STU33009 Week 9 Assignment

Efeosa Louis Eguavoen - 17324649

April 1, 2020

1 Question 1

(a)

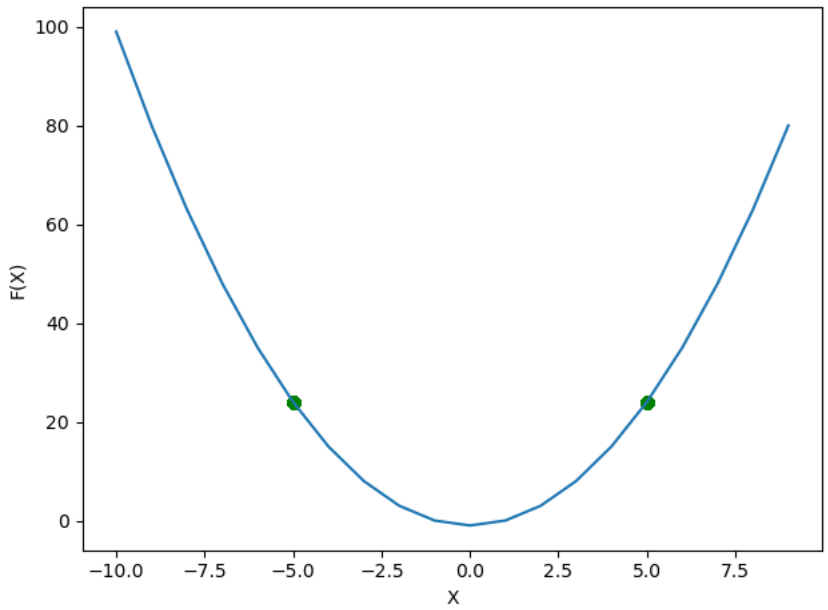
```
def gradient_descent():
    function = lambda x: (x**2) -1
    xvals = []
    yvals = []
    for i in range(-10,10,1):
        xvals.append(i)
    yvals = list(map(function,xvals))
   # plt.plot(xvals,yvals)
    #plt.show()
   # print(xvals,'\n',yvals)

    startX = 2.5
    learningRate = float(input('Enter Learning Rate: '))
    precision = 0.000001
    prev_size = 1
    max_iters = 1000
    cur_Iter = 0
    func = lambda x: 2*x

    while prev_size > precision and cur_Iter < max_iters:
        prev_x = startX
        inter = float(func(prev_x))
        startX = startX - learningRate * float(func(prev_x))
        prev_size = abs(startX-prev_x)
        cur_Iter = cur_Iter+1
        print("Iteration", cur_Iter, "\nX value is", startX)
    print("The local minimum occurs at", startX)
```

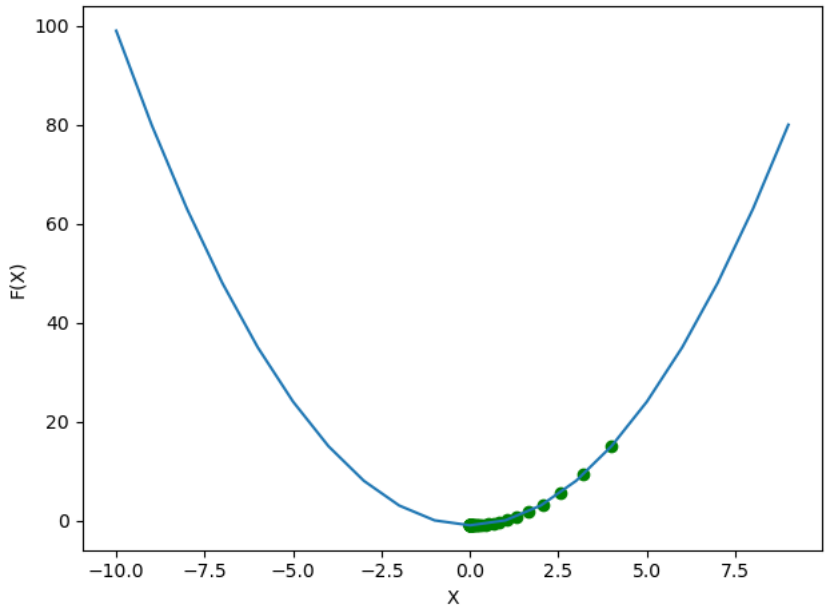
(b)

For a= 1 we get the following plot:



As we can see on the graph, F(x) swithces between -5 and 5 when the learning parameter is 1 as we essentially keep shooting past the minnimum each iteration through the loop.

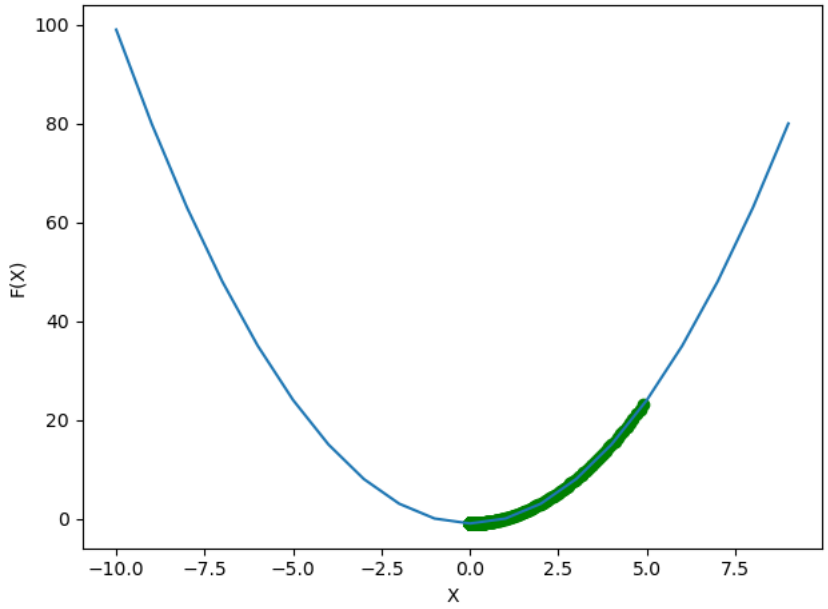
For a = 0.1 we get this plot:



From this graph, we can see the f(x) approaching our minimum each time we iterate through the loop. Our learning parameter enables us to approach the minnmum at a

good pace and enables the loop to exit in much fewer iterations than when compared to other 0.01 for instance.

For a = 0.01 we get this plot:



Again, we see f(x) approaching the minnimum but requires way more iterations of the loop to exit and progress is much slower than when we were using 0.1 as out learning parameter. We do eventually reach the minnimum but the number of iterations is way more than when we used 0.1.

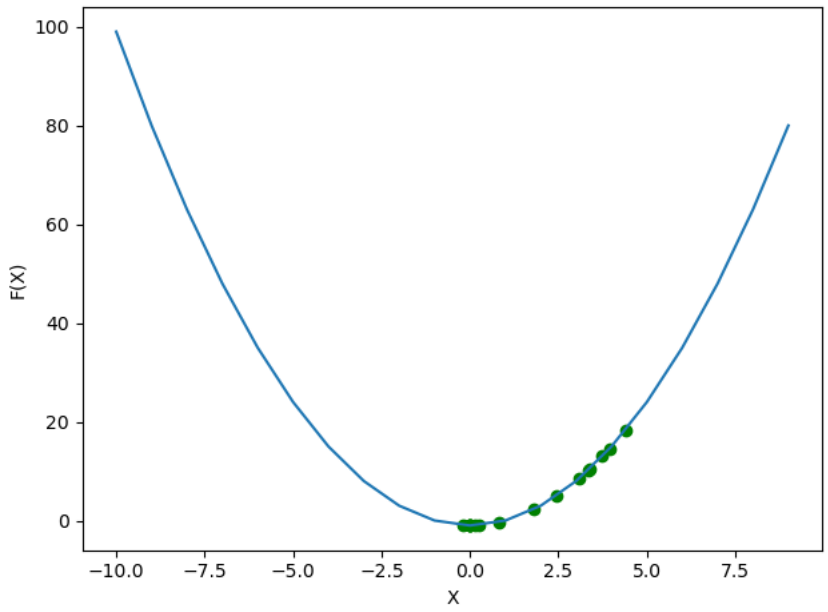
(c)

```
def gradient_descentRandom():
    function = lambda x: (x ** 2) - 1
    xvals = []
    yvals = []
    for i in range(-10, 10, 1):
        xvals.append(i)
    yvals = list(map(function, xvals))
    plt.plot(xvals, yvals)
    print(xvals, '\n', yvals)

    startX = 5
    precision = 0.000001
    prev_size = 1
    max_iters = 1000
```

```
cur_Iter = 0
derX = []
while cur_Iter < max_iters:
    randomX = random.uniform(startX-1,startX+1)
    if function(randomX) < function(startX):
        startX = randomX
        derX.append(startX)
    cur_Iter = cur_Iter + 1
derY = list(map(function, derX))
plt.scatter(derX, derY, c='g')
plt.show()
print("The local minimum occurs at", startX)
```

Example output:



We can see it's random as the gap between points isn't consistently getting smaller as a number is chosen at random each time in the vincinity of X.

(d)
The random approach can work when given enough iterations(in my code I gave it 1000 iterations) to approach the minnimum but it's a lot slower also there's no guarantee it will reach the actual minimum in the given amount of loops as it's totally random. Also there's no way of making the loop exit early reliably as there's no guarantee of the next point possibly increasing the value of x. The gradient descent approach works better as we can definitely approach the mininmum over a number of iterations but we need to

be careful to choose a learning value that won't overshoot the minimum. Also it's a lot faster as we can make it exit early if given a precision level we're looking for.

2 Question 2

(a)
To classify the hotel reviews, we can define a model

$$y = h\theta(x)$$

with it's parameters being

$$\theta 0, \theta 1...$$

Each of the values of theta equal to a value in the feature vector for that review.

$$h\theta(x) = sign(\theta 1X1, \theta 2X2...\theta nXn$$

(b)
We need to assume that the dependant variable is binary i.e that what output we get will either be positive or negative mapping each to 0 and 1 respectively.
(c)
?