Homework2 / Linear Filter

李豪韋 (HW-Lee) ID 103061527

Overview

In this homework, the goal is to implement the stochastic gradient descent method (SGD, also known as LMS) that is covered extensively in class, by Hinton, and also in Haykin’s textbook. It will be applied with real data (GradeData_HW2.csv) as well as fake data generated by the function getFakeData().

The real data are basically the same data given to you in HW1, but the .csv file now also includes the total score. The fake data are randomly generated via *principal component analysis* (PCA) to have the same 1st and 2nd-order statistics as the real data.

More specifically, the mission is to find out a set of linear combination weights that best predict the total score out of individual activities (i.e., columns).

Before proceeding to implement SGD, note that the true weights can be directly inferred by solving a pseudo inverse problem in the common least-square sense. This least-square procedure is coded at line 16 of Hw2_Starter.m:

$$\mathsf{w\_ls} = \mathsf{inv}(\mathsf{inp}' * \mathsf{inp}) * (\mathsf{inp}' * \mathsf{des});$$

Therefore, with this in mind, the result would include comparison between the weights obtained by SGD and the **true** weight w_ls. Hopefully the weights given by SGD will converge to w_ls in the stochastic sense.

Implementation

1. **Functions**

   - **Hw2LinearFilter.train(X, y)**

     Return a reference of a linear predictor trained with training set X and corresponded ground truths y.

   - **Hw2LinearFilter.predict(X)**

     Return a vector of values corresponding to unknown testing set X.

   - **Hw2LinearFilter.visualInfo()**

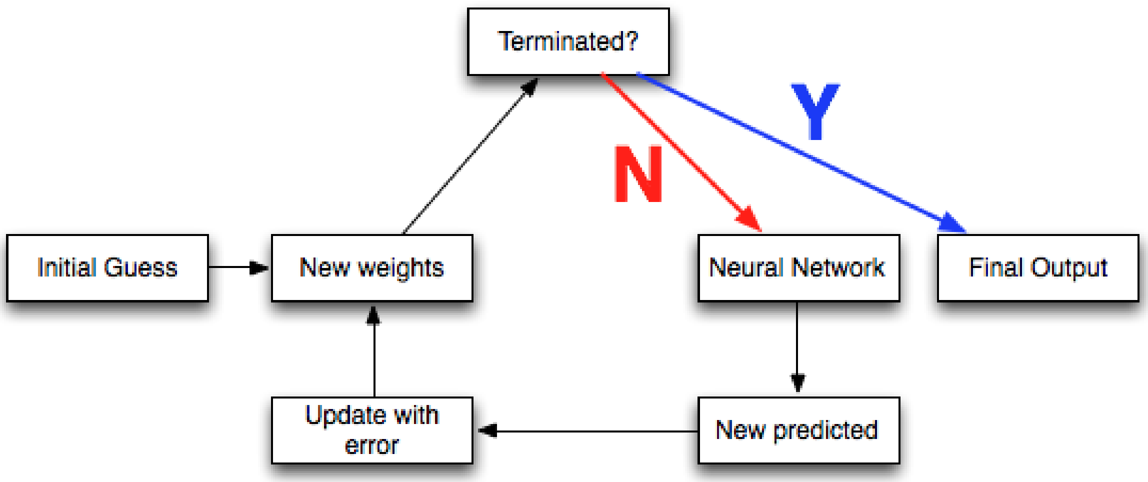
     Show the property of the predictor in a visual way, including parameters w, b, and all w’s, b’s, and e’s during training process.

2. **Process**

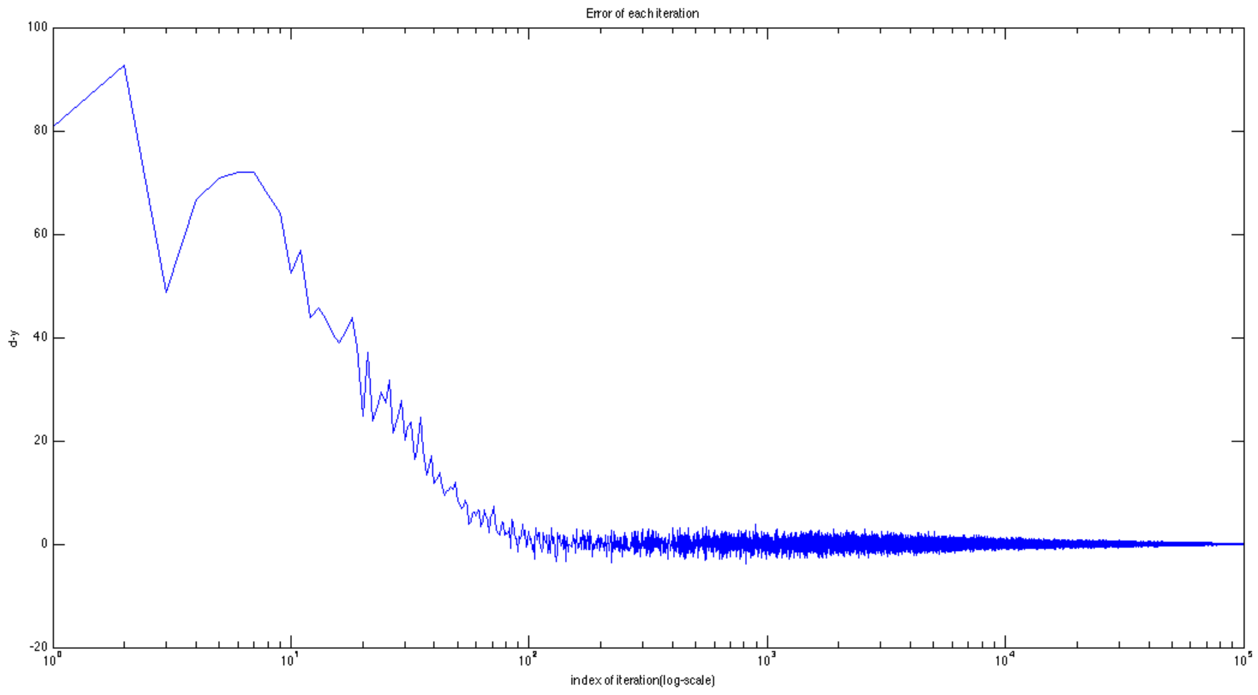
- **Initialize the weights**
- **Update the weights with error until terminated condition is satisfied**

$$\text{In batch-mode: } w = w - \eta\left(-\frac{1}{N}\sum_i (d_i - y_i)x^{(i)}\right)$$

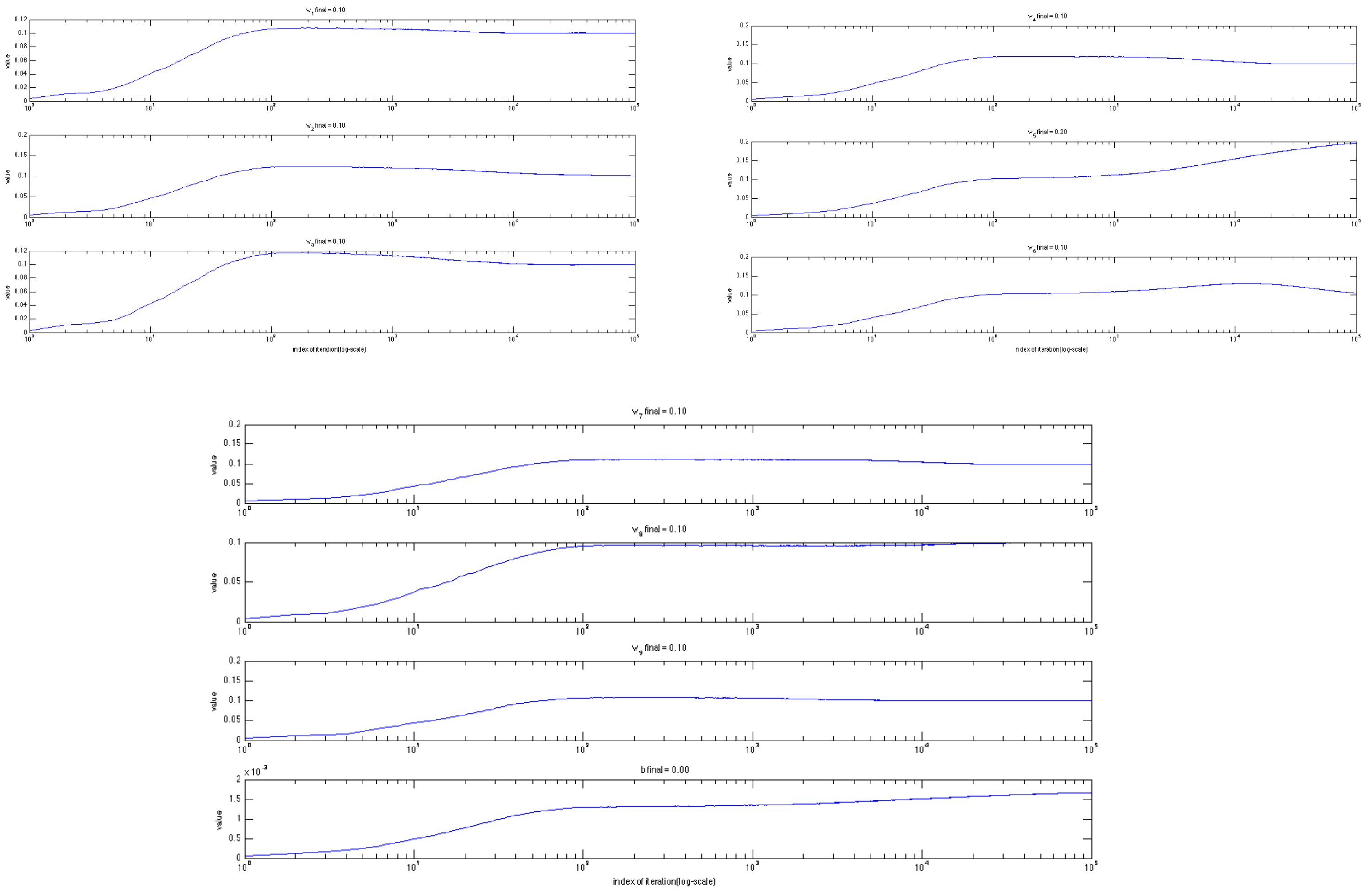
$$\text{In online-mode: } w = w + \eta(d - y)x$$



Results



[Fig.1: Error value (d−y) w.r.t. iteration index]



[Fig.2: Parametrogram of the linear filter]

The figures above are: 1) difference between desired and predicted value v.s. iteration index plot, note that the x-axis is log-scale (for ease of observing the decreasing trajectory); and 2) parameters of the linear filter v.s. iteration index, respectively. They have proved that the linear filter did work, being able to decrease the difference between desired and predicted values, and will get converged as the iteration time gets larger. In the results, η is given with $7e-7$, $1e5$ iterations, and the final weights are $w_{\text{online}} = \begin{pmatrix}0.1001 & 0.1006 & 0.0998 & 0.0997 & 0.1961 & 0.1043 & 0.0996 & 0.0998 & 0.1001\end{pmatrix}^T$, $w_{\text{batch}} = \begin{pmatrix}0.0999 & 0.0999 & 0.1000 & 0.0997 & 0.1956 & 0.1041 & 0.0993 & 0.0997 & 0.1001\end{pmatrix}^T$.

Discussion

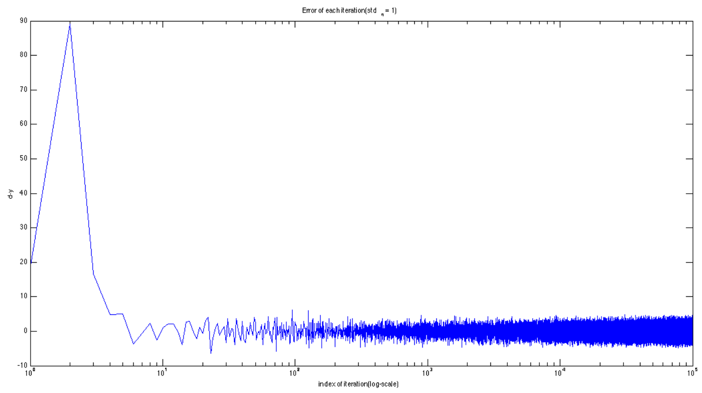
1. How much does your weight vector deviate from the **true** weight w_ls?

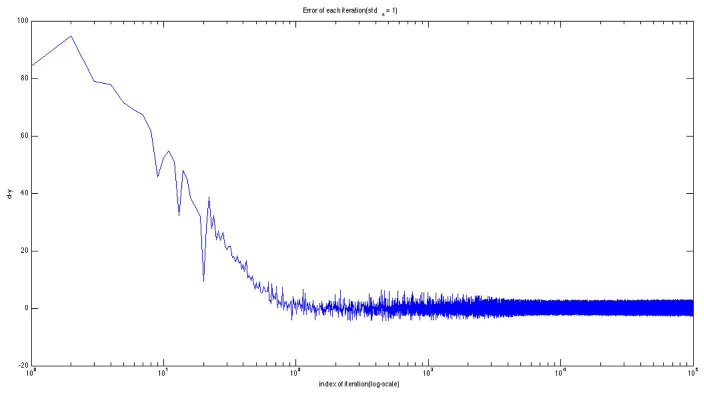
   After iterating the training algorithm with a large enough times, the weight vector will be very 'close' to w_ls.

2. What is the maximal learning rate (η) you can allow before the system becomes unstable? Can this only be found by **trial and error**? Or is there a guideline for the choice of an appropriate learning rate?

   I simply calculate the trace of $X^T X$: (Use MATLAB command $\mathsf{sum}(\mathsf{diag}(\mathsf{X}' * \mathsf{X}))$), and the 'conservative' convergence upper bound $\frac{2}{tr[R_x]} = 3.0873e-07$. Therefore, there is a strategy determining the value of learning rate: η should be close to the convergence upper bound and can be slightly larger than it to reduce the number of iterations needed to get the weights converged.

3. Is there a trade off between the learning rate and how small the approximation error can get?





   Take the original training data plus standard deviation 1 noise for example: the left-side figure is applied with learning rate $1e-5$ while the right-side figure is applied with $7e-7$. Obviously, the error margin of the right-side figure is smaller than the right-side. Then, we can conclude that the increase of learning rate will accelerate the speed of convergence while sacrifice the error margin after large iterations.

4. When sampling the real data, does it help to keep drawing the samples randomly?

   In practice, this operation can only accelerate the speed of convergence. (It will be discussed more specifically below)

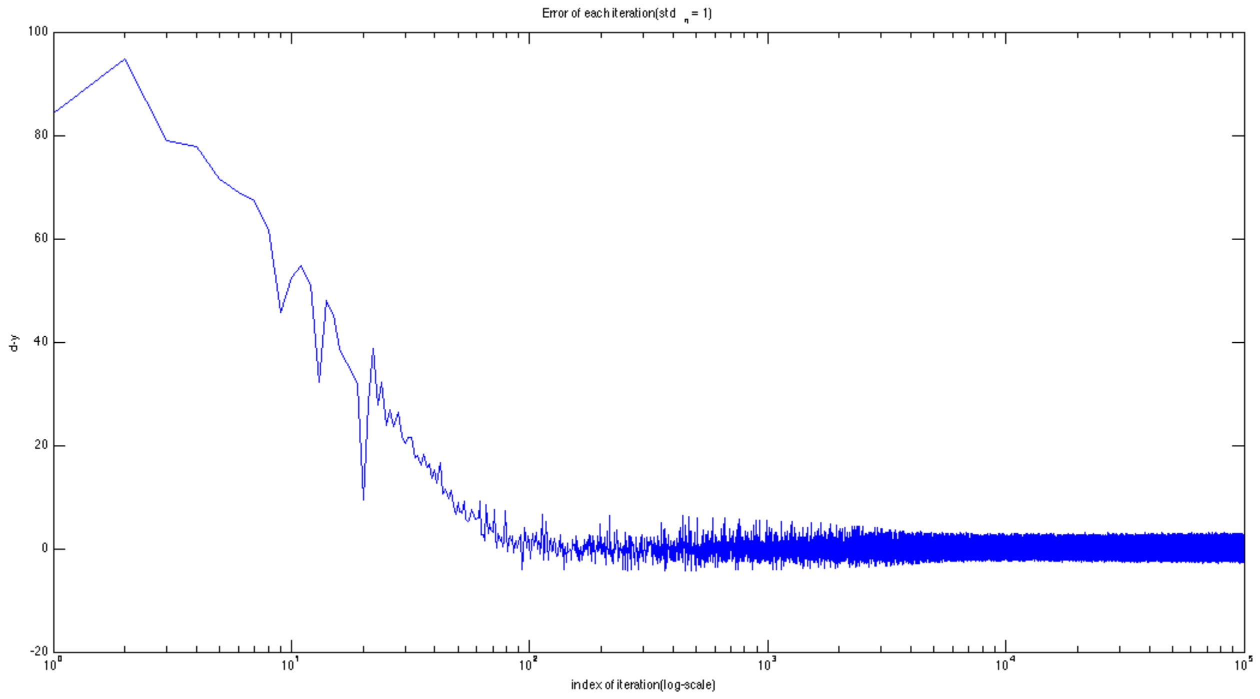
5. (Continuing from above) If so, does it help to draw the samples repeatedly in the same order?

   Yes. Under a large iteration time, the weights will get converged in any way.

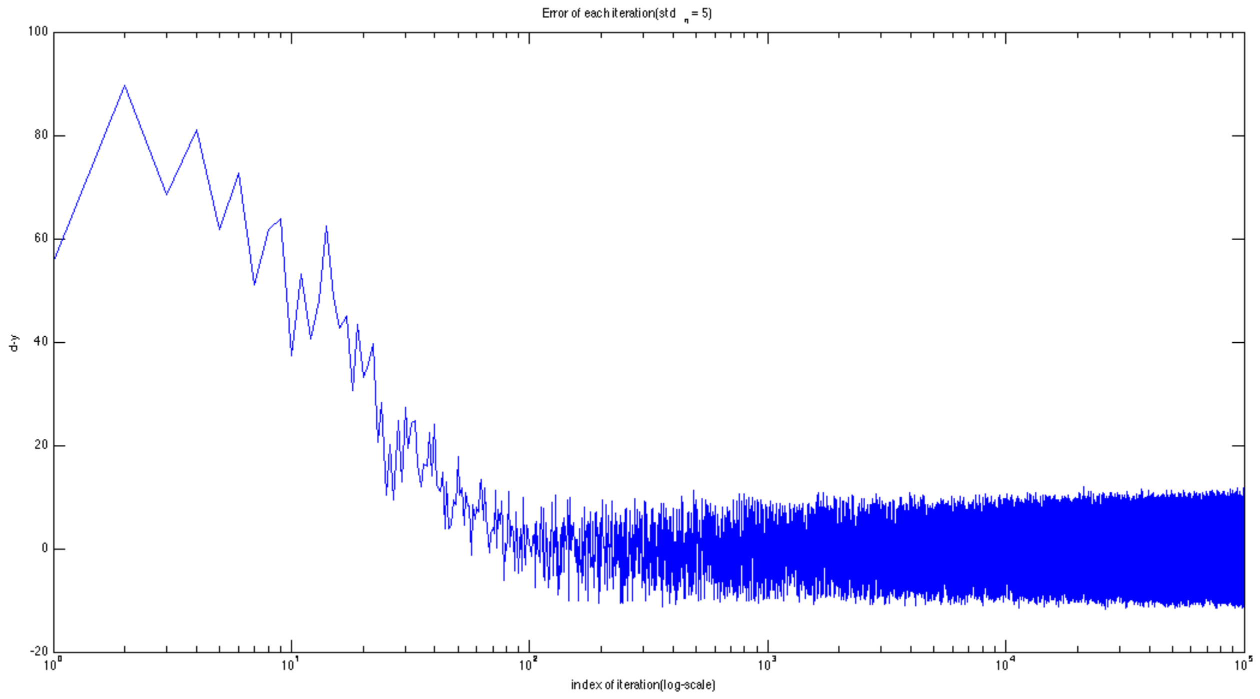
6. In the for-loop, how would the performance be changed if the observation vector d is subject to additive noise? E.g., let $\mathsf{d} = \mathsf{d} + \mathsf{randn}(5, 0)$ under a predefined standard deviation $\sigma = 5.0$. Would SGD fail to converge? Would the performance degrade?

   According to the Figure.3 and Figure.4 below, it is possible that SGD will fail to converge when noise is added. Form.1 shows the error with various noise (different standard deviation), and we can know std of noise is positively correlated to error.

More detailed implementation is available on
$\mathsf{https://github.com/HW\text{-}Lee/2015\text{-}NN\text{-}Homeworks}$.



[Fig.3: Error w.r.t. iteration index with signal plus 1 std noise]

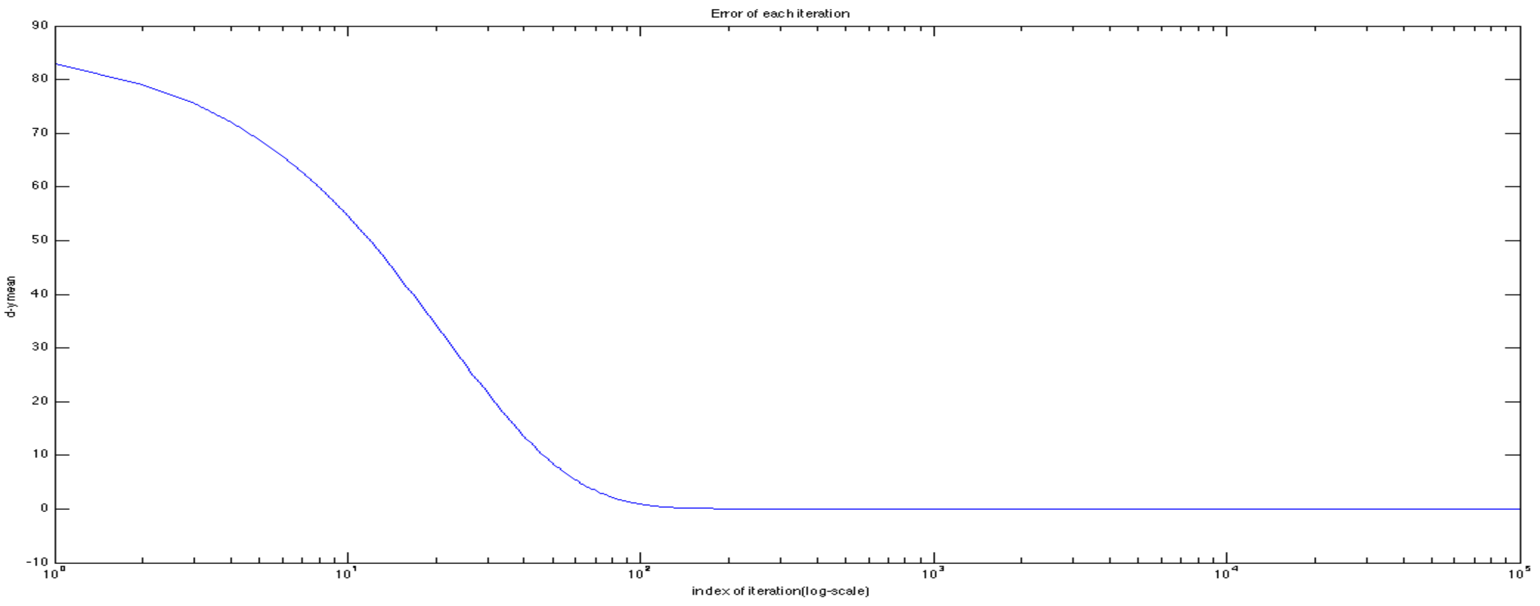


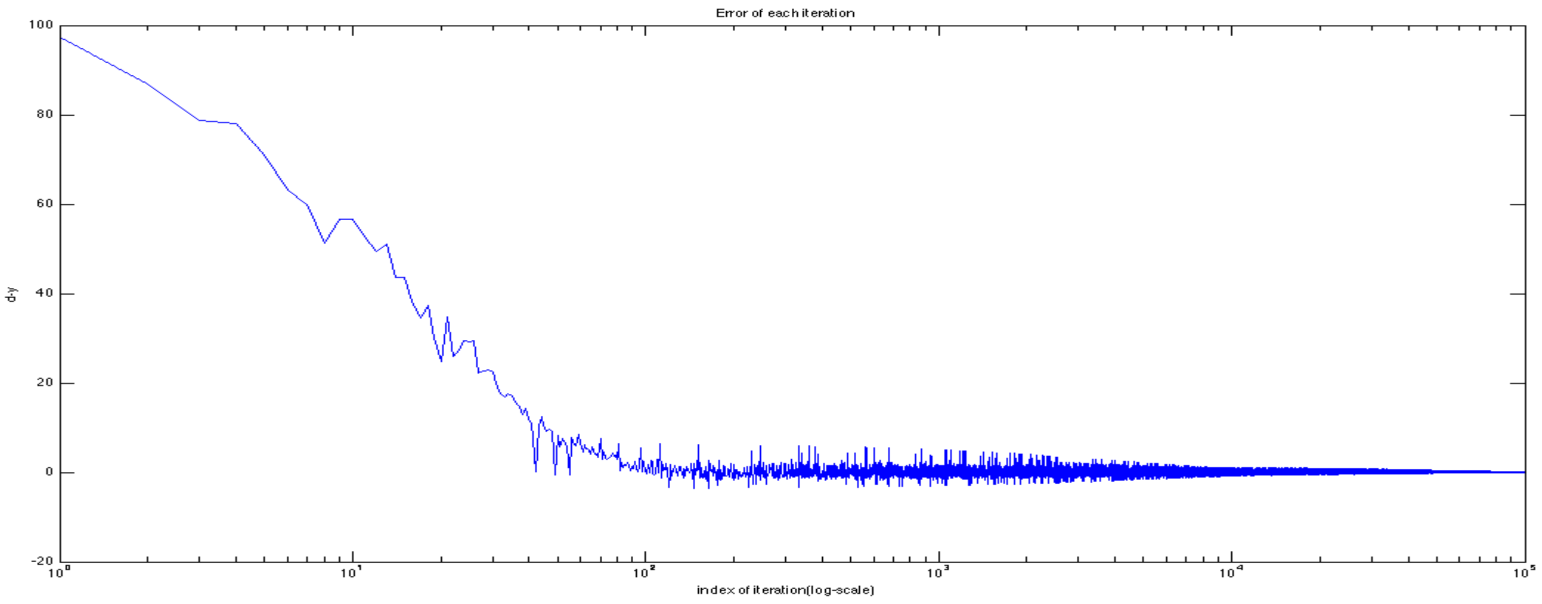
[Fig.4: Error w.r.t. iteration index with signal plus 5 std noise]

σ	1	2	3	4	5
ϵ	0.7227 ± 0.3937	1.5506 ± 0.9835	2.1620 ± 1.1115	2.3936 ± 1.2747	3.0411 ± 1.5018

[Form.1: Error with various noise added]

Additional Discussion:
Results from batch-mode and online-mode





In this section, there are two figures shown above: 1) the upper one shows error value with batch-mode, and 2) the lower one shows with online-mode . According to these figures, we can get:

- The speed of convergence is not significantly different in these two different method.

- The error curve is smoother in batch-mode than in online-mode, because batch-mode updates weights after looking through all data.

- Batch-mode trains the linear filter by repeatedly feeding the whole data **in the same order**, it shows that the 5^{thit} question in discussion section is true.

- Followed by above, batch-mode also shows that randomly sampling from real data cannot change the performance we finally get. (the 4^{th} question in discussion section)