Assignment 5

Gradient descent for single variable

The function generic_gradient_descent takes the function (`f`), its derivative (`df`), range of values within which to find the minima (`x_range`), learning rate (`learning_rate`), number of iterations (`num_steps`). It computes the function value at the present value of x and stores it in best_cost, present x in best_x. Then it adds these values to x_values and y_values which store the values over all iterations. It then finds the next value of x by subtracting it with the product of the derviative at the point and learning rate.
x = best_x - df(best_x) * learning_rate
This is continues till `num_steps` iterations are over. If any value goes out of given input range, we use break out of loop. Note that to get accurate minima the starting point should be appropriate.

In [ ]:
```
def generic_gradient_descent(frames, f, df, x_range, bestx, learning_rate, num_step
    global xall, yall, lnall, lngood
    xall = []  # Initialize xall as an empty list before the animation
    yall = []  # Initialize yall as an empty list before the animation
    for _ in range(num_steps):
        x = bestx - df(bestx) * learning_rate
        bestx = x
        bestcost = f(bestx)
        xall.append(bestx)
        yall.append(bestcost)
        lnall.set_data(xall, yall)
        lngood.set_data([bestx], [bestcost])
        if x < x_range[0] or x > x_range[1]:
            break
```

Gradient descent for a double variable function

The function gradient_descent takes the function (`f`), its partial derivative with respect to x (`df_dx`),its partial derivative with respect to y(`df_dy`), learning rate (`lr`), number of iterations (`num_steps`), starting points for x and y (`best_x,best_y`). It computes the function value at the present value of `x,y` and stores it in best_cost, present value of x in best_x, present value of y in best_y. Then it appends these values to x_values, y_values and z_values which store the values over all iterations. It then finds the next value of x by subtracting it with the product of the partial derviative wrt x at the point and learning rate.
x = best_x - df_dx(best_x, best_y) * lr
Similarly it uses gradient descent along the y axis to find new value of y. This is continues till `num_steps` iterations are over. Note that to get accurate minima the starting point should be appropriate.

In [ ]:
```
def gradient_descent(frames, f, df_dx, df_dy, bestx, besty, lr, num_steps):
    global xall, yall, zall, lnall, lngood
    xall.append(bestx)
    yall.append(besty)
    zall.append(f(bestx, besty))
    lnall.set_data(xall, yall)
    lnall.set_3d_properties(zall)
    for _ in range(num_steps):
        x = bestx - df_dx(bestx, besty) * lr
        y = besty - df_dy(bestx, besty) * lr
        bestx = x
        besty = y
        z = f(x, y)
        bestcost=z
        xall.append(x)
```

```
        yall.append(y)
        zall.append(z)
        lnall.set_data(xall, yall)
        lnall.set_3d_properties(zall)
        lngood.set_data([bestx], [besty])
        lngood.set_3d_properties([bestcost])
```

Restrictions:

- These functions are defined to find the local minimas and not global minimas. Thus according to initial value, they get struck at a minima.
- One should give appropriate learning rate as a huge learning rate doesn't letvthe function converge at minima
- Using a very small learning rate could increase time to converge
- It could get struck at saddle points(point of inflection) though it is not a point of minima
- Gradient descent requires the function to be differentiable

Deriavtive of a function

The derivative can to found close to real values by using
f'(x)=f(x+h)-f(x-h)/2*h
for a very small value of h(1e-13)

In [ ]:
```python
import numpy as np
```

In [ ]:
```python
def f1(x):
    return x ** 2 + 3 * x + 8
```

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

In [ ]:
```python
def deriv(x,f):
    df=(f(x+1e-13)-f(x-1e-13))/2/1e-13
    return df
```

In [ ]:
```python
print(deriv(2,f1))
```

```
6.998845947236987
```

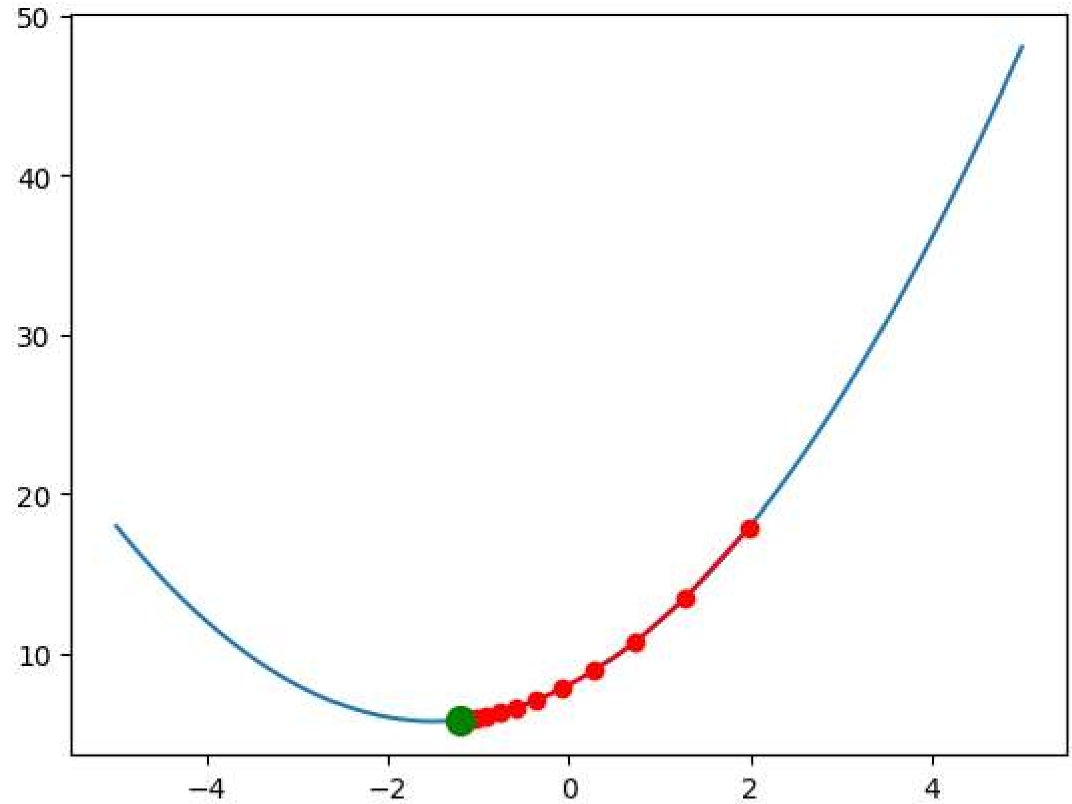
In [ ]:
```python
print(deriv(np.pi/2,f5))
```

```
-0.9992007221626409
```

We see that it gives values correct to a great precision

Problem 1 - 1-D simple polynomial

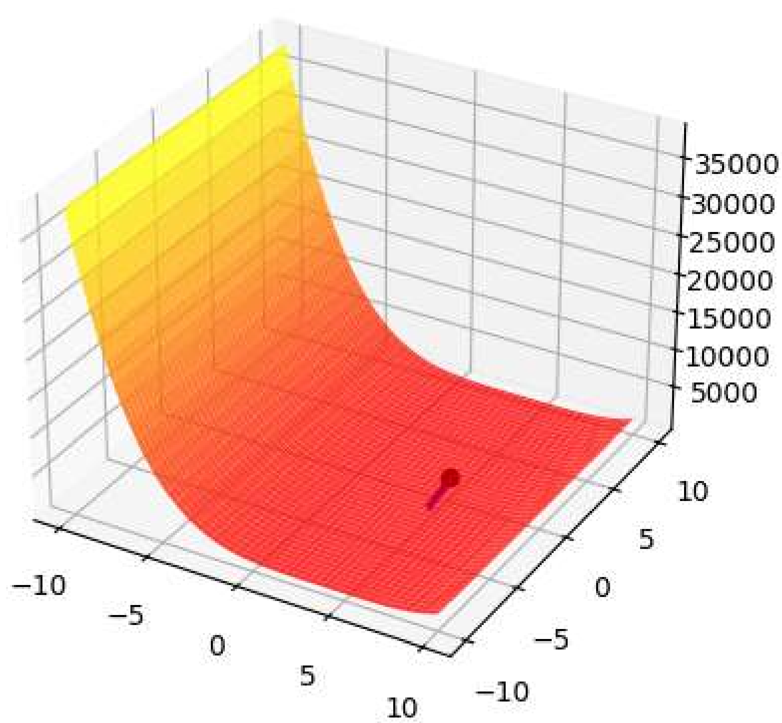
We use the generic_gradient_descent to get to the minima. Then we use the function onestepderiv(frames) to plot the image using FuncAnimation(). The values in each iteration has been updated to the lnall and the present value to lngood.



Minima of f1 occurs at x=-1.4965535749306293 and is y=5.750011877845759

Note:
Any starting point works as it has only 1 minima at x=-1.5 with value 5.75

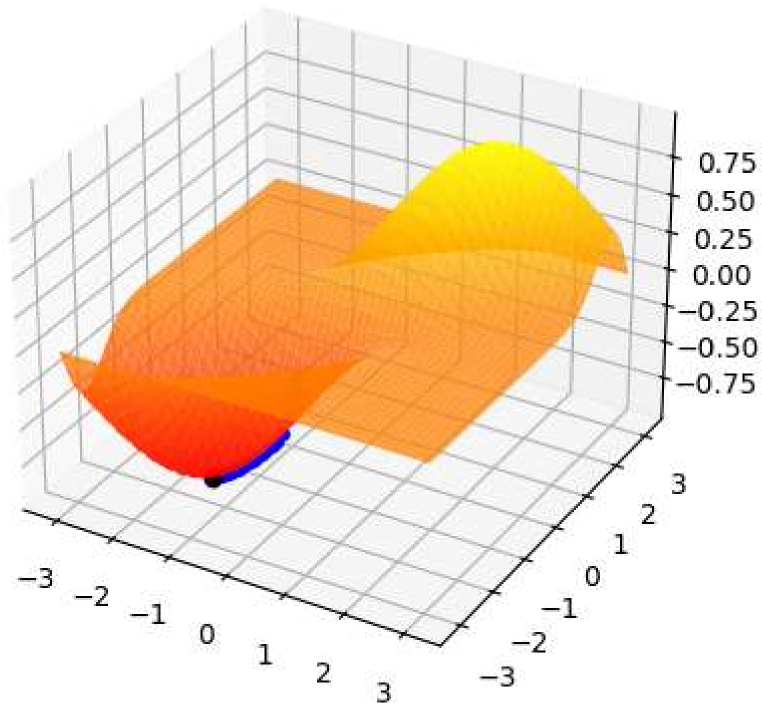
Problem 2 - 2-D polynomial



Minima of f3 occurs at x=4.3034016955593195,y=1.5705032704000002 and is z=2.1929411198543676

Note:
(5,-2) has been used as initial point. The minima is at (4,2) of value 2

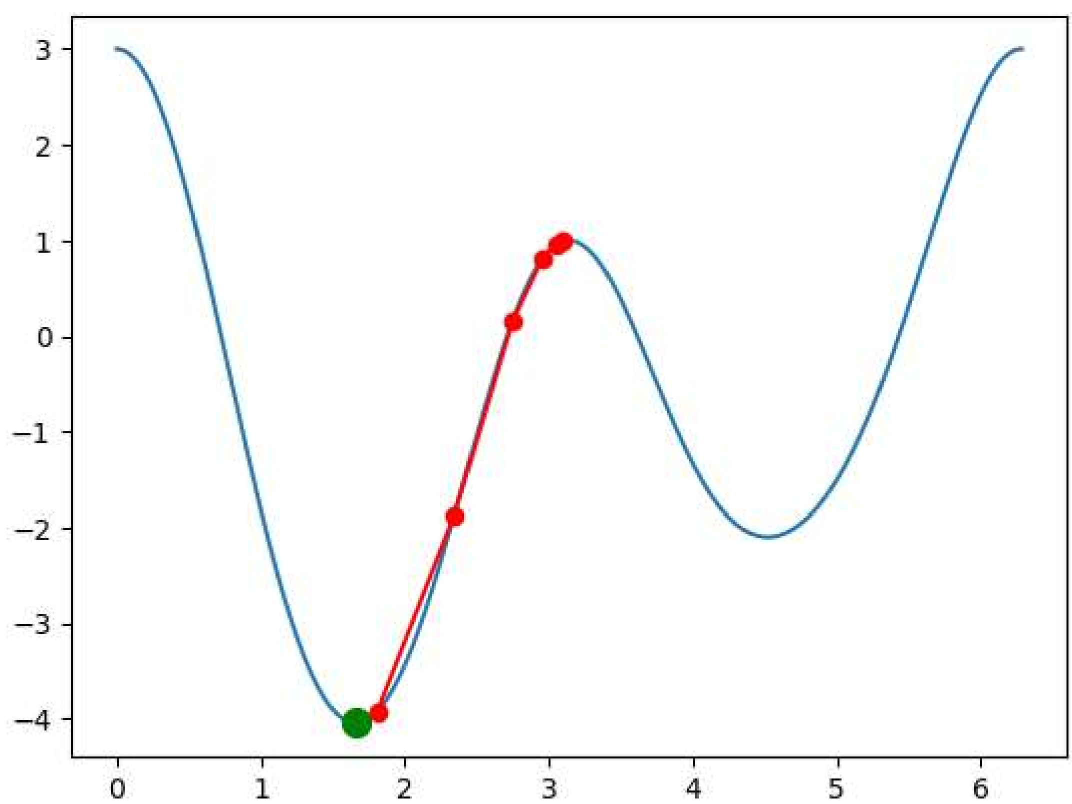
Problem 3 - 2-D function



Minima of f4 occurs at x=-1.5306563322436095,y=-1.5394590225008375 and is z=-0.9994315805498616

Note:
(0,-1.5) has been used as initial point. The minima is at (-π/2,-π/2) with value -1

Problem 4 - 1-D trigonometric



Minima of f5 occurs at x=1.661660906697188 and is y=-4.045412051572503

Note:
Any value before 3.1 in the given range for x gives the global minima at about (1.662,-4.045); Values greater than 3.1 leads to local minima at about (4.519,-2,098)