Factorization-Based Data Modeling

Practical Work 2

Students : ZHAO Mengzi, ZHOU Juncheng

I – Introduction :

The problem that we aim to solve :

$$(W^*, H^*) = arg \min_{W,H} \frac{1}{2} \|M \odot (X - WH)\|^2$$

$X \in R^{I*J}$, X is the data matrix, $W \in R^{I*K}$ and $H \in R^{K*J}$ are 2 unknown factor matrixes. $\| A \|_F$ is the Frobenius norm of matrix A. M is the"mask"matrix, it is used to denoting if a particular entry of X is observed or not, if X_{ij} is observed, $m_{ij} = 1$, if not, $m_{ij} = 0$.

At the beginning, we want to introduce the structure of the code.

By using the code given by our professor, at first, we have a document which contains lots of data in format matrix, this is why we use stochastic gradient descent, when we have amount of data, if we use the general gradient descent, it will be too slow, because every time, the GD uses all data to calculate the gradient, but SGD just takes a sample to calculate the gradient.

After loading the document of matrix, we transform these data in format matrix, this matrix is sparse, because there are some zero elements. We can obtain the mask matrix, the mask matrix has 2 possibilities of value, if the value of a data exists in the data matrix (X), the value of the corresponding position in the mask matrix will be 1.

Then we calculate the number of the non-zero elements in the data matrix. We find the coordinate of every value represented by variables xi and xj, the value is represented by xs, by using xi, xj and xs, we create a list Xlist containing these 3 types of values :

Xlist = [Xi,Xj,Xs];

These variables will be use later to calculate the position of elements that we will use to calculate the gradient in the 2 factor matrixes W and H.

For matrixes W and H, we create them according to the size of the data matrix and the number of the rank k. k is to do the reduction of dimension, it represents the feature of the data matrix, in general, k is much smaller than I and J, so for us, it will be simpler to calculate with 2 smaller matrixes than a big matrix.

Both the stochastic gradient descent and the gradient descent use the same principle, the minimum value of the loss function is found in the opposite direction of the gradient vector. The difference between these two algorithms is that our common gradient descent uses all the data to calculate the gradient, but the stochastic gradient is calculated using only one sample to calculate the gradient, and the mini-batch gradient descent combines the batch gradient descent and the stochastic gradient descent, it makes a balance between the rate of each update and the number of updates, it takes m samples which is smaller than the number of data in the data matrix, it takes randomly from the training set for learning.

II – Explication of the part of code needed to complete

PART I : Matrix Factorization with Stochastic Gradient Descent

1 – Complete the stochastic gradient algorithm :

We define the size of batch as 1000, the learning rate is 0.01, the number of the iterations is 100. In every iteration, we generate a "data_index"vector containing some random numbers between 1 and N, the size of the vector is the size of the batch :

data_index = randperm(N,batchSize);

For every value n data_index vector, it will be the x-axis for the in 3 different columns in the list Xlist, as we defined before, the first column xi represents the x-axis of the data in data matrix, the second column xj is the y-axis of the data in data matrix, the third column is the value of the data in the position (xi, xj), by using every value of the vector data_index, we use it as the x-axis in the Xlist, so we want to get the value of the current position of x-axis of factor matrix W, we use Xlist(data_index(i), 1), because the first column in the Xlist represents the x-axis, the same principle for cur_j (y-axis) and cur_x (value of data in the matrix) :

cur_i = Xlist(data_index(i),1)
cur_j = Xlist(data_index(i),2);
cur_x = Xlist(data_index(i),3);

For example, if the data_index(i) is actually equal to 3, so the cur_i = Xlist(3,1), the cur_j = Xlist(3,2), the cur_x = Xlist(3,3), so the coordinate of this data that we need in

the data matrix is (Xlist(3,1), Xlist(3,2)), its value is Xlist(3,3). The value Xlist(3,3) is the value of data in the data matrix which is the true value and is used to compare to the predict value. The cur_i = Xlist(3,1) and cur_j = Xlist(3,2) is to find the position of data in the factor matrixes W and H to do the W * H by using the corresponding element in these 2 matrixes to find the predict value, the predict value is :

*cur_xhat = W(cur_i,:) * H(:,cur_j);*

Then we need to compute the gradient of W and H, we have the formula :

$$W \leftarrow W + \eta(X - WH)H^T$$
$$H \leftarrow H + \eta W^T(X - WH)$$

So we can know that the gradient is :

$$grad\_w \leftarrow (X - WH)H^T$$
$$grad\_h \leftarrow W^T(X - WH)$$

So we have :

*grad_w = - (cur_x - cur_xhat) * H(:, cur_j)';*
*grad_h = - W(cur_i, :)' * (cur_x - cur_xhat);*

According to the formula of the gradient descent, we have :

*W(cur_i,:) = W(cur_i,:) - eta * grad_w ;*
*H(:,cur_j) = H(:,cur_j) - eta * grad_h ;*

2 – Compute the roor-mean-squared-error :

$$RMSE = \sqrt{\frac{\| M \odot (X - WH) \|_F^2}{N}}$$

N is the number of the observed entries.

The RMSE is to check if the function is converged or not, if the variation of value of RMSE is small, we can know that the function is converged.

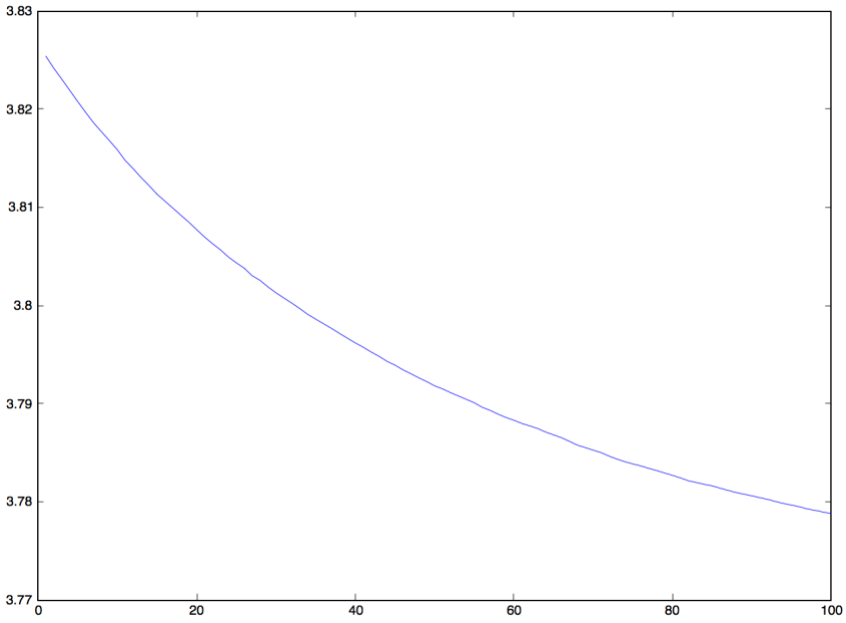
According to the formula, we have :

rmse_sgd(t) = sqrt(sum(sum(((M.(W*H))-Xsp).^2))/N);*

3 – Play with the algorithm parameters :

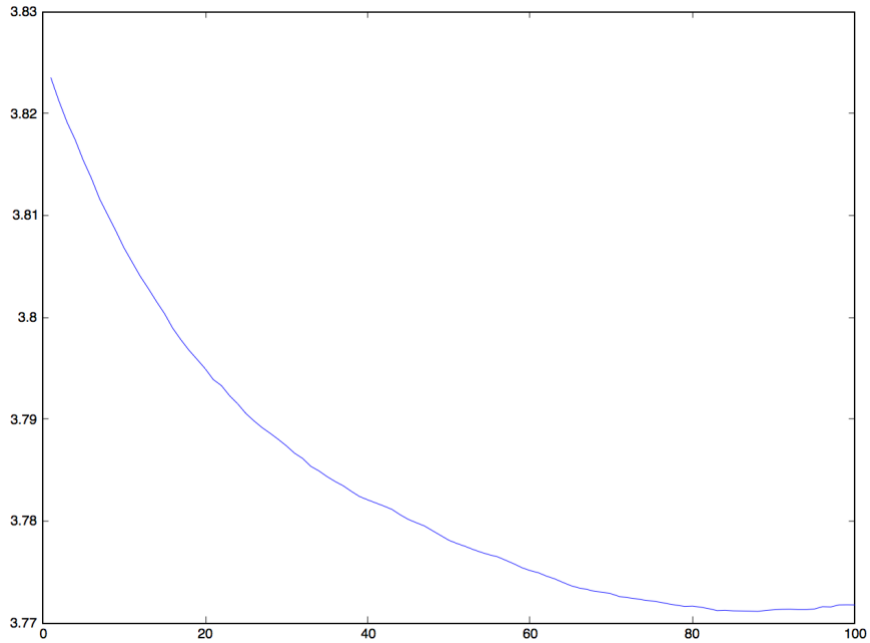
This is the original graph with original parameters :
step-size = 0.01

batch-size = 1000
rank of the factorization = 10

We notice 

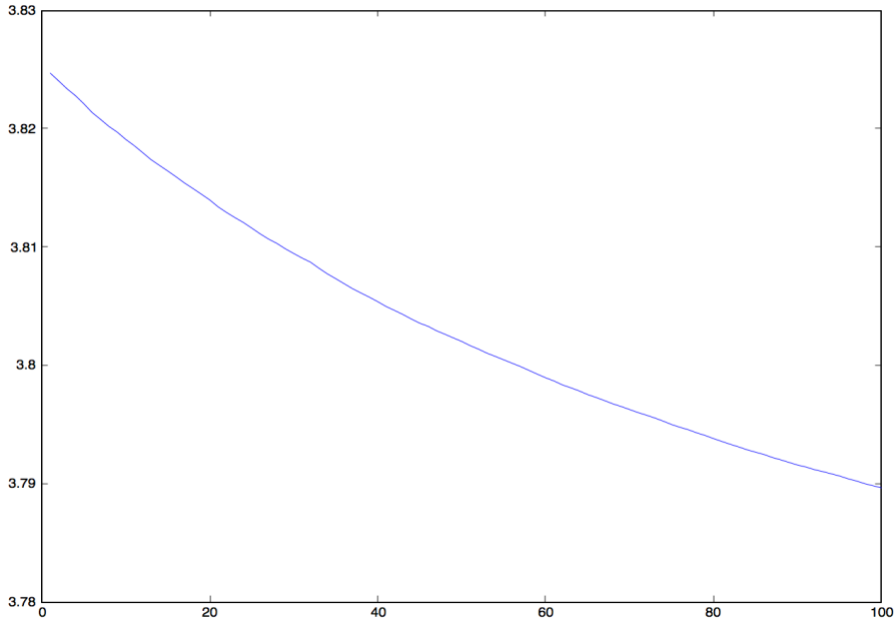
that when the number of iterations is about 100 the value of the function is about 3.779

At first, we change the parameter « step-size », we set step-size = 0.05, so we obtain a new graph :

The value 

of the objective function is about 3.771.

We set the step size = 0.005 which is smaller than the original value, we have :



The value of the objective function in this case is about 3.799.

The step-size is the length of the gradient descent for every step. If the function converges, it means the value of the step-size that people define is acceptable, if not, we need to change the value of the step-size, when the step-size is too large, it can cause the gradient descent goes too quick, it may miss the best solution, and when the value of the step-size is too small, the algorithm will use too much time to update, the function may do not converge, so maybe we can not obtain the result,
We can notice that when the step-size is larger, the function converges more quick than before, because when the number of iteration is equal to 40, in the first graph, the value of the objective function is about 3.797, but in the second graph, the value of the objective function is about 3.782. When the step-size is smaller, the function converges more slow than before.

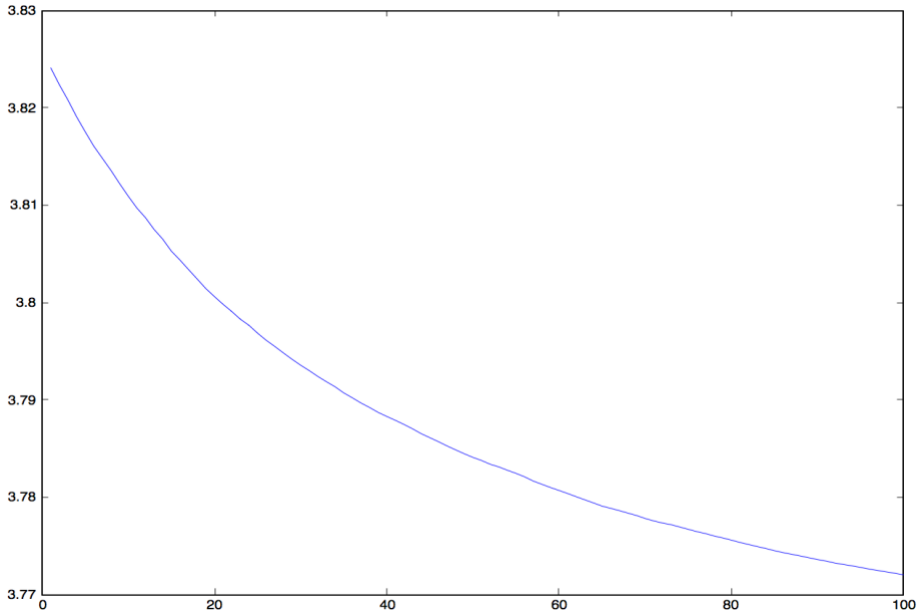
We can also notice that the first graph (step-size = 0.01) of the function is smooth, but the second graph (step-size = 0.03) has some fluctuations. This is because the stochastic gradient descent has a disadvantage, sometimes each update may not converge tending to the right direction, so it can cause some fluctuations.

We can also find the solution of the second graph is better than the first graph, the first graph is better than the third graph. We can notice that when we make the step-size larger, we obtain a better result, but we can not say this situation always happens, because when the step-size is too large, the function converges too quick, so it may

miss the best solution. So it is important to find a good value of step-size.

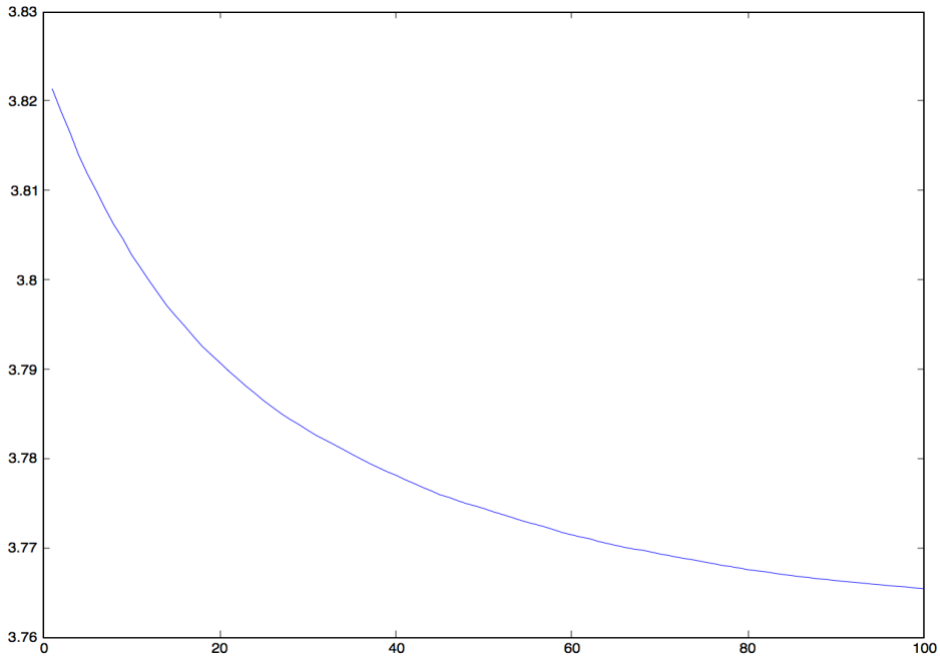
Then we change the batch-size, we set batch size = 1500, and we did the second test, we set the batch size = 2500. We have :

When batch size = 1500 :



We can notice that the value of the objective function until the 100th iteration is about 3.7703.

When the batch-size = 2500

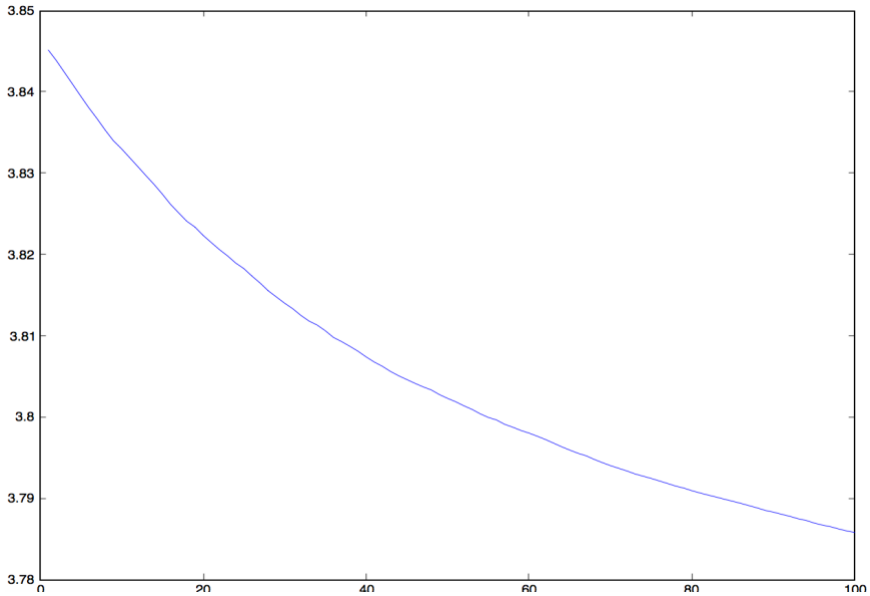


We can notice that the value of the objective function is about 3.7656.

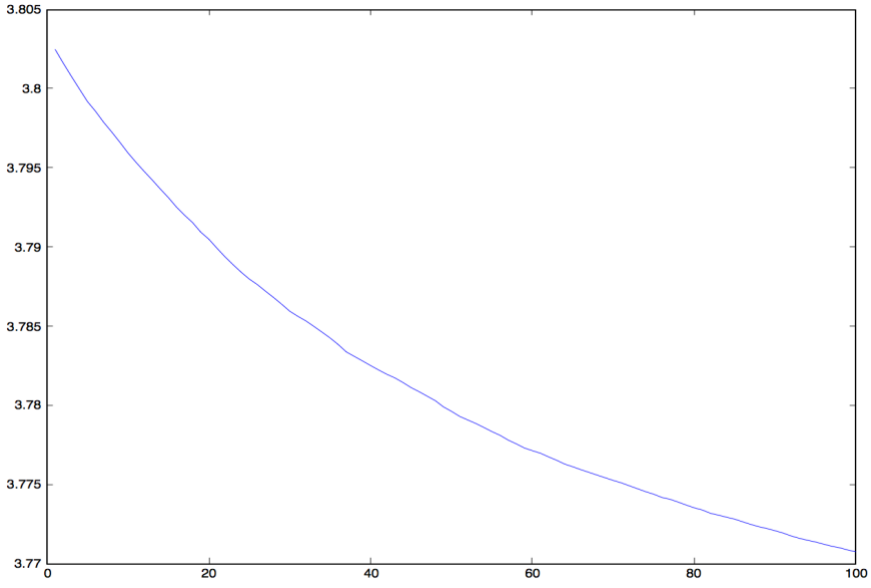
Comparing these 2 graphs to the original graph, we can find the result of the third graph is better than the second graph, and the second graph is better than the original graph. The larger the size of the batch is, the better result can we have.
The mini-batch gradient descent takes some samples from the data to calculate the gradient, so when we augment the number of the samples, it can use more data to calculate the gradient, so the gradient will be more correct.

Now we will change the value of rank of the factorization, we set k = 13, and k = 7.
When k = 13, we have :

3.786

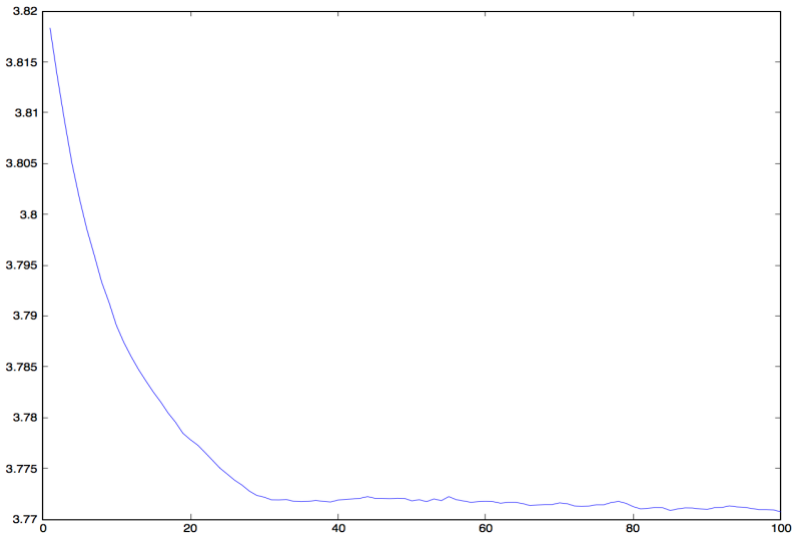


When k = 7, we have :



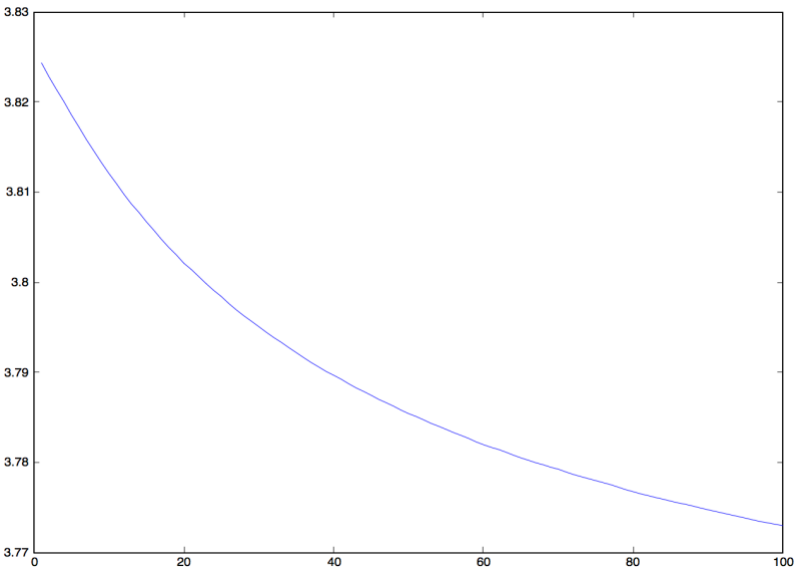
When k = 7, the value of the objective function is 3.7705. When k = 13, the value of the objective function is about 3.7860.
We can notice that when k = 7, the result is better than the original graph, when k =13, the result is worse than the original graph.

how do the step-size and the batch-size interact :
when step-size = 0.03, the batch-size = 2500, we have :



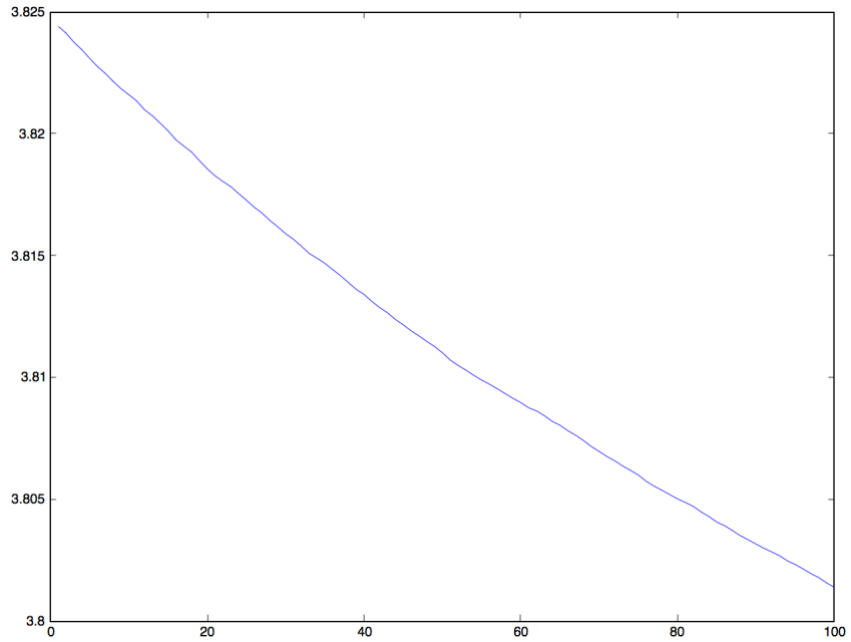
In this case, we increase the step-size, we decrease the batch-size, we can notice the result is better than the original graph. The result is about 3.7705.

When step-size = 0.005, the batch-size = 2500, we have :



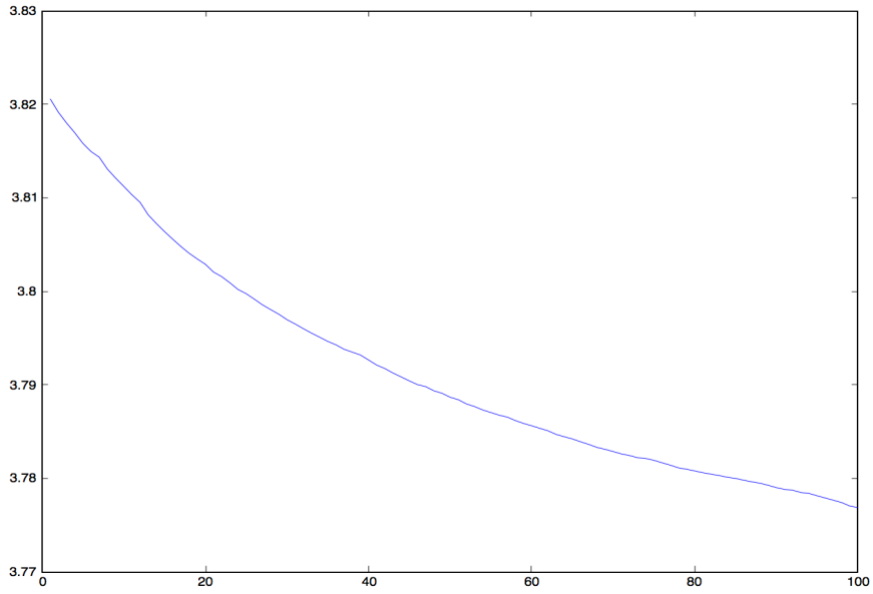
The result is about 3 .772. It is better than the original result.

When step-size = 0.005, the batch-size = 500, we have :



The result is about 3.802. It is worse than the original result.

When step-size = 0.03, the batch-size = 500, we have :



The result is about 3.777. It is better than the original result.

According to these 4 graphs, graphs which are concerned to the parameters step-size and the batch size and the original graph.
When we make step-size and batch-size larger, we can notice that the result is better

than the original graph. It is the best result. When we just make the step-size larger, we have the result better than the original result, but it is worse than the case that we make these 2 parameters larger. When we make the step-size and the batch-size smaller, we can notice that this case has the worst result. When we make the step-size larger and the batch-size smaller, it is better than the original graph, but it is not better than the case when we make the 2 parameters bigger. So these 2 paramters is positive correlation.

4 – After estimating W and H, use them to recommend a movie for a given user :

```
user_xhat = (1-M(:,user_index)).*(W*H(:,user_index));
[xx,movie_index] = max(user_xhat);
```

W is the matrix of the movies, H is the matrix of users, every column of the matrix H represents the degree of liking films of users, so we take the column concerned about the user that we want to recommend the film. We multiply the matrix W with the column that we take from the matrix H.