Factorization-Based Data Modeling
Practical Work 1

CHEN Hang, FAN Zheng

November 2018

1 Matrix Factorization with Alternating Least Squares and Gradient Descent

1.1 Implement the ALS update rules

In this section we need to apply the formula of Alternating Least Squares algorithm:

$$\mathbf{W} = \mathbf{X}\mathbf{H}^\top(\mathbf{H}\mathbf{H}^\top)^{-1}$$

$$\mathbf{H} = (\mathbf{W}^\top\mathbf{W})^{-1}\mathbf{W}^\top\mathbf{X}$$

According to formula, we have code :

```
Wals = X.dot(Hals.transpose())
       .dot(np.linalg.inv(Hals.dot(Hals.transpose())))

Hals = np.linalg.inv(Wals.transpose().dot(Wals))
       .dot(Wals.transpose()).dot(X)
```

Wals and Hals present 2 unknown factor matrices.

We need to define their initial values and to use the formulas above to update their value. In each iteration, the current values of Wals and Hals are calculated by their previous values, so they will be updated continuously.

Xhat,the value of prediction, is the product of the 2 matrices.

1.2 Compute the objective function values for each iteration

We take a matrix 'obj_als' with 20 rows and 1 column to record each objective function value in each iteration.
According to the Frobenuis norm, we get the value :

$$\frac{1}{2} \parallel \mathbf{X} - \mathbf{W}\mathbf{H} \parallel_F^2$$

So we have the code :

```
obj_als[i] = 0.5 * (np.linalg.norm(X - Xhat))**2
```

After 20 iterations with updating the value of Wals and Hals, we can get 20 values of the objective function. The graph below shows the result :

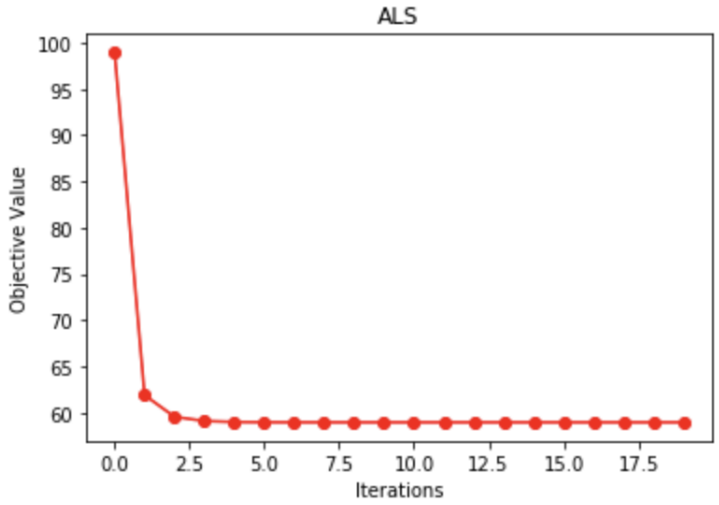

Figure 1: ALS

From the graph we can know that the objective function converges.

1.3 Implement the GD update rules

In this section we need to apply the formula of Gradient Descent algorithm :

$$\mathbf{W} \leftarrow \mathbf{W} + \eta(\mathbf{X} - \mathbf{WH})\mathbf{H}^\top$$

$$\mathbf{H} \leftarrow \mathbf{H} + \eta\mathbf{W}^\top(\mathbf{X} - \mathbf{WH})$$

According to the formulas, we can have the code :

```
Wgd = Wgd + eta * (X - Wgd.dot(Hgd)).dot(Hgd.transpose())
Hgd = Hgd + eta * Wgd.transpose().dot(X - Wgd.dot(Hgd))
```

η is a fixed step-size, and Wgd and Hgd present 2 unknown factor matrices. Wgd and Hgd can be updated in each iteration.

1.4 What is the effect of η

η is a fixed step-size, it also means the learning rate which controls the speed of gradient descent.

There are the results when $\eta = 0.005$, 0.01 and 0.02 :

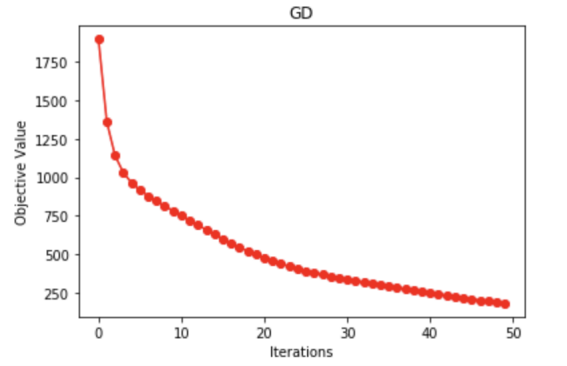

Figure 2: $\eta = 0.005$

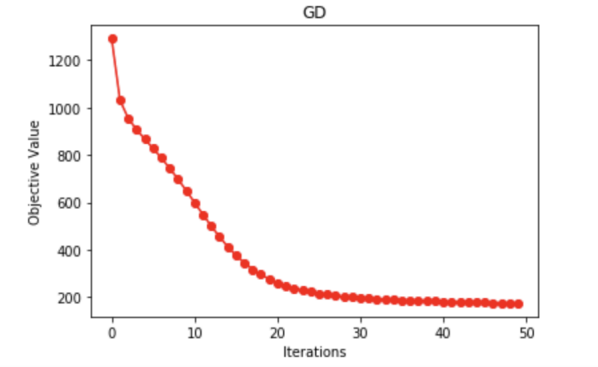

Figure 3: $\eta = 0.01$

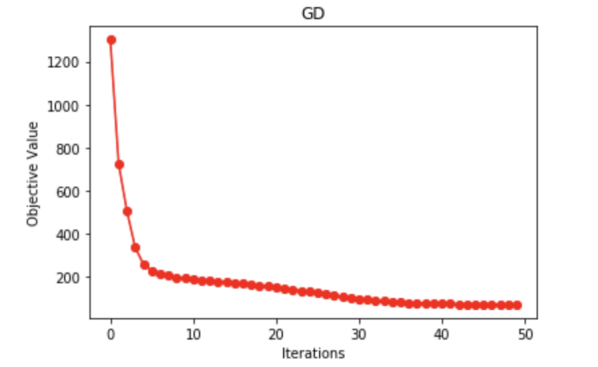

Figure 4: $\eta = 0.02$

From the result of the graphs, we notice that as the learning rate grows, the objective function converges more quickly. What's more, when η is smaller, the curve converges at a bigger value. For example, when $\eta = 0.005$, the function converges about at 250, when η grows to 0.01, the function converges about at 200, and when η is 0.02, we get a better result.
But when $\eta = 0.03$:

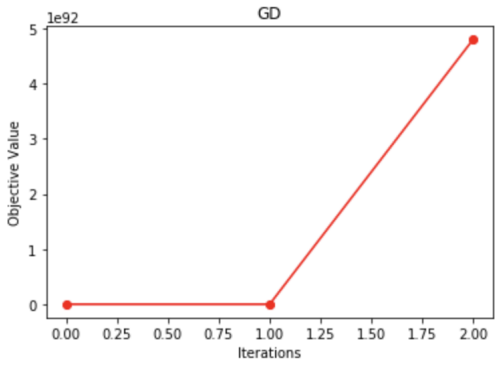

Figure 5: $\eta = 0.03$

We can see that the learning rate is too large and this function can not converge. In the above figure 5, we can only see the two points, iterations=1 and iterations=2. The reason is that when iterations=3 or bigger, the objective function values are inf or nan, so we can't show it in the graph.

So η can be neither too small to make sure the descent speed and the converge value nor too big to avoid not converge.

1.5 Compute the objective function values for each iteration

According to the Frobenuis norm, we get the value :

$$\frac{1}{2} \parallel \mathbf{X} - \mathbf{WH} \parallel_F^2$$

So we have the code :

```
obj_gd[i] = 0.5 * (np.linalg.norm(X - Xhat))**2
```

After 50 iterations with updating the value of Wals and Hals, we can get 50 values of the objective function. The figures are shown in the question 1.4.

1.6 Play with the variable 'dataNoise'

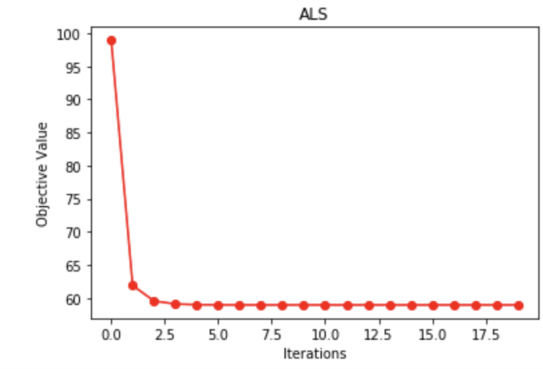

Figure 6: ALS dataNoise=1

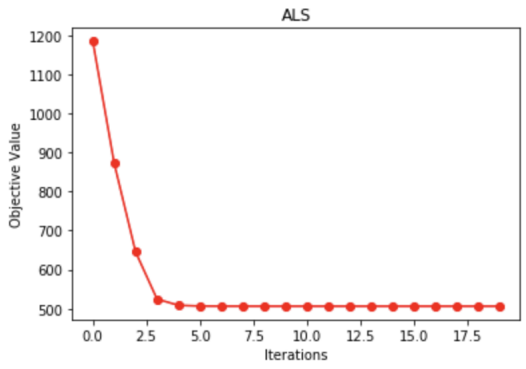

Figure 7: ALS dataNoise=3

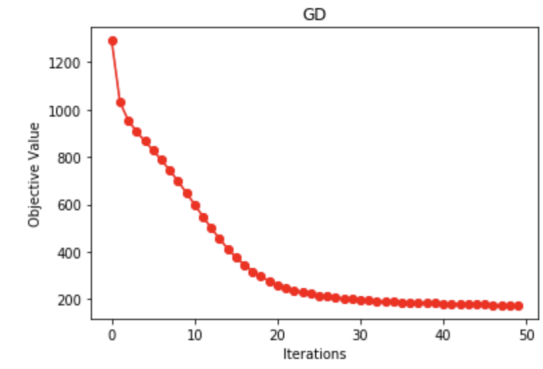

Figure 8: GD dataNoise=1

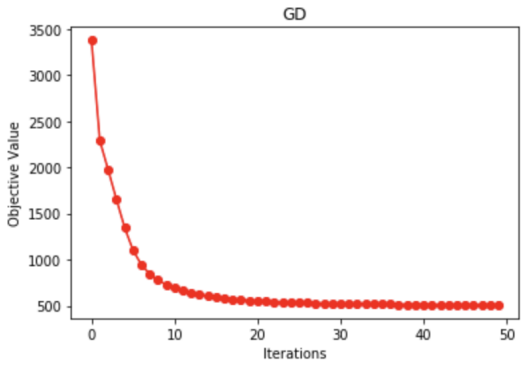

Figure 9: GD dataNoise=3

For ALS, comparing these 2 graphs, we can notice that when the value of dataNoise is larger, the value of convergence that we obtain is larger. When dataNoise = 1, the curve of the objective function converges at approximately 60. When dataNoise = 3, the curve of the objective function converges at approximately 500. What's more, the curve with dataNoise = 1 converges more quickly than the curve with dataNoise = 3. According to the aim of the objective function, we can notice that the result when dataNoise = 1 is much better than the result when dataNoise = 3.

For GD, it is the same. When dataNoise = 1, the curve is stable when the value of the objective function is about 200, when dataNoise = 3, the curve is stable when the value of the objective function is about 500.

So we can know that the dataNoise can influence the peformance of the algorithm, when the value of dataNoise is larger, the result will be worse.

1.7 Play with the size of data (I,J) and the rank of factorization K

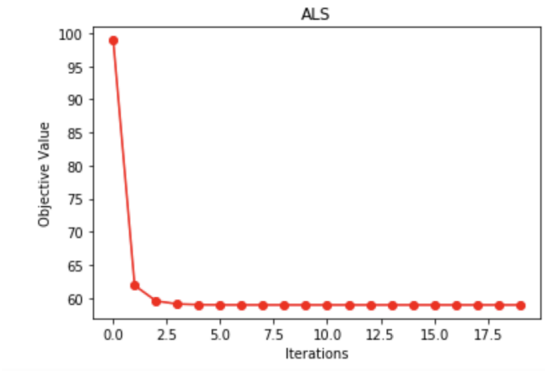

Figure 10: ALS I=10,J=20,K=3

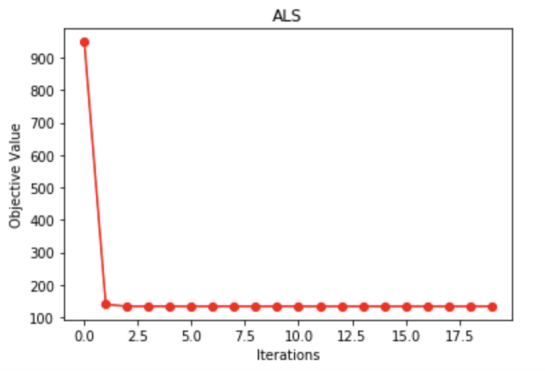

Figure 11: ALS I=15,J=25,K=3

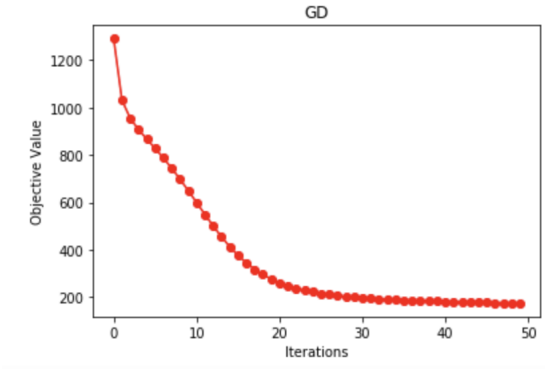

Figure 12: GD I=10,J=20,K=3

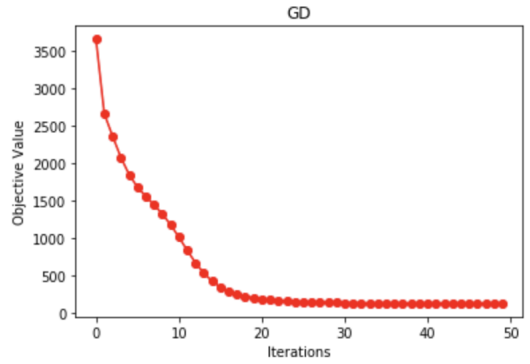

Figure 13: GD I=15,J=25,K=3

When playing with the size of data(I,J), for ALS, we can notice that the objective function with I=10,J=20 begins with a lower value and it converges at 60, which is smaller than that in I=15,J=25.

For GD, ibid.

So according to the graphs, we can know that when the size of data is smaller, we can get a better result. Because this process is to separate a large matrix to 2 small matrix with lower dimension, in other word, this process is to do dimensionality reduction. So when the size of the data is larger, it will be more difficult to do the dimensionality reduction.

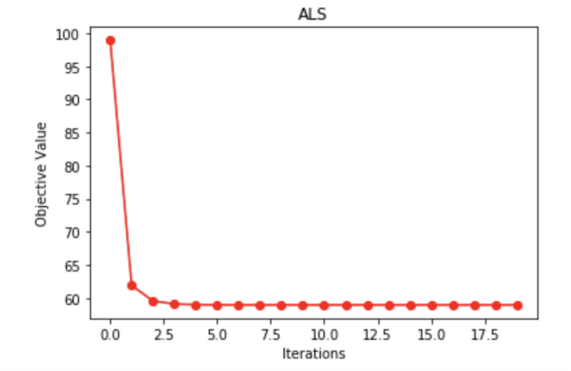

Figure 14: ALS I=10,J=20,K=3

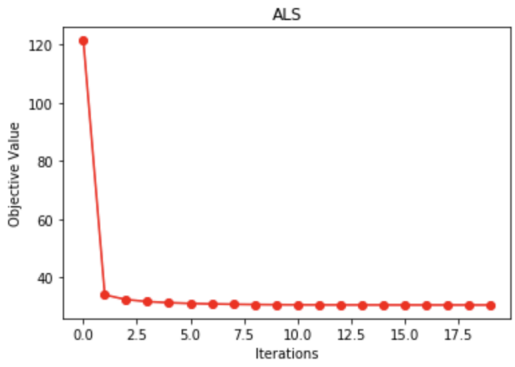

Figure 15: ALS I=10,J=20,K=6

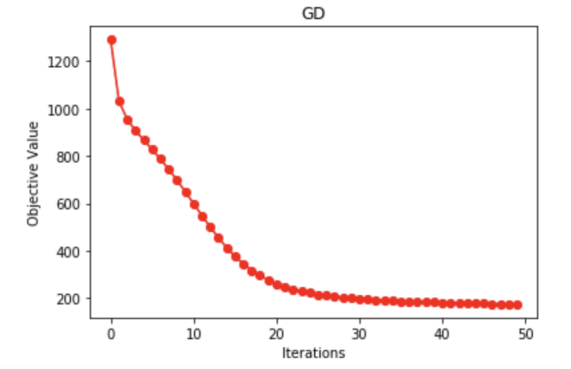

Figure 16: GD I=10,J=20,K=3

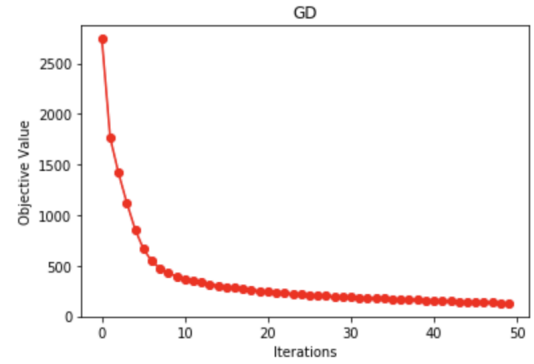

Figure 17: GD I=10,J=20,K=6

When playing with the rank of factorization K, for ALS, we can notice that the objective function with K=6 has a better descent speed and its converge value is smaller than K=3. Because when K is bigger, the dimension of the matrix W and H is bigger, there are more features so it has a better result. The result of GD is the same.

So we could say that when K is bigger, we will have a better result. But when K is too big, it maybe cause overfitting.

1.8 Which algorithm do you think is better

In conclusion, for this question, we think the algorithm ALS is better than the algorithm GD. First of all, we found that the value of convergence of algorithm ALS is samller than algorithm GD, which means that Xhat in ALS is closer to X than Xhat in GD.

Then, in ALS, we don't need to worry about the η. In GD, if η is too small, the rate of convergence is very small. If η is too big, we cannot assure whether the objective function is converge or not. So, we need to debug this parameter to get a satisfactory result.

In the nutshell, the algorithm ALS is better for this question than the algorithm GD.

2 Non-Negative Matrix Factorization with Multiplicative Update Rules

2.1 Implement the MUR update rules

According to the multiplicative update rules, we have the code :

```
Wmur = np.multiply(Wmur,np.divide(np.divide(X,Xhat)
       .dot(Hmur.transpose()),O.dot(Hmur.transpose())))
Hmur = np.multiply(Hmur,np.divide(Wmur.transpose()
       .dot(np.divide(X,Xhat)),Wmur.transpose().dot(O)))
```

Wmur and Hmur present 2 unknown non-negative factor matrices. O is a I*J matrix whose elements are 1.

2.2 Compute the objective function values for each iteration

According to the formula, we have the code :

```
tmp = np.multiply(X,(np.log(np.divide(X,Xhat))))-X+Xhat
obj_mur[i] = sum(sum(tmp))
```

After 100 iterations, we get the result :

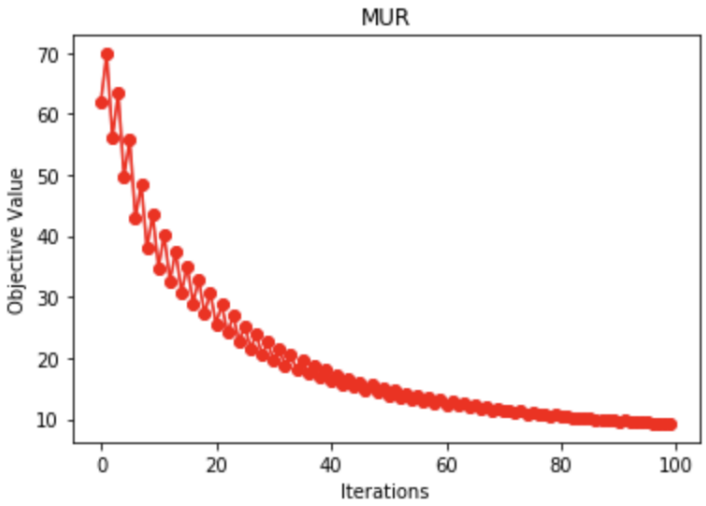

Figure 18: MUR

2.3 Play with the variable 'dataNoise'

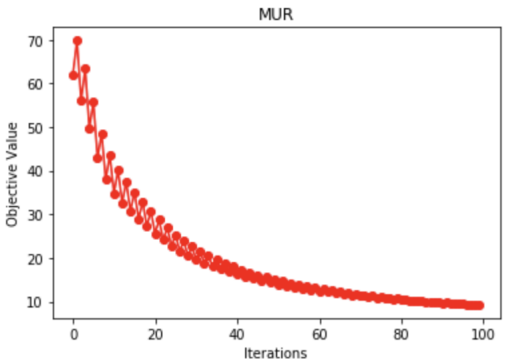

Figure 19: MUR dataNoise=1

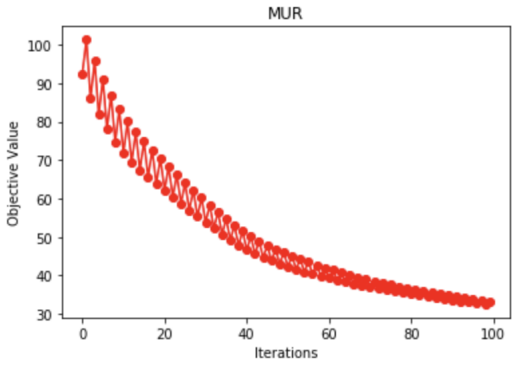

Figure 20: MUR dataNoise=3

We can notice that when dataNoise = 1, the curve will converge at approximately 10, and when dataNoise = 3, the curve will be stable when the objective value equals to about 30. So we can say when the dataNoise is larger, the result is more affected by the dataNoise. When the dataNoise is smaller, the result is better.

2.4 Play with the size of data (I,J) and the rank of factorization K

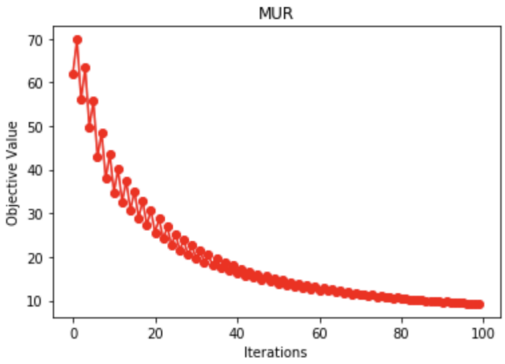

Figure 21: MUR I=10,J=20,K=3

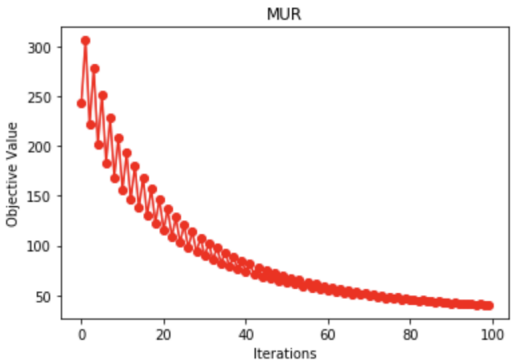

Figure 22: MUR I=20,J=30,K=3

When playing with the size of data(I,J), from the figures above, we can know that when (I,J) is smaller, the objective function begins at a smaller value and it will converges at a smaller value. In the left figure, when I=10 and J=20, the objective function value begins at about 70 and converges at about 10. In the right figure,

when I=20 and J=30, the objective function value begins at about 240 and converges at about 40.

In conclusion, we can get a better result with a smaller I and J.

As we know, this process is to do dimensionality reduction. If the size of the data is large, it will be difficult to do the dimensionality reduction.

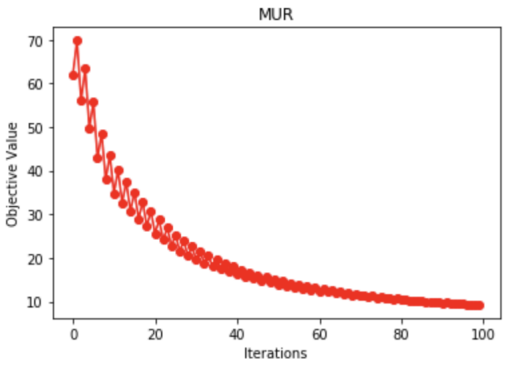

Figure 23: MUR I=10,J=20,K=3

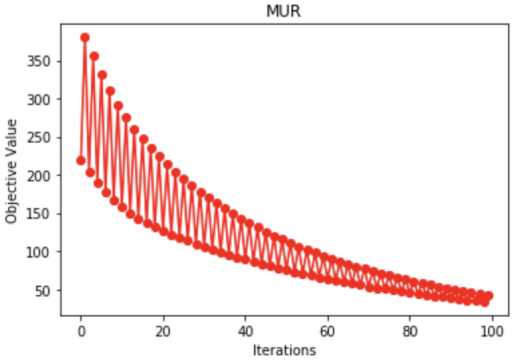

Figure 24: MUR I=10,J=20,K=8

When playing with the rank of factorization K, we can know that when K=3, after 100 iterations, the curve of objective function converges at about 10, but when K=8, after 100 iterations, we can't know its value of convergence cause the curve has not yet finished converging.

When we set iterations=200, we found that the curve with K=8 will converge at about 0, and convergence value of curve with K=3 doesn't change.

In conclusion, when k is bigger, the speed of converge is slower with a smaller convergence value.