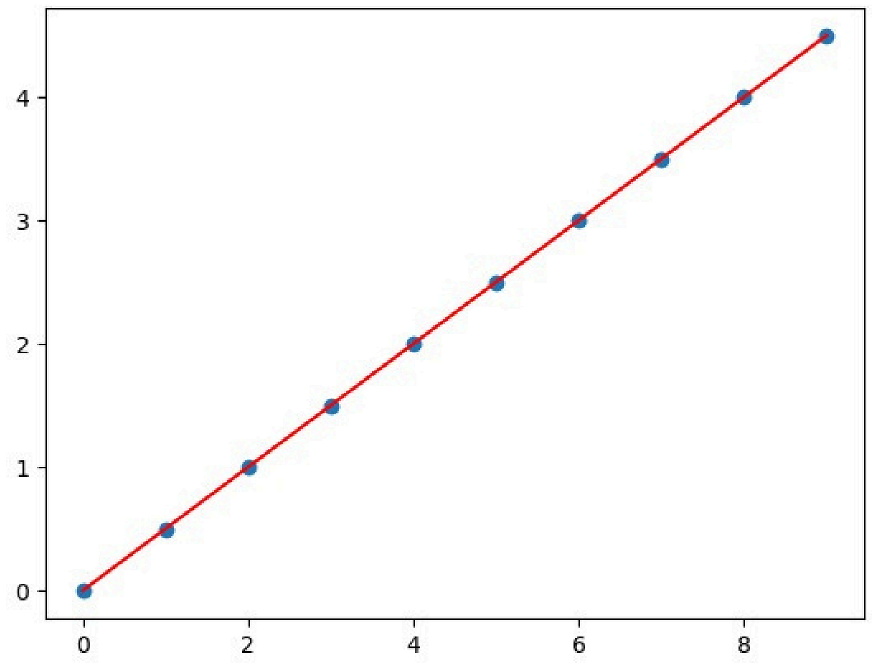
Homework 4 12/04/21

Linear Regression

A.Here's our graph for 1D-no-noise-lin



And here's our picture for 2D-noisy-lin

We have not been able to get a plot to work for 2D, please check again later.

For the 2D data we are getting a scaled loss of 0.10759283, this matches numpy's lstsq method loss exactly.
For the 1D data we are getting a scaled loss of 0. The actual unscaled loss (calculated with numpy's lstsq method) is [6.24001302e-32], a very minor difference

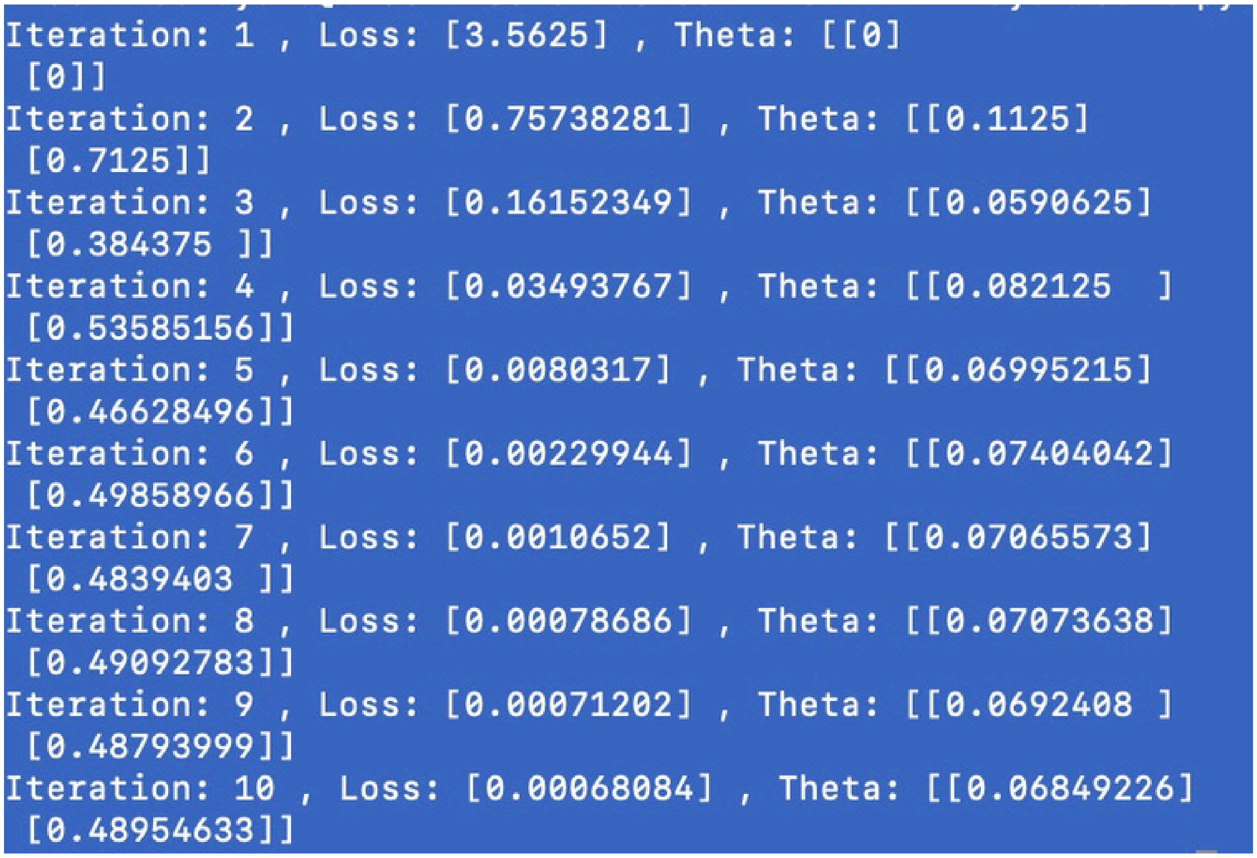
B.This throws errors with shaping, and when it doesn't it likes to throw more because you end up making the matrix uninvertible.

C.No, when adding a duplicated row the calculations remain unaffected. When we ran our method with a duplicated row, it still returned the proper values.

D.No. Gradient descent doesn't use the inverse of a matrix in its calculations. The extra column can mess up the inverse. Initial shaping errors can be present, however.

Gradient Descent

A.This is the output:



```
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.

a. For the 1D data, we are getting a loss of [6.0173026e-10] with gradient descent and a loss of 0 with the closed form. For parameters, gradient descent give us [[6.44700512e-05] [4.99989719e-01]] while the closed model gives us [[0.] [0.5]]. In short, they are NOT the same but very close to each other, with gradient descent giving a much more precise value

b. For the 2D data, gradient descent gives us a loss of [0.11078752] while the closed form gives a loss of [0.10759283]. For parameters, gradient descent gives us [[2.75594278] [2.09675389] [-0.16343704]] and the closed model gives us [[2.93987438] [2.04156149] [-0.43683838]]. Again, the values are very close but not the same

c.

   i. 2D dataset

      1. By setting the alpha to 1, we notice that the values converge to the correct answer at iteration 378/500 and stay the for the rest of the 500 interactions. This is due to the algorithm taking much bigger steps and still being able to land on the correct value, which causes it to get to that value much quicker. However, large steps are not always good.

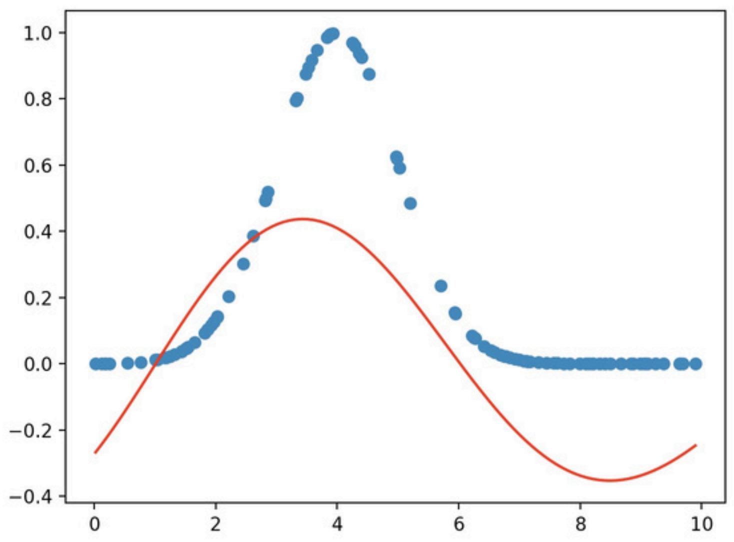
2.We noticed that an alpha of 1.2 also converges quickly, but going up by just 0.1 to an alpha value of 1.3 results in massive values of [[2.45191011e+19] [1.40519141e+19] [1.36585972e+19]]. This is due to the gradient only being able to take these massive steps and, one realizing it has taken a large step one way, attempts to take a large step the other way and continues a cycle of overstepping.

ii. 1D dataset

1. By setting the alpha to 0.01 and setting the num_iters to 10000, we get the correct but more precise answers of [[6.59700806e-16] [5.00000000e-01]]. This precision is due to the fact that the steps taken are much smaller and the iterations are much larger, allowing for a much more accurate solution.
2. Unlike the 2d values, this dataset has no leeway for large alphas. By simply putting the value to 0.1, it returns the massive incorrect values of [[2.86444118e+140] [1.79617243e+141]]. The one-dimensional nature of the data does not allow for large steps.

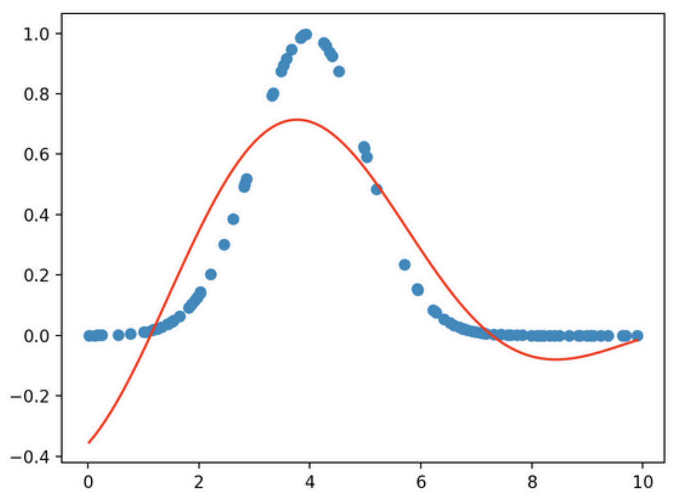
Random Fourier Features

- 1D-exp-samp.txt
  - K = 1

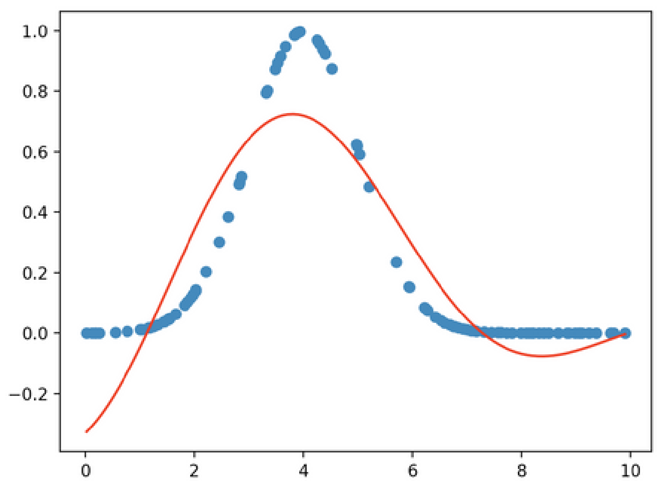


▪

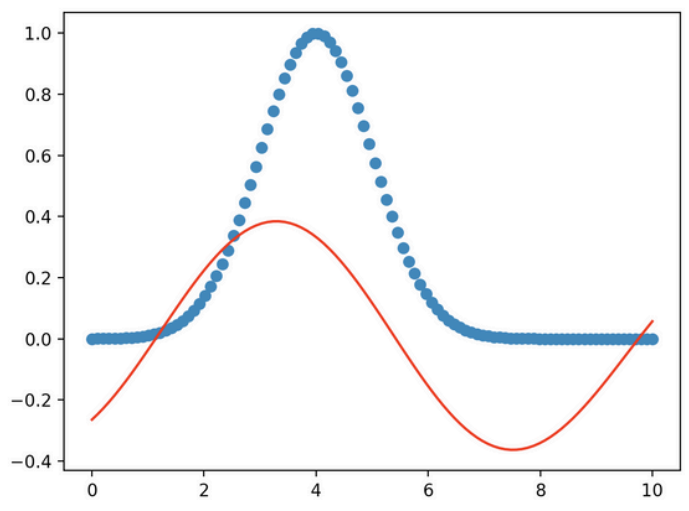
  - K = 1000



- K = 10000

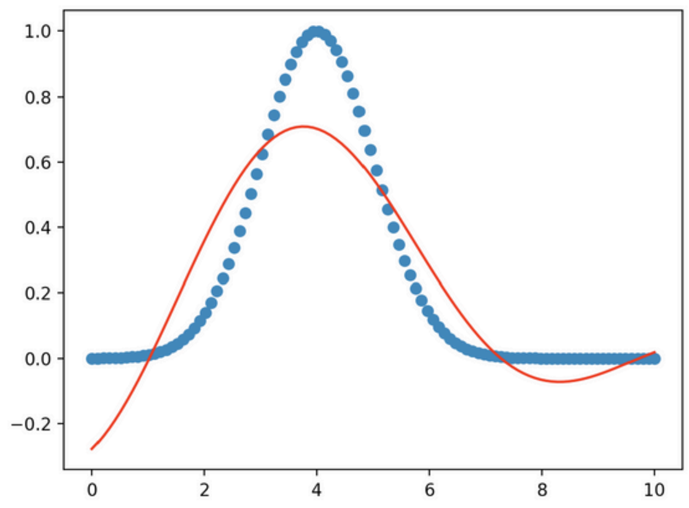


- 1D-exp-uni.txt
  - K = 1



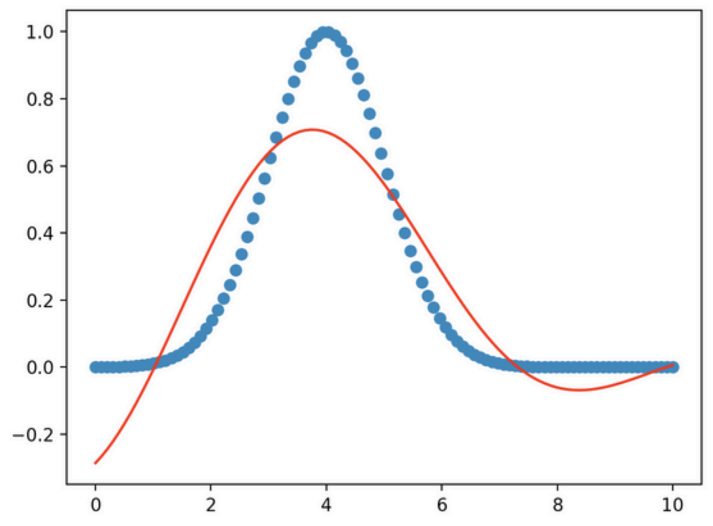
■

◯ K = 1000



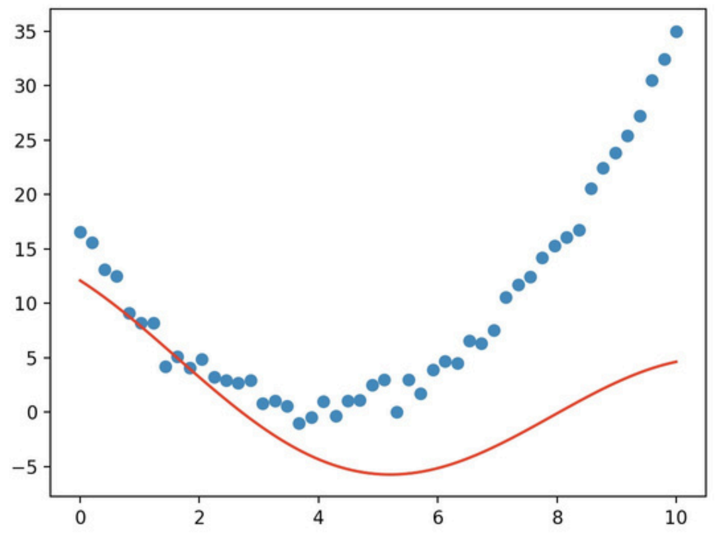
■

◯ K = 10000



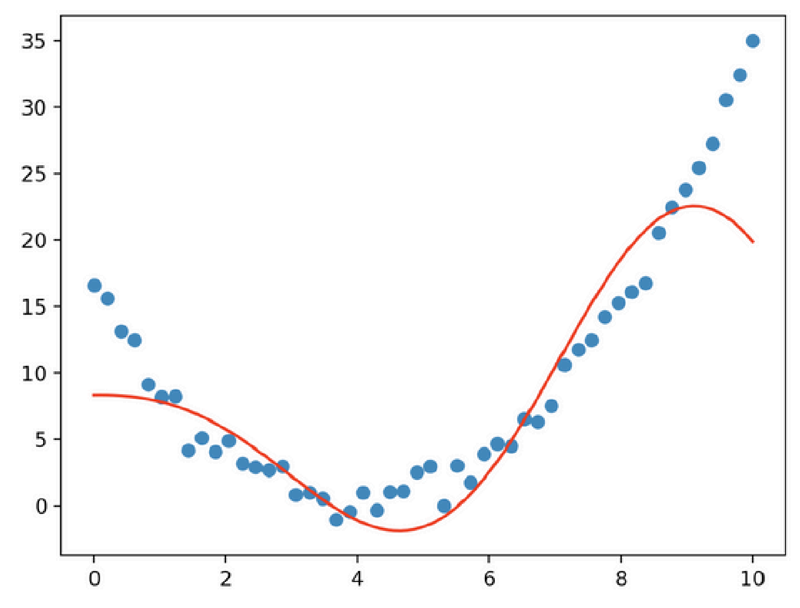
▪

- 1D-quad-uni.txt
    - K = 1



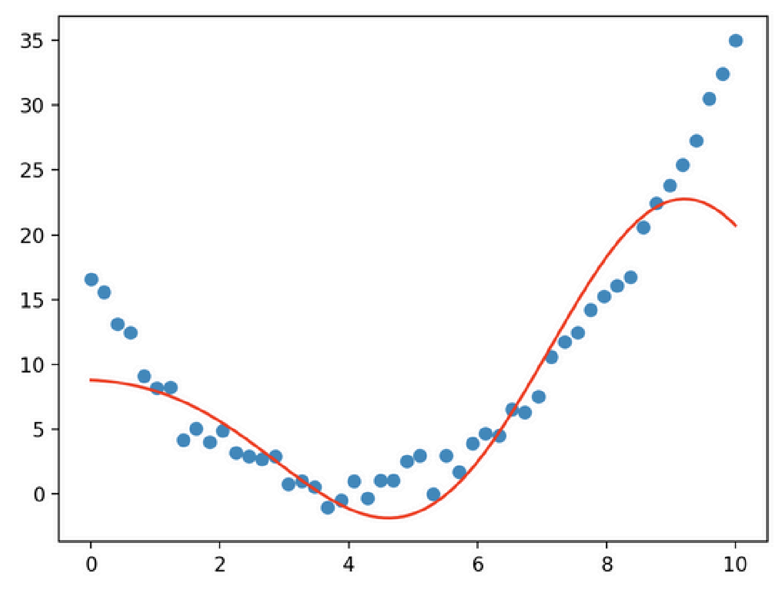
▪

- K = 1000



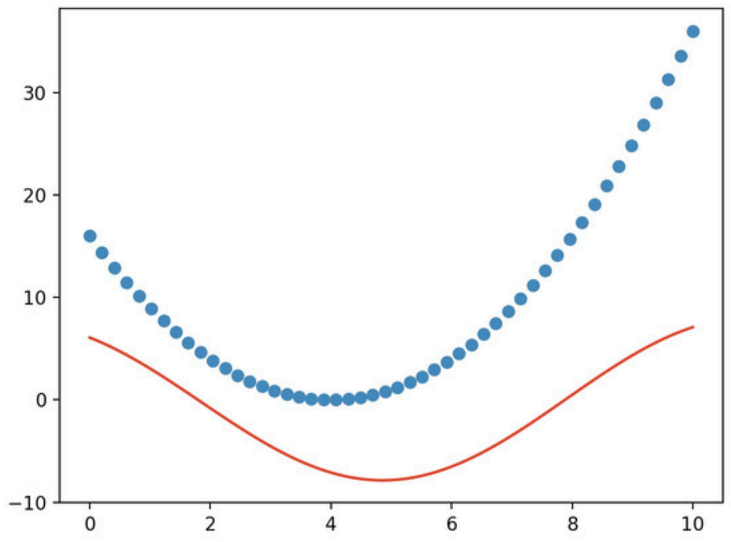
▪

- K = 10000



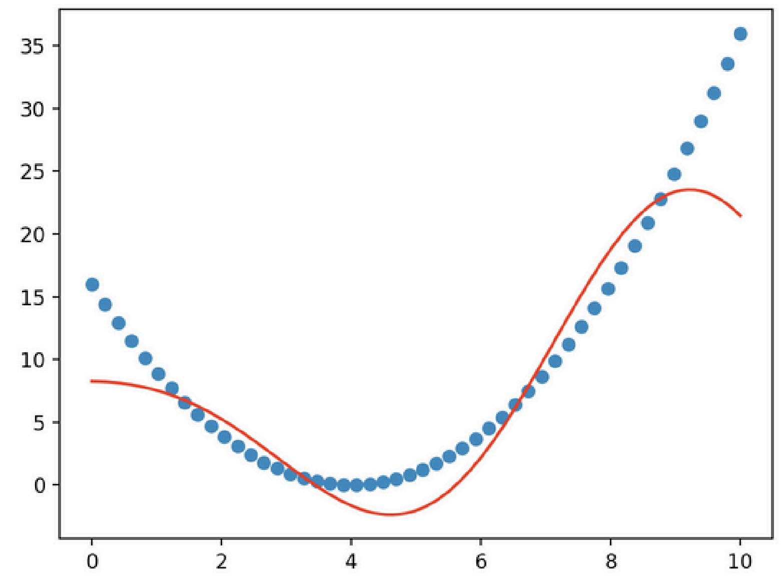
▪

- 1D-quad-uni-noise.txt
    - K = 1



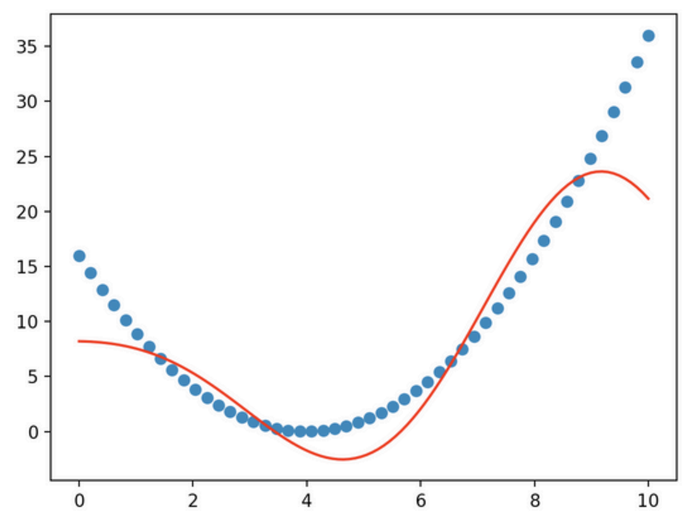
■

○ K = 1000



■

○ K = 10000



■

There was a lot of confusion for the random Fourier features portion of this homework. We could not determine what the mean and covariance matrix values for numpy.random.multivariate_normal() should be, so we used another random distribution we found in the numpy documentation. We have used features of 10 and 100 thousand as well and we didn’t see them changing our random Fourier features much either so we’re fairly sure that the issue has to do with the random() we’re using for Omega, not the data size. But we could be wrong.