Homework 4 - Regression

Q1. Linear Regression

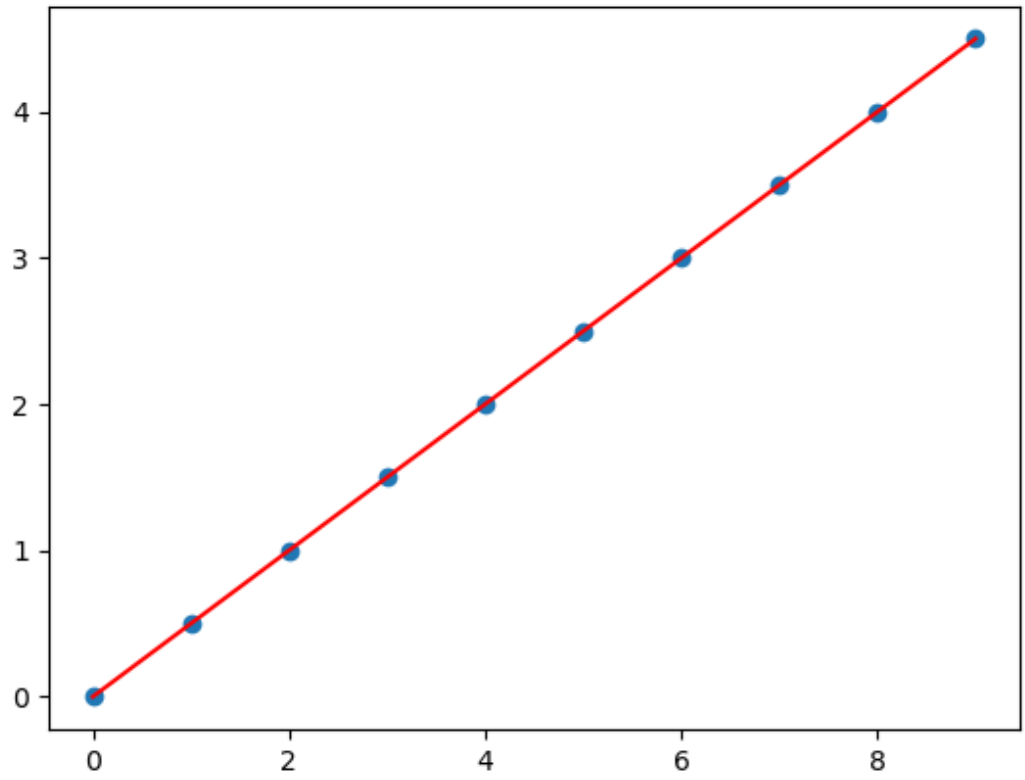
(a) Use your closed form implementation to fit a model to the data in 1D-no-noise-lin.txt and 2D-noisy-lin.txt. What is the loss of the fit model in both cases? Include plots for both. Note: for some of these datasets the y-intercept will be non-zero, make sure you handle this case!

Solution:

Loss of fit closed-form model for 1D-no-noise-lin.txt:

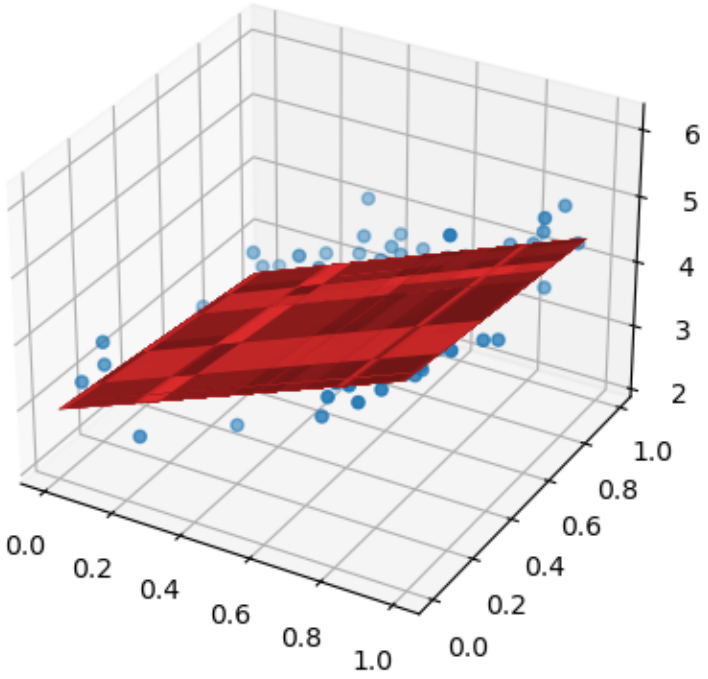
Theta: [[0.], [0.5]]
Loss: 0.0

Plot for 1D-no-noise-lin.txt



Loss of fit closed-form model for 2D-noisy-lin.txt:

Theta = [[2.93987438], [2.04156149], [-0.43683838]]
Loss: 0.2151856695040176



Plot for 2D-noisy-lin.txt

(b) What happens when you're using the closed form solution and one of the features (columns of X) is duplicated? Explain why. Note: you may want to test this out with your code as a useful first step, but you should think critically about what is happening and why.

Solution:

When one of the features were duplicated, the program returned numpy. linalg. Singular Matrix error. This is because, when there exists a duplicate feature in X, it returns a Singular Matrix when multiplied by its transpose (X^TX), where determinant of X^TX will be 0. We know that inverse of a matrix doesn't exist when its determinant is zero. Therefore, the solution is not possible in closed form since we can't compute the value of Theta using inverse of X^TX.

(c) Does the same thing happen if one of the training points (rows of X) is duplicated? Explain why.

Solution:

Output for 1D-no-noise-lin.txt:
Theta = [[8.8817842e-16], [5.0000000e-01]]
Loss: 4.392520949526088e-31

Output for 2D-noisy-lin.txt:
Theta = [[2.9328412], [2.03334514], [-0.42065597]]
Loss: 0.21393978089092

When one of the training points were duplicated, there is not much of a difference seen from the previous values. Since the same points are just repeating, the distance between them won't change and so it doesn't really affect the fitted line. However, having duplicate points in a dataset, might make a model likely to overfit training set by seeing the same examples again.

(d) Does the same thing happen with Gradient Descent? Explain why.

Solution:

Testing with 2D-noisy-lin.txt

	Theta	Loss
Original dataset	[[2.93987438] [2.04156149] [0.43683838]]	0.2151856695040176
Feature duplication	[[2.82307853] [2.11017553] [-0.151103] [-0.151103]]	0.21699917954546855
Training point duplication	[[2.75423648] [2.09471344] [0.16106392]]	0.21980678524364192

By duplicating a feature and training point on 2D-noisy-lin.txt and fitting it with gradient descent, we can see that there is not a big difference between the scores. They are almost similar. There is a very little increase in the loss for feature duplication because we now have one more feature to be considered for gradient descent to converge. There is no Singular Matrix error in this case because we don't compute inverse of X^TX in gradient descent. And again, because of a duplicate training point, the gradient descent model couldn't generalize and hence will start overfitting the data by learning same points again.

Q2. Gradient Descent

(a) Now use your Gradient Descent implementation to fit a model to 1D-no-noise-lin.txt with alpha=0.05, num iters=10, and initial Theta set to the zero vector. What is the output (the full list of (θ, loss) tuples for each of the 10 iterations)?

Solution:

```
Iteration: 1 , Loss: 7.125 , Theta: [[0.]
 [0.]]
Iteration: 2 , Loss: 1.5147656250000001 , Theta: [[0.1125]
 [0.7125]]
Iteration: 3 , Loss: 0.3230469726562498 , Theta: [[0.0590625]
 [0.384375 ]]
Iteration: 4 , Loss: 0.06987533596801745 , Theta: [[0.082125  ]
 [0.53585156]]
Iteration: 5 , Loss: 0.016063408298435153 , Theta: [[0.06995215]
 [0.46628496]]
Iteration: 6 , Loss: 0.0045988720769591 , Theta: [[0.07404042]
 [0.49858966]]
Iteration: 7 , Loss: 0.002130391801865711 , Theta: [[0.07065573]
 [0.4839403 ]]
Iteration: 8 , Loss: 0.0015737151106616106 , Theta: [[0.07073638]
 [0.49092783]]
Iteration: 9 , Loss: 0.0014240352995616161 , Theta: [[0.0692408 ]
 [0.48793999]]
Iteration: 10 , Loss: 0.0013616878728702903 , Theta: [[0.06849226]
 [0.48954633]]
```

Gradient Descent on 1D-no-noise-lin.txt

(b) Using the default parameters for alpha and num iters, and with initial Theta set to 0, do you get the same model parameters as you did with the closed form solution? the same loss? Report this for both 1D-no-noise-lin.txt and 2D-noisy-lin.txt.

Solution:

Dataset	Theta	Loss
1D-no-noise-lin.txt		
Gradient Descent	[[6.44700512e-05] [4.99989719e-01]]	1.203460519328615e-09
Closed form	[[0.], [0.5]]	0.0
2D-noisy-lin.txt		
Gradient Descent	[[2.75594278] [2.09675389] [-0.16343704]]	0.2215750342043043
Closed form	[[2.93987438] [2.04156149] [-0.43683838]]	0.2151856695040176

The table shows that the parameters and loss for both the datasets are very similar that they differ only by tiny marginal values. The closed-form solution seems to provide us with accurate low loss values because, the parameter theta can be directly calculated from the dataset. However, for gradient descent, we need to run multiple times to find the optimal learning rate and iterations for optimizing the overall loss.

(c) Find a set of values of the learning rate and iterations where you get the same answers and a set of values where the answers are noticeably different. Explain what's going on in both cases, and for both datasets 1D-no-noise-lin.txt and 2D-noisy-lin.txt.

Solution:

Different set of parameters for same output values

	Alpha	Iters	Theta	Loss
1D-no-noise-lin.txt				
Default parameters	0.05	500	[[6.44700512e-05] [4.99989719e-01]]	1.203460519328615e-09
Changed parameters	0.06	416	[[6.47338400e-05] [4.99989677e-01]]	1.213328939459853e-09
2D-noisy-lin.txt				
Default parameters	0.05	500	[[2.75594278] [2.09675389] [-0.16343704]]	0.2215750342043043
Changed parameters	0.06	416	[[2.75556512] [2.09669249] [-0.16269593]]	0.22160886677137148

Found a different set of parameters with learning rate (alpha) = 0.06 and 416 iterations to get the similar results (Loss and Theta).

Since we increased the learning rate from 0.05 to 0.06, we reduced the iterations because the gradient descent will converge faster than before. Otherwise, running for a lot of iterations would make the model get past the minima because of a higher learning rate. So, after running multiple times with different set of parameters, we were able to find this instance that matches with the default parameter results.

Different set of parameters for different output values

	Alpha	Iters	Theta	Loss
1D-no-noise-lin.txt	0.07	5	[[-0.02102143] [-0.09344239]]	10.149672391240482
2D-noisy-lin.txt	0.07	5	[[0.90387343] [0.53944514] [0.4597673]]	6.171593265545774

Here, since the learning rate is high compared to the previous ones, the model was starting to jump over the minima and move to random points when it ran for more than 100 iterations. Again, having a high learning rate, will not allow gradient descent to converge and constantly get past the minima if it runs for multiple times. But when we look at the table, for this learning rate even with just 5 iterations we have a very high loss on training data (10.14 & 6.17). This is because the model is still getting past the convergence. So, we must either keep a low learning rate or run it for a smaller number of iterations.

Q. Random Fourier Features: Next page

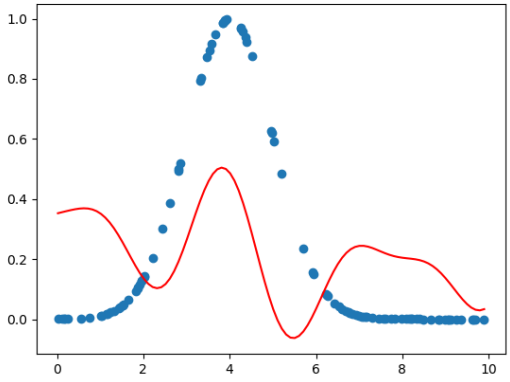
Q3. Random Fourier Features

(a) Use your Random Fourier Features implementation to fit models for 1D-exp-samp.txt, 1D-exp-uni.txt, 1D-quad-uni.txt, and 1D-quad-uni-noise.txt, each with 3 different values for K: small, medium, and large. The small value should noticeably under-fit the data, the large value should fit almost perfectly, and the medium value should be somewhere in between. Report the values for K and include graphs for each of the four datasets (so you should report 12 values of K and have 12 graphs in total).

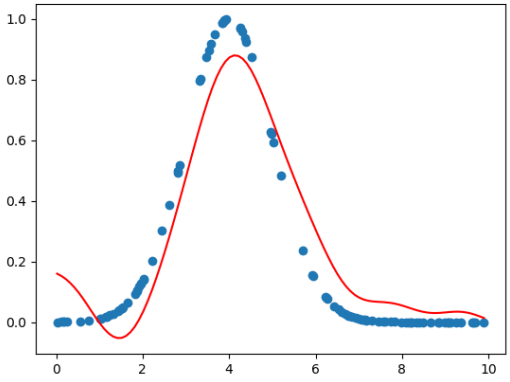
Solution:

1d-exp-samp.txt:

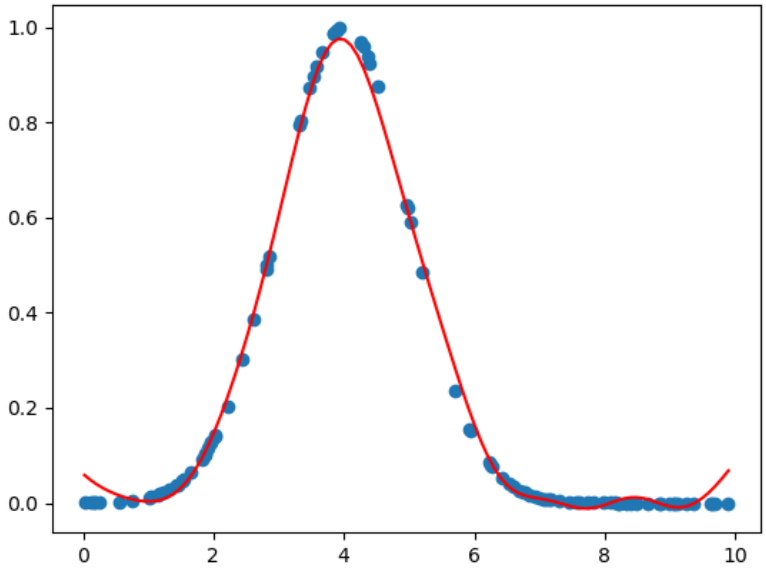
K=7, Iterations=500 (Underfit)



K=20, Iterations=500 (In-between fit)

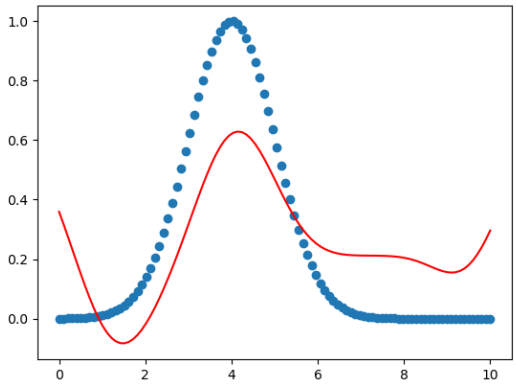


K=4000, Iterations=1000 (Perfect fit)

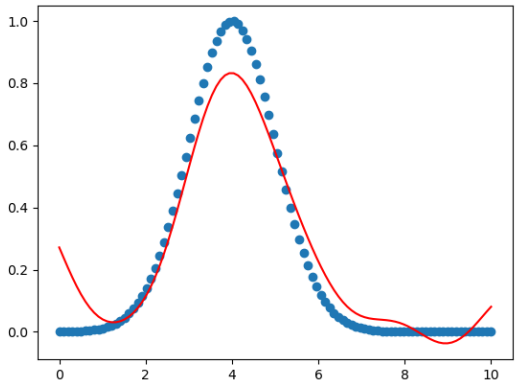


1d-exp-uni.txt:

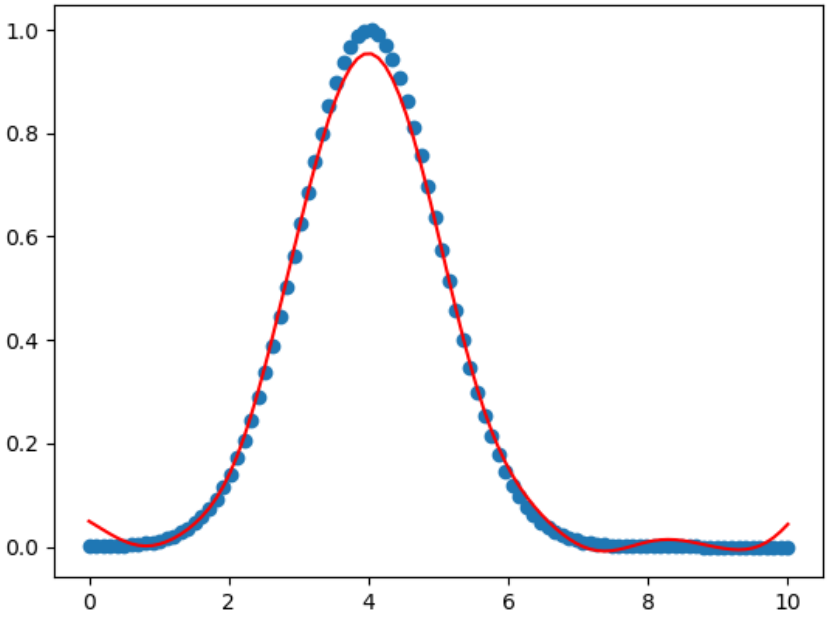
K=8, Iterations=500 (Underfit)



K=35, Iterations=500 (In-between fit)

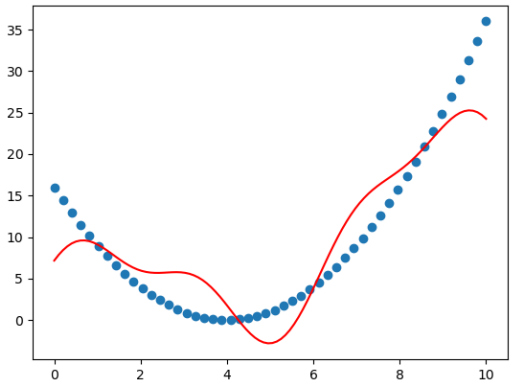


K=6000, Iterations=1000 (Perfect fit)

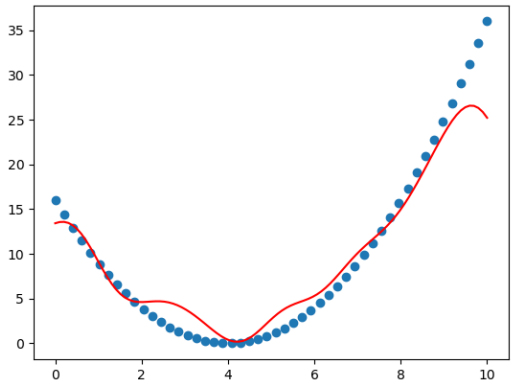


1d-quad-uni.txt

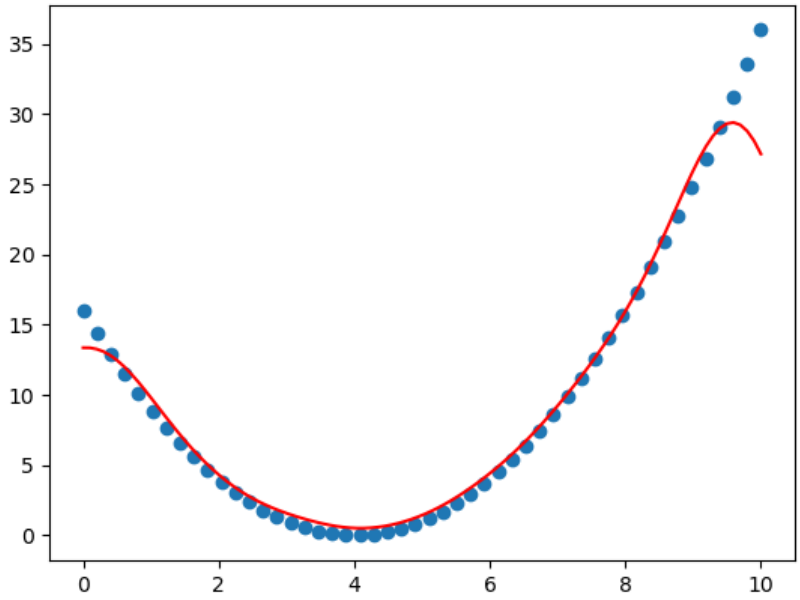
K=5, Iterations=500 (Underfit)



K=30, Iterations=500 (In-between fit)

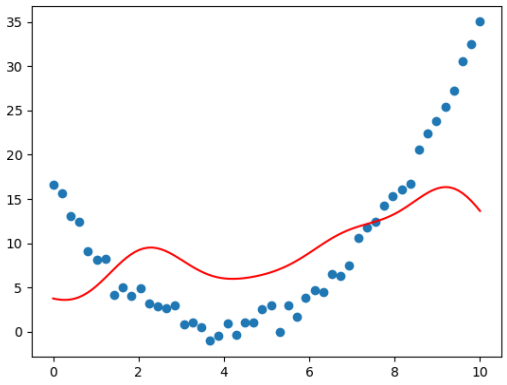


K=5000, Iterations=800 (Perfect fit)

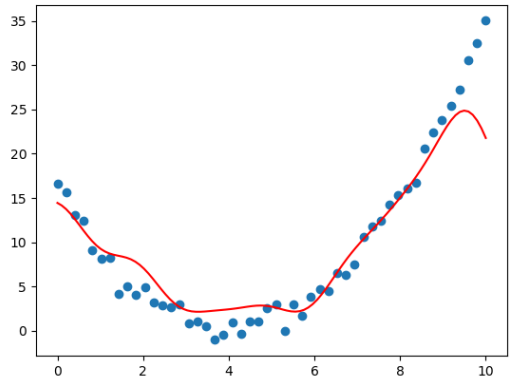


1d-quad-uni-noise.txt

K=10, Iterations=500 (Underfit)



K=50, Iterations=500 (In-between fit)



K=2000, Iterations=1000 (Perfect fit)