# How to run the code?

You can Simply run the [W,log] = myoptimization(Y,X,W0,C,incremental) function.

**Input:**

Y: N\*d2 target values.

X: N\*d1 training values.

W0: d2\*d1 Matrix, which is the initial value for the optimization

C: The sparsity value

Incremental: Option to choose if an incremental method be used to optimize or not. By turning on this option, more iterations are needed to optimize. But, at each iteration it just use a few number of samples.

**Output:**

W: The optimized weights vector.

Log: the true error value after each 100 iterations. This values are used to draw the progress of the optimization and comparisions.



# The problem definition:





We are trying to solve the regression problem. And also we have sparsely constraint on the mapping function. (W)

For the sparsity part, we used the l1-ball projection [1] method, which has a linear running time to project a point from the outer space of the l1 ball to the inside of that. The method simply find a subset of dimensions which do not violate the constraint. Then using that, it finds a value to map each dimension value to a new value. You can find the algorithm in the figure 2 of [1]. This projection is done in each iteration, when the W is not in the l1-ball, to hold the constraint. The function which performs the projection named W = projection(W,C);

We are going to make some examples of the code running to know more about the effect of the value C and also to use incremental subgradient. After that, we will discuss about the details of the implementation.

We built an X and a W and produced Y respect to W and X. Then, we also add a small noise to the Y.

When the noise is considerable, for example in our case it is about 1% of the real value of Y, my code find a W, even better than the original W.

|  |  |
| --- | --- |
| D:\Screen Shot 2015-04-23 at 8.50.54 PM.png | D:\Screen Shot 2015-04-23 at 8.52.16 PM.png |

When there is no additional noise, the found W is very similar to the ground truth.

D:\Screen Shot 2015-04-23 at 8.46.27 PM.png

Another experiment that I had done was about the cases that the original W is outside of the l1-ball. In these cases, in all the iterations, the optimum W does not change. It just alternate in one direction to the outside of the l1-ball and the projection brings it back to the first place.

When our original W is sparser, this effect decreases.

Convergence rate and the value of C:

* In all the curves, I am showing the error value and the iterations number. I got an instance each 100 iterations from the error.

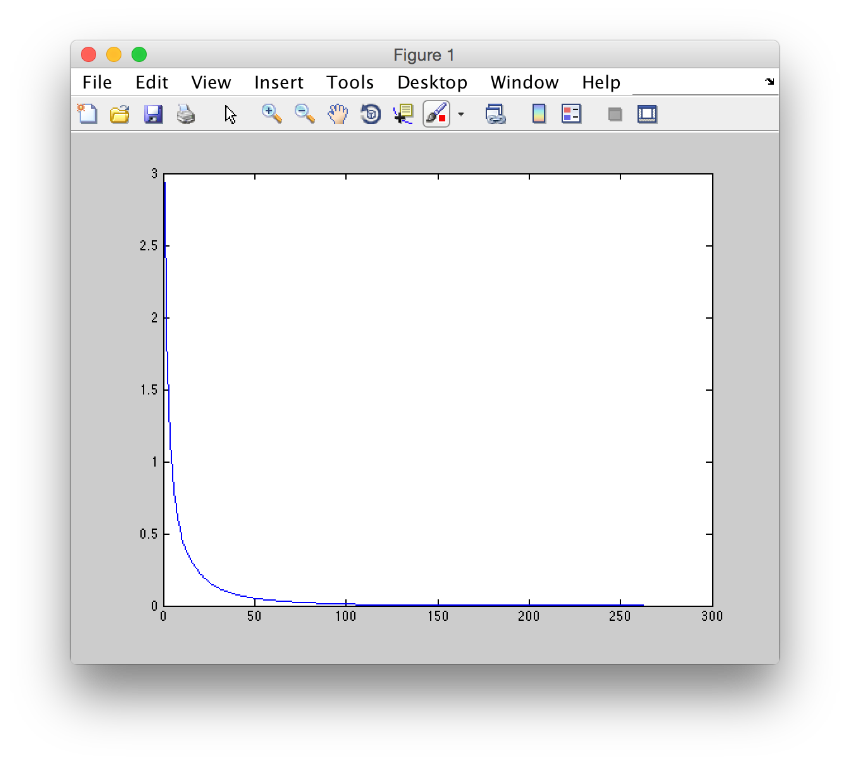
The slope of the curve shows the convergence rate.

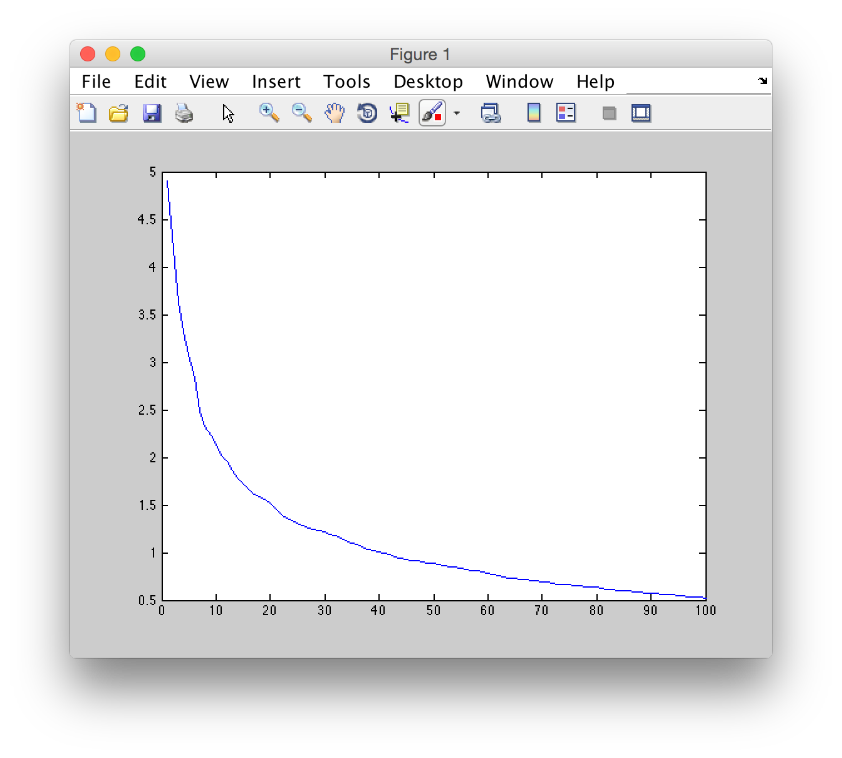
In this experiment the sparsity of the original W was 36.28. I built the curve using different values of C which you can see the |W| and also the slope of convergence.

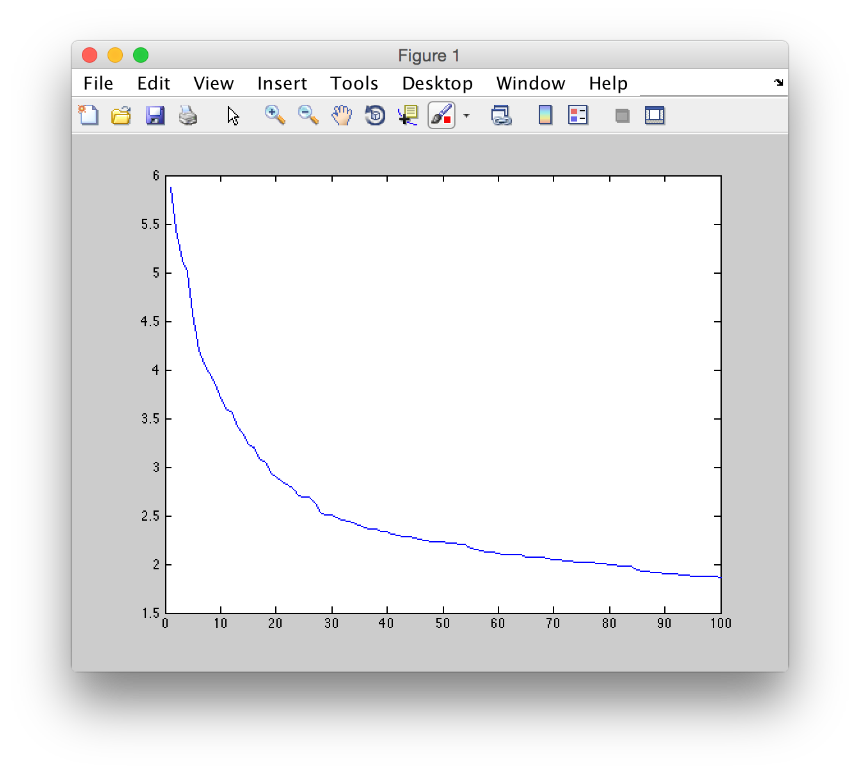
* A very important point is that, when the real W is inside the l1-ball, there may be more iterations but finally we get the better answer. But in other cases there may be even less iterations to the convergence which is on the border of the ball and it has a bigger error, but this is the best one inside the ball.
* If we consider the equal number of iterations the comparison will be more meaningful.
* For example you can compare the rate of the second one and third one rate before they reach 2 (200’Th iteration). Also, the first one obviously has the largest convergence rate.
* Also, you can observe the final error value when the ball becomes smaller and the original W value is outer of the ball.
* An important point which I discussed with Dr. Qi, is that, this data set built based on a Gaussian distribution. If it W was built using a more sparse distribution, the error rate would be less sensitive to the value of the C.

|  |  |
| --- | --- |
| D:\Screen Shot 2015-04-24 at 12.27.48 AM.png | D:\Screen Shot 2015-04-24 at 12.29.01 AM.png |
| D:\Screen Shot 2015-04-24 at 12.33.24 AM.png | D:\Screen Shot 2015-04-24 at 12.33.09 AM.png |
| D:\Screen Shot 2015-04-24 at 12.34.09 AM.png | D:\Screen Shot 2015-04-24 at 12.33.55 AM.png |

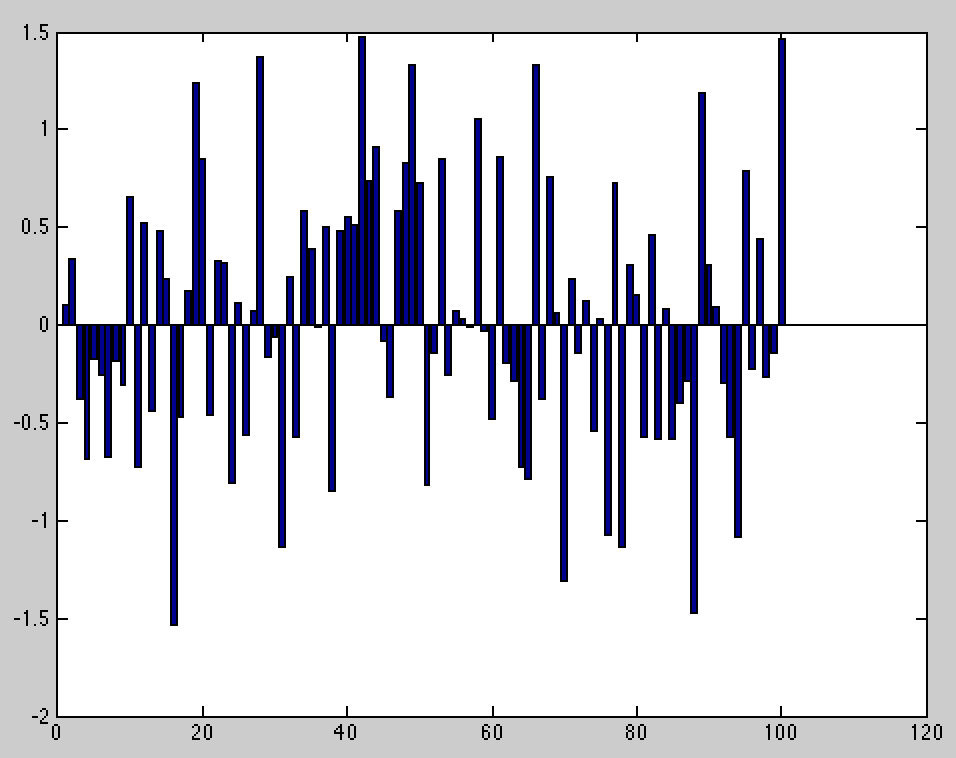
Another experiment was on the effect of the incremental algorithm. In case of incremental, in each iteration, we choose a few of the data and we just get the derivatives and updates of W using those samples. The following figures show that, when we use incremental method we need more iterations to converge and also the number of samples that we use matters in our convergence speed. The first figure is non-incremental and the second one is incremental using 30 samples in each iteration and the last one is incremental with using just 3 samples at each iteration.

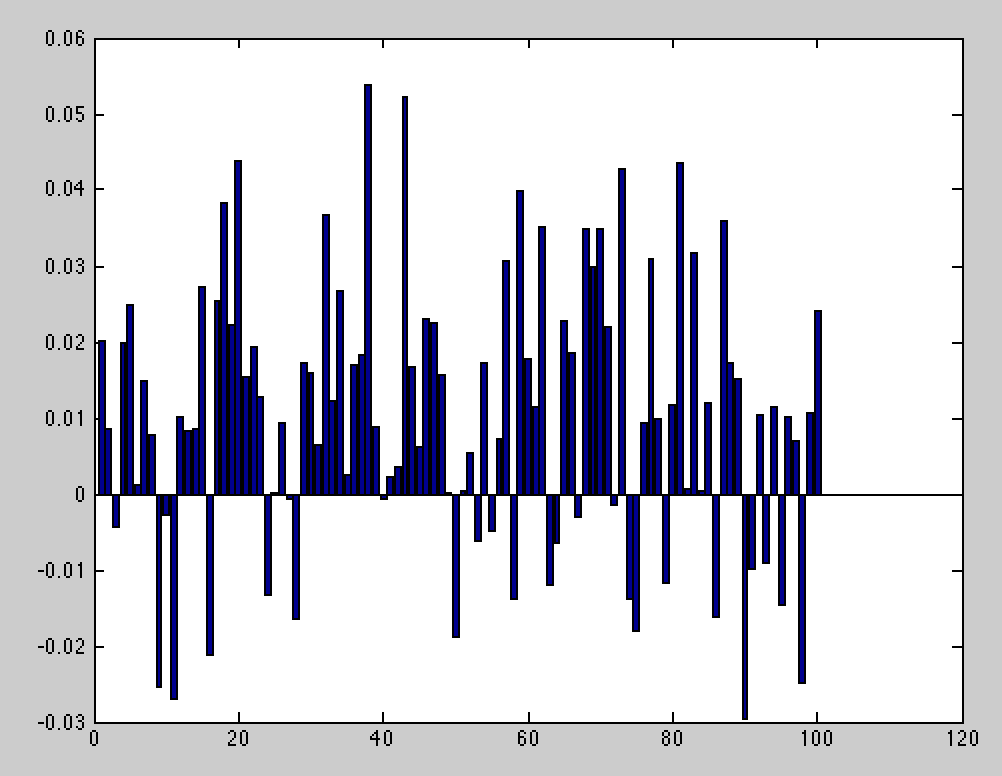






The following figures show the improvement on all the samples during the iterations. Each bar shows one data sample error. The first figure is for one of the first iterations and the second figure shows the errors for one of the last iterations.





**Implementation details:**

**Matrix Support:**

The projection to l1-ball algorithm is for when the W is vector (Y is a one dimensional label).

I used the W in a way that each column is multiplying to one dimension of the data. (W\*X’). So, in case of having matrix instead of vector for Y and W, I take a sum of columns of W and pass this vector to the projection algorithm. After getting the output of the projection, I compare the original sum of each column with the output of the projection and then scale the elements of that column in order to have the same size as the output of the projection code.

**Step parameter:**

This parameter determine the amount of movement in the direction of the gradient. If the step be too big, we may see an increase in the error, even after moving toward the gradient direction. If the step size be too small, the running becomes too long.

In order to regulate the size of the step parameter, we dynamically change it. When in an iteration the error increases we divide the step size by two and when for some iterations the amount of step does not change, we increase it by 3/2.

**Gradient:**

I used the sub-gradient descent method. We get the derivative of the norm-2 regression function as shown above and move to the opposite direction of that to reach the minimum. Because this function is differentiable, the gradient is a sub-gradient by itself.

**Incremental part:**

The part for optimization with the incremental method, I used a copy of the original code. The only difference is that, in each iteration, just a few, for example 3, samples are used to get the derivative and compute the new error.

* We pass all the data to the function in order to find the true error in each step to draw the figures to compare with the non-incremental mode. But, just a few of samples are used for derivatives in incremental mode.

References

[1] “Efficient projections onto the l 1-ball for learning in high dimensions” J Duchi, S Shalev-Shwartz, Y Singer,

T Chandra. Proceedings of the 25th international conference on Machine learning, 272-279