Gradient descent Model for a linear least squares model

We will find the parameters in a linear least squares model using a gradient descent method

In practice, we have an analytics solution to the least squares model, we are looking at this approach to simply see how gradient descent works

Here is the set up for this exercise

We want a data set with a known solution to work with

generate n random x values between 0 and 10

generate y= m*x+b+error

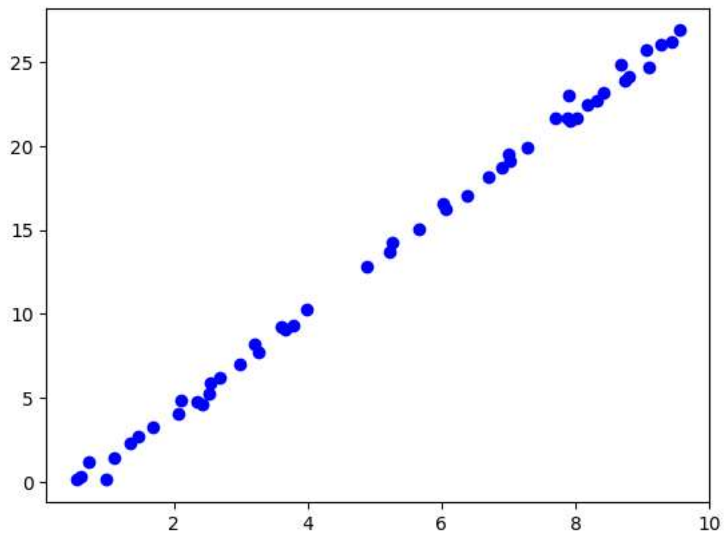
```
    where m and b are chosen and error is a random normal variable with mean=0 and std=sigma
```

```
import numpy as np

n=50

x=np.random.uniform(0,10,n)

m=3
b=-2
sigma=0.5

y= m*x + b+np.random.normal(0,sigma,n)
```

Plot y vs x, you can mess with n, m, b and sigma to change the data as desired

```
from matplotlib import pyplot as plt
%matplotlib inline

plt.plot(x,y,"bo")
```

```
     [<matplotlib.lines.Line2D at 0x7af7580d4370>]
```



Compute the means of x, y, x*y and x^2, since we use these over and over in the calculations

```
ybar=sum(y)/n
xbar=sum(x)/n
x2bar=sum(np.square(x))/n
xybar=sum(x*y)/n
```

Since we have the derived equations for m and b for a linear least squares model, we can compute the algebraic solutions to compare with the solutions we get by gradient descent

```
m=(xybar-xbar*ybar)/(x2bar-xbar*xbar)
print(m)
```

```
    2.9897936301281374
```

```
b=ybar-m*xbar
print(b)
```

```
    -1.8144944165378671
```

Here is gradient descent-

Complete the code below, so that this runs

Now use gradient descent method

a.) pick the initial guess at m and b, call them mg, bg

b.) set the tolerance and the delta values (control parameters in the algorithm)

c.) compute an initial L

d.) compute the derivatives and update mg and bg

e.) compute the next L

f.) Start a while loop, while abs(initial L - next L) <tolerance

```
 1.) set initial L to next L
 2.) find the derivatives and update mg, and bg
 3.) compute the next L
```

g.) Once a-f are working, alter my code to add a count of the number of steps needed to converge

h.) Try altering the values of tol and delta slightly to see what happens

```
#initial m and b,  called mg and bg for gradient
mg=0
bg=0

# descent control parameters
#tol is compared to the change in L,  the descent stops when the change
# in L is below the tolerance
# delta is the step size in the descent
tol= 1e-5
delta=1e-4 #the step at which we are moving

# compute the initial objective function the sum of squares
#  see the lecture notes for this week

L_now= sum((y-mg*x-bg)**2)  # fill in the equation for L_now here
print(L_now)


# compute the derivatives,  using xbar, ybar, xybar, x2bar,n and the
# current value of mg and bg
#   (these do appear in the lecture notes for this week)

dLdb=  2* sum(y-mg*x-bg) # fill in the equation for the derivative of L with respect to b
dLdm=  2* sum((y-mg*x-bg)*x)# fill in the equation for the derivative of L with respect to b

#update the mg and bg values

bg=bg-dLdb*delta
mg=mg-dLdm*delta

# compute the update objective function L,   calling it Lnext


L_next= sum((y-mg*x-bg)**2) # fill in the equation for L_now here
print("Starting objective function value")
print(L_next)

# now use a while loop.   repeat the loop while the absolute value, np.abs(L_now-L_next) is less
# thah the tolerance level.  When this value falls below the tolerance, we must be close to the minima
# stop the loop and print the results

while( np.abs(L_now-L_next)>tol):
    L_now=L_next
    dLdb=  2* sum(y-mg*x-bg) # fill in the equation for the derivative of L with respect to b
    dLdm=  2* sum((y-mg*x-bg)*x) # fill in the equation for the derivative of L with respect to b

    bg=bg+dLdb*delta
    mg=mg+dLdm*delta

    L_next= sum((y-mg*x-bg)**2) # fill in the equation for L_now here
    print(L_next)


print("\n\n")
print("Estimates of the slope and intercept")
print(mg)
print(bg)
```

```
7.95532011822712
7.955308686156663
7.955297306789655
7.955285979883088
7.955274705195139
7.955263482485077
7.955252311513251
7.955241192041166
7.955230123831378
7.955219106647594
7.95520814025455
7.95519722441809
7.9551863589051575
7.95517554348375
7.95516477792293
7.955154061992859
7.955143395464707
7.955132778110731
7.9551222097042205
7.955111690019534
7.955101218832049
7.9550907959182044
7.955080421055435
7.955070094022217
7.95505981459807
7.955049582563487
7.955039397700013
7.95502925979016
7.955019168617489
7.955009123966523
7.954999125622796
```