Project 2: Learning to rank using Linear Regression

CSE 574 – Machine Learning

Karanjeet Singh (Person No: 50169916); karanjee@buffalo.edu

Overview:

The project was to implement linear regression to solve regression problem and evaluate its performance.

We were provided with a real world set data from Microsoft LeToR MQ2008 dataset and a synthetic data.

Evaluation of the closed form solution for both was done and then stochastic gradient descent algorithm was applied to both the datasets so as to get similar result to closed form.

Closed Form solution to the real world dataset:

Firstly we parsed the data into the desired form and imported the data into matlab. Did the parsing of data using Python. I had then separated the input and output features. The data was also divided into three sets: training (80%), validation (10%) and testing (10%).

There were 46 input and 1 output features in the dataset.
In closed form solution Gaussian radial basis functions were used to create a design matrix and hence forth used the matrix to obtain the weights.

Gaussian radial basis function takes one observation at a time and we see the difference with some random values picked from the datapoints. The number of these values decides out M factor of linear regression. Then we find the sigma, whose inverse is multiplied by the calculation we did above and with the transpose of the above calculation. Here we assumed the sigma to be normally distributed and used it as diag(var(training_data)). We also introduced a factor to the above formula to remove singularity of the matrix.

Gaussian radial basis function formula:

```
exp(((-0.5)*((training_input1(i,:)-mu1(j,:))*(inv(sigma1)*transpose((training_input1(i,:)-mu1(j,:)))))));
```

```
sigma1=diag(var(training_input1))+eye(46)*0.3;
```

we loop through the data and get the design matrix.

I had chosen M as 6 in the end after comparing the erms of various M values.

In the end we get a design matrix of size N*M; where N is the number of observations.

```
for j=1:M1
   Sigma1(:,:,j)=sigma1;                    // creating sigma as required in the mat file
end

M1=6
m1=randi([1 55698],1,M1);                   // random numbers generation
mu1=zeros(M1,46);

for j=1:M1
   mu1(j,:)=training_input1(m1(j),:);       // random datapoints generation
end

design_matrix=zeros(55698,M1);              // initializing design matrix

// creating the design matrix here using the basis functions

for i = 1:55698
   for j= 1:M1
      if(j==1)
         design_matrix(i,j) = 1;
      else
         design_matrix(i,j)=exp(((-0.5)*((training_input1(i,:)-
mu1(j,:))*(inv(sigma1)*transpose((training_input1(i,:)-mu1(j,:)))))));
      end
   end
end

// after creation of the design matrix, we generate the weights after using a regularizer. I
used a combination of lambda and M values and found the optimized error rate for them
and chose lambda which would suffice for stochastic gradient descent too.

l=0.3;
lmda=eye(M1)*l;

// finding the weights here using the formula provided

w1=(inv((transpose(design_matrix)*design_matrix) + lmda)) * (transpose(design_matrix)
* training1_output);

// finding the squared error for the weights found
```

```
sqared_error=(transpose(training1_output) - (transpose(w1) *
transpose(design_matrix))).^2;
trainPer1 = sqrt(sum(sqared_error)/55698);                // sum of squared error for the
weights
```

validation error calculation:

// creating the design matrix for the validation data, using the same sigma and mu points, and finding Gaussian radial basis functions.

```
validation_design_matrix=zeros(13926,M1);
for i = 1:13926
   for j= 1:M1
      if(j==1)
         validation_design_matrix(i,j) = 1;
      else
         validation_design_matrix(i,j)=exp(((-0.5)*((validation_input(i,:)-
mu1(j,:))*(inv(sigma1)*transpose((validation_input(i,:)-mu1(j,:))))))));
      end
   end
end
```

// calculating the error on the validation data, using the weights calculated in the training step.

```
validation_sqared_error=(transpose(validation_output) - (transpose(w1) *
transpose(validation_design_matrix))).^2;
validPer1 = sqrt(sum(validation_sqared_error)/13926);

mu1=transpose(mu1);                // to get the same form as required
```

Finding the values of M and lambda using the closed form was pretty straight forward, where we had to apply the formulas using a combination of M and lambda.

I ran a loop for various combinations and chose the one with optimal error on the training data. Then we validate out weights on the validation set.

Variables:

M1=6
Lambda1 = 0.3
trainPer1 = 0.5615
validPer1 = 0.5951
w1 = [0.2933, 0.8937, -0.3793, -0.1976, -1.1472, 1.7340]

Batch 2 (Closed form linear regression on the synthetic data):

Here we use the same approach as done in the real world dataset. We import the data in the correct format. Create the features matrix and apply the formulas as above.
Here again we try to find the best combination of the M and lambda. After testing various values on the test set and validation set I found the values to be:

M2=6 and lambda2=6

Here I have used a larger regularizer as we had only 10 input features and the dataset was also small comparatively.

The data was separated into the training set (80%) and validation set (20%)

We use the basis functions again with a sigma related to the data in this dataset.

```
sigma2=diag(var(synthetic_input1))+eye(46)*.3;

for j=1:M2
   Sigma2(:,:,j)=sigma2;                    // creating sigma as required in the mat file
end

for M2=6
m1=randi([1 1600],1,M2);                    // generating 6 random values from 1 to 1600
mu2=zeros(M2,10);

for j=1:M2
   mu2(j,:)=syn_train_inp(m1(j),:);         // creating the mu points from the random values
end

syn_design_mat=zeros(1600,M2);          // initializing the design matrix

// creating the design matrix for the synthetic data

for i = 1:1600
   for j= 1:M2
      if(j==1)
         syn_design_mat(i,j) = 1;
      else
         syn_design_mat(i,j)=exp(((-0.5)*((syn_train_inp(i,:)-
mu2(j,:))*(inv(syn_sigma)*transpose((syn_train_inp(i,:)-mu2(j,:)))))));
      end
   end
end

now we find the lambda, and then the weights
l=6;                                                    // optimum value for me
```

```
lmda=eye(M2)*l;

// finding the weights for the synthetic matrix

w2=(inv((transpose(syn_design_mat)*syn_design_mat) + lmda)) *
(transpose(syn_design_mat)*syn_train_out);

// finding the squared error for the given weights on the training data

syn_sqared_error=(transpose(syn_train_out) - (transpose(w2) *
transpose(syn_design_mat))).^2;

trainPer2 = sqrt(sum(syn_sqared_error)/1600);          // erms error for synthetic data

// creating the design matrix for the validation data

syn_valid_design_matrix=zeros(400,M2);
for i = 1:400
   for j= 1:M2
      if(j==1)
         syn_valid_design_matrix(i,j) = 1;
      else
         syn_valid_design_matrix(i,j)=exp(((-0.5)*((syn_valid_inp(i,:)-
mu2(j,:))*(inv(syn_sigma)*transpose((syn_valid_inp(i,:)-mu2(j,:)))))));
      end
   end
end

// finding the validation error using the same weights found in the training set

syn_validation_sqared_error=(transpose(syn_valid_out) - (transpose(w2) *
transpose(syn_valid_design_matrix))).^2;

// erms error on validation data

validPer2 = sqrt(sum(syn_validation_sqared_error)/400);

mu2=transpose(mu2);                //transposing to provide variable in the required format
```

Variables:

```
M1=6
Lambda1 = 6
trainPer1 = 0.1446
validPer1 = 0.1548
w2 = [ 0.6079, -0.0014, 0.0642, 0.0727, 0.0330, 0.893 ]
```

Stochastic Gradient Descent on Real World Data set:

Here we take some random weights in the beginning and start looking at the datapoints one by one. We use a learning rate which determines the impact of error on our new weights. I stochastic we run through the training set and one datapoint at a time and find the error in the actual output and our prediction using the current weights. We find the delta weights ie the difference and update the weights accordingly. The learning rate determines the effect of each delta and helps us to find the global minimum of the error rate. Learning rate plays a key role in finding the minimum. A large learning rate would make the weights fluctuate a lot at each step and a global minimum is not guaranteed. A small learning rate would make our convergence very slow with the error rate. Thus we have to choose an optimized learning rate.

I used an adaptive learning rate which decreases when the difference in new weight and the previous weight is too high and increases when the difference is too low. The program is as follows:

```
// counter for the matrices to store the factors of learning rate

c=1;
for up=1.0:.1:2;                          // loop to increase the learning rate
for down=1.00:-0.1:0.00;                  // loop to decrease the learning rate
for nn = 1.0:0.1:2;                       // loop to change the learning rate

// I am using these loops to get the best combination of adaptive learning rate

w01=[5;5;5;5;5;5];                        // initial weight wector

n1=nn;
iterations=55698;                         // number of iterations
dw1=zeros(6,iterations);                  // initializing delta_w matrix
eta1=zeros(1,iterations);                 // initializing eta1 matrix

// we iterate through the datapoints, use the formula to calculate the delta_w
// we update the weight. We check the difference in updated and current weight
// if difference is small, then we need to increase the learning rate
// we are here applying difference combinations of this parameter (up)
// if difference in weights is not so small we decrease the learning rate by a factor, down
// in the end we find the best combination of learning_rate, up and down by seeing which
one gives us the minimum error.

for j=1:55698
        delta_w=((n1*(((training1_output(j) - (transpose(w01) *
        transpose(design_matrix(j,:))))*transpose(design_matrix(j,:)))-(0.3*w01))));
        updated_wt = w01+delta_w;
```

```
        eta1(:,j)=n1;
        dw1(:,j)=delta_w;

        if sqrt(sum((updated_wt).^2)-sum((w01).^2))<0.001
                n1=n1*up;
        else
                n1=n1*down;
         end
        w01=updated_wt;
end
```

// in the above when we have found the best combination, then I am populating the eta1 and the dw1 matrix.

// finding the error rate for the final updated weights, after running through dataset.

```
stochastic_sqared_error=(transpose(training1_output) - (transpose(w01) *
transpose(design_matrix))).^2;

stocastic_error_2 = sqrt(sum(stochastic_sqared_error)/55698);                    // erms error

w01=[5;5;5;5;5;5];                                  // reinitializing the weights, that were choses
```

// here we have the matrixes where we store the result of each combination of up and down. We use the error function to find the min error and its index. Through this we find the up, down and learning rate which are the most optimum ones.

```
errorrate(c)=stocastic_error_2;
upp(c)=up;
downn(c)=down;
nnn(c)=nn;

c=c+1;                    // counter to save values to the matrixes
end
end
end
```

After doing the above, we find the eta1 and dw1 for the optimized set of n1, up and down.

Thus we find the delta weights and the learning rate which is adaptive in my case.

Took w01 = [5;5;5;5;5;5]

Eta1 = 55698 X 1 matrix   // has the learning rate at each instance

Dw1 = 6 X 55698 matrix // has the delta_w of each instance
Similarly we do for the Synthetic dataset.

Stochastic Gradient Descent on Real World Data set:

We initialize the weight vector, here I took it as all 5s. We find the best learning rate and iterate though the data as well. After finding the best adaptive eta for the dataset. We use it and find eta and dw matrixes.

```
c=1;                                    // counter to update the eta parameter matrixes
```

// final value of optimized up counter, that increases the learning rate when it becomes very low. The up value was found by looping over a combination of values and then finalizing one which gives the minimum error.

```
up=1.75
```

// final value of the down factor which decreases the learning rate when it goes too high.

```
down=.59
```

// final value of the initial learning rate, this value is applied to the first row of the dataset.
// using the above combination of nn, up and down i found the gradient to descent to the minimum error position.

```
nn = 1.8

w02=[5;5;5;5;5;5];                      // initial weight vector
   n2=nn;                               // using a variable as the learning rate in the iteration
   iterations_2=1600;                   // total number of iterations
   dw2=zeros(6,iterations_2);           // initializing the dw2 matrix
   eta2=zeros(1,iterations_2);          // initializing the eta2 matrix
```

// looping over the datapoints to update the weights and eta, storing the desired values.

```
for j=1:1600

      // finding the delta_w for a given weight vector and input features

    delta_w_2=((n2*(((syn_train_out(j) - (transpose(w02) *
    transpose(syn_design_mat(j,:)))) * transpose(syn_design_mat(j,:)))-(6*w02))));

      // updating the weight vectors with the newly learnt weights

    updated_wt_2 = w02+delta_w_2;
    eta2(:,j)=n2;                       // updating the eta matrix for each iteration
```

```
    dw2(:,j)=delta_w_2;                    // updating dw2 matrix for each iteration
    // updating the learning weights depending on the weight difference

      if sqrt(sum((updated_wt_2).^2)-sum((w02).^2))<0.0001
            n2=n2*up;
      else
            n2=n2*down;
      end
      w02=updated_wt_2;
end

// finding the erms for the finally learnt weights.

   stochastic_sqared_error_2=(transpose(syn_train_out) - (transpose(w02) *
   transpose(syn_design_mat))).^2;
   stocastic_error_2 = sqrt(sum(stochastic_sqared_error_2)/1600);

// reinitializing the weight vector to what we had assumed initially

w02=[5;5;5;5;5;5];
```

Model Complexity and Performance of the models:

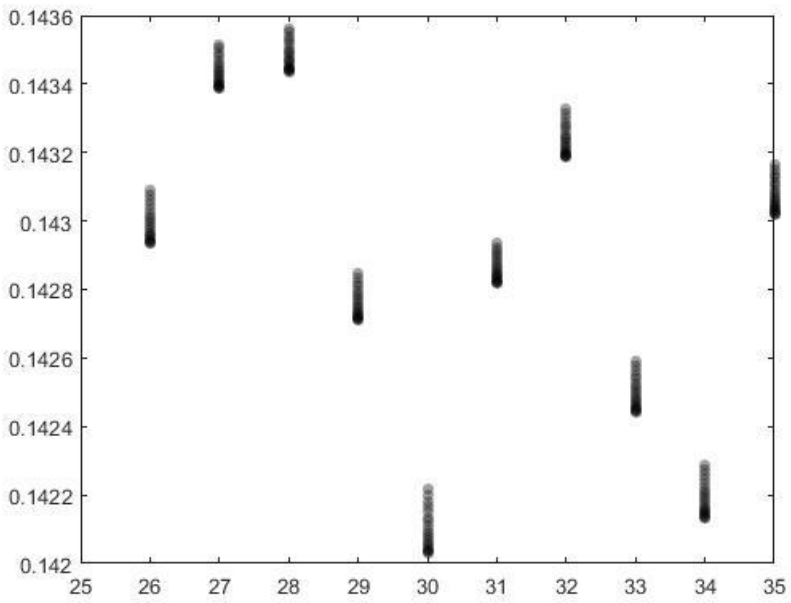
The Closed form solution to a dataset is pretty straight forward and we just have to design the matrix and use formulas to decide the optimum weights. This is a feasible approach when we are dealing with a comparatively small dataset. Applying the formulas incurs high computational cost and we don't know till the end what are results are. But for a few thousands of datapoints and less features, closed form solution should be used.

When we are dealing with a very large dataset with many features then we should use the stochastic gradient descent approach. From the very first observation, the weights start aligning themselves according to the observations and this goes on till the last observation till then, the weights are pretty much optimal. The learning rate defines the convergence of the solution but a high rate may not converge and a low rate might take a lot of time and we might have to do multiple iterations of the data. Thus, here we have one more parameter to tune, however, this provides the solution to very large scale machine learning problems.

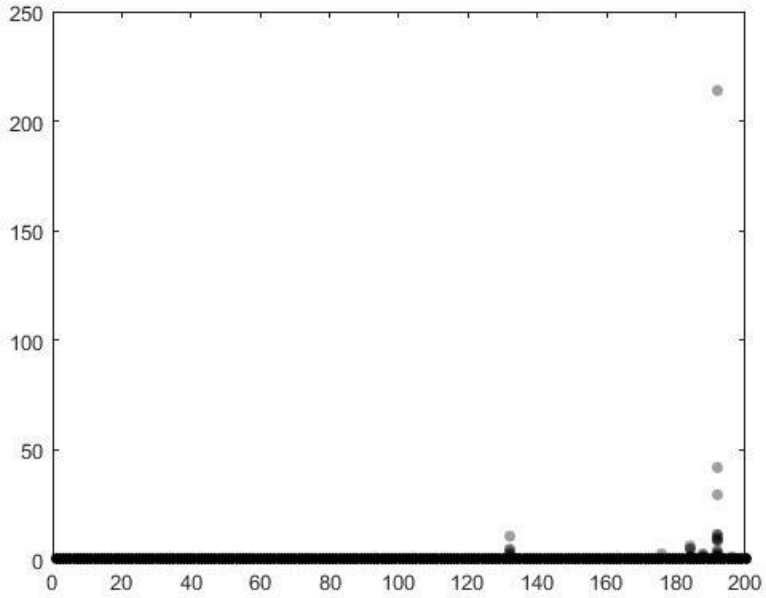
Tuning of the learning rate is very important and we look at the trend generated for a few samples and decide on what the learning rate it or how it changes.

Thus, closed form solution gives a better performance for a medium sized machine learning problem while stochastic gradient descent is used for large scale machine problems.

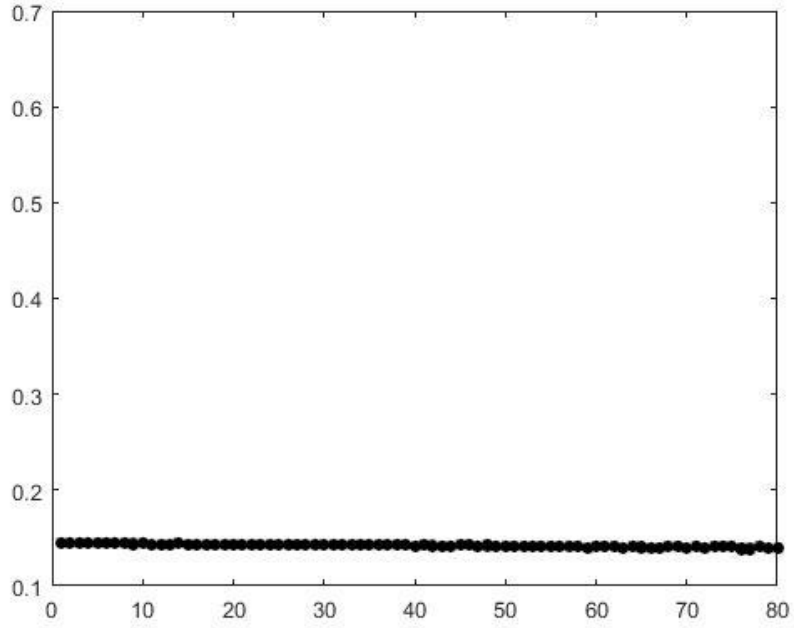
Figures:



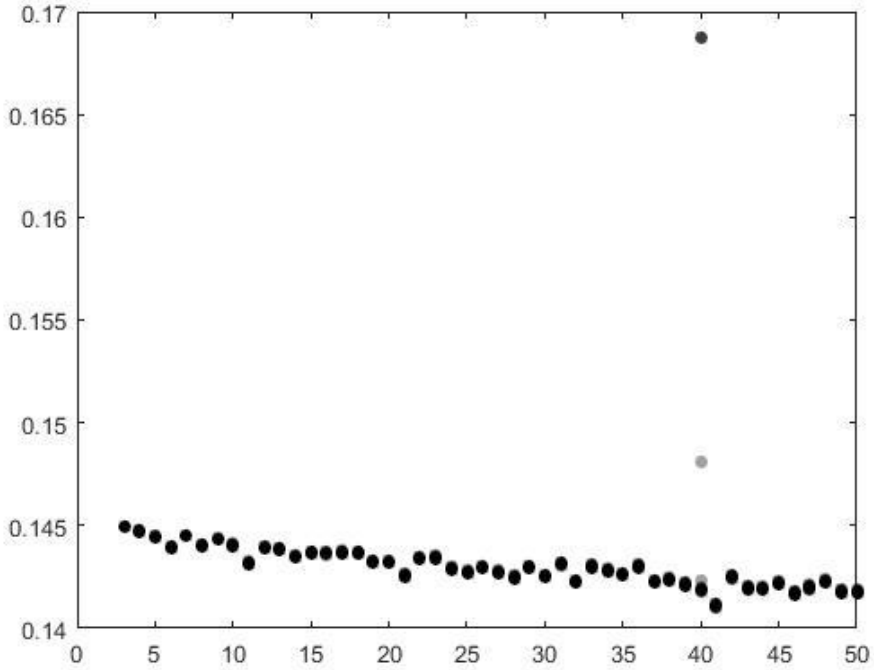
Plot between M and error rate for synthetic data



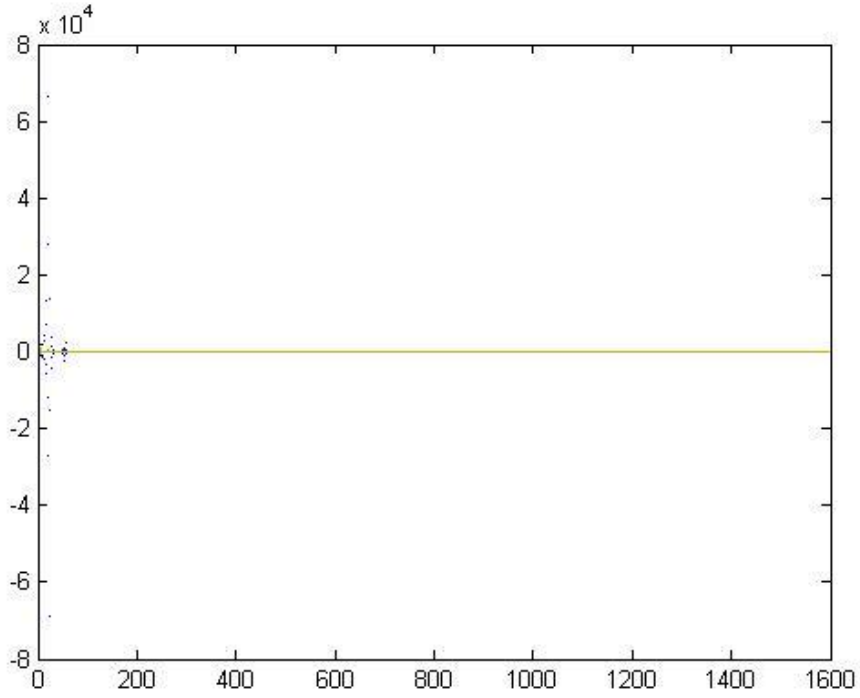
Plot between M and lambda



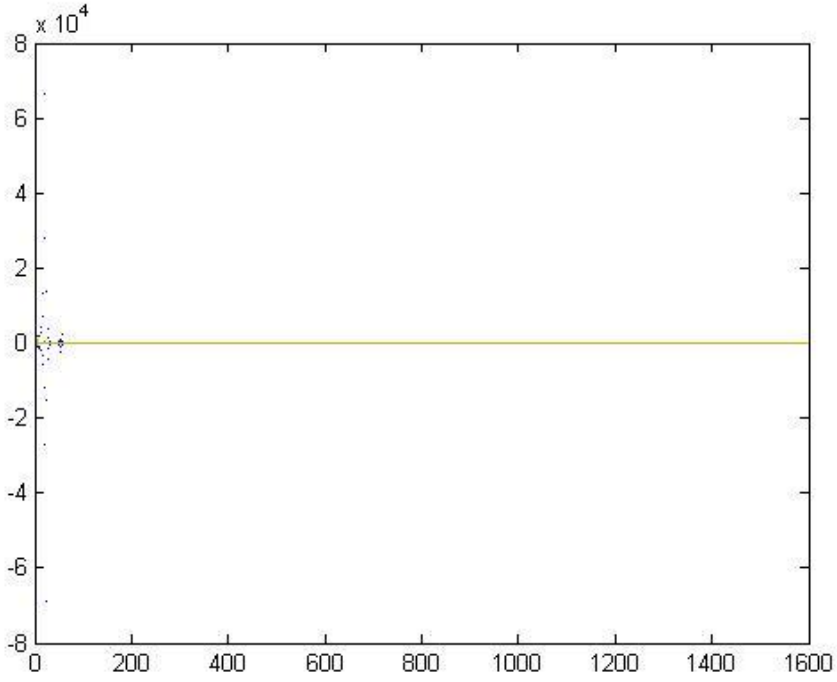
Error plot between M and error for stochastic



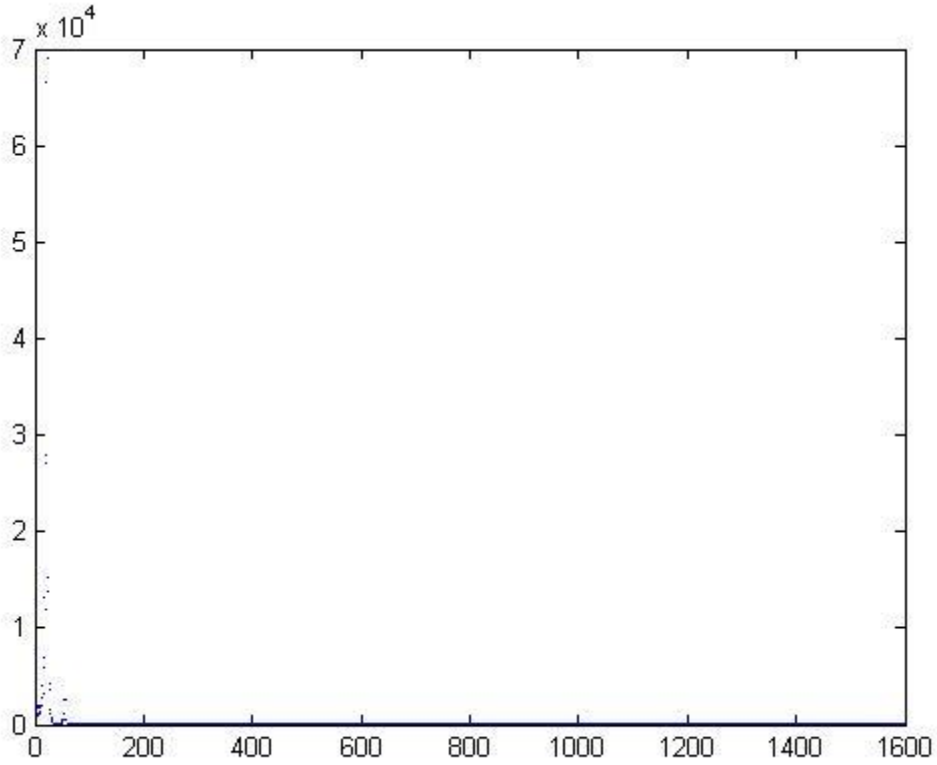
M and error for stochastic



Eta2 and iteration



Dw2 and iteration



Erms plot with iterations