Factorization-Based Data Modeling
Practical Work 1

HEI Xiaoxuan, HE Yuru

1 Matrix Factorization with Alternating Least Squares and Gradient Descent

1.1 Implement the ALS update rules

```
Wals = X*Hals'/(Hals*Hals');

Hals = (Wals'*Wals)\Wals'*X;
```

1.2 Compute the objective function values for each iteration.

```
Xhat = Wals * Hals;
obj_als(i) = 1/2 * sum(sum((X - Xhat).^2));
```

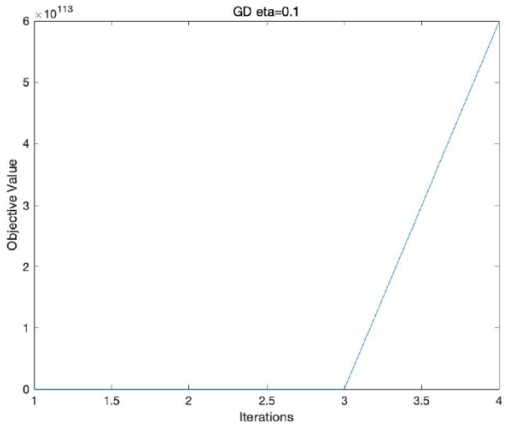
1.3 Implement the GD update rules.

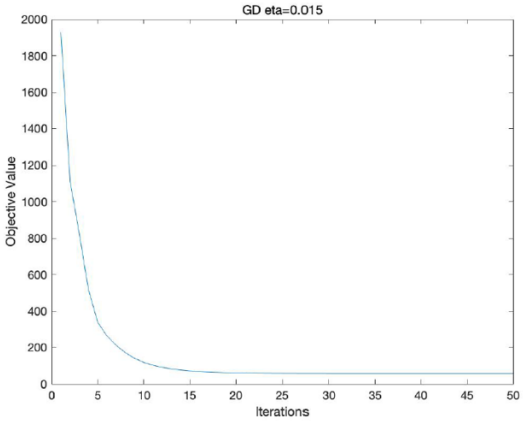
```
%compute the gradient
%todo
W = (X - Wgd*Hgd) * Hgd';
H = Wgd' * (X - Wgd*Hgd);

%take the small step
%todo
Wgd = Wgd + eta * W;
Hgd = Hgd + eta * H;
```

1.4 What is the effect of η?

η is the learning rate. If η is too small, we need more iterations to find the optimal result. The large learning rates result in unstable training, even can't get the result. In this lab, we kept other parameters and only change η, we found that, when η = 0.001, we can't get a good result within 50 iterations. When η = 0.1, there's no result. Compared with the original case η = 0.01, η = 0.015 performs better.



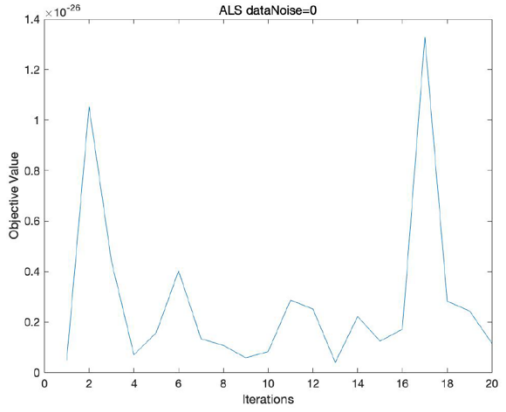


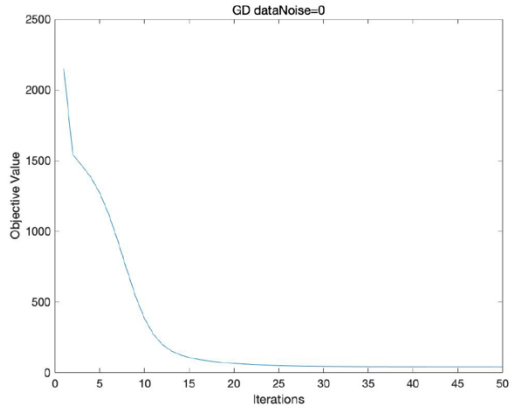
1.5 Compute the objective function values for each iteration.

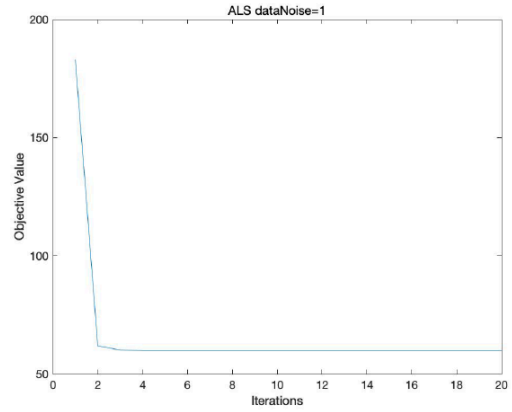
```
Xhat = Wgd * Hgd;
obj_gd(i) = 1/2 * sum(sum((X - Xhat).^2));
```

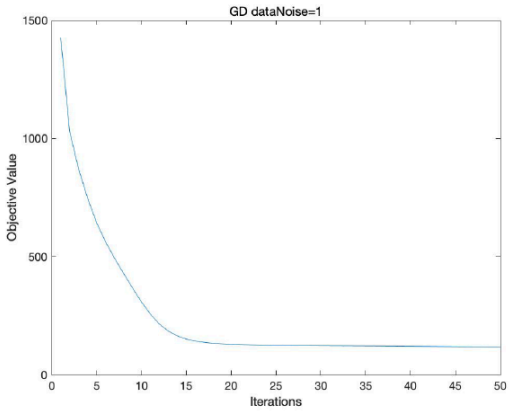
1.6 Play with the variable 'dataNoise'. What is the effect of this parameter on the algorithms.

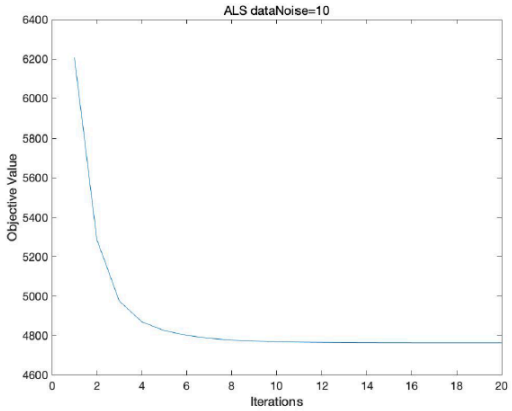
In this lab, "dataNoise" has a huge impact on the data. If we change the noise, the results will be changed completely. When we change the dataNoise to 0, ALS can't get the result. But GD can still iterate correctly.

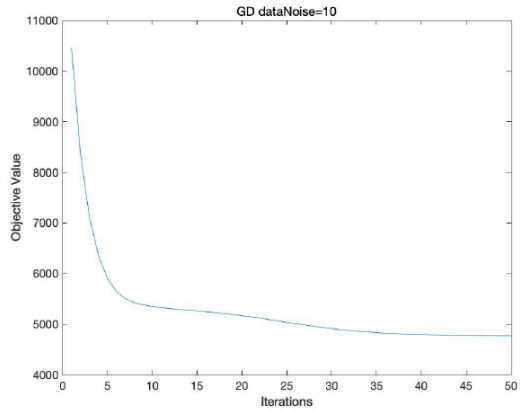








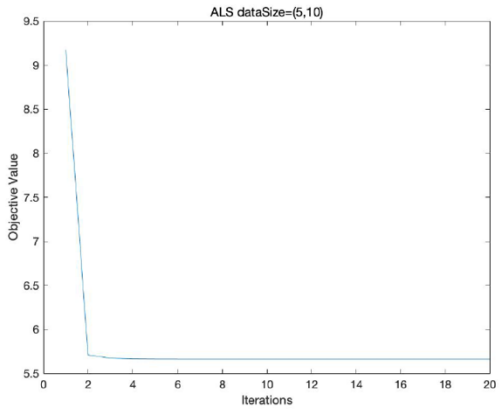


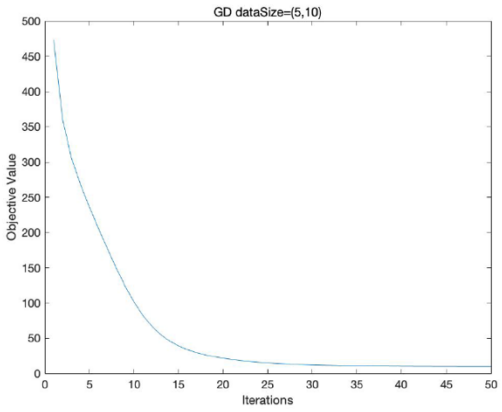


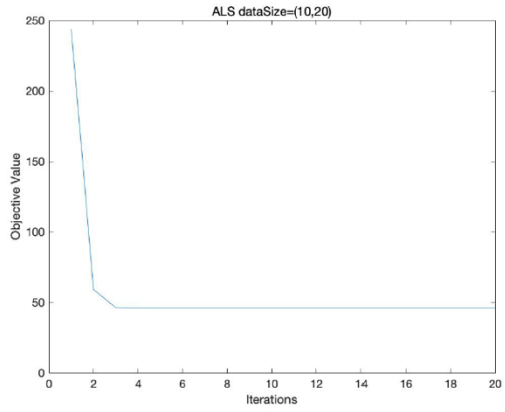
1.7 Play with the size of the data (I, J) and the rank of the factorization K. What behavior do you observe?

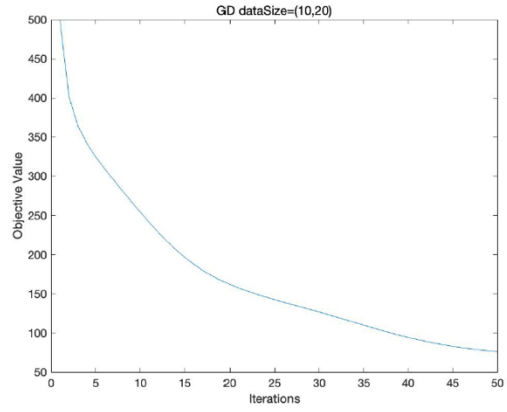
When we augment the size of the data (I, J), the algorithm GD can find the optimal value within less iterations. Otherwise, it needs more iterations. As for ALS, the change of data size doesn't make sense.

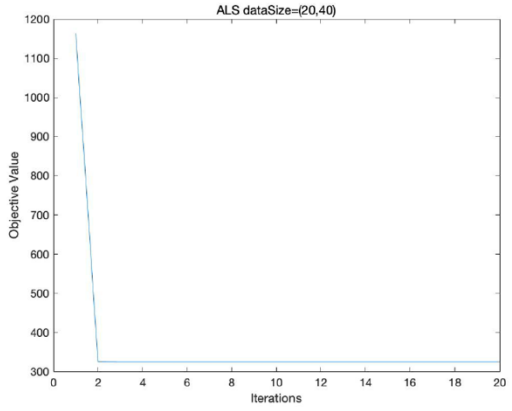
When K becomes bigger, the optimal objective function values become better.

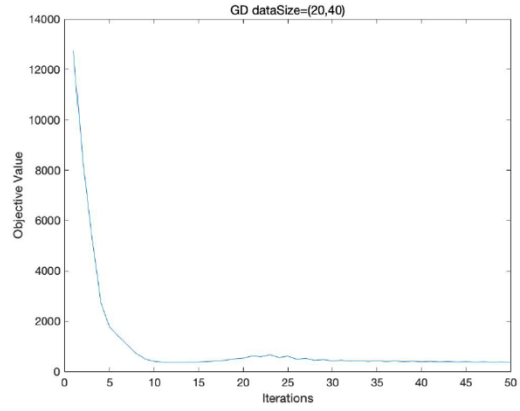

ALS dataSize=(5,10)
Objective Value
Iterations

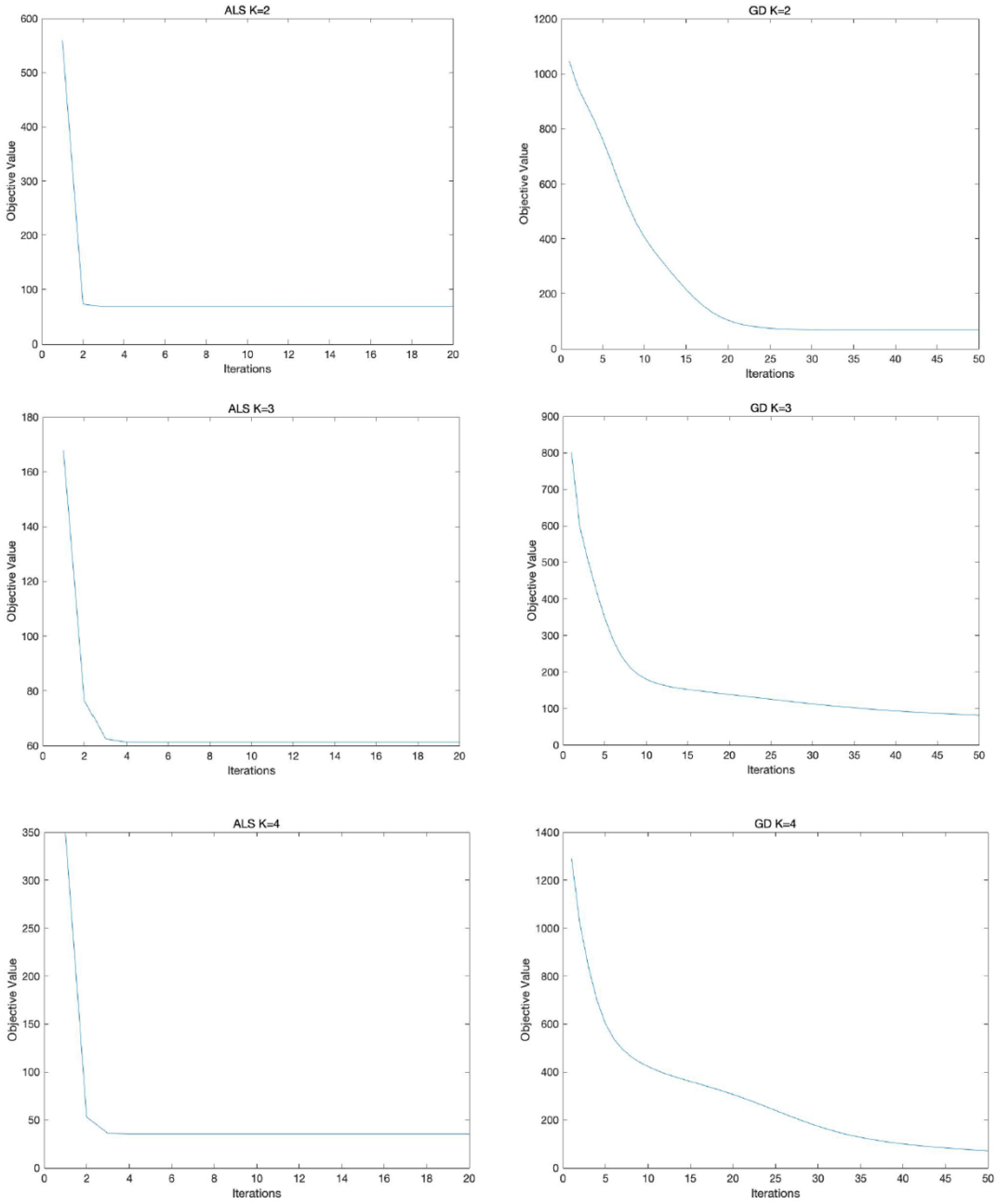

GD dataSize=(5,10)
Objective Value
Iterations


ALS dataSize=(10,20)
Objective Value
Iterations


GD dataSize=(10,20)
Objective Value
Iterations


ALS dataSize=(20,40)
Objective Value
Iterations


GD dataSize=(20,40)
Objective Value
Iterations



1.8 Which algorithm do you think is better? Why?

In this lab, we think ALS is better. Because we can get the result faster. As for GD, the learning rate need to be adjusted to a proper value or it needs more iterations. And ALS can find the global optimal value, but GD can only find the local optimal value.

2 Non-Negative Matrix Factorization with Multiplicative Update Rules

2.1 Implement the MUR update rules.

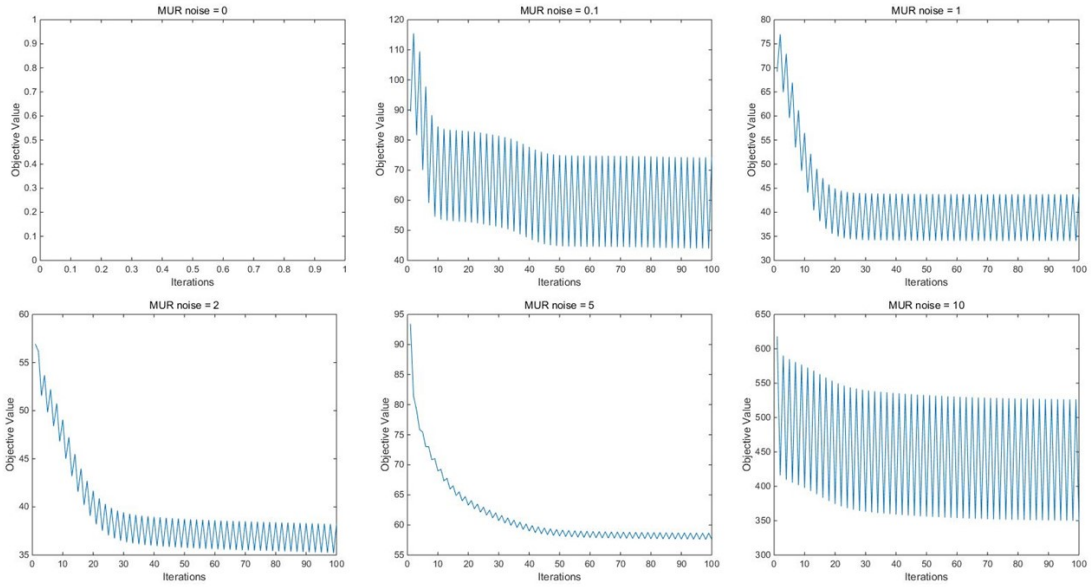
```
%Update W
Wmur = Wmur .* (((X./Xhat)* Hmur') ./ (O * Hmur'));

%Update H
Hmur = Hmur .* ((Wmur' * (X./Xhat)) ./ (Wmur' * O));
```

2.2 Compute the objective function values for each iteration.

```
Xhat = Wmur * Hmur;
Xhat = Xhat + eps; %for numerical stability
obj_mur(i) = sum(sum(X .* log(X./Xhat) - X + Xhat));
```

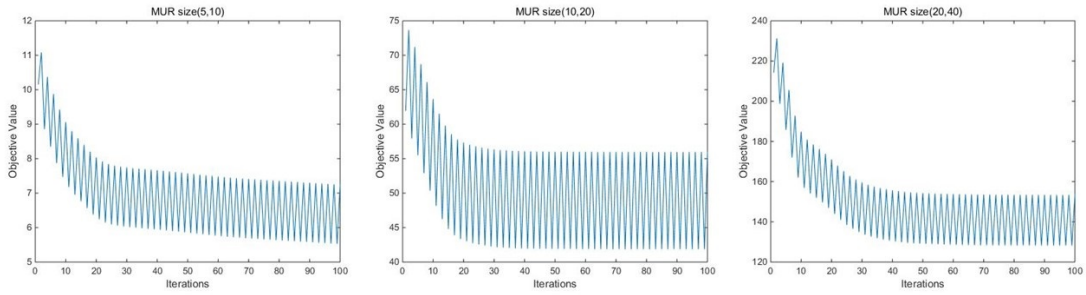
2.3 Play with the variable 'dataNoise'. What is the effect of this parameter on the algorithms.



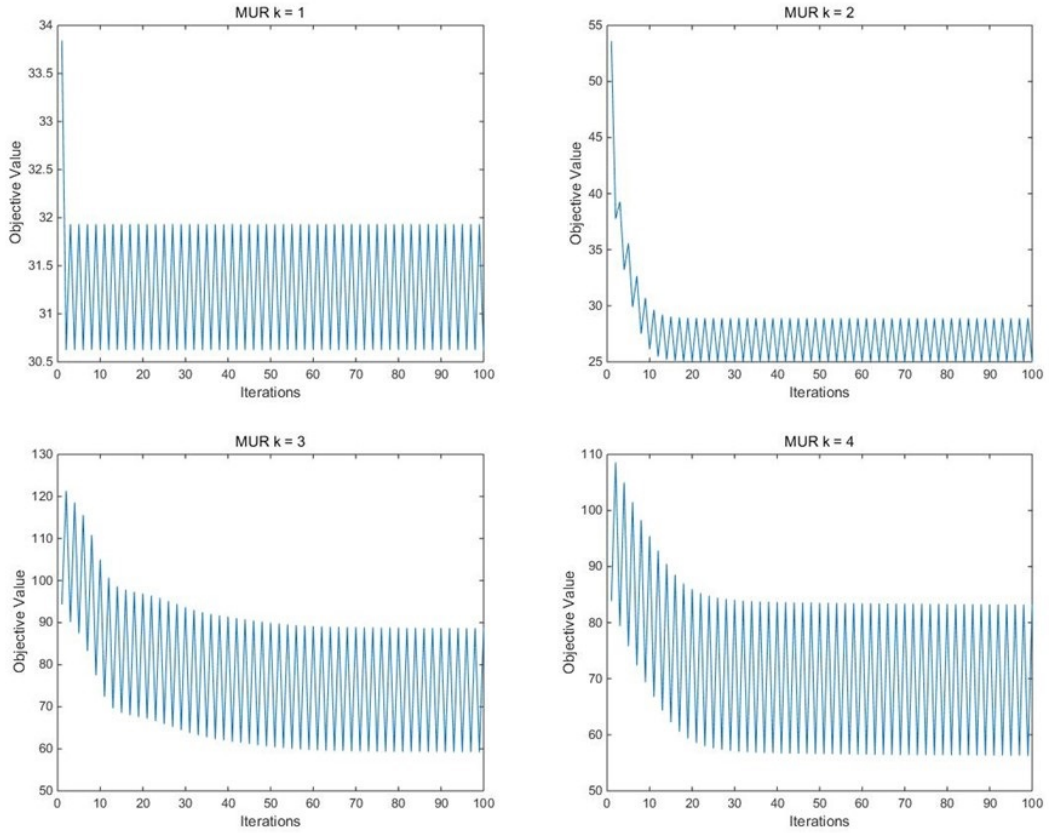
When noise = 0, the objective function value is also 0. When noise is very big, such as 10, the objective function value is also big. But we didn't find any trends when we augment or reduce noise.

2.4 Play with the size of the data (I, J) and the rank of the factorization K. What behavior do you observe?

The larger the data size is, the bigger the objective function value is, and the shock range is.

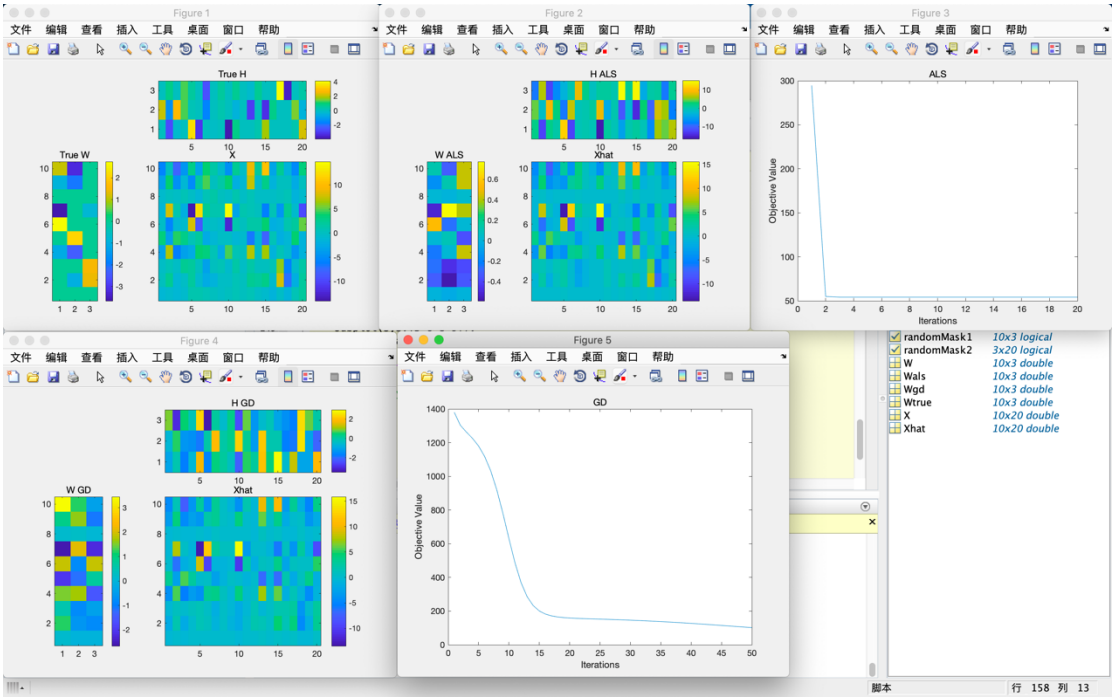


When we play with K, we observed that, K is related to the shock range of the objective function value. We speculated that, when K is smaller, the shock range is also smaller.

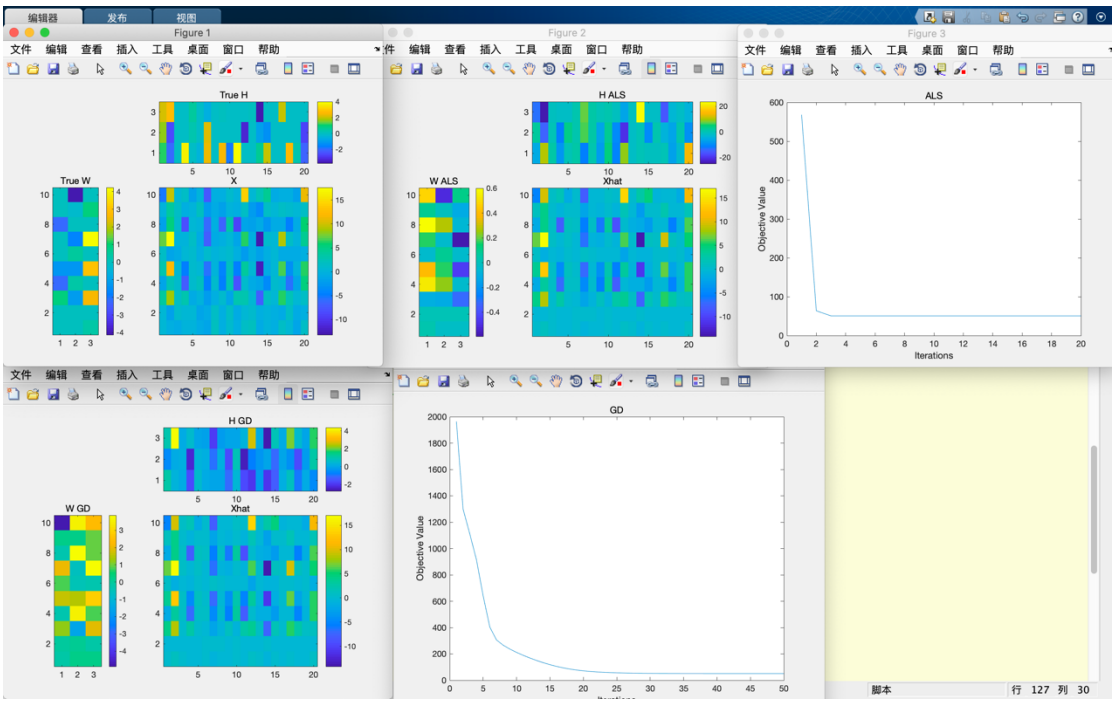


Appendix

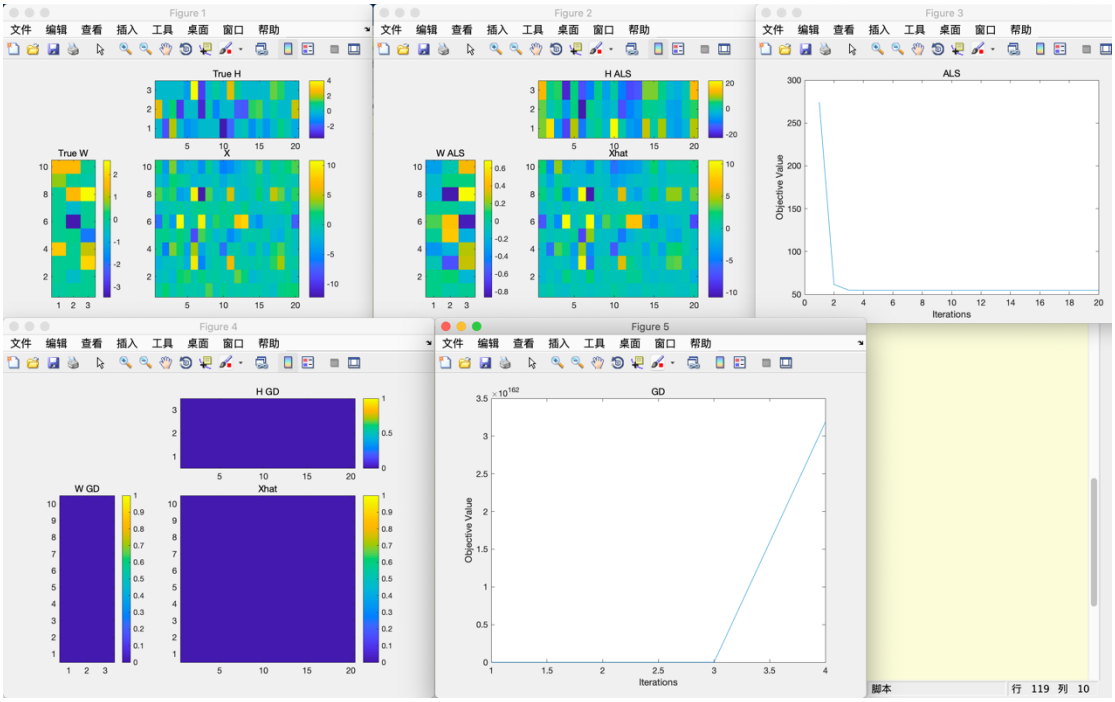
(I, J) = (10, 20), K = 3, eta = 0.01, dataNoise = 1



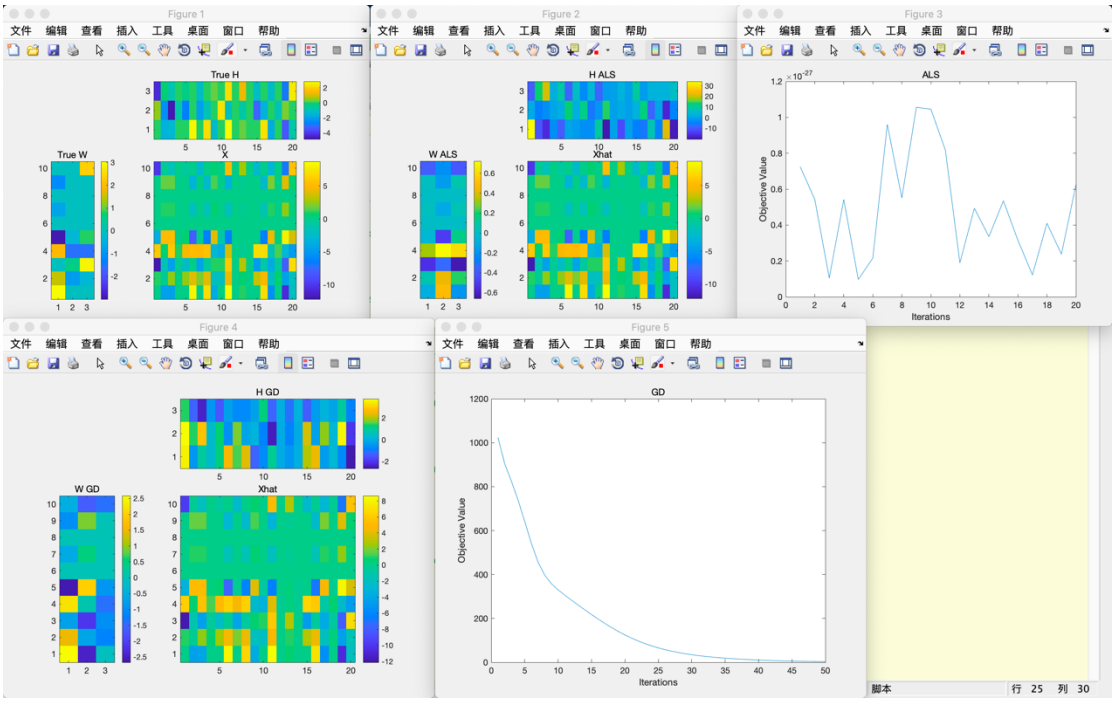
(I, J) = (10, 20), K = 3, eta = 0.15, dataNoise = 1



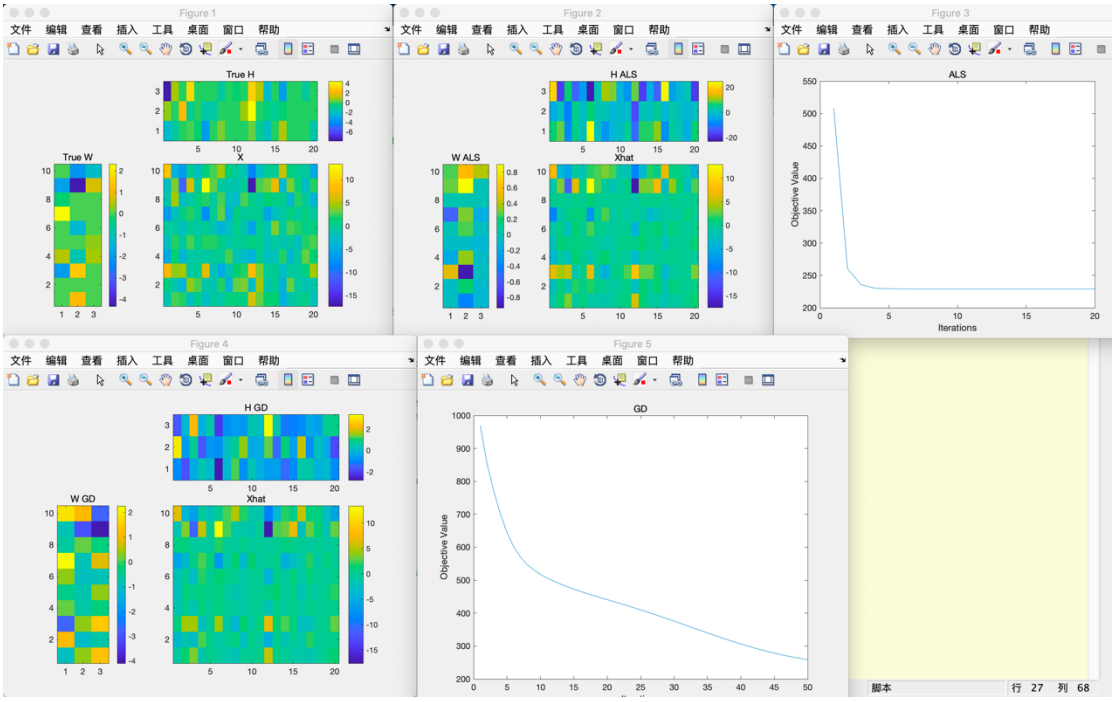
(I, J) = (10, 20), K = 3, eta = 0.1, dataNoise = 1



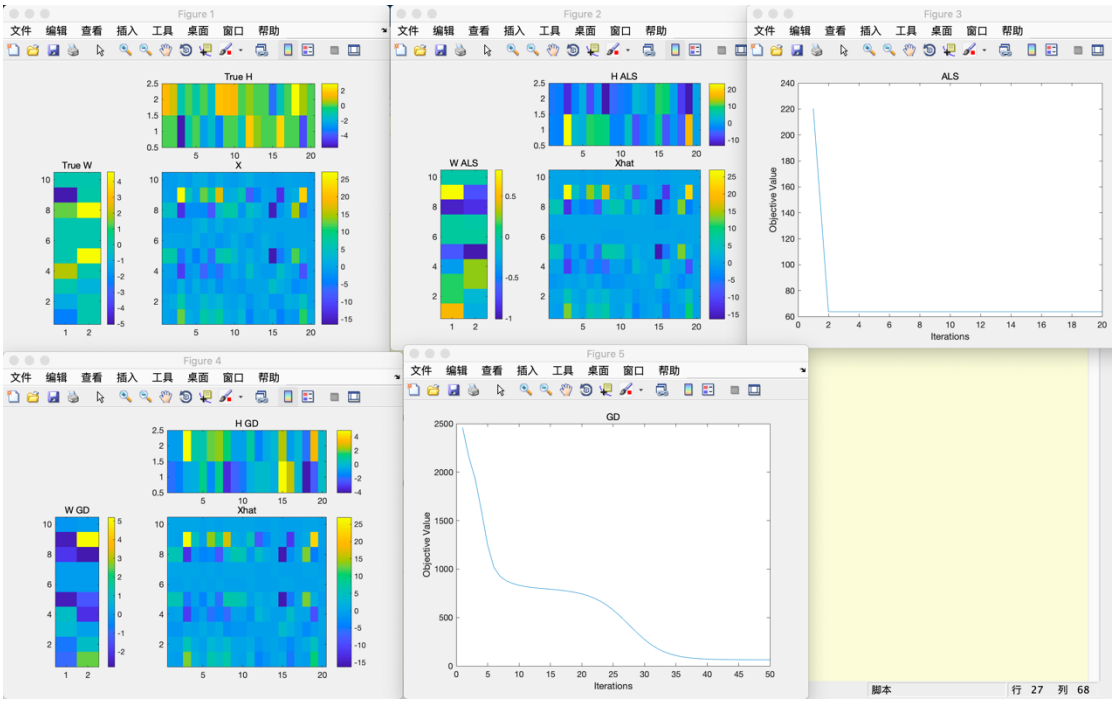
(I, J) = (10, 20), K = 3, eta = 0.01, dataNoise = 0



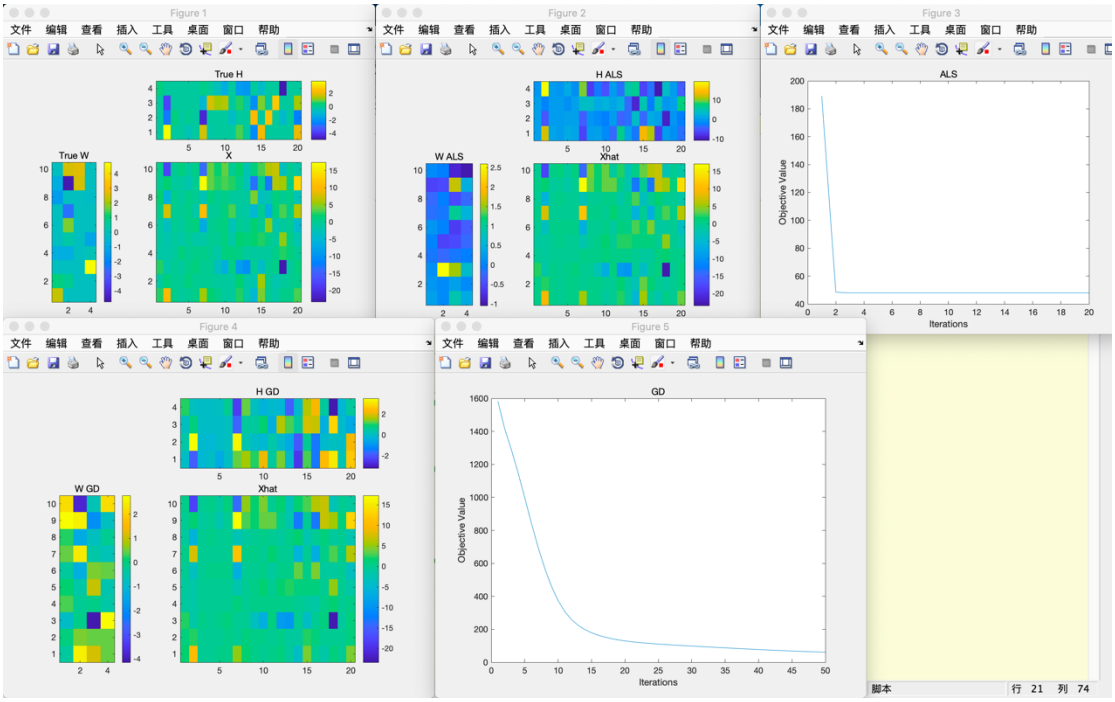
(I, J) = (10, 20), K = 3, eta = 0.01, dataNoise = 2



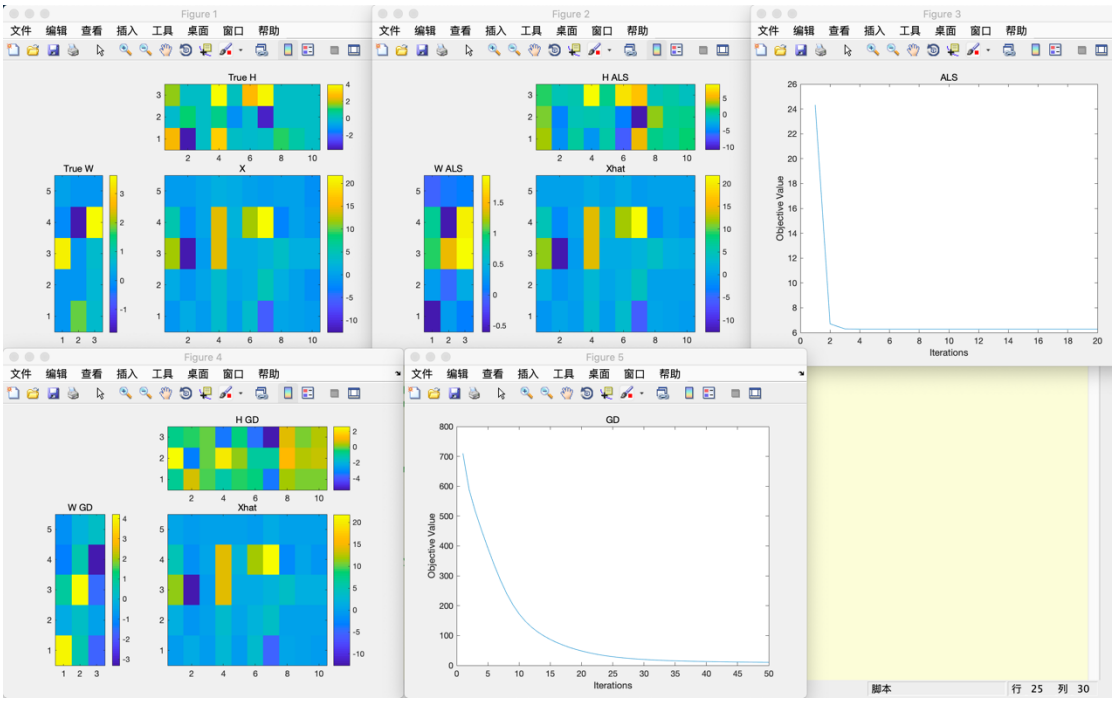
(I, J) = (10, 20), K = 2, eta = 0.01, dataNoise = 1



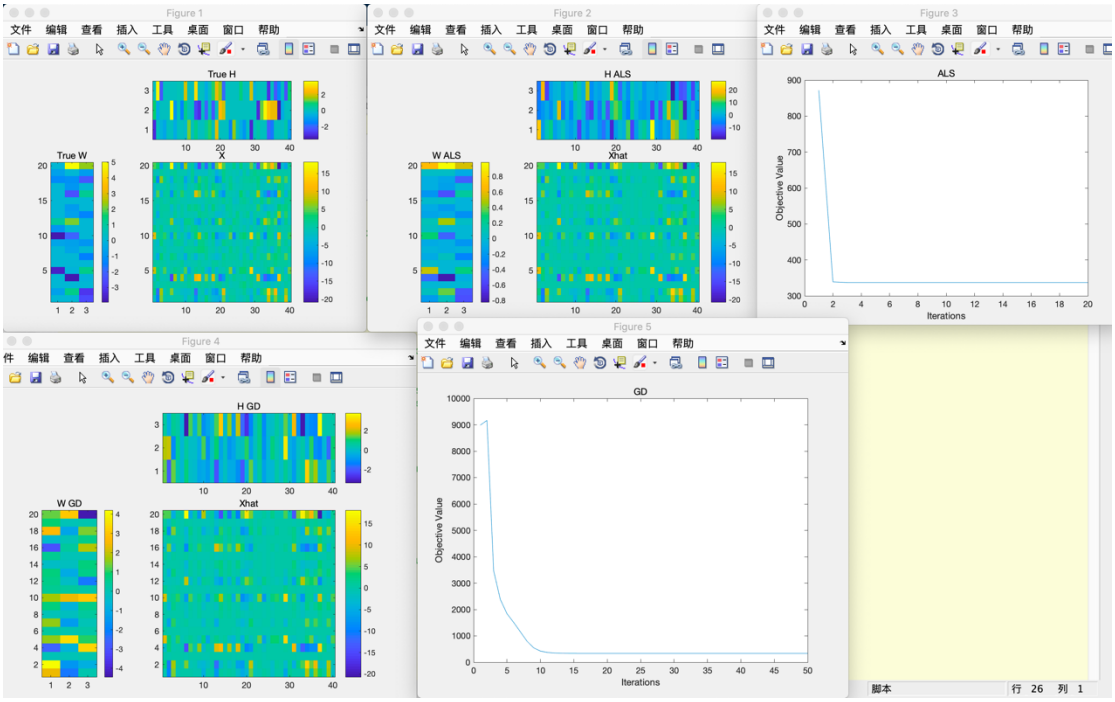
(I, J) = (10, 20), K = 4, eta = 0.01, dataNoise = 1



(I, J) = (5, 10), K = 3, eta = 0.01, dataNoise = 1

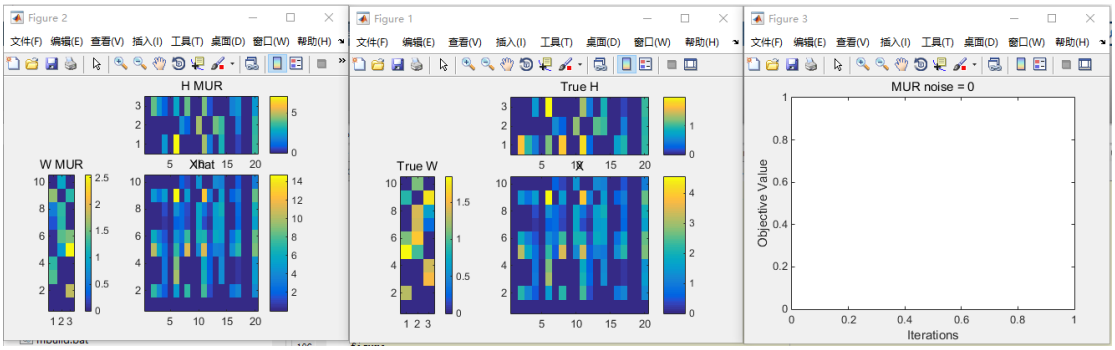


(I, J) = (20, 40), K = 3, eta = 0.01, dataNoise = 1

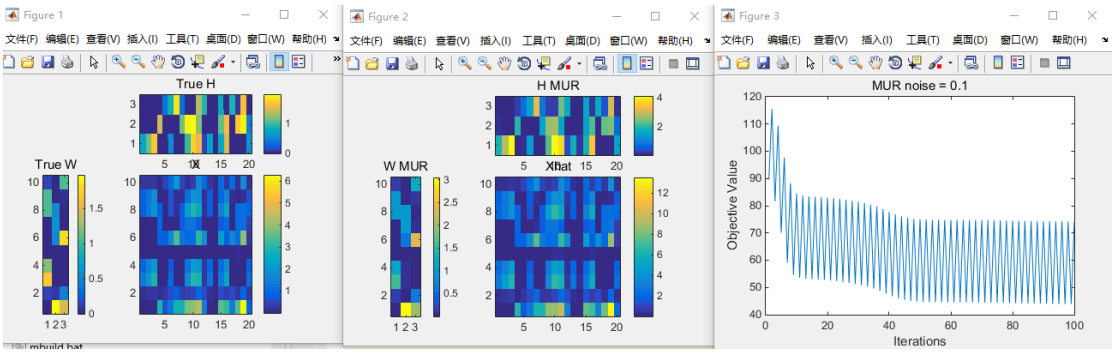


MUR

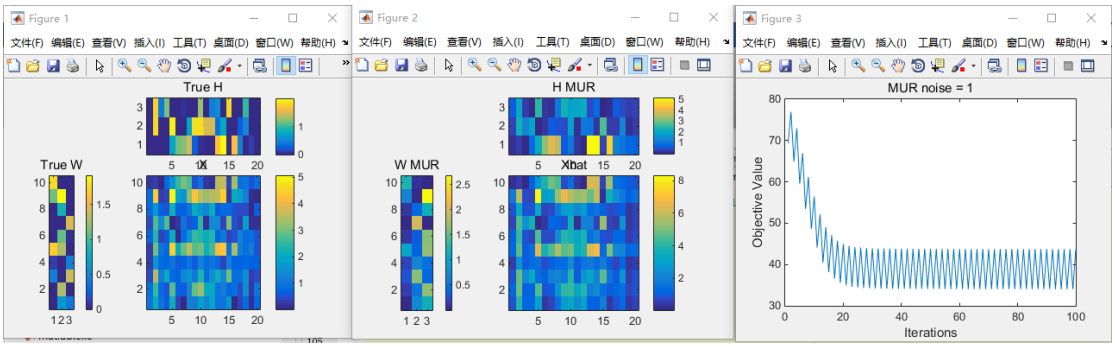
(I, J) = (10, 20), K = 3, dataNoise = 0



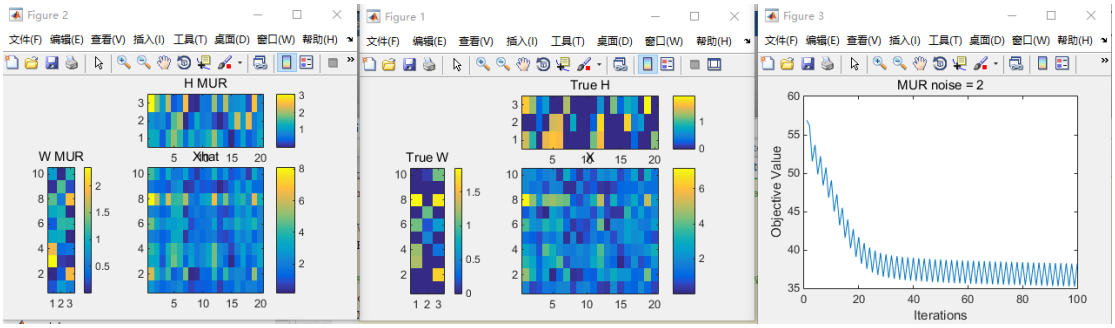
(I, J) = (10, 20), K = 3, dataNoise = 0.1



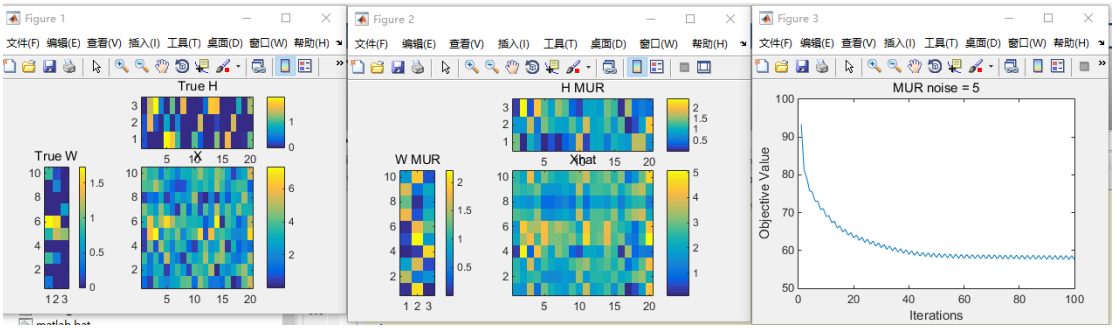
(I, J) = (10, 20), K = 3, dataNoise = 1



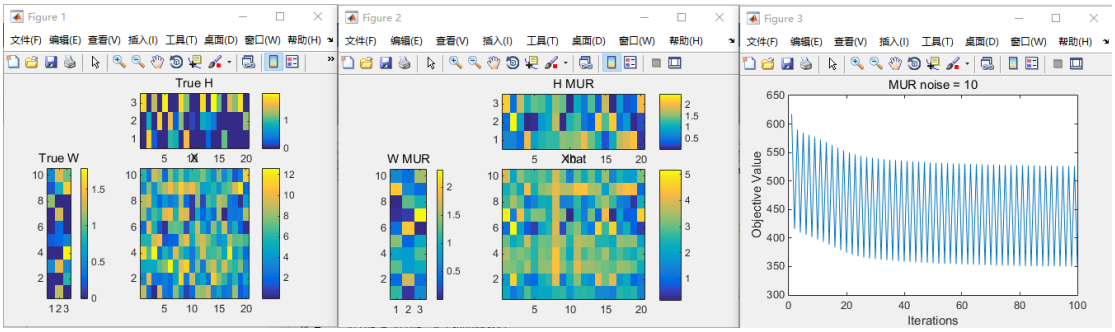
(I, J) = (10, 20), K = 3, dataNoise = 2



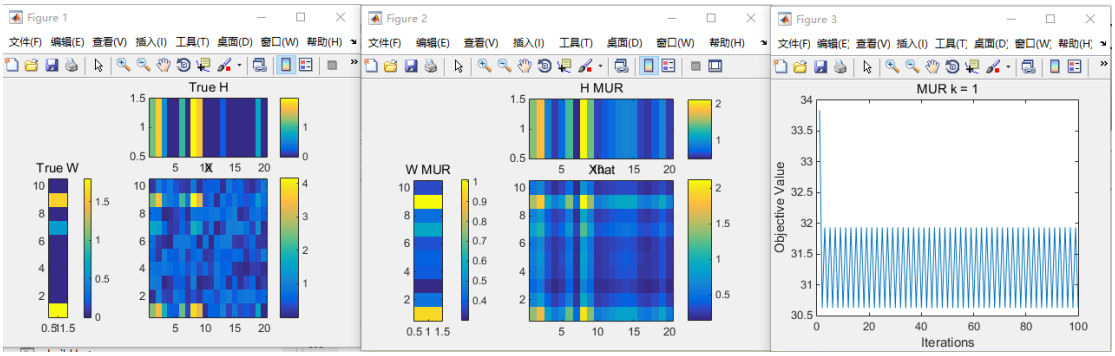
(I, J) = (10, 20), K = 3, dataNoise = 5



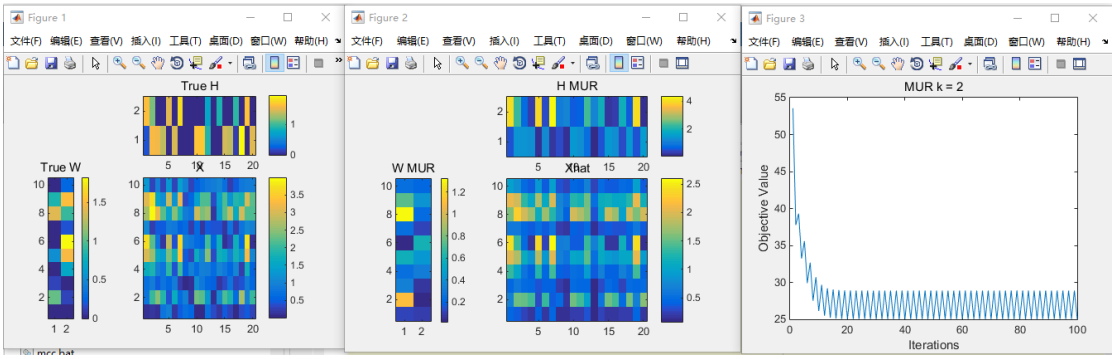
(I, J) = (10, 20), K = 3, dataNoise = 10



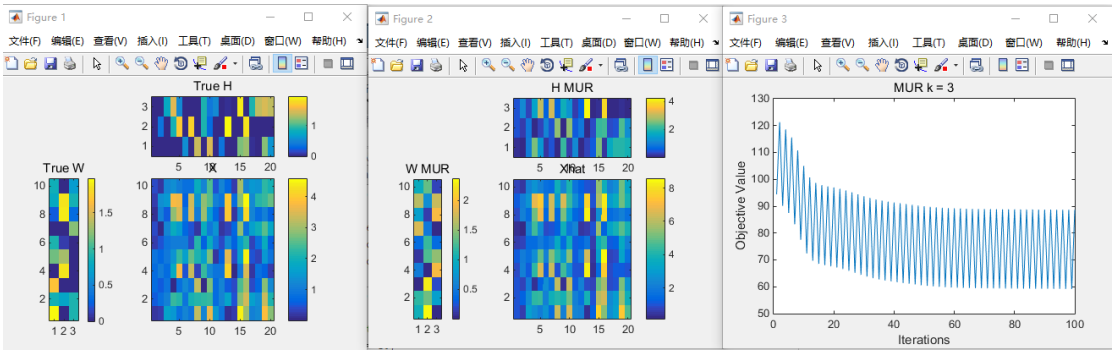
(I, J) = (10, 20), K = 1, dataNoise = 1



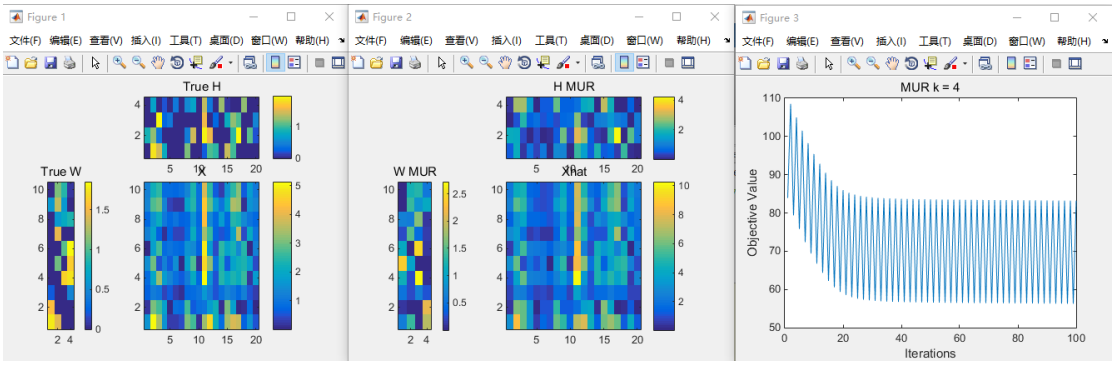
(I, J) = (10, 20), K = 2, dataNoise = 1



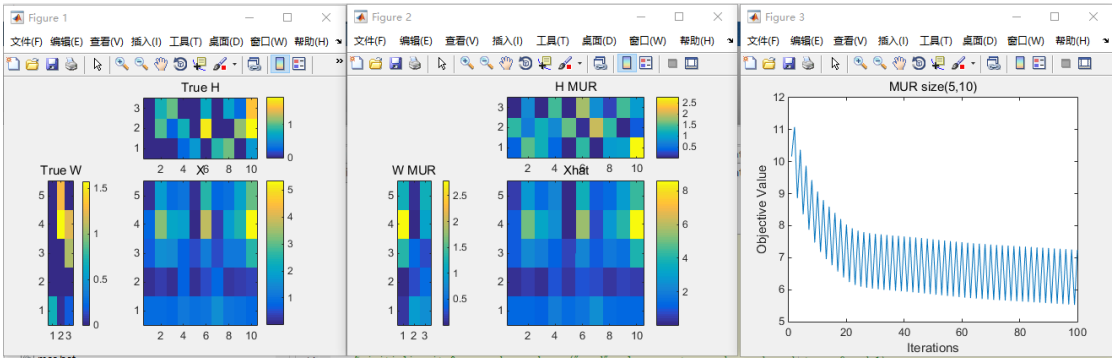
(I, J) = (10, 20), K = 3, dataNoise = 1



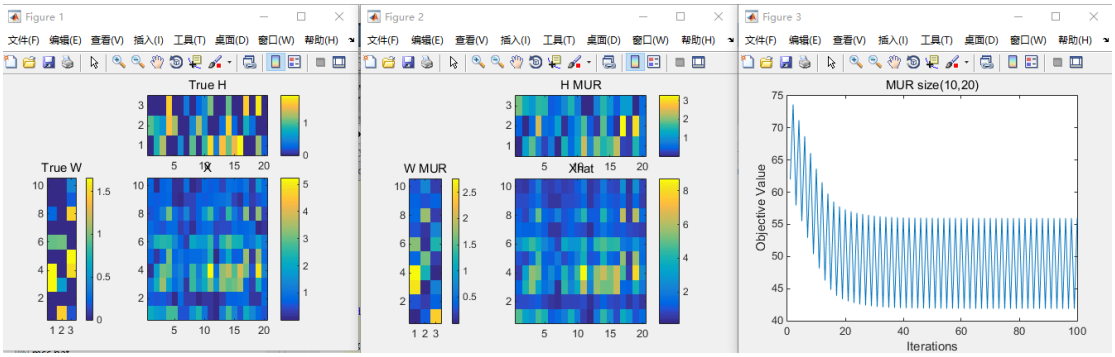
(I, J) = (10, 20), K = 4, dataNoise = 1



(I, J) = (5, 10), K = 3, dataNoise = 1



(I, J) = (10, 20), K = 3, dataNoise = 1



(I, J) = (20, 40), K = 3, dataNoise = 1