

Assignment 5

Aim: Optimize a given function and find its minimum value using gradient descent

Approach

- For the given problem I have provided a single .py file in which all the given functions by sir are present.
- It can handle both 1D and 2D functions. The main function needs 3 parameters **function to optimize: f**, **a list containing calls of derivatives: der** and **limits within which to search: lim**.
- **f** can take any 1D or 2D function call
- **der** should either be empty list for a 1D function or a list with 2 derivative calls for x and y respectively for a 2D function.
- **lim** list should contain a list of limits of x and y values. If 1D only one element.
- For all the functions starting point is a random value from the range provided thus there might be different outputs as gradient decent largely depends on the initial guess.
- You can change the no. of iterations in the function as well.
- Gradient descent largely depends on the initial guess thus it is not that dependable.
- After the execution of the program it will output a file named **"Animation.gif"**
- All the animations for the given functions can be accessed by uncommenting the function calls at the bottom.
- It may take like 10 seconds for the animation gif to be saved
- Custom derivative using **Five-point stencil method**

```
def derivative(f, x):  
    h = 1e-10  
    return (-f(x + 2 * h) + 8 * f(x + h) - 8 * f(x - h) + f(x - 2 * h)) / (12 * h)
```

Optimization Reports

1D Functions

- For these functions derivatives with respect to x were not given.

1. Problem 1


```
def f1(x):  
    return x ** 2 + 3 * x + 8  
  
xlim1 = [-5, 5]
```

- For this problem the graph is -  Graph 1
- Minimum = 5.750000000000028 at $x = -1.4999998324425592$

2. Problem 5

```
def f5(x):
    return np.cos(x) ** 4 - np.sin(x) ** 3 - 4 * np.sin(x) ** 2 + np.cos(x) + 1

xlim5 = [0, 2 * np.pi]
```

- For this problem there are two graphs, dependent on the initial random value -  Graph 2
- Minimum = -4.0454120515725425 at $x = 1.6616608535627833$

 Graph 3

- Minimum = -2.097968346611845 at $x = 4.51901289203324$
- There is a 3rd graph for this as well where the initial guess results in 0 derivative value and thus the gradient descent stops.

2D Functions

- For these functions derivatives with respect to x and y were given.

1. Problem 3


```
def f3(x, y):
    return x ** 4 - 16 * x ** 3 + 96 * x ** 2 - 256 * x + y ** 2 - 4 * y + 262

These all are derivatives -

def df3_dx(x, y):
    return 4 * x ** 3 - 48 * x ** 2 + 192 * x - 256

def df3_dy(x, y):
    return 2 * y - 4




xlim3 = [-10, 10]
ylim3 = [-10, 10]
```

- For this problem there is a single 3D graph -  Graph 4
- Minimum = 2.0000008696432587 at $x = 3.9694623709895764$ and $y = 2.0000000000000107$
- In this function due to high derivative value sometimes the function overshoots.

2. Problem 4

```
def f4(x, y):
    return np.exp(-(x - y) ** 2) * np.sin(y)
```

```
def df4_dx(x, y):  
    return -2 * np.exp(-(x - y) ** 2) * np.sin(y) * (x - y)  
  
def df4_dy(x, y):  
    return np.exp(-(x - y) ** 2) * np.cos(y) + 2 * np.exp(-(x - y) ** 2) *  
    np.sin(y) * (x - y)  
  
ylim4 = [-np.pi, np.pi]  
xlim4 = [-np.pi, np.pi]
```

- For this function there are 3 graphs, due to value of the random initial guess - Graph 5
- Minimum = -0.0006334503494897465 at x = 0.9524758743124799 and y = 3.2959506697110292 Graph 6
- Minimum = -1.0 at x = -1.570796326794924 and y = -1.5707963267949185 Graph 7
- Minimum = -0.001237369930940134 at x = 1.1031057431514422 and y = -1.4835831195999876