_1D* :
  - We start with an initial guess of bestx and then check whether $x \in (RangeMin, RangeMax)$.
  - If yes, then we update the bestx by using the equation

$$bestx = bestx - f'(x)dx$$

  - If no, then we update the bestx and make it equal to Range_min.
  - This keeps repeating until we reach the loop limit.
  - The idea is to keep finding an x such that $f(x)$ is smaller than the previous *bestcost* value until we reach a **local/global minima**.
- For *Grad Desc_2D* :
  - This function also follows the same logic as its 1D counterpart, with the difference being an extra variable z.
  - Here, we have inital guess of *bestx* and *besty* along with *bestcost*.
  - Again, we check $x \in (RangeMin, RangeMax)$ and $y \in (RangeMin, RangeMax)$ and update them using the same equation as given above.
  - Again, this keeps repeating till we reach the loop limit.

What Parameters does the Gradient Descent Function take?

1) ***Lr*** - This is the Learning Rate which tells us how big the step size between each iteration is.
2) ***bestcost, bestx , besty*** have been explained above.
3) ***func*** - This is the given function which we need to optimise.
4) ***ddx/ddy*** - This is again the (partial) derivative of the given function.
5) ***range_min and range_max*** - These give the range in which we need to optimise the given function.

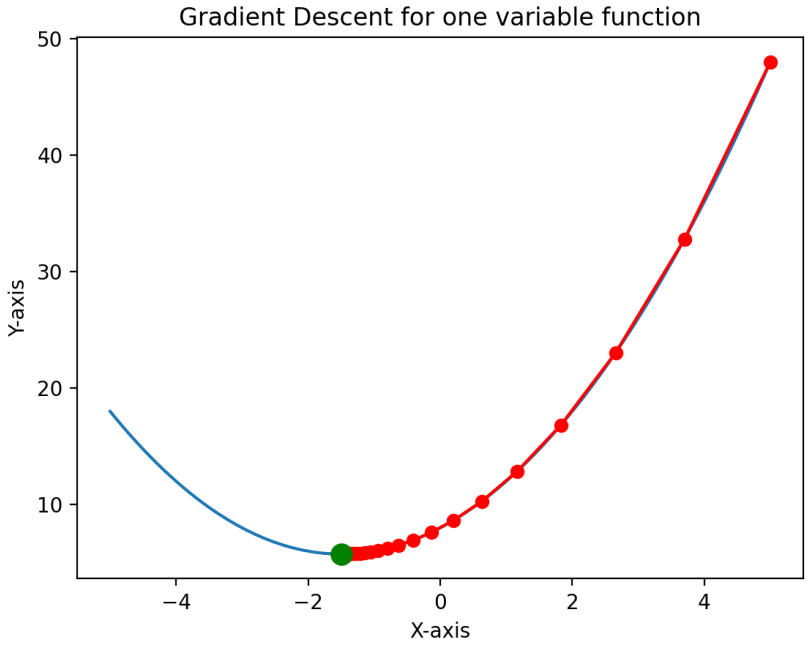
What types of Function can Gradient Descent Function take?

- Usually functions which are ***differentiable***, ***smooth***, ***well behaved*** are taken as inputs to the Gradient Descent function.
- There should be ***well defined gradient*** everywhere in the optimisation range.
- Functions which are ***discontinuous***, ***having peaks and valleys*** with ***not a well-defined derivative*** are usually restricted for the use of Gradient Descent.
- When a function has too many *local minima/maximas* and if the bestcost value reaches a local optima, then it doesn't move from there again as the *gradient is 0* at that point and we end up having a **wrong answer**.

- Also, if we keep the *Learning Rate* very high, then we won't be able to find a good optimum solution because the *bestx* value will keep jumping.

PROBLEM 1

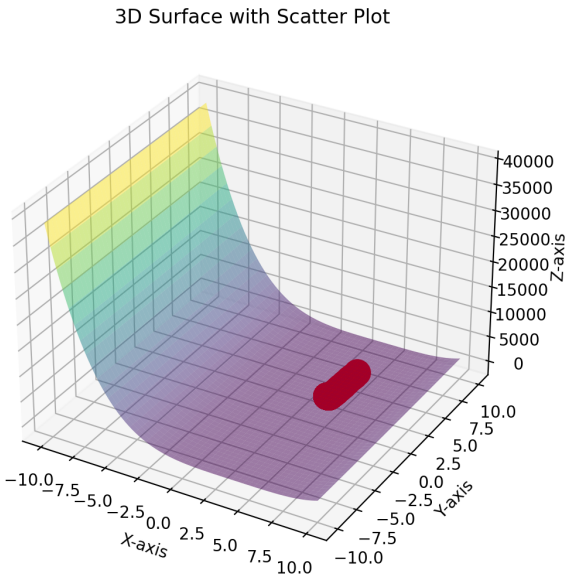
- This function is a simple **Quadratic 1D Polynomial**. The **gradient** has also been provided.
- Since this is a 1D polynomial, I have used the grad_Desc_1D function to implement Gradient Descent.
- The parameters I used are :
  - LR = 0.1
  - bestcost = 10000
  - bestx = 5
  - Range of Optimisation = $[-5, 5]$
- The Final PLOT after Gradient_Descent is :



- The RESULT is :
  - **Minima of f1 = 5.75**
  - **Point of Minima = -1.4999999999999996**

PROBLEM 2

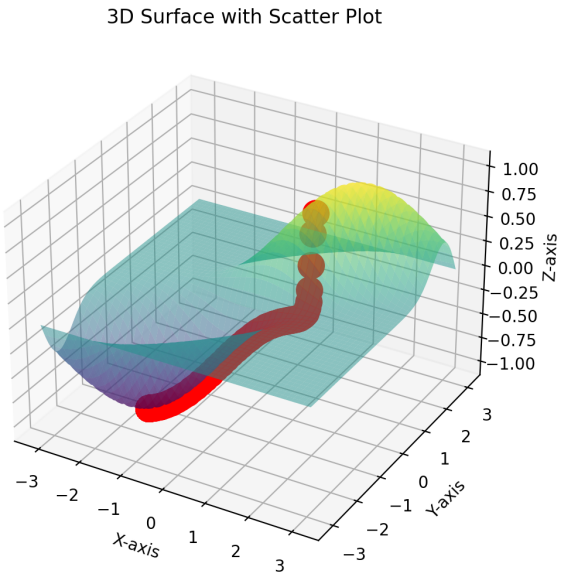
- This function is a simple **2D Polynomial**. The **Partial Derivatives** have also been provided.
- Since this is a 2D polynomial, I have used the grad_Desc_2D function to implement Gradient Descent.
- The parameters I used are :
  - LR = 0.05
  - bestcost = 10000
  - bestx = 6 , besty = 6
  - Range of Optimisation = $[-10, 10]$
- The Final PLOT after Gradient_Descent is :



- The RESULT is :
  - Optimum of f2 occurs at **(x,y) = (4.049561394942729,2.0000000000000018)**
  - Optimum value of f2 = **2.000006033566251**

PROBLEM 3

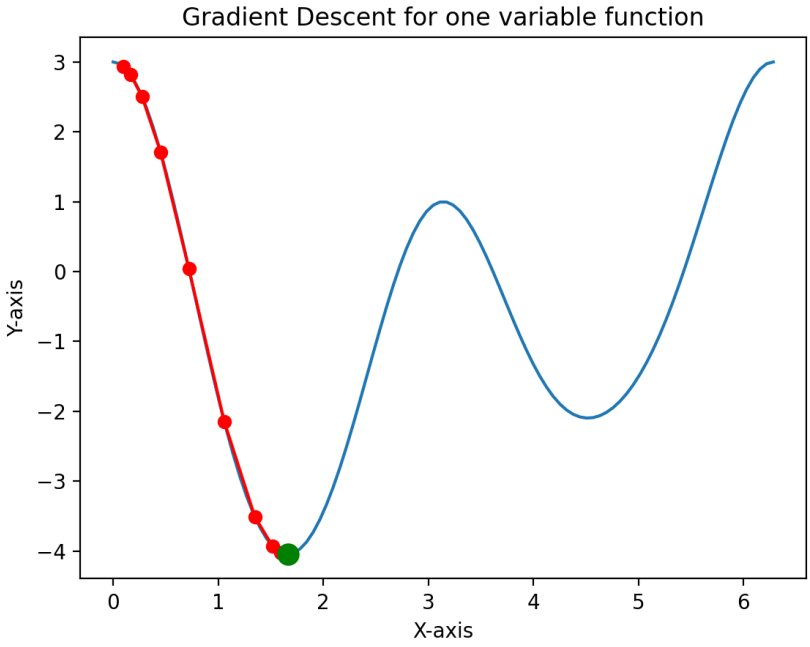
- This function is a complicated **2D Function**. The **Partial Derivatives** have been provided.
- Since this is a 2D function, I have used grad_Desc_2D() to implement Gradient Descent.
- The parameters I used are :
  - LR = 0.05
  - bestcost = 10000
  - bestx = 6 , besty = 6
  - Range of Optimisation = $[-10, 10]$
- The Final PLOT after Gradient_Descent is :



- The RESULT is :
  - Optimum of f4 occurs at **(x,y) = (-1.5707963267948954,-1.5707963267948957)**
  - Optimum value of **f4 = -1.0**

PROBLEM 4

- This function is a 1D ***Trigonometric Function***. The **gradient** has not been provided, so I created the `f5_prime() function`.
- Since this is a 1D function, I have used grad_Desc_1D() to implement Gradient Descent.
- The parameters I use for this function are :
  - LR = 0.2
  - bestcost = 1000
  - bestx = 1, besty = 1
  - Range of Optimisation = $[-pi, pi]$
- The reason for a smaller learning rate is because this function, being trigonometric, must have a lots of peaks and valleys and hence if we take longer steps, we might end up at the wrong optima. So if we take smaller Learning rate, the steps will also be smaller.
- The Final PLOT after Gradient_Descent is :



- The RESULT is :
  - **Optima of f5 = -4.045412051572552**
  - **Point of Optima = 1.661660812043789**

INSTRUCTIONS ON HOW TO RUN THE CODE:

- You need to have the `numpy`, `matplotlib` and `FuncAnimation` libraries in the environment.
- You can then run the code and you will be able to see the **final plots** and **optimum values**.