Assignment-5

Gradient Desceding

Mithesh M | EE22B060 ***

Problem 1: 1-D simple polynomial

1. Approach

Gradient Descent Funtion: The `grad_desc` function in the code performs gradient descent optimization.

`grad_desc(cfunc,xrange,initialx,lr)`

It takes the following parameters:

- `cfunc`: This is a function that represents the cost or loss function. In this case, it's defined as `f1(x)`, which is a quadratic function `x^2 + 3x + 8`.
- `xrange`: This is a list containing two elements, `[rangemin, rangemax]`, which define the range of x values for which the cost function will be plotted.
- `initialx`: This is the initial value of `x` from which the optimization will start.
- `lr`: This is the learning rate, which determines the step size in each iteration of gradient descent.

Derivative Function: `cfuncd(x)` function which is defined inside the `grad_desc` function calculates the derivative of `cfunc(x)` using the first principle deferentiation.

$$\lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Plot initialisation:

- The function cfunc(x) is plotted by generating random 100 values of x in the given range using `np.linspace()`.
- Also the plot for the dynamic marking of the points to find the bestx during animation is also initalised using

```
lnall,  = ax.plot([], [], 'ro-')
lngood, = ax.plot([], [], 'go', markersize=10)
```

`onestepderiv` Function:

- This function is called by the animation for each frame.
- It performs one step of gradient descent
- Updates bestx using the gradient descent formula:

```
bestx = bestx - cfuncd(bestx) * lr.
```

- Updates the position of the red dot (lngood) representing the current best point.
- Updates the line plot showing the trajectory (lnall).

Creating the Animation:

- `FuncAnimation` is used to create the animation. It calls the onestepderiv function for each frame.
- It runs for 100 frames.
- The animation is saved as animation1.gif using the Pillow writer at 2 frames per second.

Restrictions:

- It saves the final animations in gif format.
- It need pillow writer to save the animation (works fine in the jupyter server).
- While passing the learning rate we should be careful such that the plot don't jumps out the given range.
- The starting point (initial value) of x should be in the given range.

2. Output

The outputs for the starting value of x = 4 are:

- Best-x is = -1.5000000491745027
- Best-cost at best-x is = 5.750000000000002

The final plot:

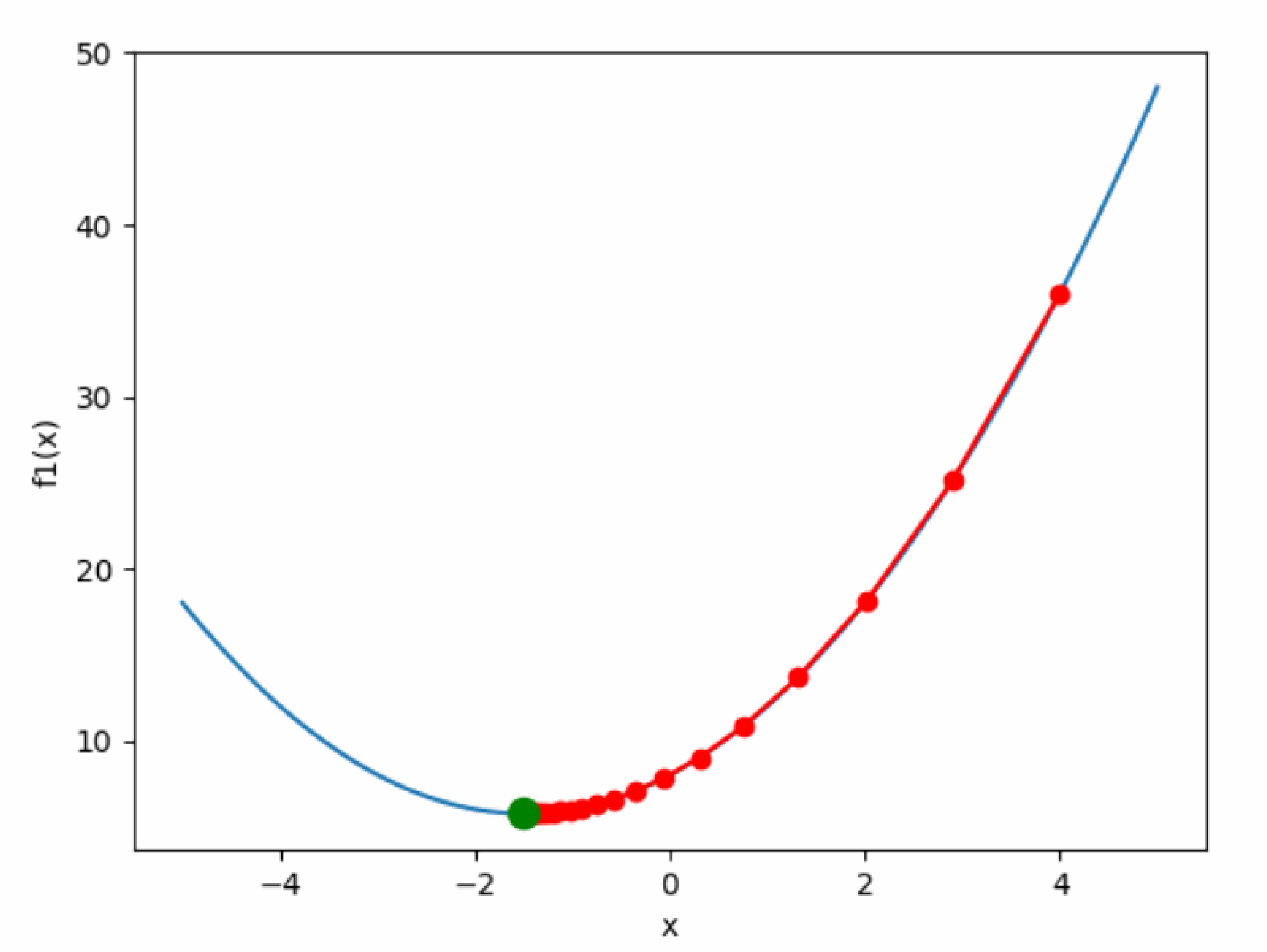


Figure 1: img

Problem 2: 2-D polynomial

Gradient Descent Funtion: The `grad_desc` function in the code performs gradient descent optimization.

`grad_desc(cfunc, cfuncd_x, cfuncd_y, xlim, ylim, initial_vals, lr):`

It takes the following parameters:

- `cfuncd_x` and `cfuncd_y`: These parameters are functions that compute the partial derivatives of the objective function with respect to x and y, respectively. In the code, `cfuncd_x` corresponds to `df3_dx(x, y)` and `cfuncd_y` corresponds to `df3_dy(x, y)`.
- `xlim` and `ylim`: It is the list ocontaining the range of x and y respectively.
- `inititial_vals`: It is the list containing the starting value of x and y as first element and second element respectively.
- Other parameters are similar the one explained in the problem-1.

Plot initialisation:

- xbase and ybase are created by 100 - x and y values within the specified ranges xlim and ylim.

- X and Y are created using np.meshgrid to represent a grid of (x, y) coordinates.
- Z is computed using the objective function cfunc based on the grid of (x, y) values.
- A 3D plot is initialized with the surface plot of the objective function.

`onestepderiv` Function:

- This function is ran in for loop for 10,00,000 to give better approximation.
- It performs one step of gradient descent
- Updates bestx and besty using the gradient descent formula:

```
bestx = bestx - cfuncd(bestx) * lr.
besty = besty - cfuncd(besty) * lr.
```

- Updates the position of the red dot (lngood) representing the current best point.
- Updates the line plot showing the trajectory (lnall).

Animation:

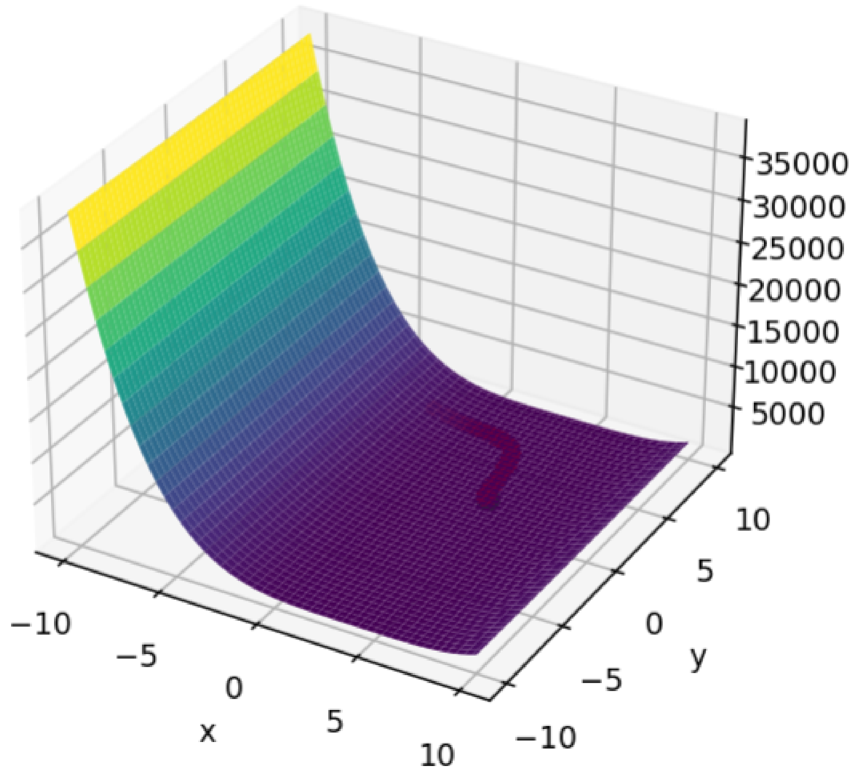
- I have commented out the animation in the code as it takes long time to run the program.
- In order the run the animation part, uncomment the part which is described as animation in the program and comment the for loop which calls the onestepderiv function.

Restrictions:

- The animation part takes long time to run.
- It need pillow writer to save the animation (works fine in the jupyter server).
- Learning rate is taken very low (0.001). For greater values of learning rate the program overflows out of the range.
- The starting point (initial value) of x should be in the given range.

Results: The output for starting points x = -7 and y = 7 are :

- Best-x = 3.9888197289788336
- Best-y = 2.000000000000111
- Best-cost at best-x and best-y is = 2.0000000156246642



The final plot obtained is:

Problem 3: 2-D function

Gradient Descent Funtion:

- Similar to that mentioned in problem 2

Plot initialisation:

- Similar to that mentioned in problem 2

`onestepderiv` Function:

- This function is ran in for loop for 1,000 to give better approximation.
- It performs one step of gradient descent
- Updates bestx and besty using the gradient descent formula:

```
bestx = bestx - cfuncd(bestx) * lr.
besty = besty - cfuncd(besty) * lr.
```

- Updates the position of the red dot (lngood) representing the current best point.
- Updates the line plot showing the trajectory (lnall).

Animation:

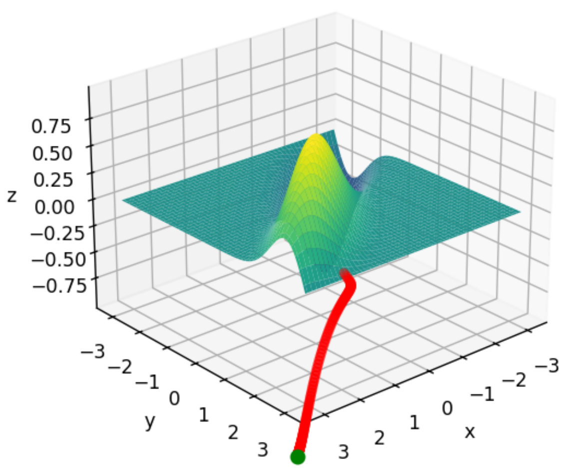
- I have commented out the animation in the code as it takes long time to run the program.
- In order the run the animation part, uncomment the part which is described as animation in the program and comment the for loop which calls the onestepderiv function.

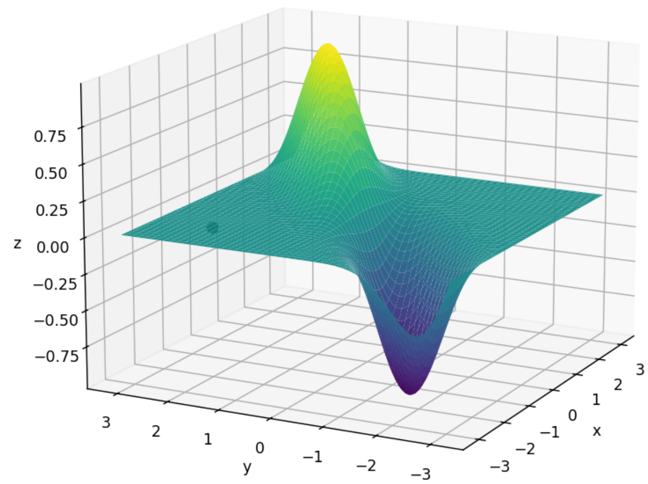
Assumption:

- Since range of y is not given, I have assumed the range of y is same as that of x.

Restrictions:

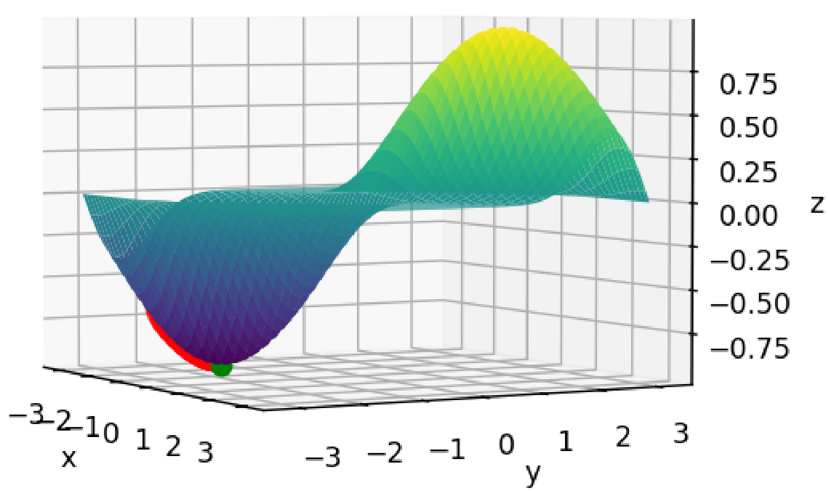
- The animation part takes long time to run.
- It need pillow writer to save the animation (works fine in the jupyter server).
- The starting point (initial value) of x should be in the given range.
- There is a possibility of the plot to overflow when the starting point lies in a certain region, eg: (3,2).(diagram on left)
- There is a possibility of the plot to remain static when the starting point lies in a certain region, eg: (-2,2).(diagram on right)





Results: The output for starting points x = -2 and y = 0 are :

- Best-x = -1.5707963267948943
- Best-y = -1.5707963267948948
- Best-cost at best-x and best-y is = -1.0



The final plot obtained is:

Problem 4: 1-D simple trigonometry

1. Approach

Gradient Descent Funtion:

- Similar to that in Problem-1.

Derivative Function:

- Similar to that in problem-1

Plot initialisation:

- Similar to that in Problem-1.

`onestepderiv` Function:

- Similar to that in Problem-1.

Creating the Animation:

- The animation is saved as animation4.gif using the Pillow writer at 2 frames per second.

Restrictions:

- It saves the final animations in gif format.
- It need pillow writer to save the animation (works fine in the jupyter server).
- While passing the learning rate we should be careful such that the plot don't jumps out the given range.
- The starting point (initial value) of x should be in the given range.
- There are the two minimas in the given range. We have to explicitly pass the correct starting point to get the correct min value

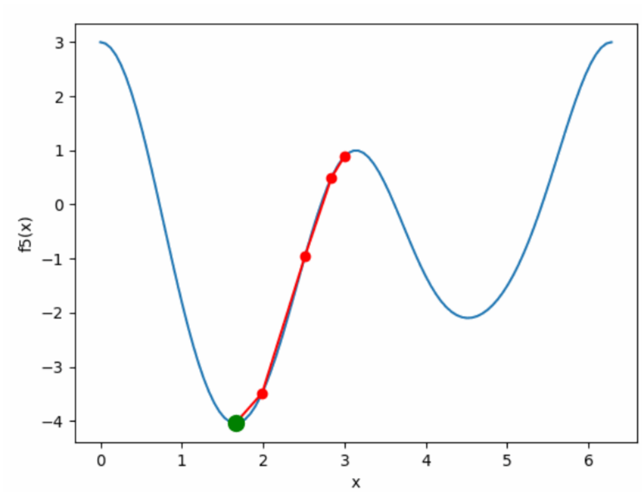
2. Output

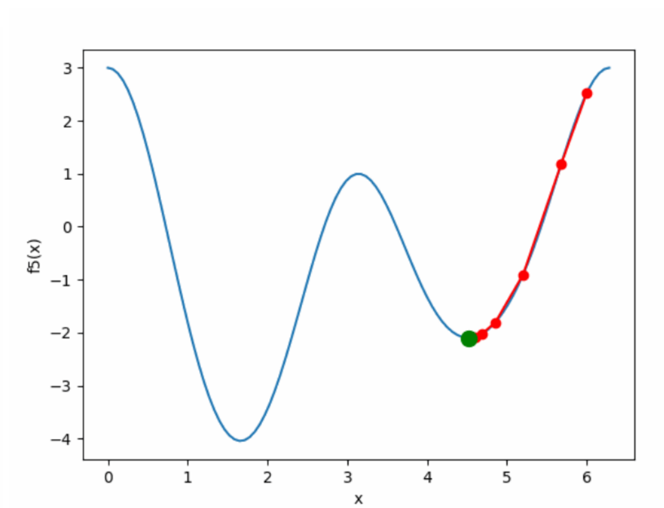
The outputs for the starting value of x = 3 are(Correct one):(Plot on the left)

- Best-x is = 1.6616607615674823
- Best-cost at best-x is = -4.045412051572538

The outputs for the starting value of x = 6 are:(plot on the right)

- Best-x is = 4.51901278180577
- Best-cost at best-x is = -2.097968346611848





s