Homework 4

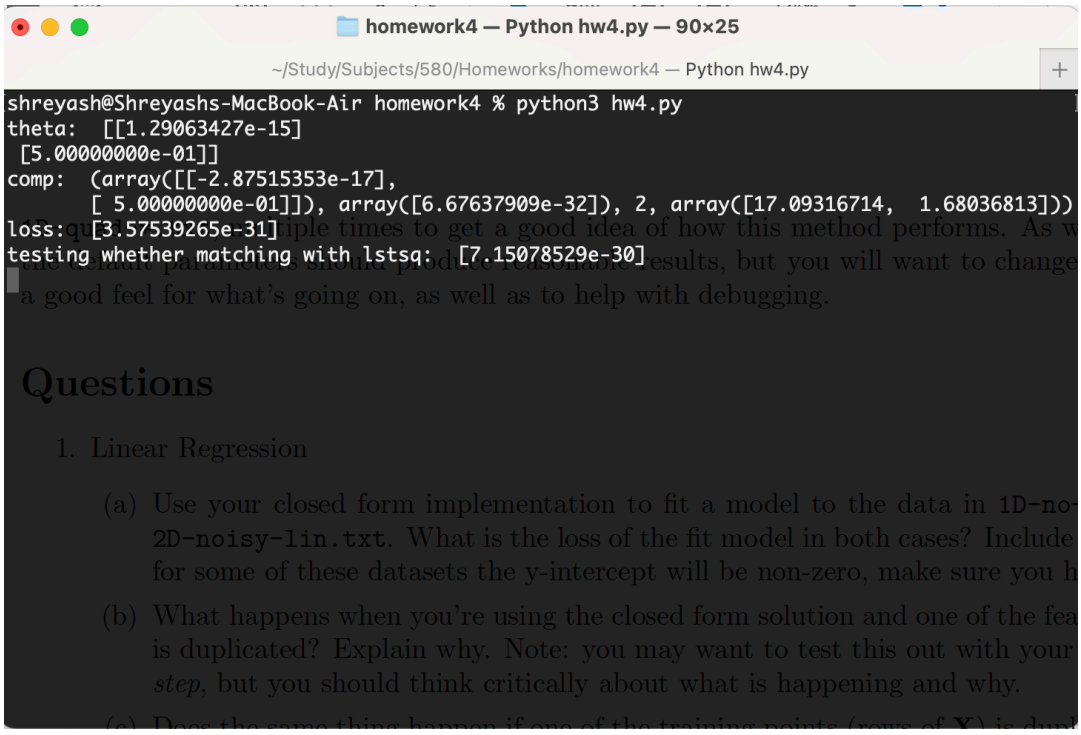
Shreyash Solunke
G01325631

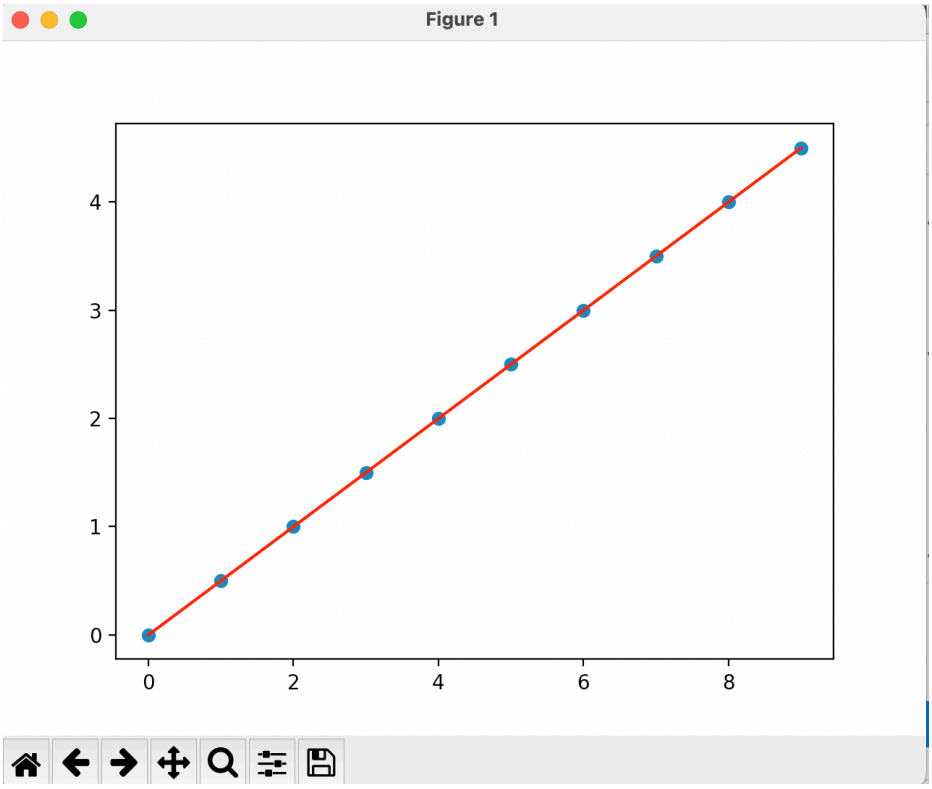
Q1. Linear Regression

(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:
Plot for *1D-no-noise-lin.txt*, Loss and other outputs:

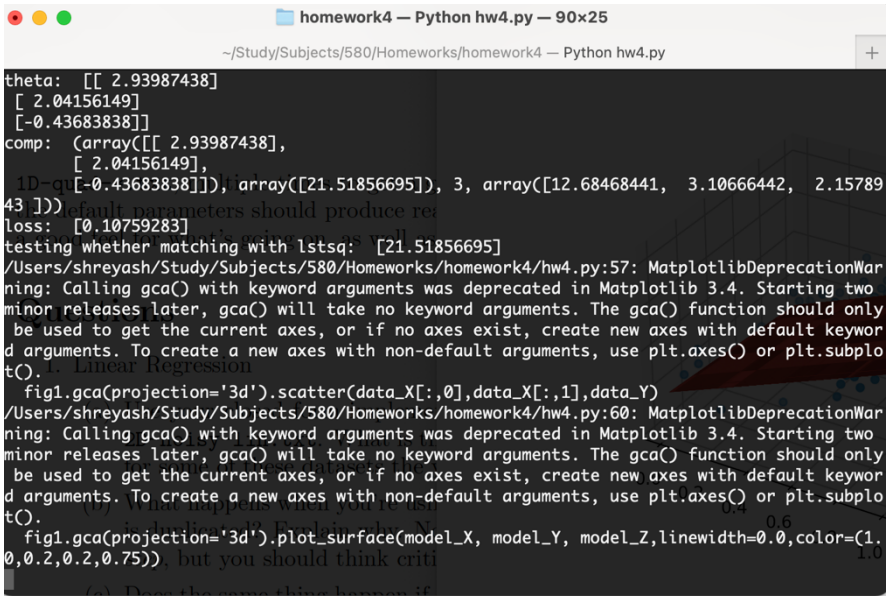
Loss = 0 (3.57539265e-31 to be exact)
Theta = [[1.29063427e-15] (0, 0.5 converting to floating point)
[5.00000000e-01]]

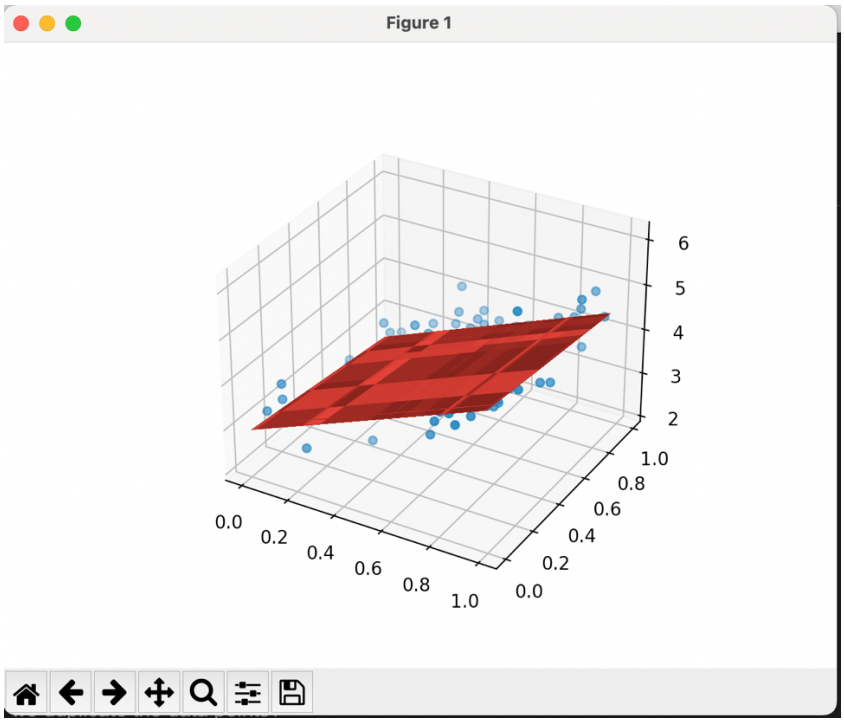


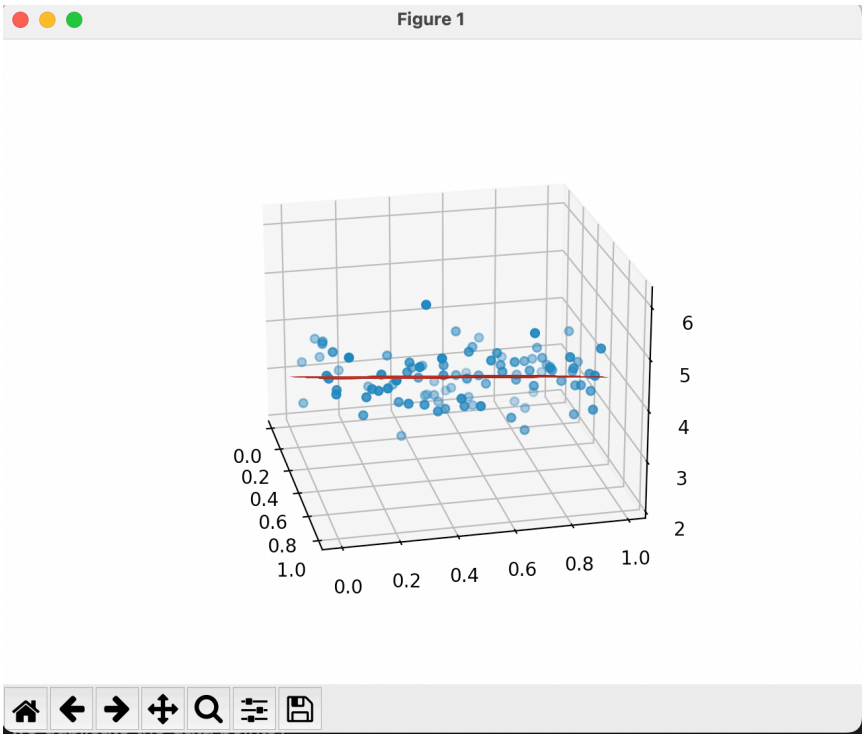


Plot for *2D-noisy-lin.txt*, Loss and other outputs:

Loss = 0.10759283
Theta = [[2.93987438]
[2.04156149]
[-0.43683838]]







(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 was duplicated on the 1D data, it generated a **singular matrix** when it was multiplied with its transpose ($\mathbf{X^T.X}$). Since its determinant is zero, and no inverse of the matrix is available, we cannot calculate the Theta value using the matrix method.

When the feature was duplicated on the 2D data, I got the following results.
theta: [[-0.31793461]
[1.95940297]
[0.72233963]
[0.]]
loss: [3.77557245]

As you can see, the loss was increased by a lot. I think the reason behind this is that the fit distance will be counted more which results in more loss.

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

Solution:

When one of the features were duplicated, I got the following results.
theta: [[1.58206781e-15]
[5.00000000e-01]]
loss: [8.44362705e-31]

As you can see, there is not much of a difference in values compared to the original data. This is because the fit distance calculated will not vary much as the same distance between the previous data points will just repeat. This will make the model much less generalized though, due to data duplicity as the real-world data will differ.

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

Solution:

- Feature duplication:

  Original Results:
  Theta: [[2.82307853]
  [2.11017553]
  [-0.151103]
  [-0.151103]]
  Loss: [0.10849959]

  After feature duplication:
  Theta: [[2.75594278]
  [2.09675389]
  [-0.16343704]]

Loss: [0.11078752]

We see that the loss has increased by small amounts but not by a lot. This is because the gradient descent will just converge quickly compared to the original data, as one more feature is considered now.

- Row duplication:
  Original Results:
  Theta: [[2.75594278]
  [2.09675389]
  [-0.16343704]]
  Loss: [0.11078752]
  After row duplication:
  Theta: [[2.79389748]
  [2.08058115]
  [-0.20298115]]
  Loss: [0.11235462]

  As we can see, the results are almost similar with very low variation.

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 (Theta, loss) tuples for each of the 10 iterations)?

Solution:

Iteration: 1 , Loss: [3.5625] ,
Theta: [[0.]
[0.]]
Iteration: 2 , Loss: [0.75738281] ,
Theta: [[0.1125]
[0.7125]]
Iteration: 3 , Loss: [0.16152349] ,
Theta: [[0.0590625]
[0.384375]]
Iteration: 4 , Loss: [0.03493767] ,
Theta: [[0.082125]
[0.53585156]]
Iteration: 5 , Loss: [0.0080317] ,
Theta: [[0.06995215]
[0.46628496]]
Iteration: 6 , Loss: [0.00229944] ,
Theta: [[0.07404042]
[0.49858966]]
Iteration: 7 , Loss: [0.0010652] ,
Theta: [[0.07065573]

[0.4839403]]
Iteration: 8 , Loss: [0.00078686] ,
Theta: [[0.07073638]
[0.49092783]]
Iteration: 9 , Loss: [0.00071202] ,
Theta: [[0.0692408]
[0.48793999]]
Iteration: 10 , Loss: [0.00068084] ,
Theta: [[0.06849226]
[0.48954633]]

(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:

- 1D data:

  Using gradient descent with default parameters:
  Theta: [[6.44700512e-05]
  [4.99989719e-01]]
  Loss: [6.0173026e-10]

  Using closed-form solution:
  theta: [[1.29063427e-15]
  [5.00000000e-01]]
  loss: [3.57539265e-31]

- 2D data:

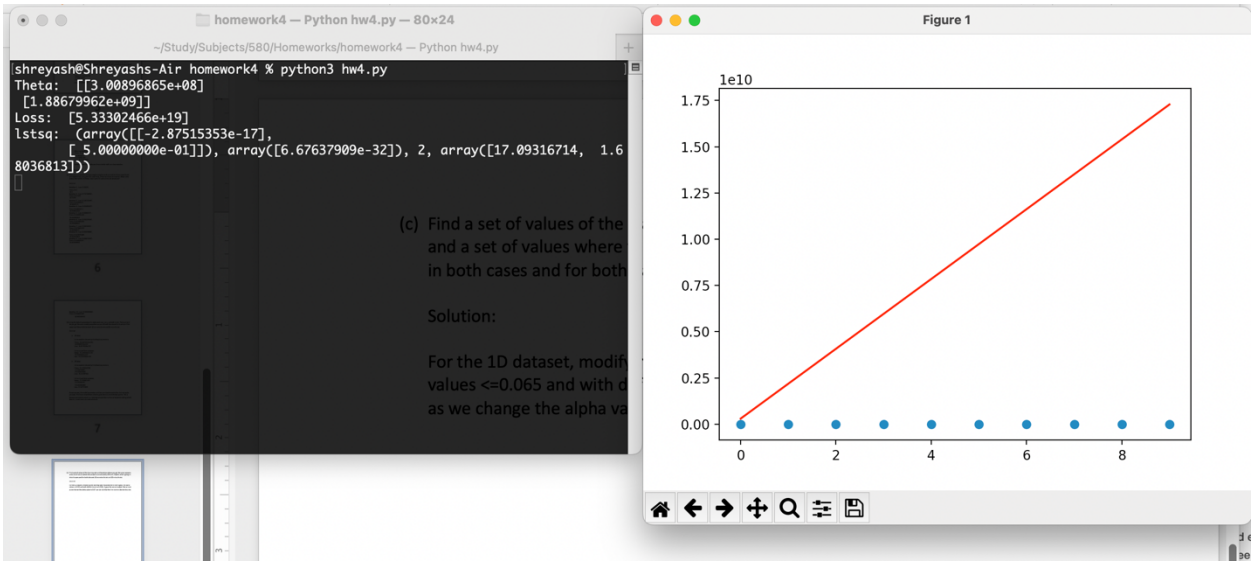
  Using gradient descent with default parameters:
  Theta: [[2.75594278]
  [2.09675389]
  [-0.16343704]]
  Loss: [0.11078752]

  Using closed-form solution:
  theta: [[2.93987438]
  [2.04156149]
  [-0.43683838]]
  loss: [0.10759283]

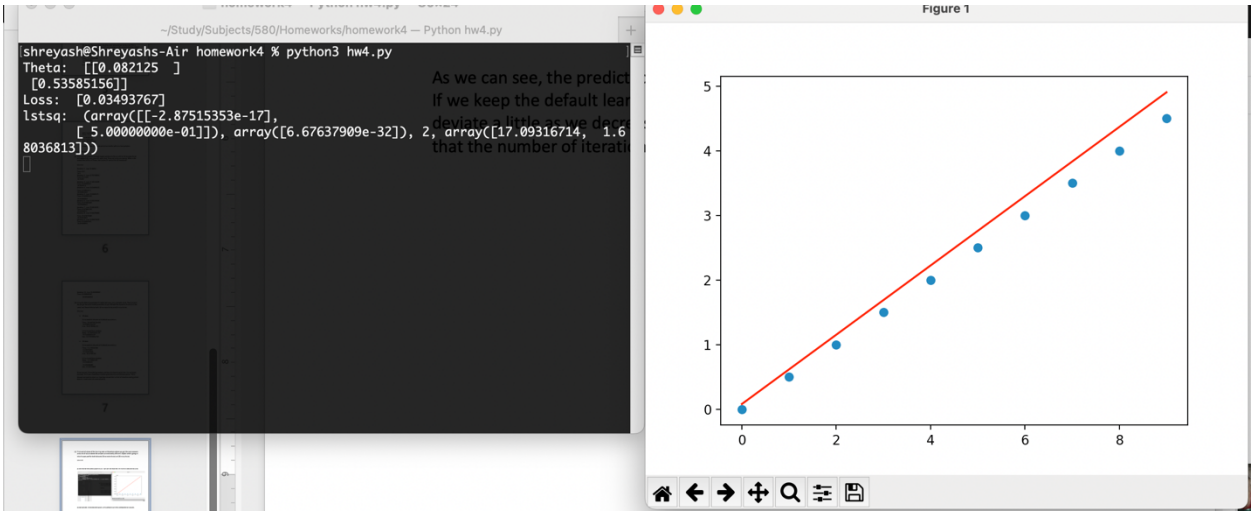
As we can see, the model parameters and loss are close to each other, but not quite accurate. The linear closed-form solution gives less loss on both data points. This is because we need to adjust our learning rate and the number of iterations with gradient descent, to get more accurate solutions.

(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:
For the 1D dataset, modifying the learning rate impacted the fit line hugely. For alpha values <=0.065 and with default iterations (500), it gave the correct output. But as soon as we change the alpha value to 0.07, we can see that the line starts to deviate by a lot.



As we can see, the predicted values jump almost e10 times compared to X values.
If we keep the default learning rate (0.05), then we can see that the line starts to deviate a little as we decrease the number of iterations is around 9-10. But when we see that the number of iterations is <= 5, the line starts to deviate a lot.



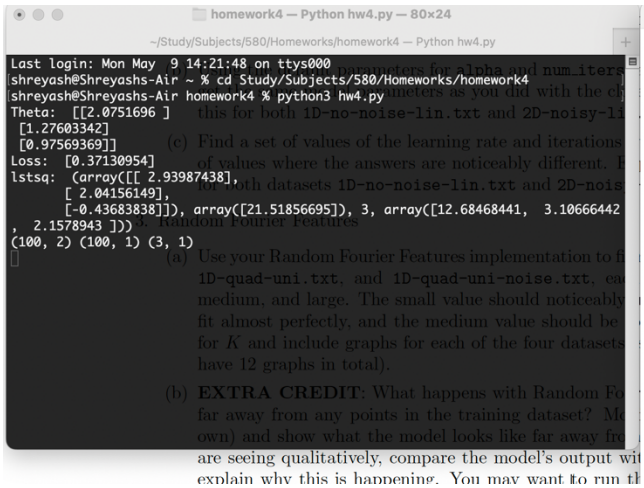
Sor for 1D data, when the learning rate is <=0.065 and the number of iterations is =>10, we got the almost similar answer, but when the learning rate is >= 0.07 and the number of iterations < 9 we started to see the deviation.

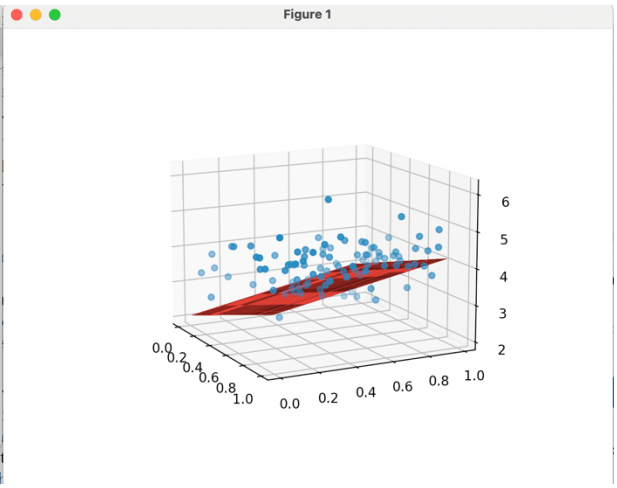
This is because we know that a smaller value of α means it will take many more iterations for it to converge than a higher value of α. If we keep a high learning rate, the

gradient values will overshoot the minima and can skip the minima point. On the other hand, if we keep the number of iterations too low, even with a low learning rate, the gradient values will never reach the minima and it is possible that the gradient values will not converge and stop at a value before the minima.

So, we need a low learning rate and a high number of iterations.

For 2D data, when the learning rate is <=1.2 and the number of iterations is =>50, we got an almost similar answer, but when the learning rate is >= 1.3 and the number of iterations <= 25 we started to see the deviation, as follows.



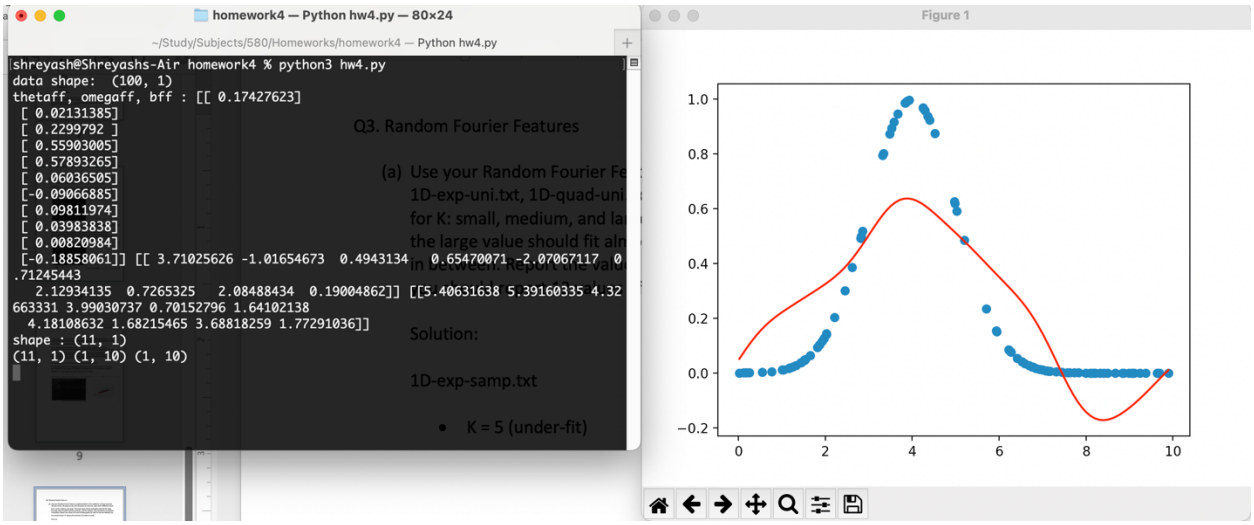


Q3. Random Fourier Features

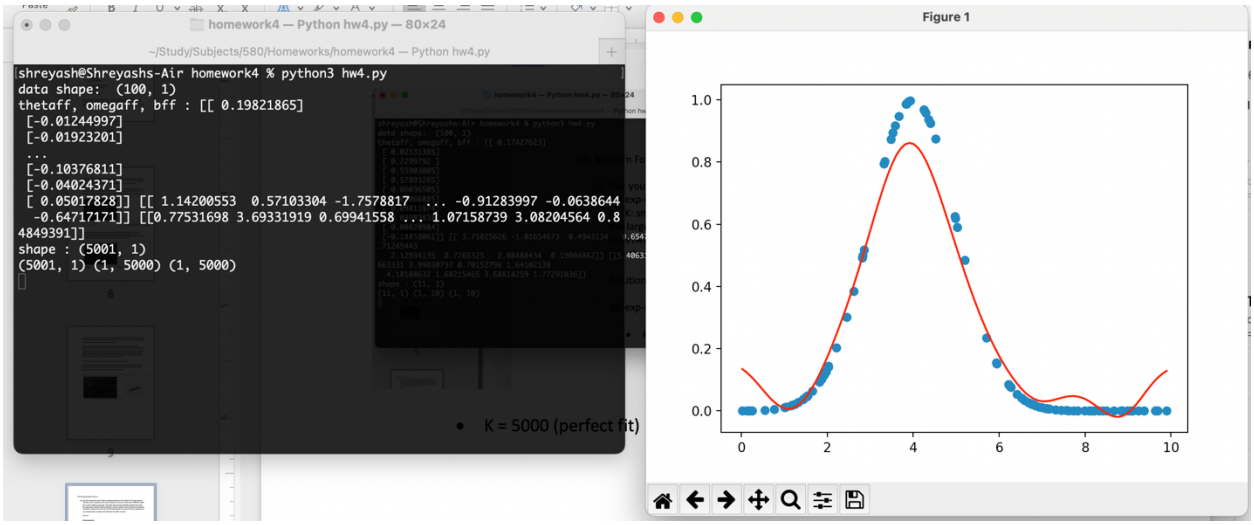
(a) Use your Random Fourier Features implementation to fit models for 1D-exp-samp.txt, 1D-exp-uni.txt, 1D-quad-uni.txt, and 1D-quad-uni-noise.txt, each with 3 different values for K: small, medium, and large. The small value should noticeably under-fit the data, the large value should fit almost perfectly, 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

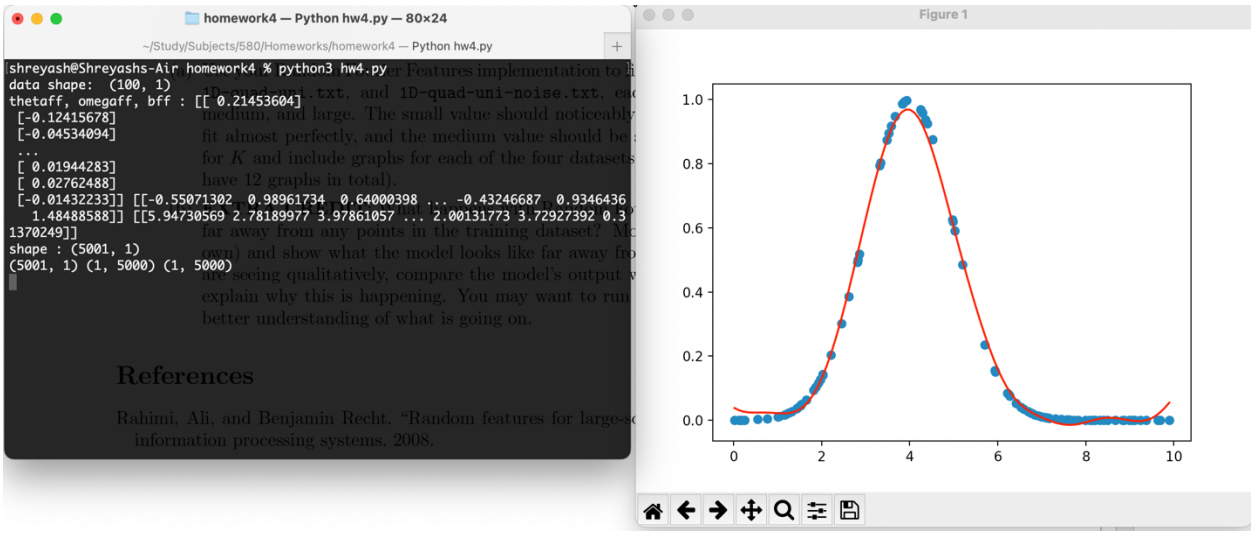
- K = 10 (under-fit)



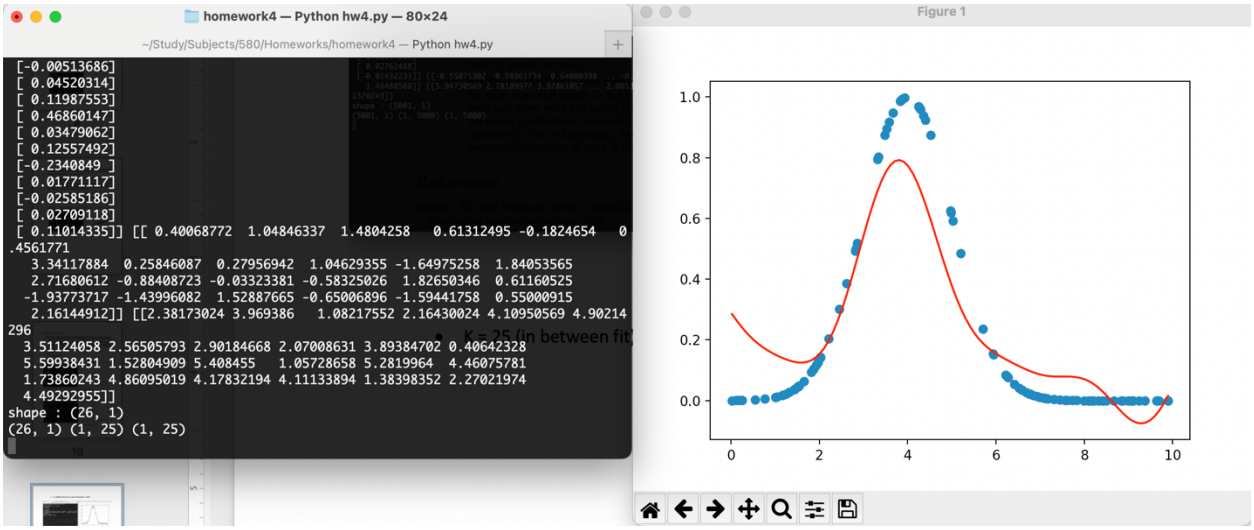
- K = 5000 (perfect fit) (number of iterations = 500)



- K = 5000 (perfect fit) (number of iterations = 1000)
When I increased the number of iterations, the line fit perfectly, otherwise there was a small gap as shown in the above image.

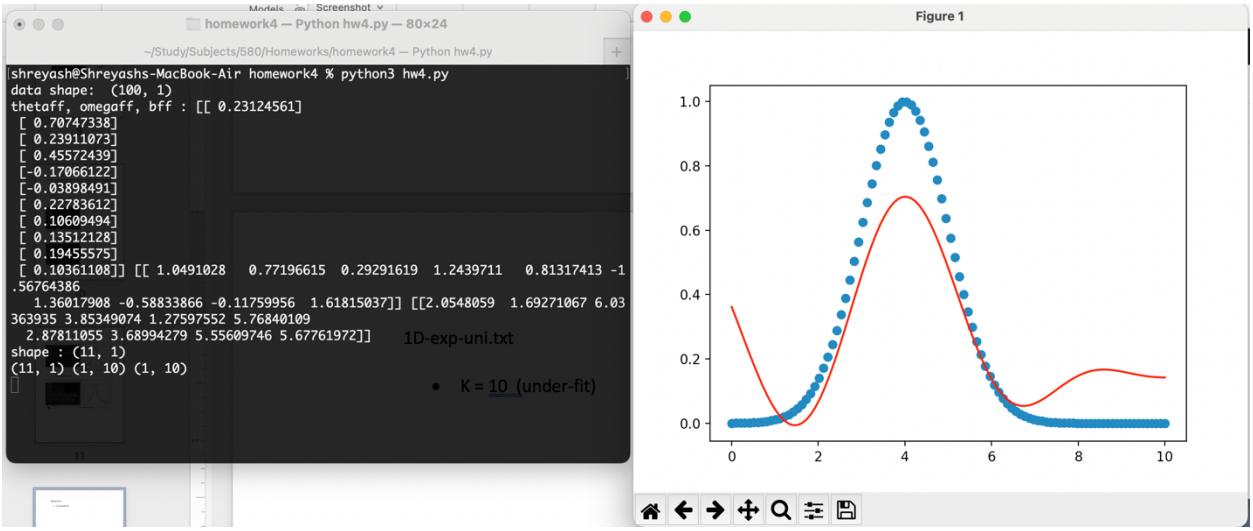


- K = 25 (in between fit)

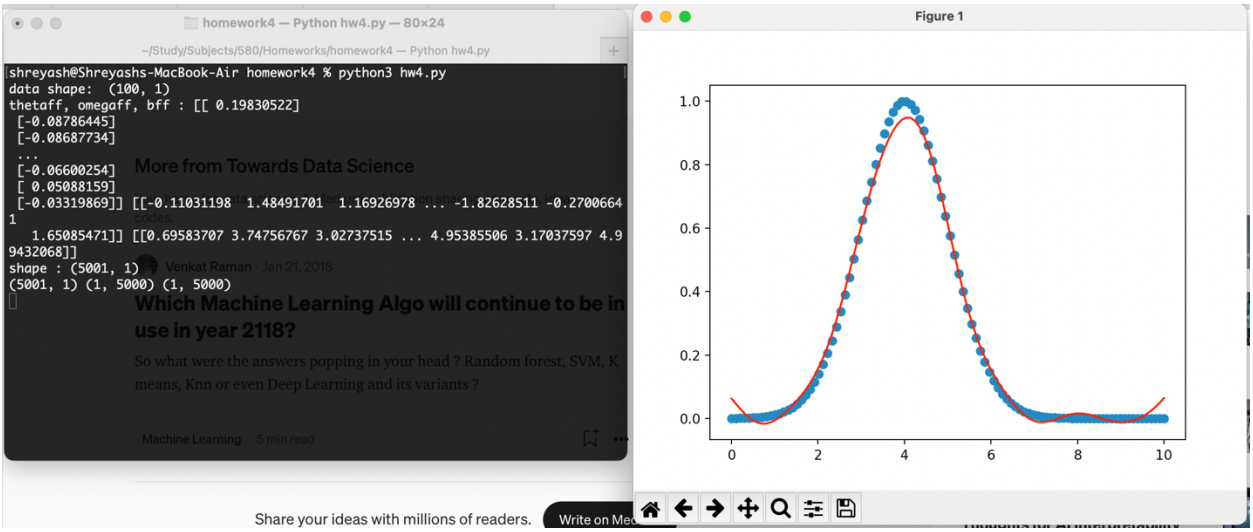


1D-exp-uni.txt

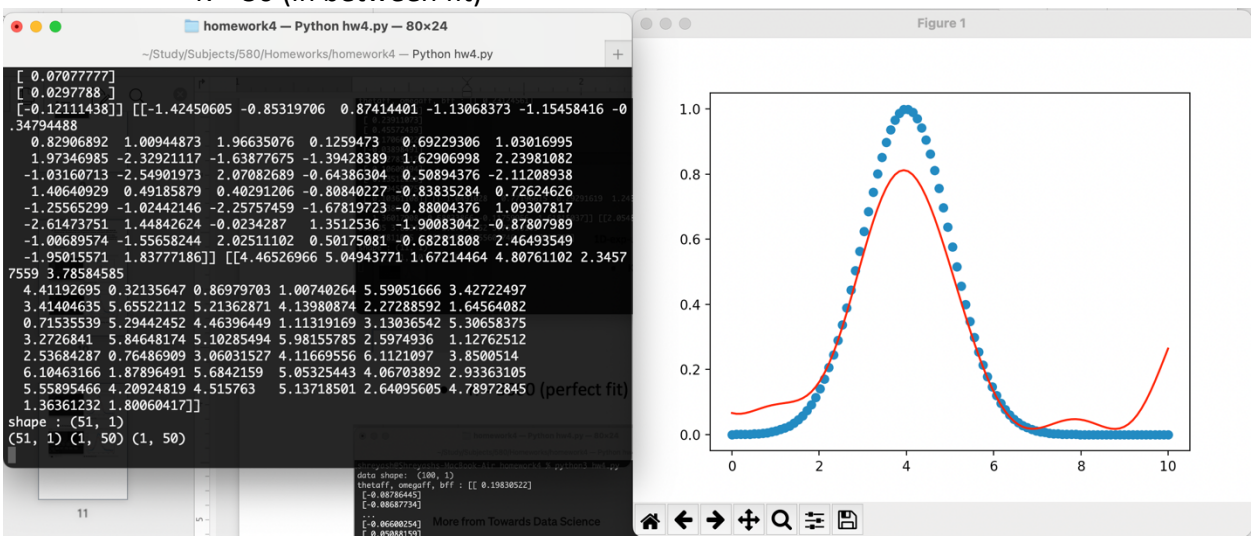
- K = 10 (under-fit)



- K = 5000 (perfect fit)

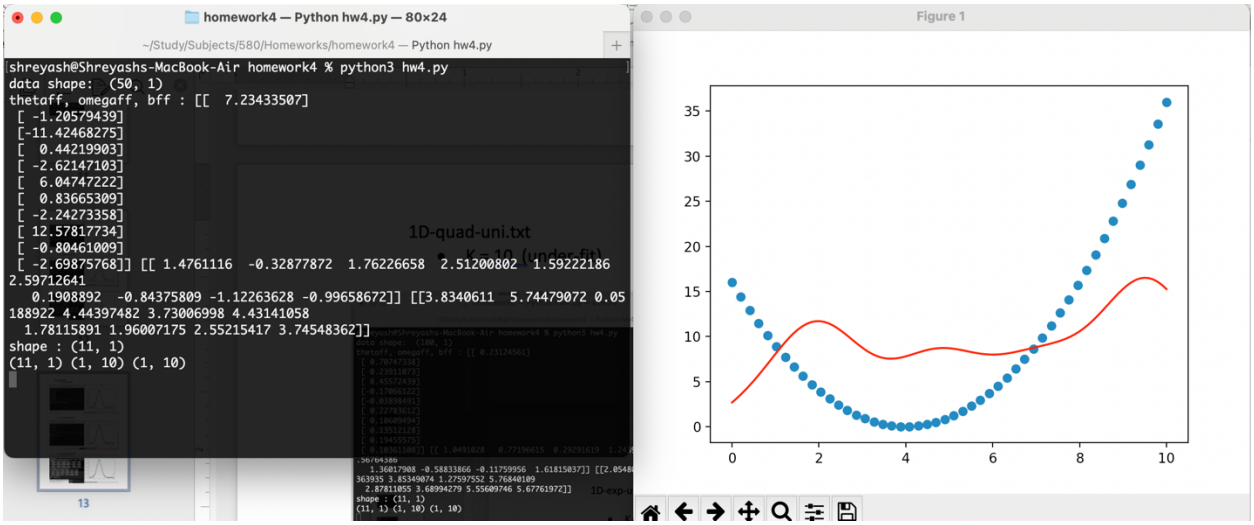


- K = 50 (in between fit)

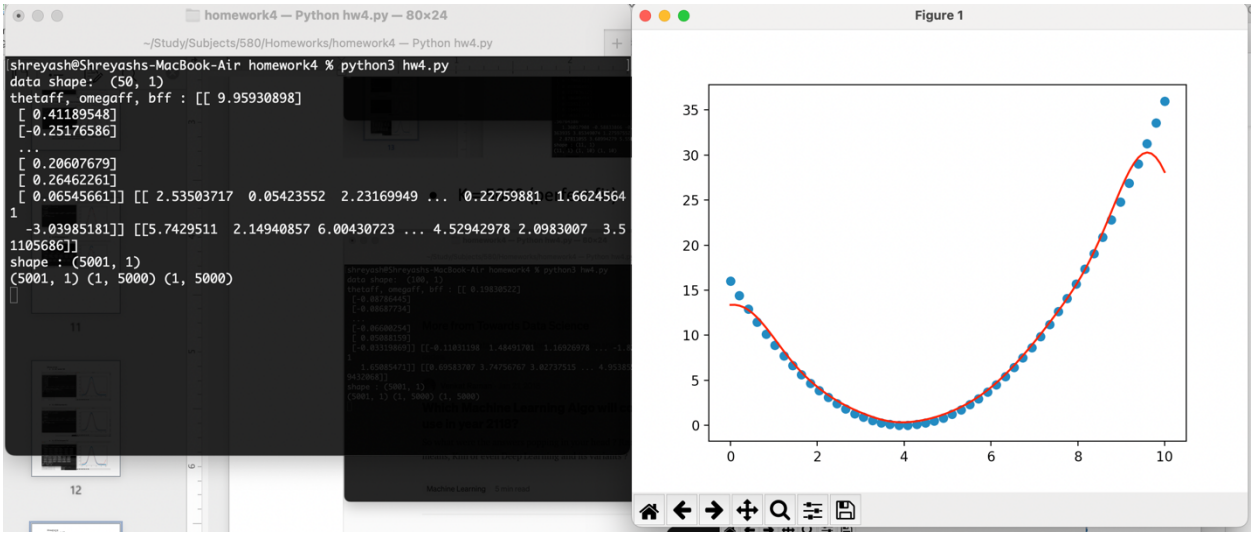


1D-quad-uni.txt

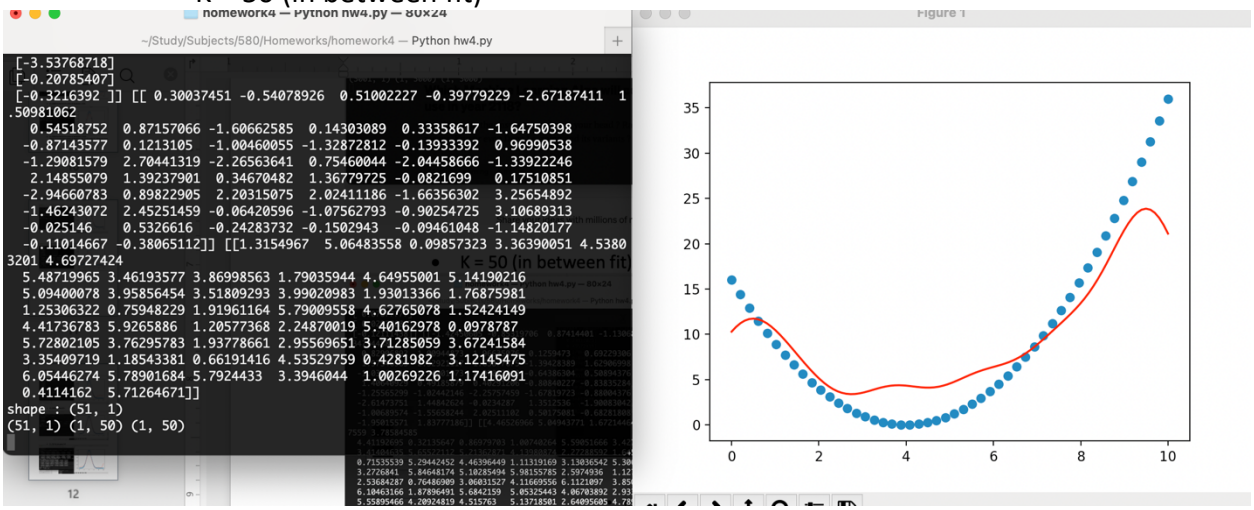
- K = 10 (under-fit)



- K = 5000 (perfect fit)

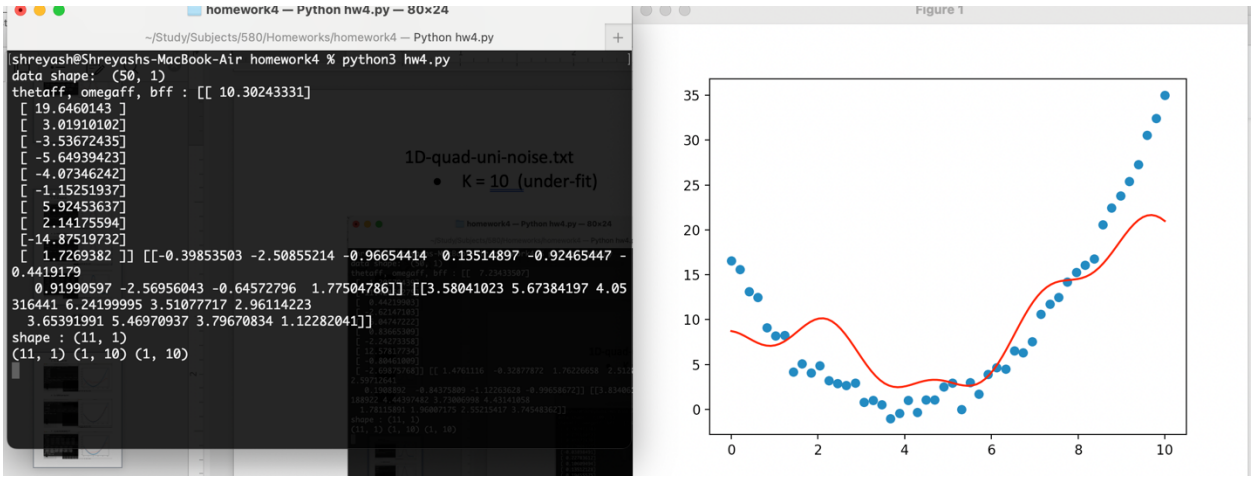


- K = 50 (in between fit)

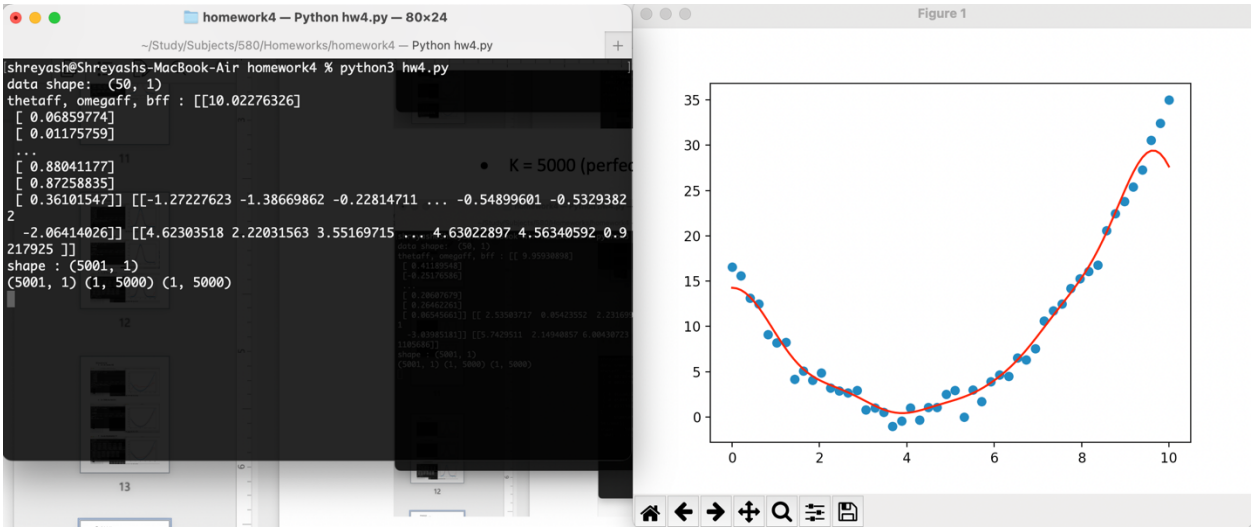


1D-quad-uni-noise.txt

- K = 10 (under-fit)



- K = 5000 (perfect fit)



- K = 50 (in between fit)