## Q1)

### a)

### b)

Since :

Now we have to add logarithm from both sides:

Now we should take the first derivation of the equation and put it 0:

Let’s first consider the other possibilities where or :

When we have:

When we have:

So we will get:

Now we have to consider three possibilities, the min/max global can be in a place where > 0 or where . We calculated these two possibilities. The other possibility is that, the global min/max is where the . Since this possibility cause a non-smooth place which it has no derivation, we have to use gradient descent algorithm with its iterative way. In other words, when we have a non-differentiable point.

We can take the partial derivation for to be zero at zero and for and for . However, this descent is too slow because it has a tendency to jump around the 0 and we cannot reach that point because of the which does not gradually decrease near zero and tend to jumps between those values for and .

So our only choice is to use proximal methods. We use a thresholding idea which works for non-smooth . We have and as two values. is our regularization parameter. So we will have:

Which

### c)

As the question mentioned, and , so we will have:

Then:

If we add logarithm to both sides, we will get:

Now we have to take gradient of both sides:

If we equate this equation to zero:

## 2-

### a)

|  |  |  |
| --- | --- | --- |
| **Number of features** | **Average Error** | **Comments** |
| 5 | **0.487565110574** | **Works well** |
| **50** | **0.203603855578** | **Works well** |
| **55** | **0.202266996658** | **Works well** |
| **60** | **0.198231318126** | **Works well** |
| **70** | **\*\*\*\*\*Error\*\*\*\*\*** | **Not working due to “Singular matrix” Error** |
| **100** | **\*\*\*\*\*Error\*\*\*\*\*** | **Not working due to “Singular matrix” Error** |
| **200** | **\*\*\*\*\*Error\*\*\*\*\*** | **Not working due to “Singular matrix” Error** |
| **300** | **\*\*\*\*\*Error\*\*\*\*\*** | **Not working due to “Singular matrix” Error** |
| **385** | **\*\*\*\*\*Error\*\*\*\*\*** | **Not working due to “Singular matrix” Error** |

Table 1 - Comparison of FSLinearRegression by increasing the features

We get singular matrix error, when the equations we are trying to solve has no unique solution, or in other words, when it is invertible and not full ranked. In the “regressionalgorithms.py”, class of “FSLinearRegression” we have this line of code:



The method “inv” in numpy, tries to invert the matrix which is not full rank probably due to dependency between rows of the matrix (which is more probable since by adding features, some of them are related to each other, so by adding new features we increase the chance of the matrix not being invertible and full rank). So the matrix , is not invertible and we encounter error. Now, the solution for non-invertible matrices is to use pseudo-inverse for them. When we use pseudo-inverse we can run the code again without any error. The method, “pinv” in numpy is made for these reasons. It first checks, whether a matrix is invertible or not, if it is not, it gets the pseudo-inverse of the matrix instead.



The table below shows the result by using pseudo-inverse:

|  |  |  |
| --- | --- | --- |
| **Number of features** | **Error** | **Comments** |
| 5 | **0.487565110574** | **Works well** |
| **50** | **0.203603855578** | **Works well** |
| **55** | **0.202266996658** | **Works well** |
| **60** | **0.198231318126** | **Works well** |
| **70** | **0.198001997835** | **Works well** |
| **80** | **0.197980536126** | **Works well** |
| **100** | **0.194747571802** | **Works well** |
| **200** | **0.157619521821** | **Works well** |
| **300** | **0.154053912132** | **Works well** |
| **385** | **0.160677468541** | **Works well** |

Table 2 - Comparison of FSLinearRegression by increasing the features (getting pseudo-inverse)

### b)

In the “script\_regression.py”, the program gets the error of the prediction and the test cases:



For finding the standard error, we need to first get the standard deviation. For getting the standard deviation, we import “utilities.py” and then get the standard deviation by this line:



And now we should divide it on square root of the number of runs:

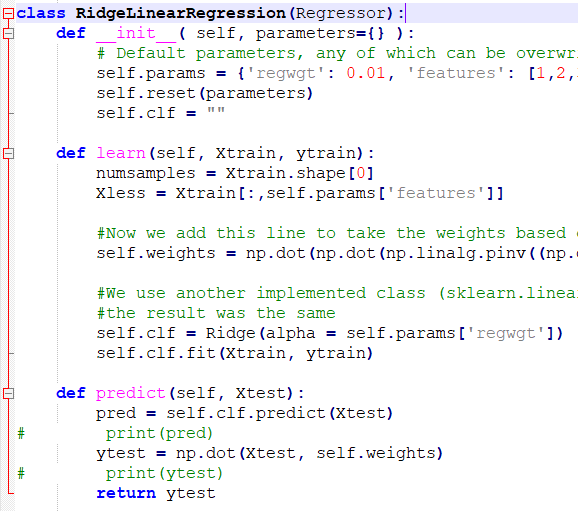


We also used the method of numpy for getting standard deviation to check it with our result. The result was the same. The output is:



### c)

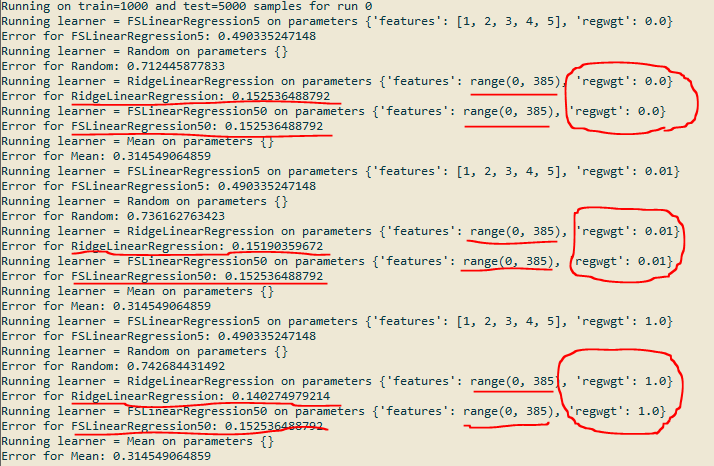
The code is implemented in “regressionalgorithms.py”, class of RidgeLinearRegression:



I also used “sklearn.linear\_model.Ridge” to test our results with implemented tools in python for ridge linear regression and I got the same result.

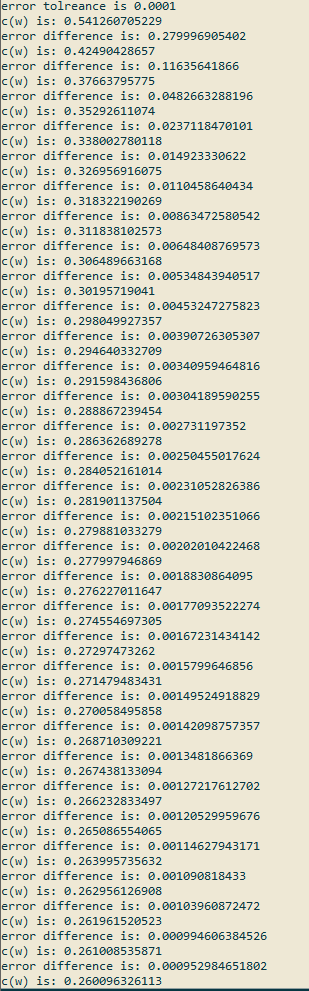
NOTE: I removed the code of sklearn since the TA may have not this library and finds error by running the code.

The result for one run is in figure below. As you can see, when the regularization parameter is 0, the feature select linear regression and the ridge regression, for 385 features, behave the same. That is because we put the regularization parameter to 0 which means ridge regularizer has no effect and is equal to 0. Now when increased to 0.01 and 1, the difference of the error between these two algorithms also increased. **It is happening** **because** the chance of overfitting is also reduced and it behaves as a new information which is added to our previous predictions.



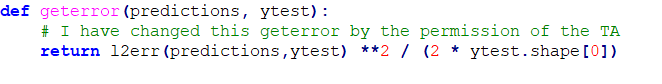
### d)

This part is implemented in regressionalalgorithms.py, with the class name of LassoRegression. As the question asked batch gradient descent is implemented. The way it converges to one point is shown in the figure below:

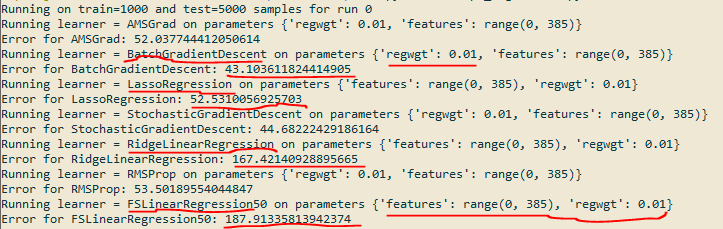


### e)

NOTE: before doing this question, with TA permission, I have changed the getError function in script\_regression.py to:



So the results will vary with previous ones that I have reported.

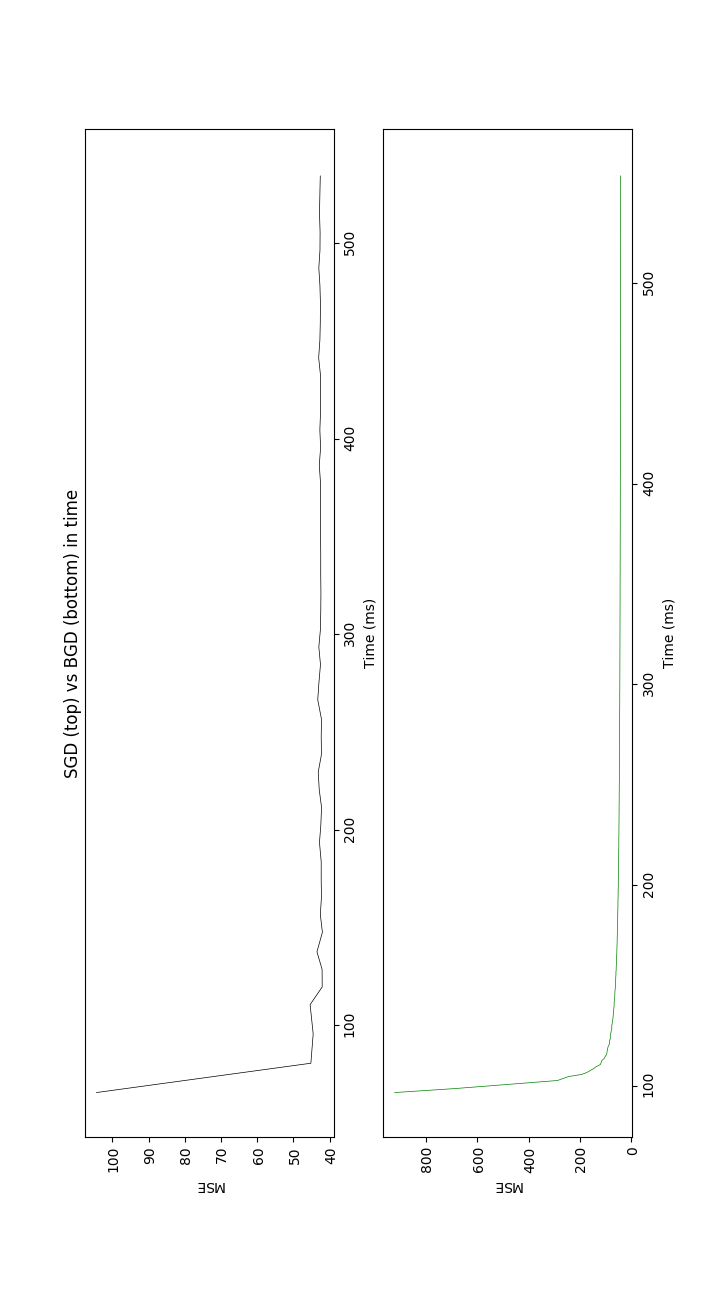
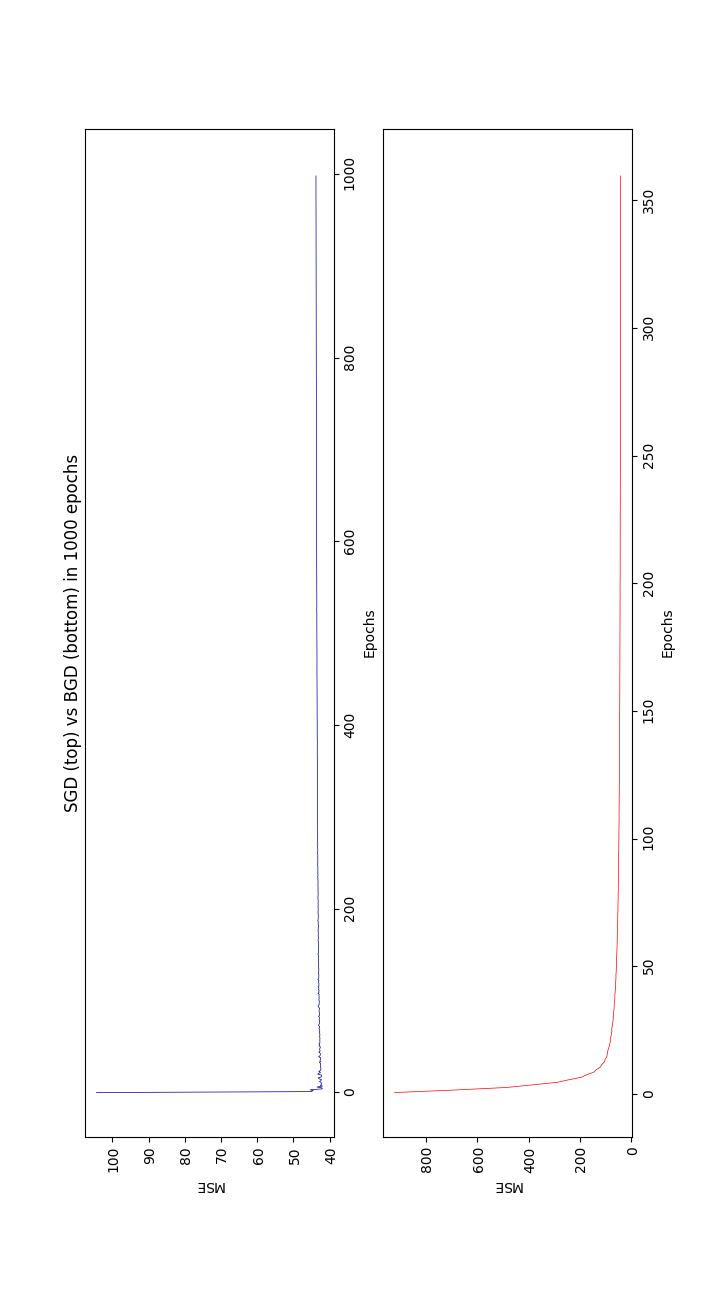
The error is: 

The error value is less than FSLinearRegression and RidgeLinearRegression as it is shown in the picture above. The step size is 0.01 as it is asked and it is calculated in 1000 epochs.

### f)

The batch gradient descent is implemented and its comparison with Stochastic Gradient Descent is shown in picture below.

NOTE: since some of the BGD experiments finished without running 1000 times on training set, I just ran it multiple times to find one BGD experiment that finished running 1000 times on training set. The result of this experiment is plotted. But the second one is about the running time comparison, and since the BGD in most of the experiments had lower running time, I just got the minimum of the running times for plotting. (second figure)



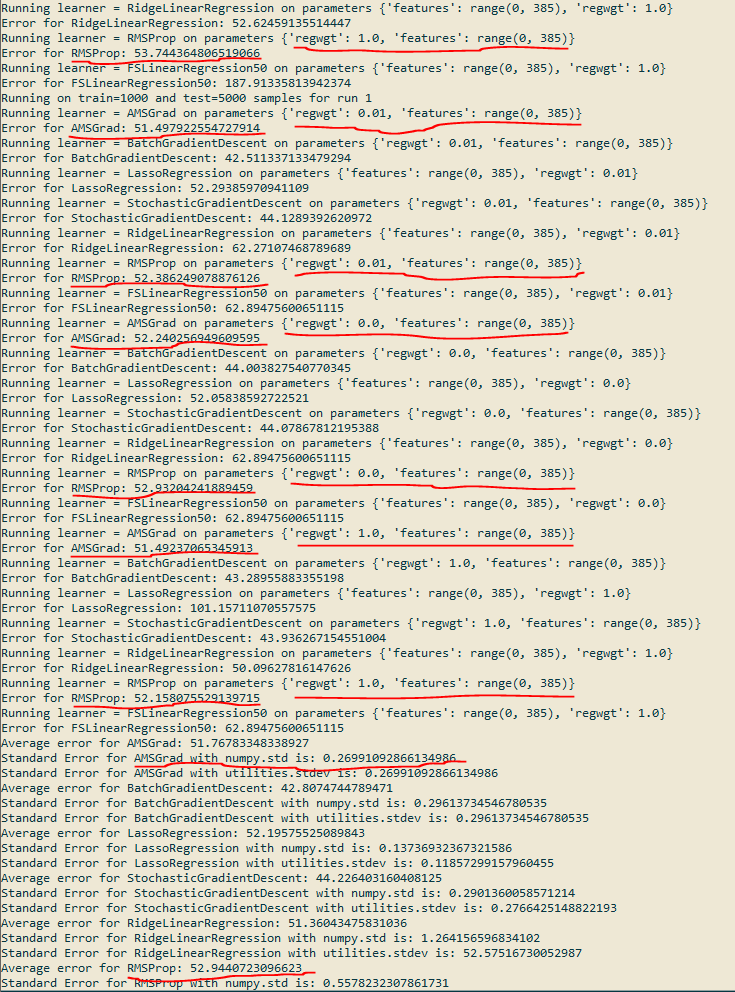
## Bonus Questions:

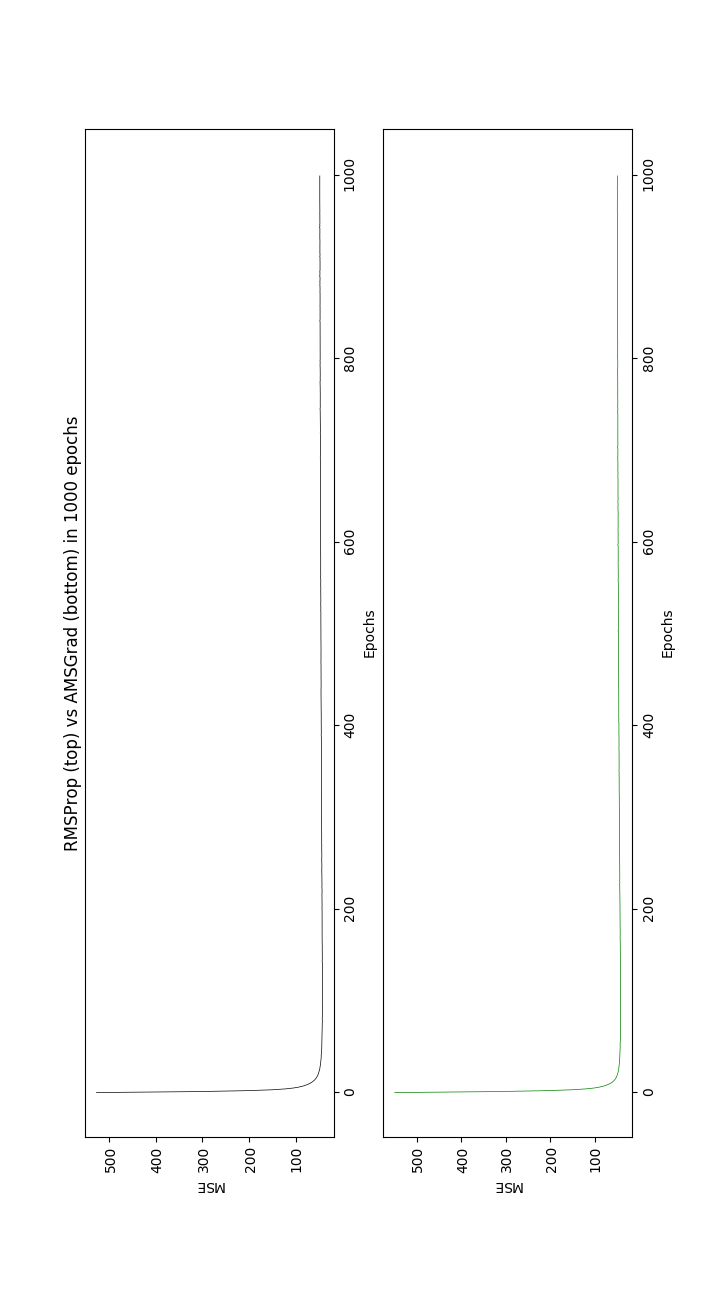
### a , b)

As you can see in the picture below, the error of the RMSProp and AMSGrad are so similar to each other:

RMSProp in 3 experiments: 52.386, 53.744, 52.158 – AVG Error: 52.944

AMSGrad in 3 experiments: 51.497, 52.240, 51.492 – AVG Error: 51.767





Contribution with Fatemeh Modares