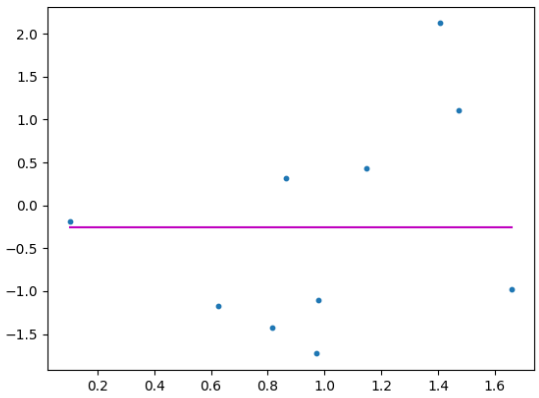
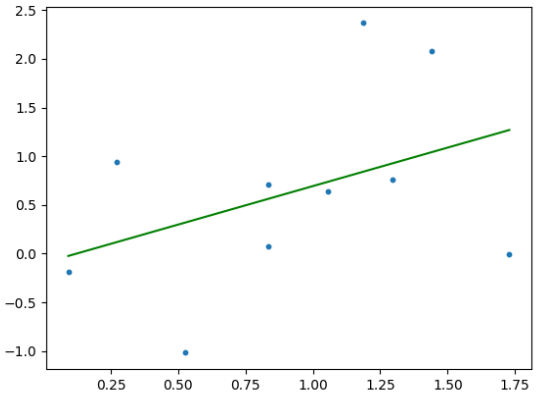
CMPE 442 Assignment #1

Yasemin Direk

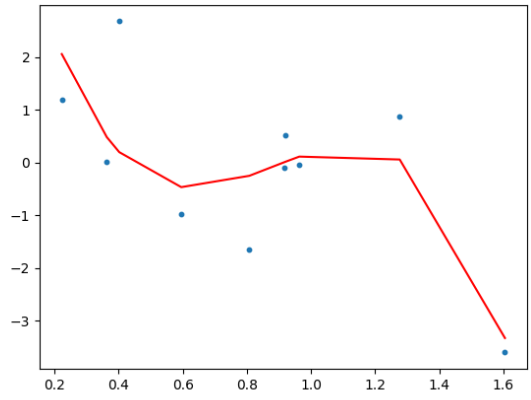
1) **I.** In the first part of the first question I used linear regression and polynomial features modes of python. Firstly, I generated synthetic data by using given function $y = \sin(2\pi x) + \varepsilon$, where m=10. Then, I applied d-order polynomial model to data set by using PolynomialFeatures and LinearRegression classes of Scikit. In first part I used m=10 and different d values such as 0, 1, 3 and 9, respectively. I first plotted the data on the same figure, then I also plotted separate graphs to see the difference more clearly.



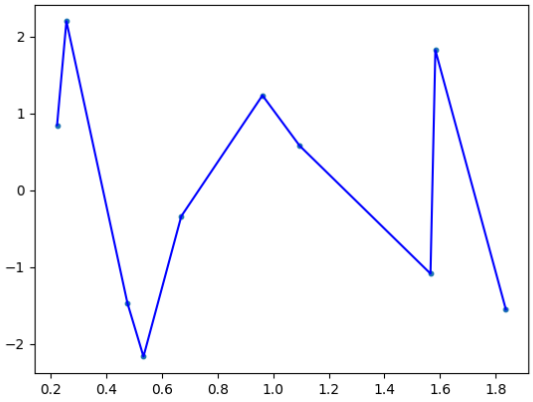
m = 10, d = 0



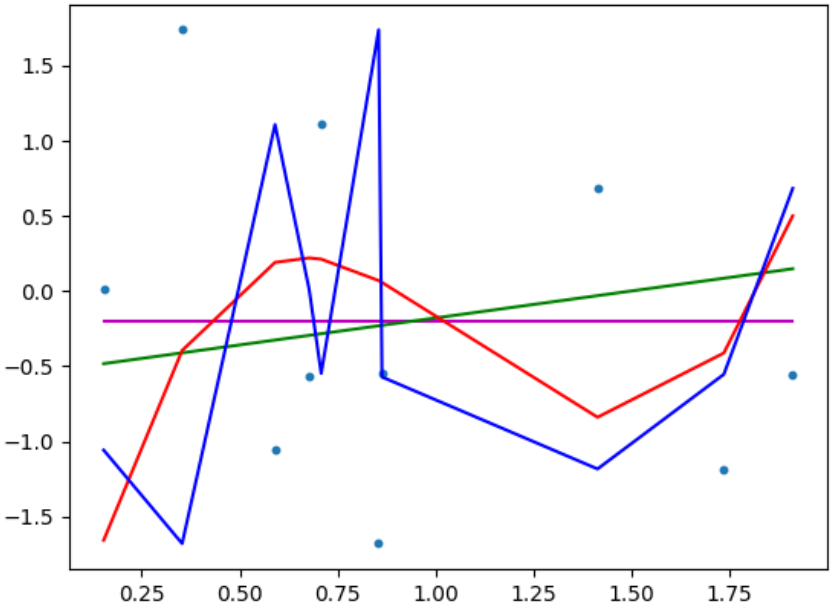
m = 10, d = 1



m = 10, d = 3

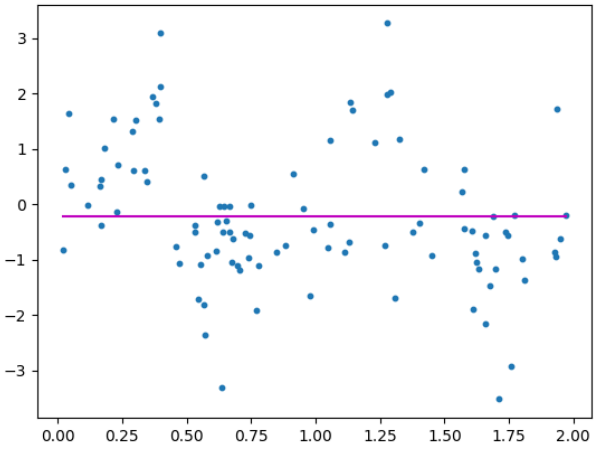


m = 10, d = 9

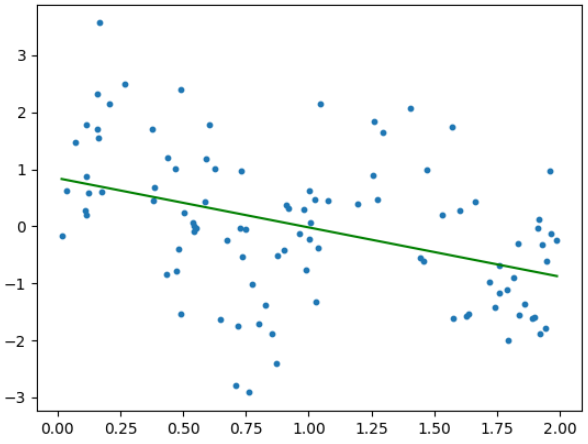


After creating the graph, I saw that while the number of training examples(m) is equal to 10, increasing the value of d causes sharp ups and downs in the graph. For the values d=0 and d=1 the line could not fit the data. So, when the straight line poorly fits nonlinear data, it is a case for underfitting. For the value d=3, the curve is closer to data points than linear plots. For the value d=9, the curve passes through more data points, but it causes sharp ups and downs in the graph. So, this is a case for overfitting. As a result, when the degree value is 0 and 1, the model is underfitting the training data. When the value for degree is 9, the model is overfitting the training data.

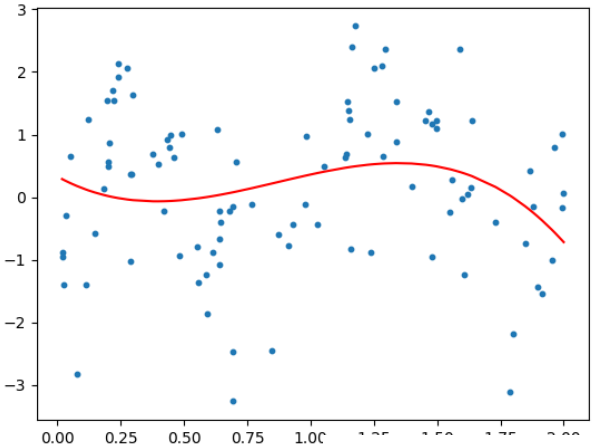
II. In the second part, I increased the number of training examples to m=100 and repeated the steps in part I. After creating new graph, I saw that for the values d=0 and d=1 the line still could not fit the data, but for the values d=3 and d=9 the curve passes through more data points and fits the data better. As a result, when the number of training examples increase, the model become more generalized, but this only works for the overfitting problem. Because if the model is underfitting the training data as in the values d=0 and d=1, adding more training examples will not help. So, with the increased number of samples, we can solve the overfitting problem we encountered with larger d values.



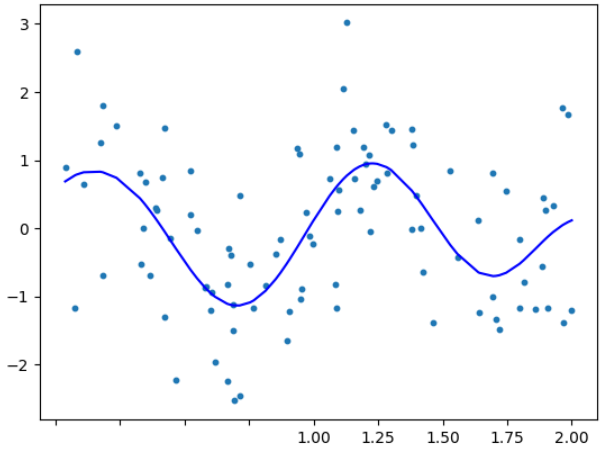
m = 100, d = 0



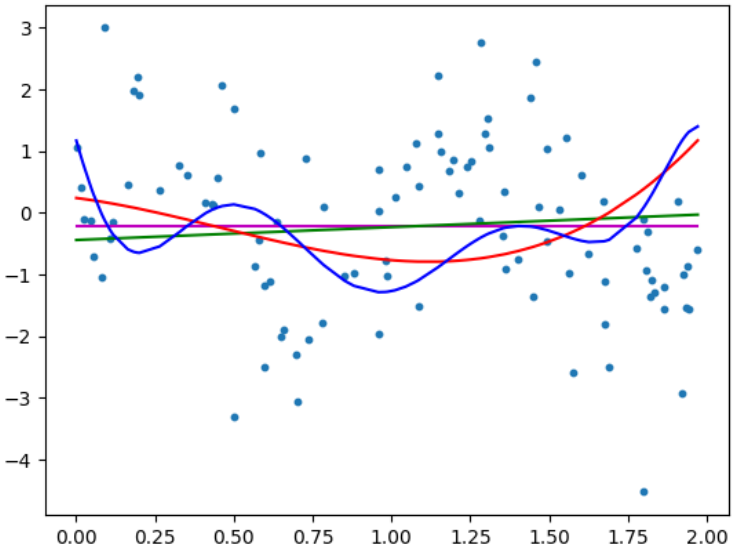
m = 100, d = 1



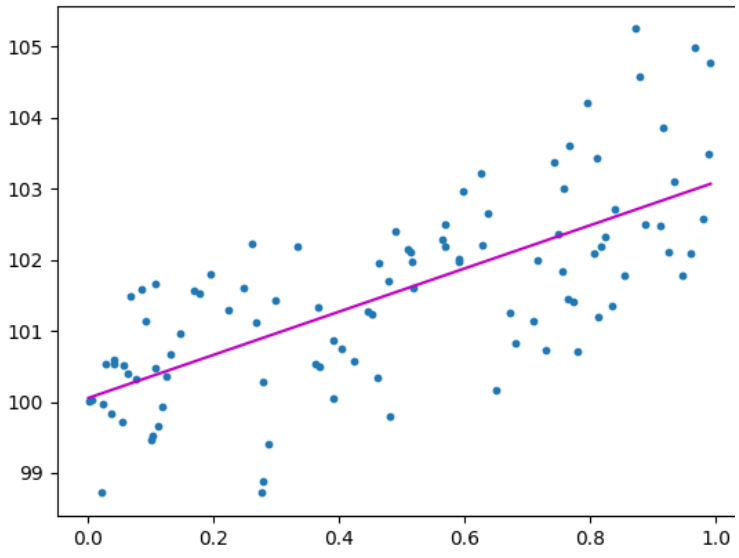
m=100,d=3



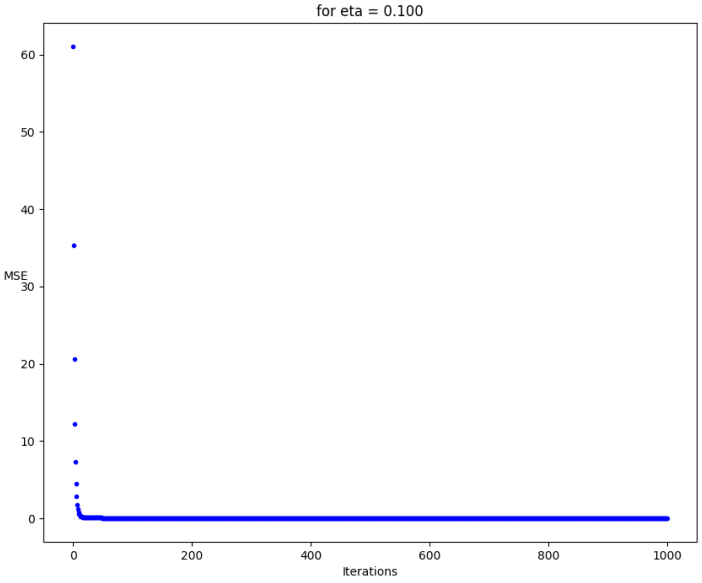
m=100, d=9



2) I. Firstly, I generated synthetic data by using given function $y = 100 + 3x + \varepsilon$, where m=100. Then, I implemented linear_regression function on this dataset using batch gradient descent. Then, I called the function with the values X, y, iterNo = 1000, eta = 0.1. Lastly, I plotted the data and the straight line on the figure and I get the hypothesis function by calculating theta[0] and theta[1].



$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$



MSE-iteration graph for eta = 0.1

(For part II)

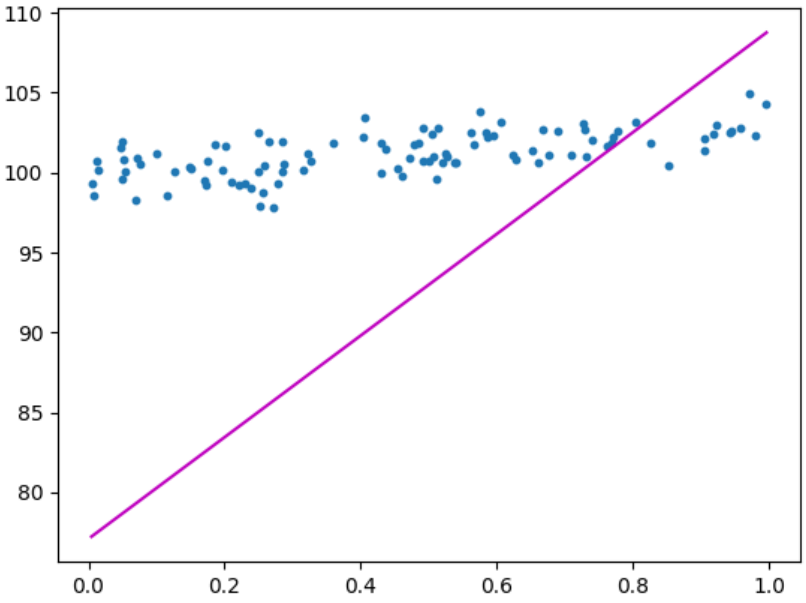
For iterNo = 1000, eta = 0.1

Hypothesis function:

h(x) = 100.0351631 + 3.31490545 * x

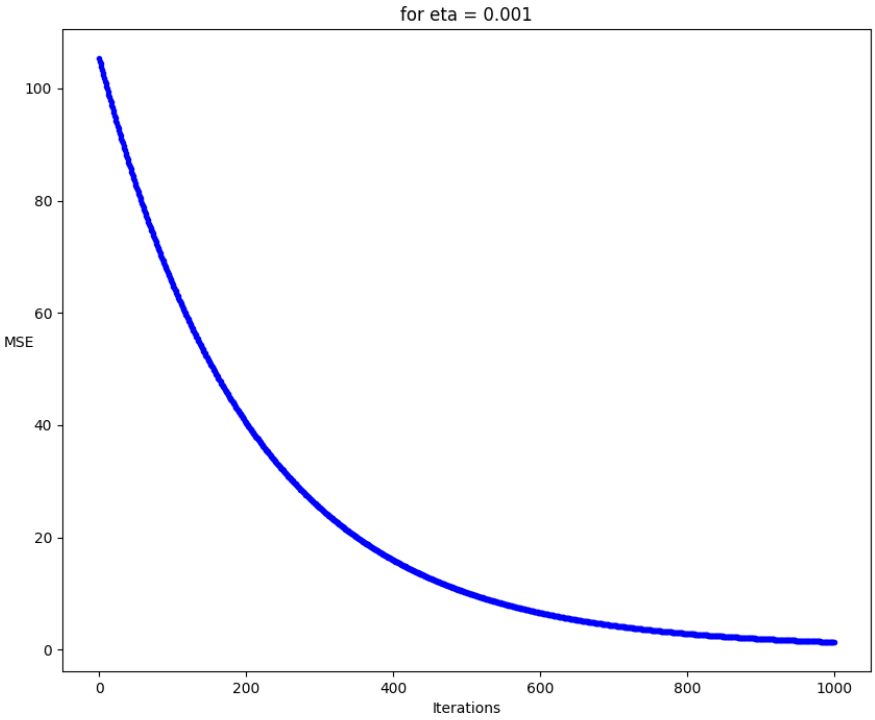
II. In the second part, I added function statement to linear_regression function that computes MSE for each iteration. I also updated the return parameter to theta and MSE. Then, I plotted the results for eta = 0.1, eta = 0,001, eta = 0.01 and eta = 0.5 while iterNo = 1000. Finally, I get the hypothesis function for every eta by calculating theta[0] and theta[1]. Also, I generated MSE-iteration graph for every eta.

For iterNo = 1000, eta = 0.001



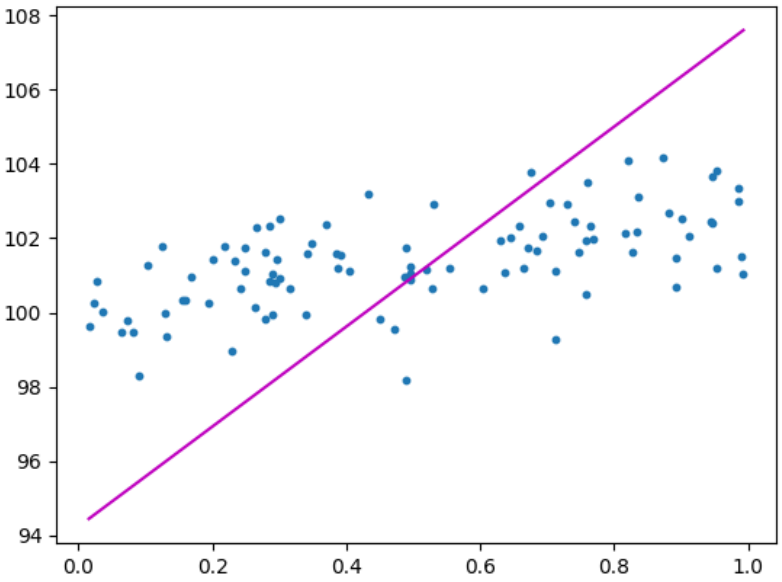
Hypothesis function:

h(x) = 75.64295137 + 33.595857*x



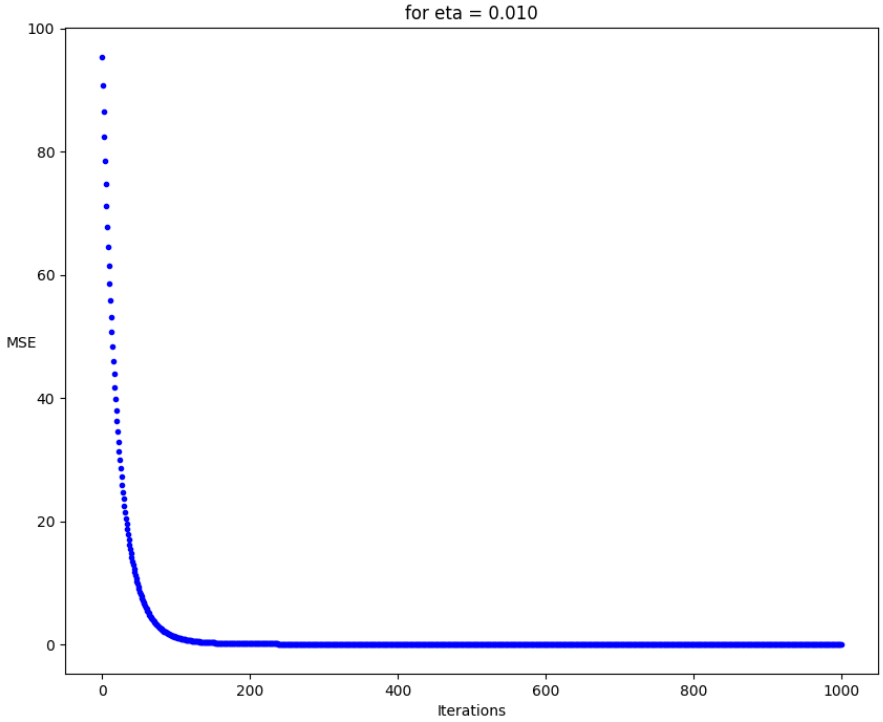
MSE-iteration graph for eta = 0.001

For iterNo = 1000, eta = 0.01



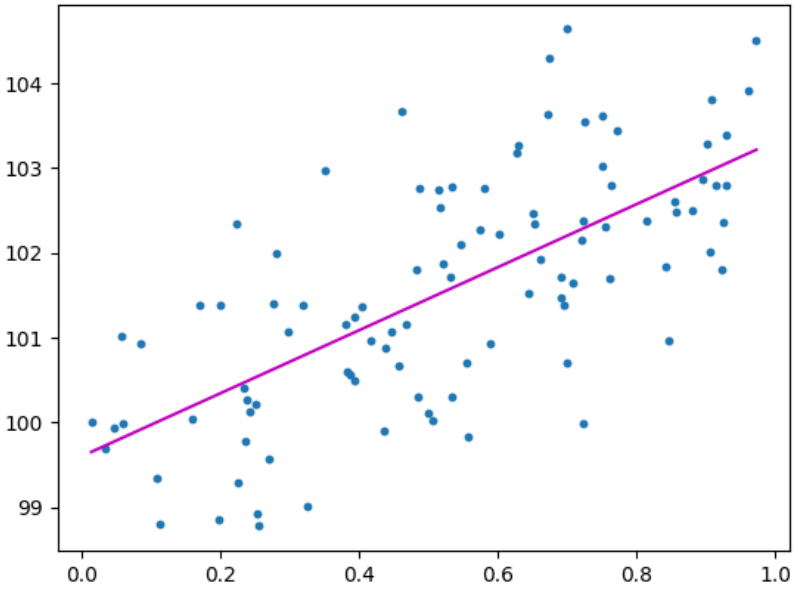
Hypothesis function:

h(x) = 94.50499278 + 13.29905356*x



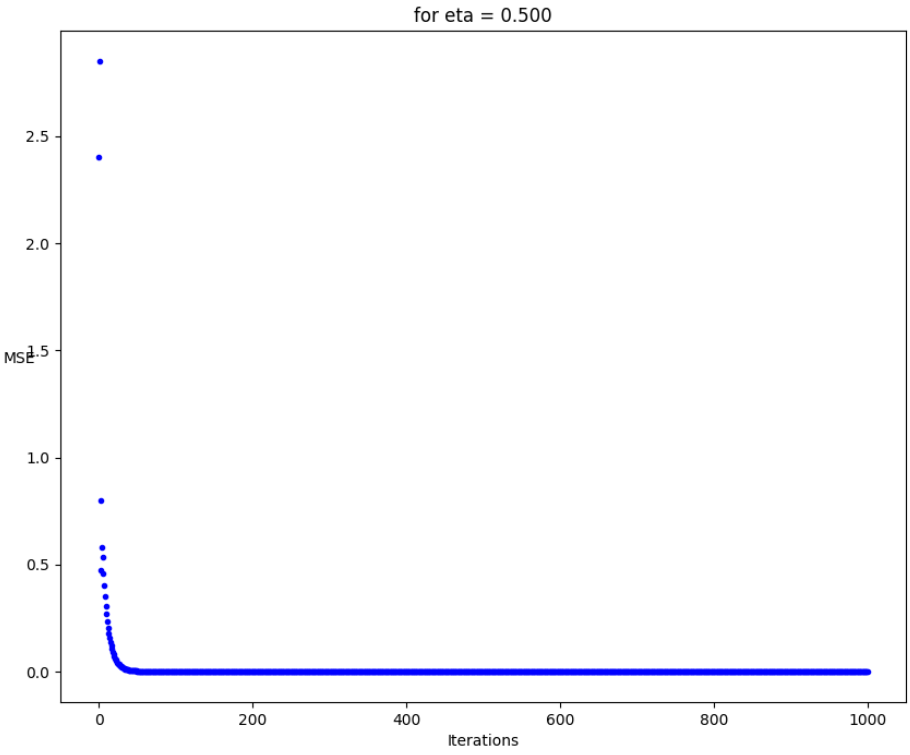
MSE-iteration graph for eta = 0.01

For iterNo = 1000, eta = 0.5



Hypothesis function:

h(x) = 99.68634761 + 3.47625993*x

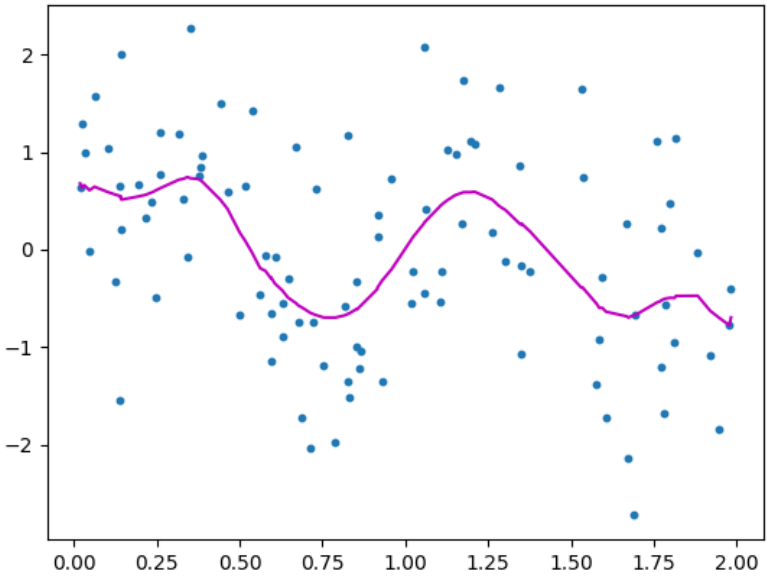


MSE-iteration graph for eta = 0.5

Discussion of Results

The learning rate hyperparameter controls the rate or speed at which the model learns. After I plotted the data and the straight line on the figure, I saw that the optimal value of the learning rate is 0.5 and 0.1. In this dataset, for the learning rate value 0.5 and 0.1, the straight line fits the data more. Also, the batch gradient descent algorithm that I used minimizes the cost function of our model at every step. Cost function indicates the difference between the predicted and the actual values for the dataset. Lower value of the cost function (MSE), the more accurate the prediction. When I looked at the MSE-iteration graphs, I saw that the MSE decreased as the number of iterations increased. In addition, MSE decreased as the learning rate increased in this dataset. As a result, if the learning rate is too low, the number of iterations to minimize the cost function could be high, but if the learning rate is too high, the cost function could increase.

3) I. In this part, first I generated synthetic data using a function $y = \sin(2\pi x) + \varepsilon$, where m=100. Then, I updated the linear regression function that I implemented in question 2. In the weigted_linear_regression function local weights are also considered in taking the gradient of a cost function. Then, I called the function with the values X, y, iteration_cnt=1000, eta=0.1. Lastly, I plotted the data and the curve on the figure, and I get the hypothesis function.

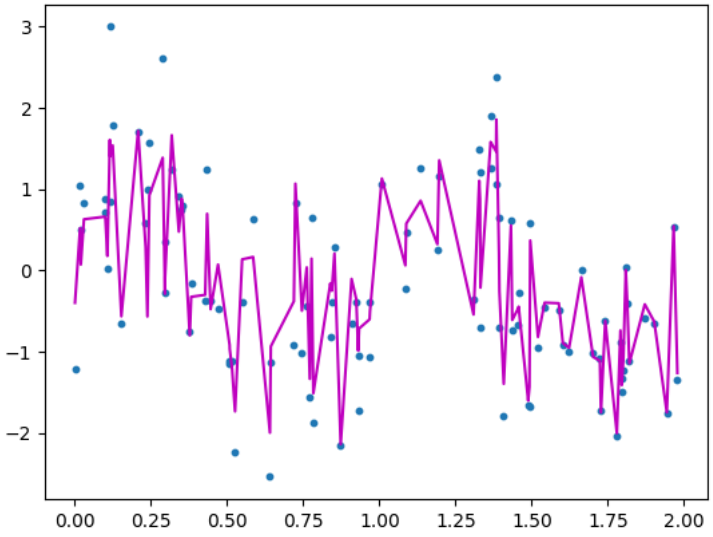


For iteration_cnt = 1000, eta = 0.4, tau=0.1

Hypothesis function:

h(x) = -2.04784665+ 1.86140495*x

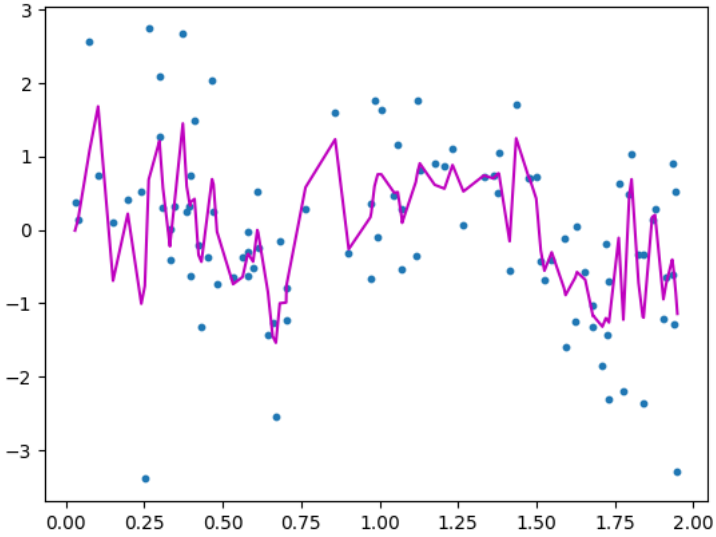
II. In this part, I repeated Part I with values τ = 0.001, 0.01, 0.3 ,1 and 10.



For iteration_cnt = 1000, eta = 0.4, tau=0.001

Hypothesis function:

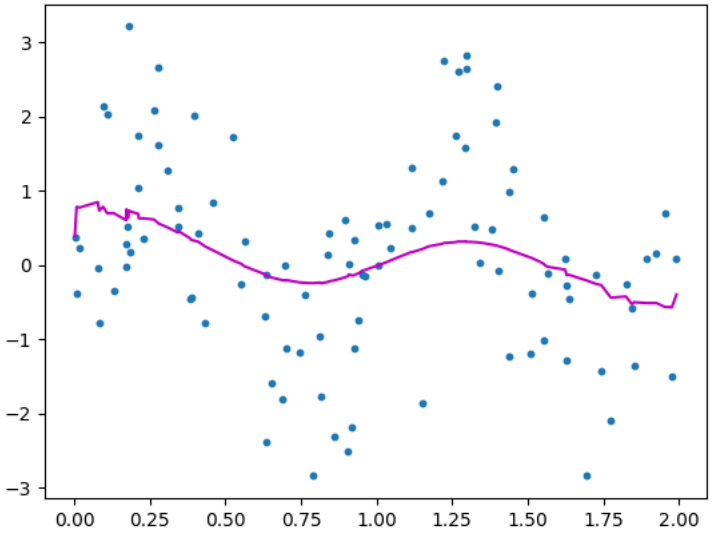
h(x) = 1.49418191+ 0.35951089 * x



For iteration_cnt = 1000, eta = 0.4, tau=0.01

Hypothesis function:

h(x) = 0.17299784 + -0.78262552 * x



For iteration_cnt = 1000, eta = 0.4, tau=0.3

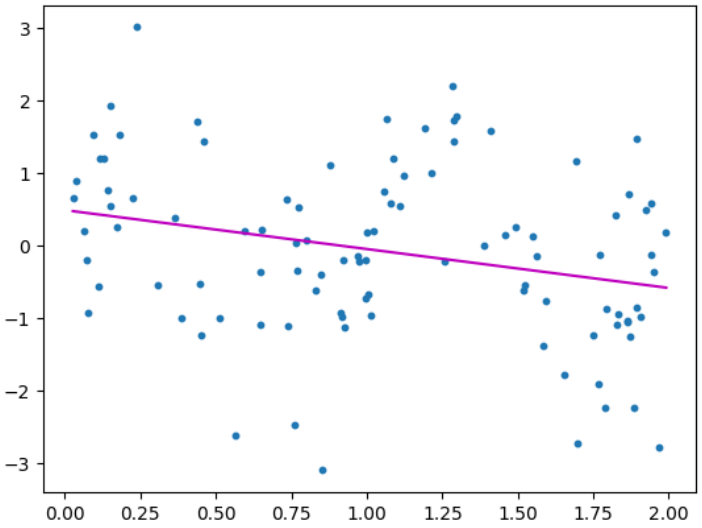
Hypothesis function:

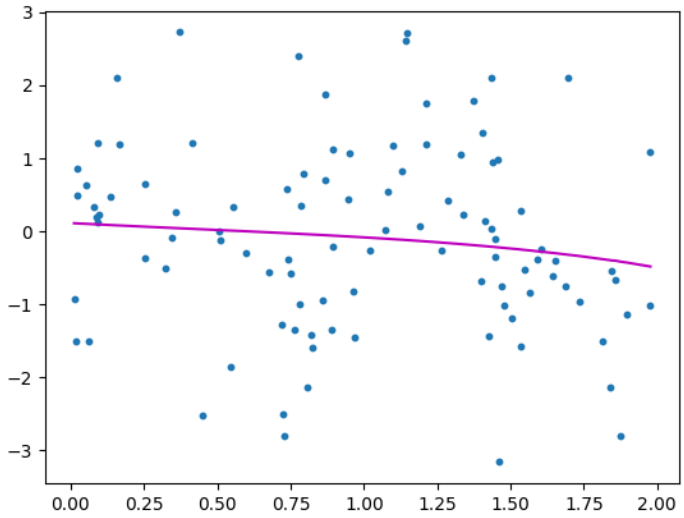
h(x) = 0.92193891 + -1.2703972 * x

For iteration_cnt = 1000, eta = 0.4, tau=1

Hypothesis function:

h(x) = 0.25127276 + -0.32642238 * x





For iteration_cnt = 1000, eta = 0.4, tau=10

Hypothesis function:

h(x) = 0.49227373 + -0.53649218* x

Discussion of Results: The difference of locally weighted linear regression from normal linear regression is that in the latter case all the weights were the same. Our data is non-linear, so we cannot use linear regression. In this part, I called the weighted_linear_regression function with fixed learning rate=0.4 and number of iterations=100. I just changed the value of hyperparameter tau(τ). When I plotted the results, I saw that with larger tau(τ) such as τ = 1 and τ = 10, the line become more general like linear regression. So, this causes underfitting problem. With smaller tau such as τ = 0.001 and τ = 0.01, the line seems perfect fit into dataset. But this causes overfitting problem. According to the results, locally weighted linear regression can solve the underfitting problem of linear regression, but it may cause overfitting problem. So, we should choose right parameters for our model. For this dataset, the best tau value to fit our synthetic data is 0.1. Because, with the τ=0.1 the curve smoothly fits the data.