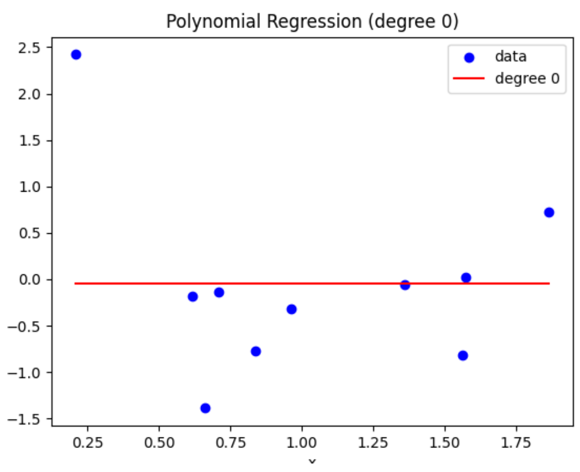
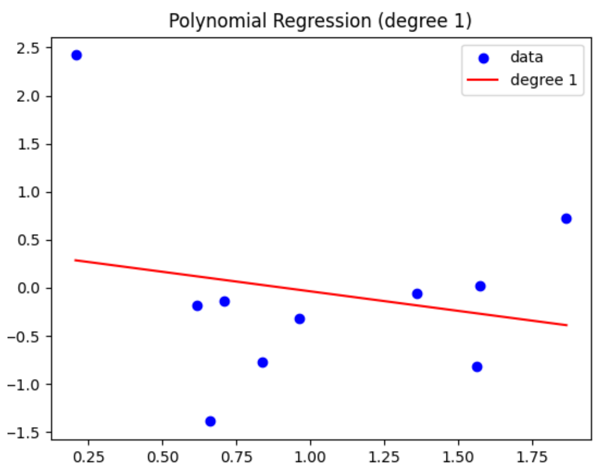
CMPE442 – Machine Learning
HW-1 Report

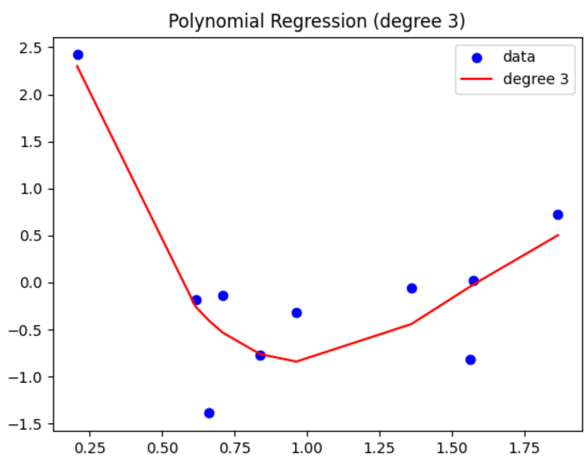
Zülal Karın 12622989076

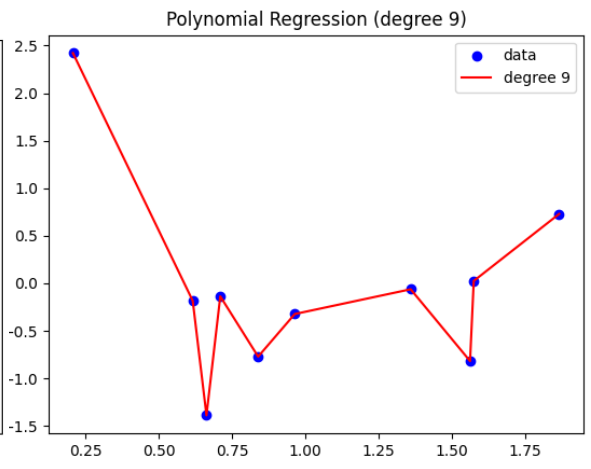
Question – 1

For this task, I am given the function y=sin(2πx)+ε and I am asked to produce synthetic data using it. After that, I will use d-order polynomial models to analyze the dataset and then plot the generated curves for different values of d. Finally, I must discuss the outcomes for various values of d and choose the right value for d.
In my python code, I first generated synthetic data using the given function with m = 10 and random noise ε. Next, I apply d-ordinary polynomial models to the dataset using PolynomialFeatures and LinearRegression classes. Here, I'm using 10 training examples and various d values (0, 1, 3, 9).



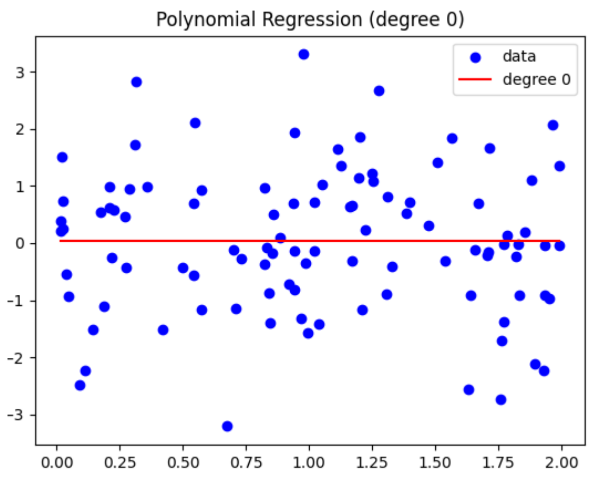


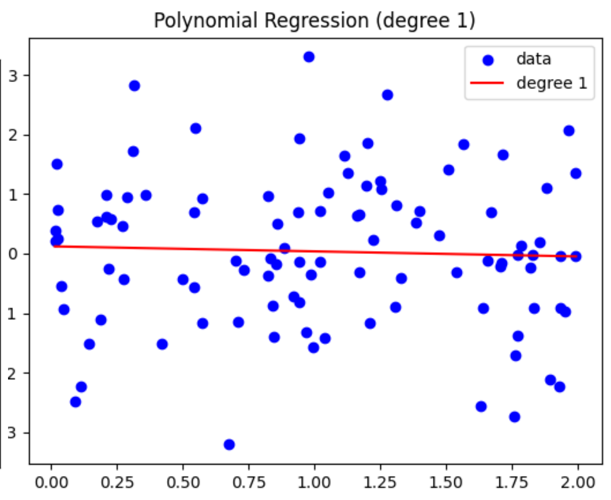


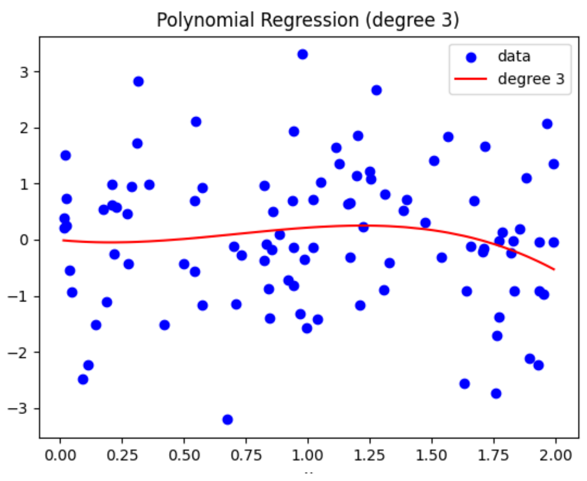


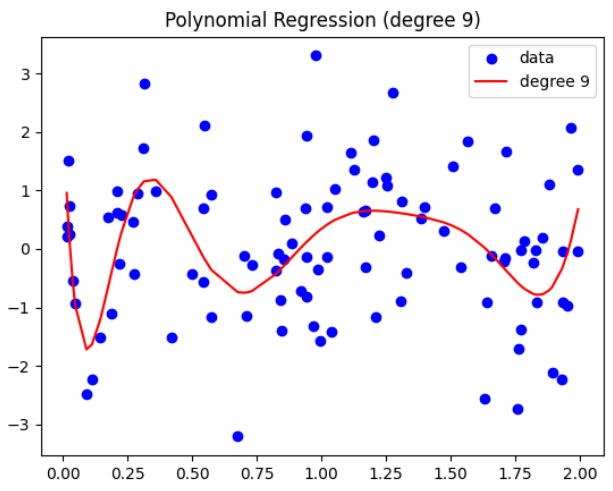
As a result, for d=0 the model is just a horizontal line that cannot fit the data. For d=1, the model is a straight line, which again does not fit the data exactly. For d=3, the model is a curve that fits the data better. For d=9, the model is a complex curve that fits the data perfectly. This means that the higher the Degree, the more accurately the curve fits the data.

II. Here, I increased the number of training examples from 10 to 100 and changed the degree to 0, 1, 3 and 9 provinces, respectively. The resulting graphics were as follows:









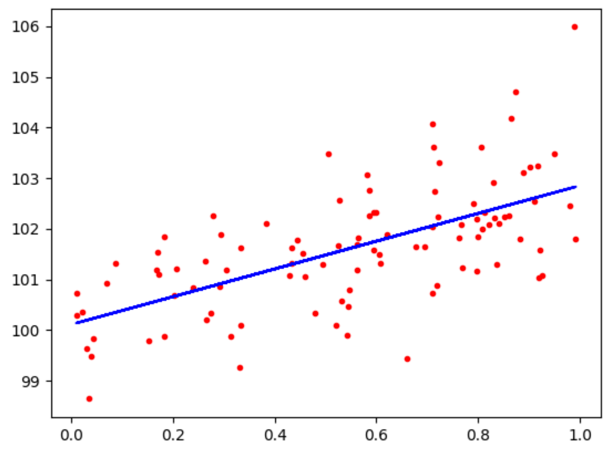
The graphic shows that increasing the number of training instances spreads the data out. We can see that as m increases, the performance of the polynomial models improves. However, even with m=100, we see that the problem of overfitting with large d values persists.

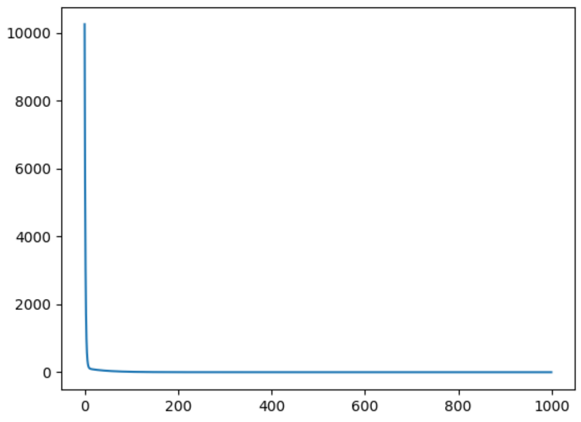
In general, increasing the number of training examples can improve the accuracy of our models by reducing the effects of random noise in the data. However, its important to balance this with the complexity of our models to avoid overfitting.

Question – 2

In this question, I used batch gradient descent to implement linear regression and plotted the data and the resulting straight line fit. For each iteration, the linear regression function returned the parameters theta and the mean squared error (MSE).

I initialized theta with random values and updated it using the gradient of the cost function with respect to theta for the batch gradient descent technique. The learning rate (eta) and number of iterations (iterNo) were both set to 0.1 and 1000. The result is presented below:



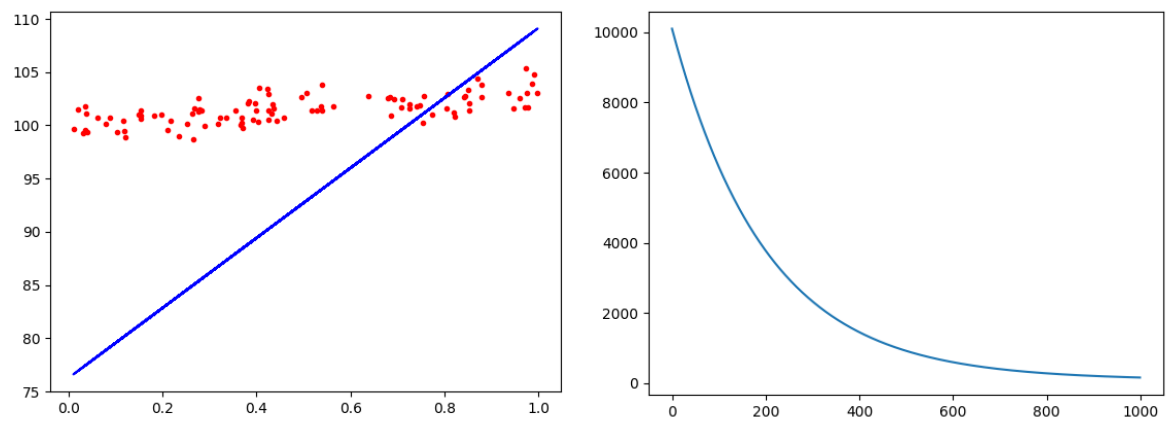


I can write the hypothesis function as: $h(x) = \theta\_0 + \theta\_1x$

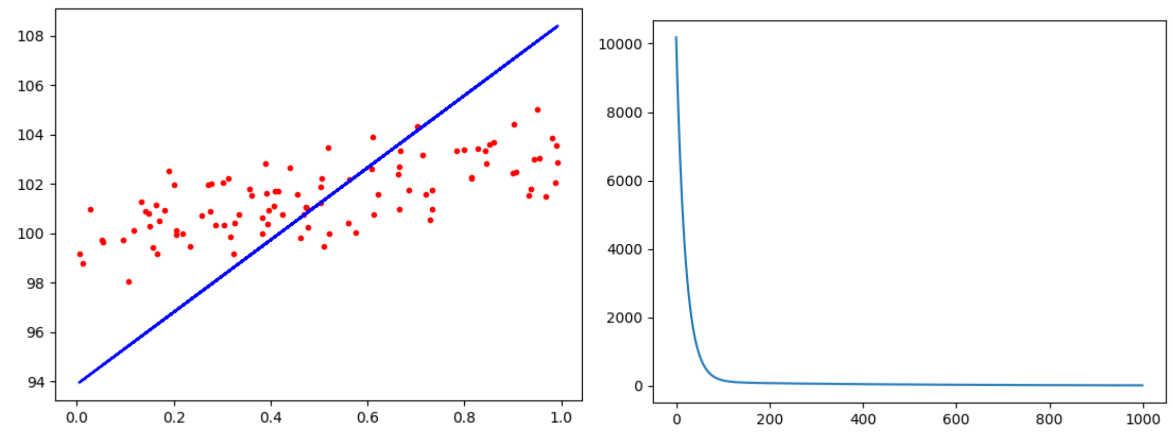
θ_0 is the intercept term and θ_1 is the coefficient for the feature X. In this question, the intercept term is 100 and the coefficient for the feature X is 3. So the hypothesis function for this dataset is: $h(x) = 100 + 3x$

Next, I tried the linear regression for various eta values (0.001, 0.01, and 0.5) while keeping iterNo constant at 1000. I plotted the data and the resulting straight line fit, as well as the MSE, for each eta value.

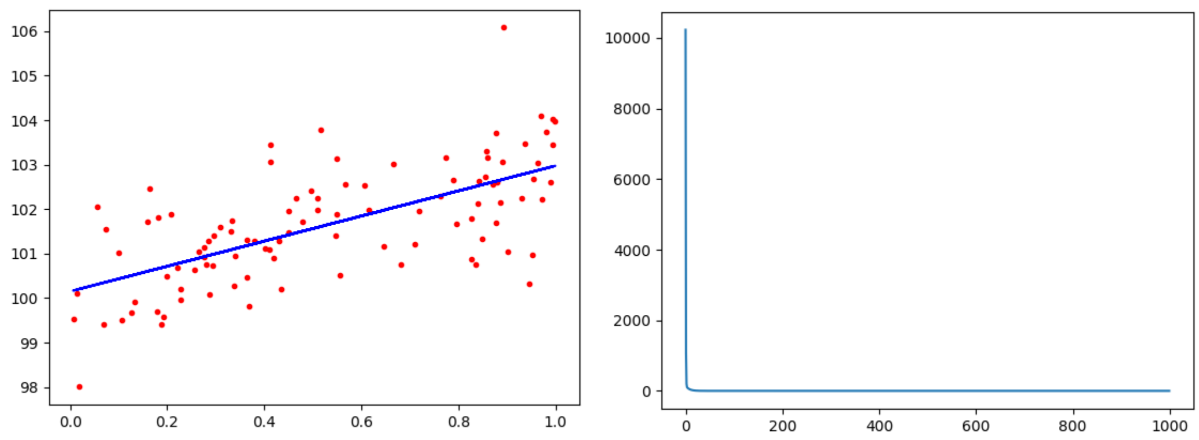
When eta was set to 0.001, the algorithm converged slowly and produced an inaccurate fit. When eta was adjusted to 0.5, however, the algorithm diverged and the MSE increased with each iteration. The best results were obtained with a learning rate of 0.01 since the algorithm converged rapidly and the final fit was very close to the actual data. You can see the related graphs below.



eta =0.001   h(x) = 76.28528054888427 + 32.86189460088479x



eta = 0.01 and   h(x) = 93.87008057534227 + 14.6329309875613x

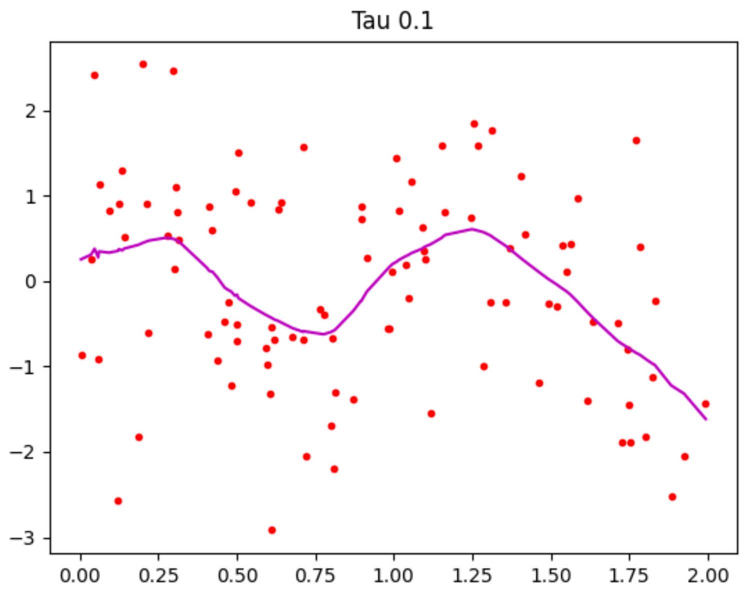


eta=0.5   and   h(x) = 100.1514921880017 + 2.8230436494488655x

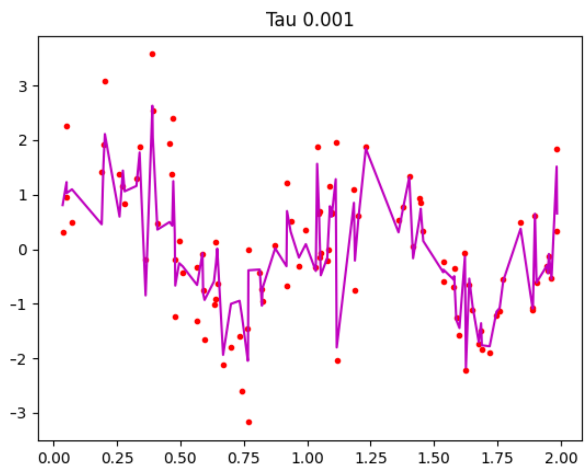
Question – 3

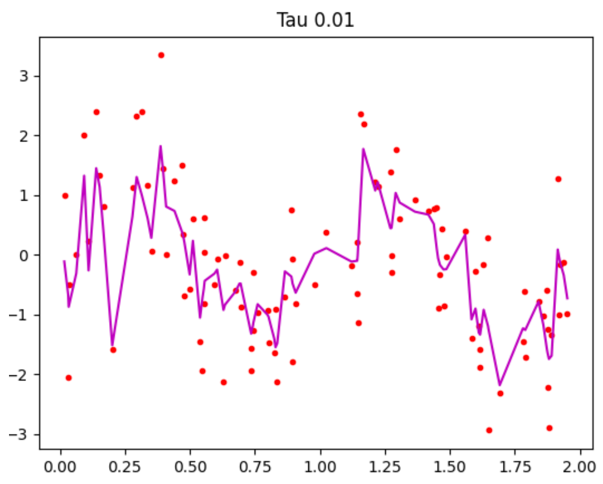
First, I generated 100 training examples and implemented the weighted linear regression function with local weights (W(i) = exp(-(x(i)-x)2/(2*tau2)), with a bandwidth parameter tau =

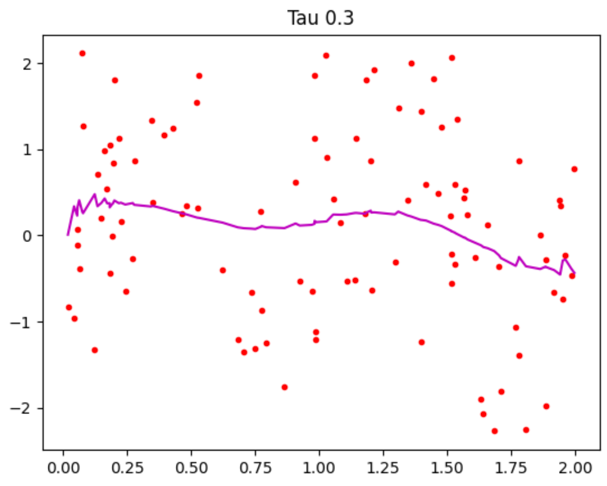
0.1. The weighted linear regression function accepts as input parameters X, Y, iteration count, testx, eta, x, and tau and returns the optimum theta value.

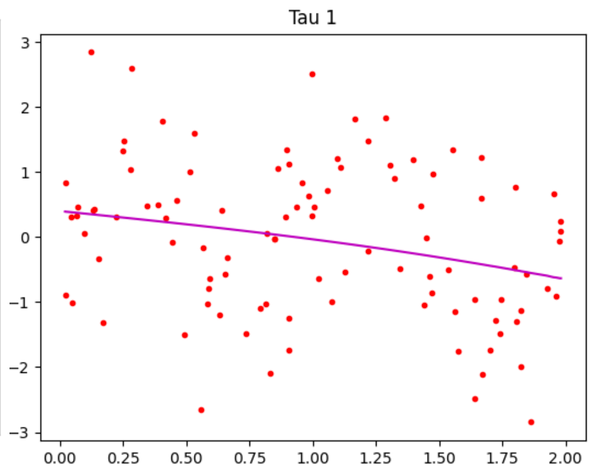


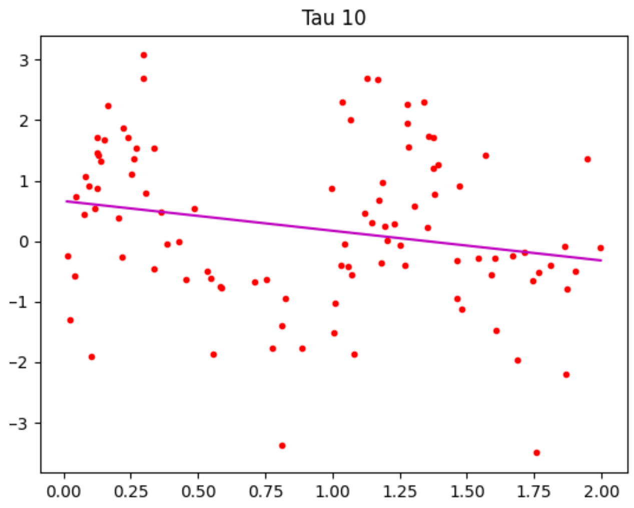
Part two involves repeating part one five times with different bandwidth parameter values (=0.001, 0.01, 0.3, 1, and 10) and observing what happens to the fit when is too little or too large.











When the weights given to the training data are too large (e.g. =0.001), the algorithm can overfit to the training data, which means it will perfectly fit the training data but perform poorly on new data. On the other hand, when the weights are too small (e.g. =10), the algorithm can underfit to the training data, which means it will be too smooth and ignore important details in the data.

The ideal value of τ is the one that balances the risk of overfitting and underfitting, and works well with new data. Our experiments revealed that τ=0.1 gave the best results. It was able to fit the training data well and it captured the high-frequency details of the data.

To sum up, the decision of τ impacts the balance between bias and variance of the algorithm. If τ is small, the bias will be low, but the variance will be high, and the model will overfit. On the other hand, if τ is large, the bias will be high, but the variance will be low, and the model will underfit. The optimal value of τ is the one that strikes a good balance between bias and variance and performs well on new data.