STA 250 Final

Chen Zihao 915490404

Probelm 1. Robust PCA

In the non-convex robust PCA problem, given an input matrix $M \in \mathbb{R}^{mxn}$, we need to solve the following optimaization problem:

$$\min_{W\in\mathbb{R}^{mxk},H\in\mathbb{R}^{nxk},S\in\mathbb{R}^{mxn}} \frac{1}{2}(||W||_F^2 + ||H||_F^2) + \lambda||S||_1,$$

$$s.t. WH^T + S = M$$

where $||S||_1 = \sum_{i,j} |S_{i,j}|$ is the element-wise l_1 norm. Note that k is small (we fix k=10 in this problem), So we expect to decompose the matrix M into the low-rank part WH^T and the sparse part S.

1.(20pt) Develop an algorithm for solving this problem. Describe your algorithm. What's the time complexity of your algorithm?

I will use ADMM algorithm to solve this problem.

Augumented-Lagragian:

$$L_\beta(W,H,S,Y) = \frac{1}{2}(||W||_F^2 + ||H||_F^2) + \lambda||S||_1 + \frac{\beta}{2}||WH^T + S - M||_F^2 + < Y, WH^T + S - M >$$

For k=0,1,2,...

$$W^{k+1} = \arg\min_W L_\beta(W^k, H^k, S^k, Y^k)$$

$$H^{k+1} = \arg\min_H L_\beta(W^{k+1}, H^k, S^k, Y^k)$$

$$S^{k+1} = \arg\min_S L_\beta(W^{k+1}, H^{k+1}, S^k, Y^k)$$

$$Y^{k+1} = Y^k + \eta(WH^T + S - M)$$

end.

let's get into details:

$$L_\beta(W,H,S,Y) = \frac{1}{2}(||W||_F^2 + ||H||_F^2) + \lambda||S||_1 + \frac{\beta}{2}||WH^T + S - M||_F^2 + < Y, WH^T + S - M >$$

$$\frac{\partial L_\beta(W,H,S,Y)}{\partial W} = W + \beta(WH^T + S - M)H + YH$$

let it equal to zero, we get

$$W = (\beta(M-S)H - YH)(I + \beta H^T H)^{-1}$$

```
Wupdate=function(W){
  W=(b*(M-S)%*%H-Y%*%H)%*%solve(diag(k)+b*t(H)%*%H)
  W
}
```

Similarly,We get

$$\frac{\partial L_\beta(W,H,S,Y)}{\partial H} = H + \beta(WH^T + S - M)^T W + Y^T W$$

let it equal to zero, we get

$$H = (\beta(M-S)^T W - Y^T W)(I + \beta W^T W)^{-1}$$

```
Hupdate=function(H){
  H=(b*t(M-S)%*%W-t(Y)%*%W)%*%solve(diag(k)+b*t(W)%*%W)
  H
}
```

$$\frac{\partial L_\beta(W,H,S,Y)}{\partial S} = \lambda\frac{\partial ||S||_1}{\partial S} + \beta(WH^T + S - M) + Y$$

we get

$$S_{ij} = sign((M - WH^T - \beta^{-1}Y)_{ij})max((|M - WH^T - \beta^{-1}Y| - \lambda\beta^{-1})_{ij}, 0)$$

```
Supdate=function(S){
  A=M-W%*%t(H)-Y/b
  for (i in 1:m){
    for (j in 1:n){
      S[i,j]=sign(A[i,j])*max(abs(A[i,j])-lambda/b,0)
    }
  }
  S
}
```

At last we update Y with a fix step size η.

```
Yupdate=function(Y){
  Y=Y+eta*(W%*%t(H)+S-M)
  Y
}
```

for each iteration, $o((mn)$

2.(20 pt) Generate synthetic data with n=m=100 and k = 10 to test your algorithm. Generate W^*, H^* from Gaussian random. Generate S^* to be a sparse matrix with uniform random nonzero entries, and random values for each entry. Generate $M = S^* + W^*(H^*)^T$ and solve the robust PCA problem (1). How well can you recover low-rank matrix $W^*(H^*)^T$ the sparse matrix S^*? Try to choose a good parameter λ that can better recover the matrices. Report your findings.

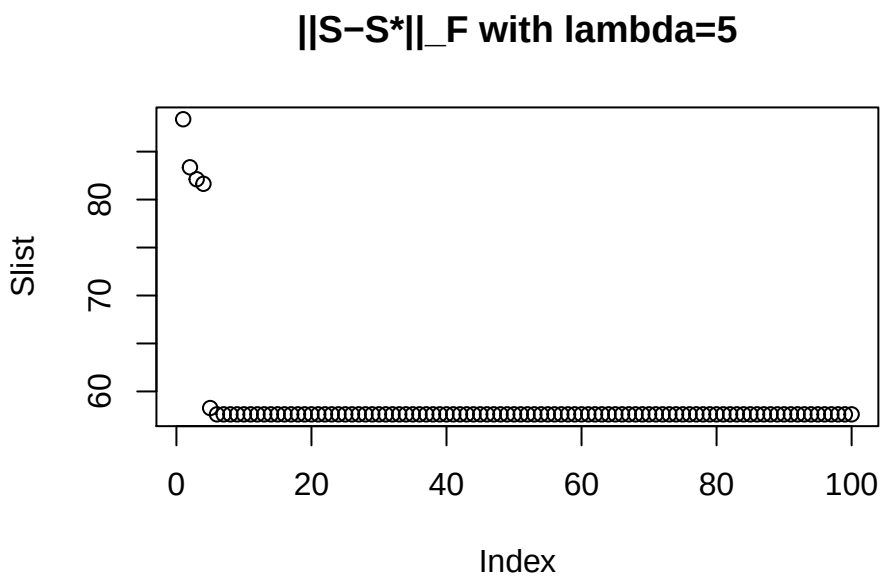
Generate synthetic data with n=m=100 and k = 10 to test your algorithm. Generate W^*, H^* from Gaussian random. Generate S^* to be a sparse matrix with uniform random nonzero entries, and random values for each entry. Generate $M = S^* + W^*(H^*)^T$.

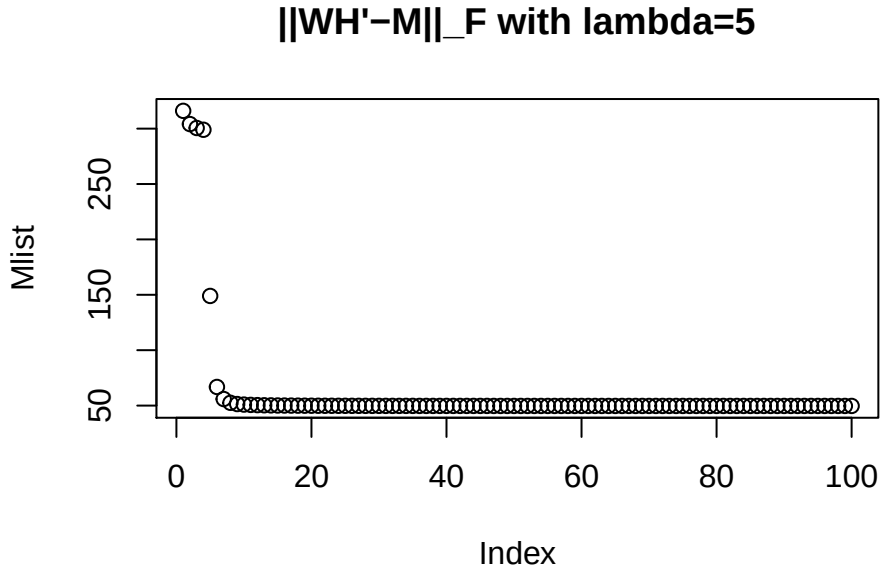
```
library("stats")
set.seed(10)
m=100
n=100
k=10

W1=matrix(rnorm(m*k),m,k)
```

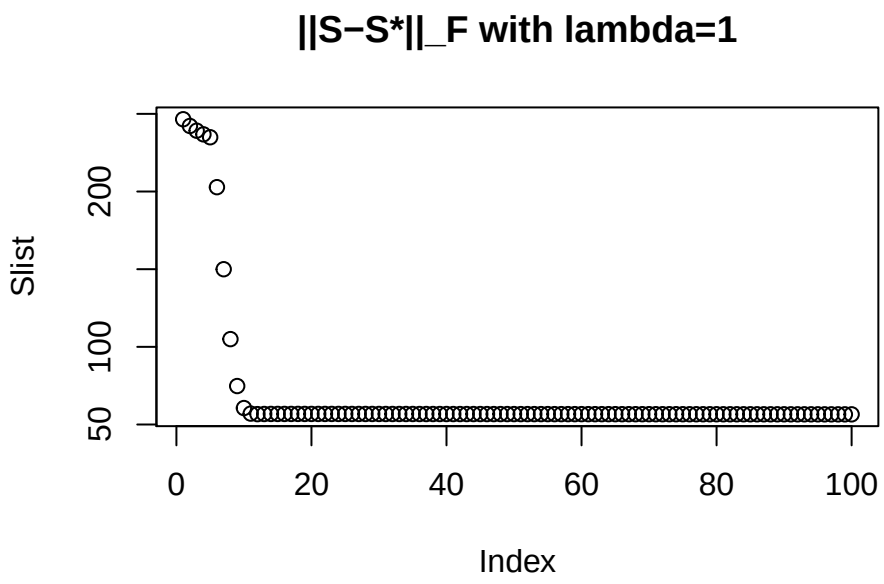
```
H1=matrix(rnorm(n*k),n,k)
S1=matrix(runif(m*n),m,n)
M=W1%*%t(H1)+S1
```

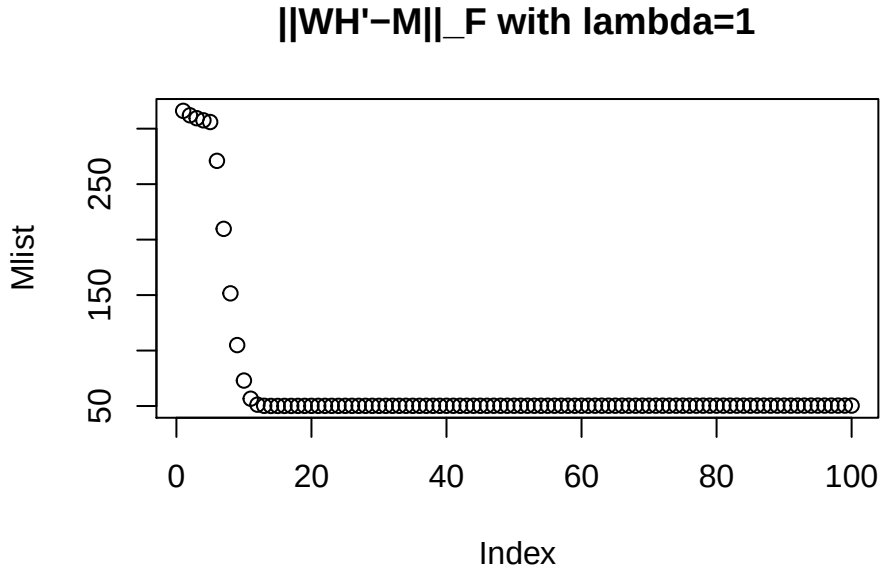
solve the robust PCA problem (1) with different λ

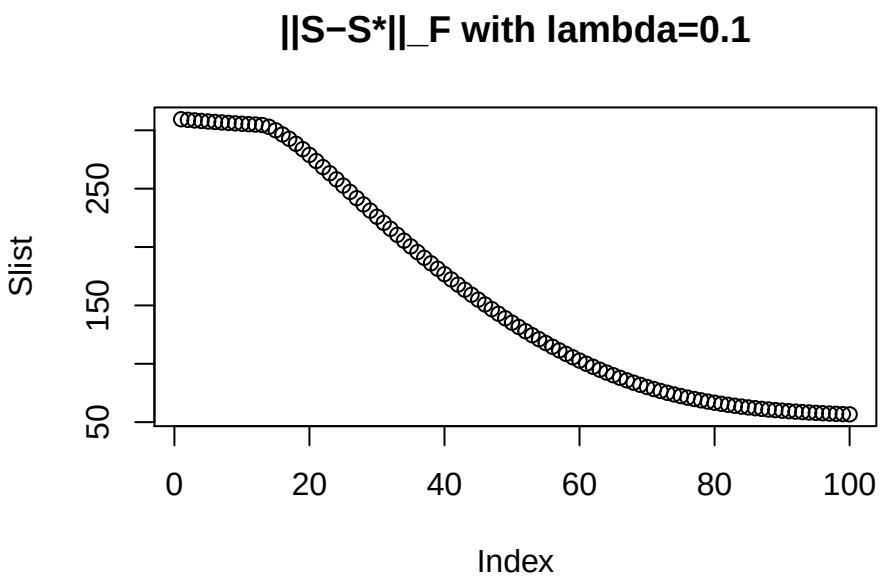
```
for (lambda in c(5,1,0.1,0.01)){
  eta=0.001
k=10
b=1
#initialize W,H,S,Y

W=matrix(numeric(m*n)+0.1,m,k)
H=matrix(numeric(m*n)+0.1,n,k)
S=matrix(numeric(m*n)+0.1,m,n)
Y=matrix(numeric(m*n),m,n)

Mlist=numeric(100)
Slist=numeric(100)

for (l in 1:100){
  W=Wupdate(W)
  H=Hupdate(H)
  S=Supdate(S)
  Y=Yupdate(Y)
  Slist[l]=norm(S-S1,type = "F")
  Mlist[l]=norm(W%*%t(H)-M,type="F")
  }
plot(Slist,main = paste("||S-S*||_F with lambda",lambda,sep = "="))

plot(Mlist,main = paste("||WH'-M||_F with lambda",lambda,sep = "="))


}
```

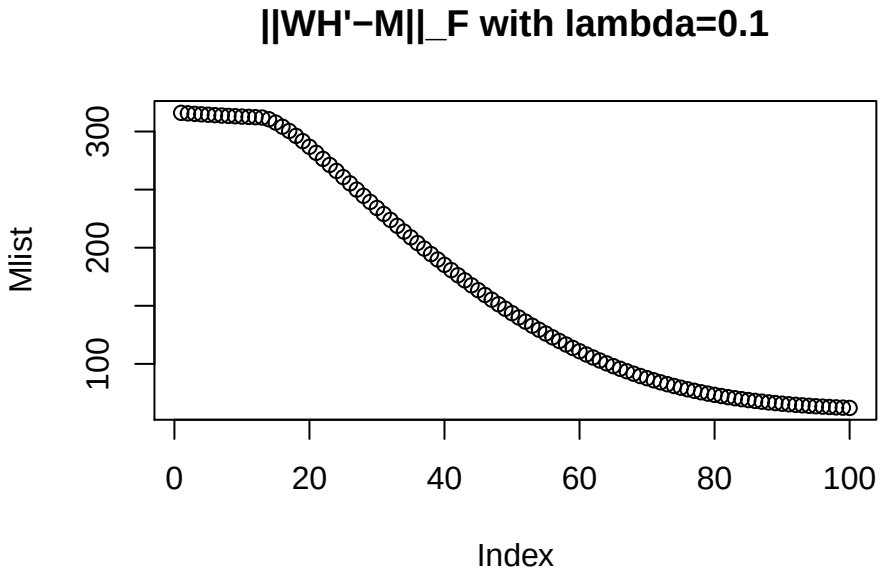

||S−S*||_F with lambda=5
Slist
60
70
80
0
20
40
60
80
100
Index

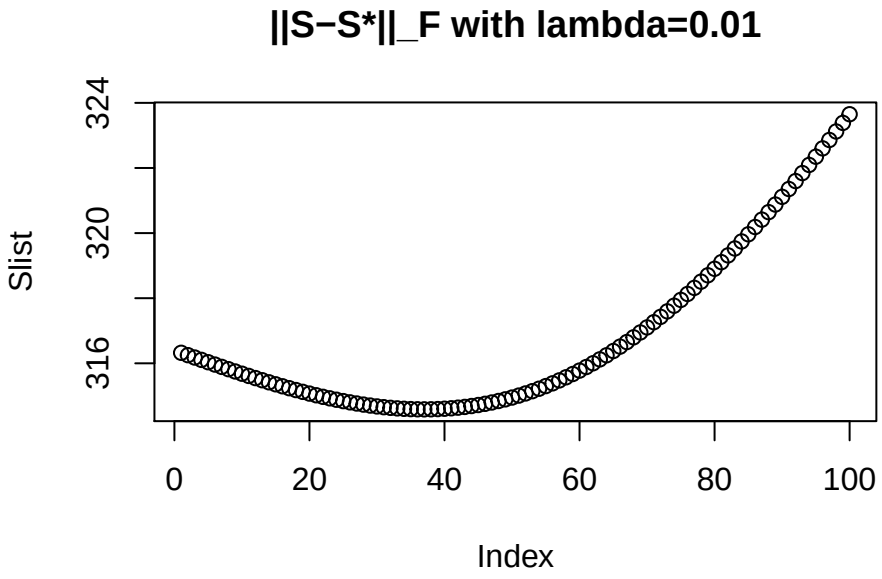

||WH'−M||_F with lambda=5
Mlist
50
150
250
0
20
40
60
80
100
Index

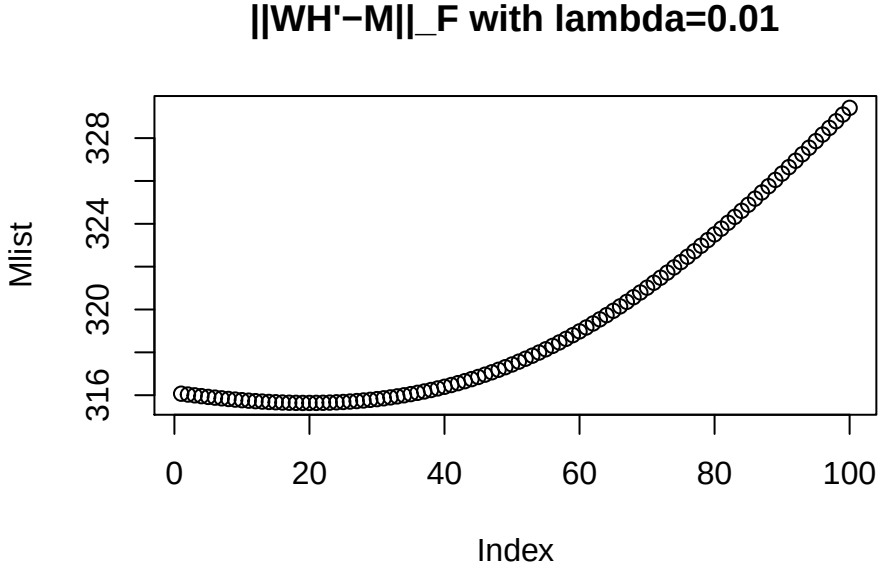

||S−S*||_F with lambda=1
Slist
50
100
200
0
20
40
60
80
100
Index


||WH'−M||_F with lambda=1
Mlist
50
150
250
0
20
40
60
80
100
Index


||S−S*||_F with lambda=0.1
Slist
50
150
250
0
20
40
60
80
100
Index


||WH'−M||_F with lambda=0.1
Mlist
100
200
300
0
20
40
60
80
100
Index

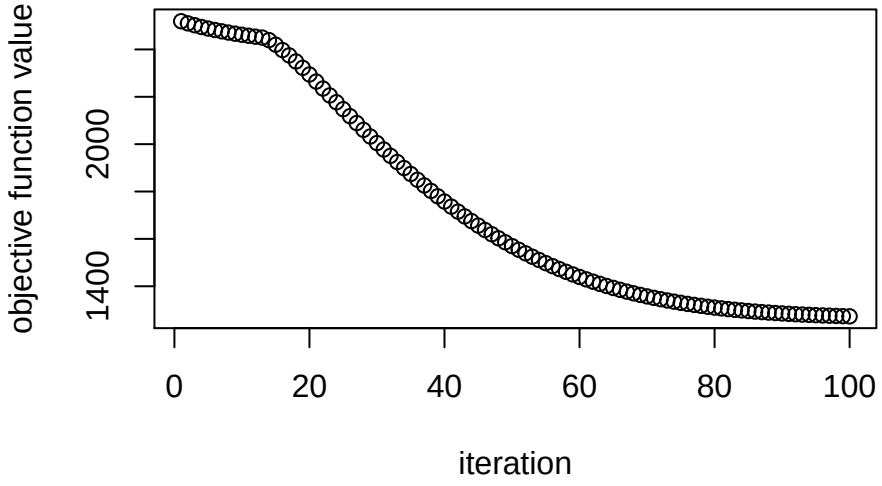




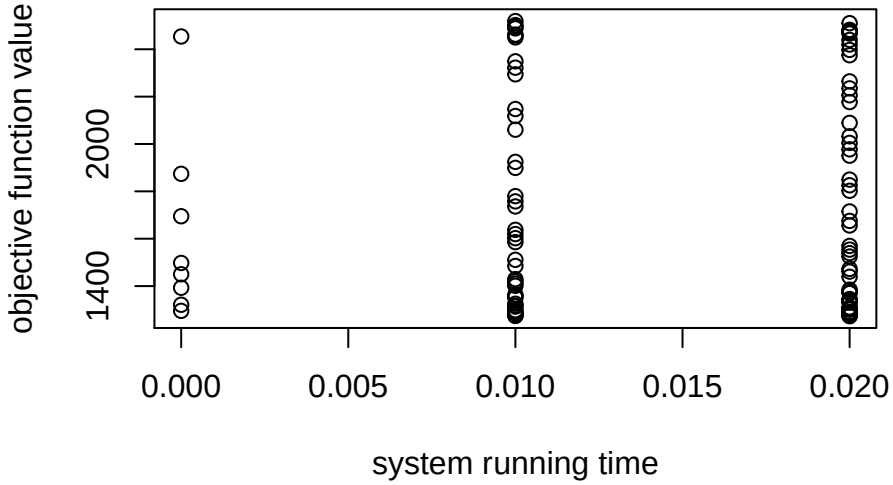
I found that with different lambda, the convergence rate, the convergence status with the same fixed step size and the values are all different. With a small lambda, the convergence rate is slower and with the same step size, it will not converge to M. I think lambda equals to 0.1 is the best for our further discussion in (3) the better convergence rate will be lambda=1.

3.(5 pt)Plot time vs objective function figures to see how fast your optimization algorithm converges.

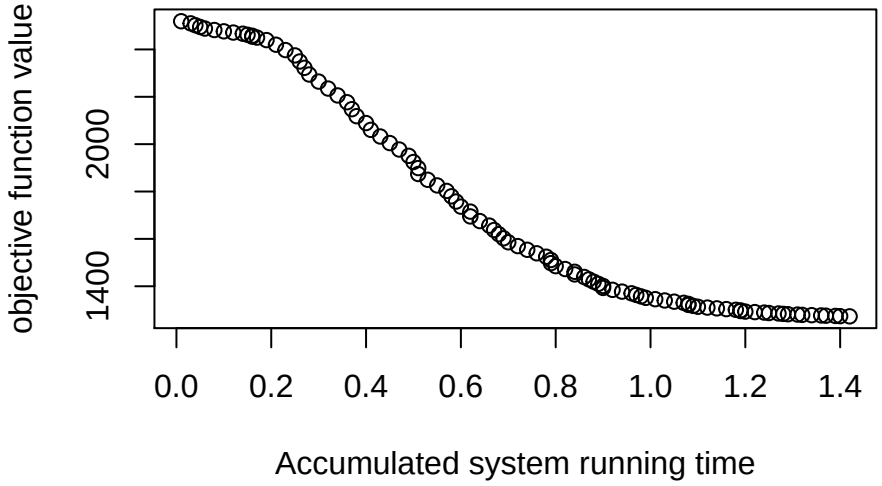
```
lambda=0.1
eta=0.001
k=10
b=1
#initialize W,H,S,Y

W=matrix(numeric(m*n)+0.1,m,k)
H=matrix(numeric(m*n)+0.1,n,k)
S=matrix(numeric(m*n)+0.1,m,n)
Y=matrix(numeric(m*n),m,n)

tlist=numeric(100)
tl=numeric(100)
Flist=numeric(100)

A=function(W,H,S,Y){
  W=Wupdate(W)
  H=Hupdate(H)
  S=Supdate(S)
  Y=Yupdate(Y)
}


for (l in 1:100){
  tl[l]=system.time(A(W,H,S,Y))[3]
  tlist[l+1]=tlist[l]+tl[l]
  W=Wupdate(W)
  H=Hupdate(H)
  S=Supdate(S)
  Y=Yupdate(Y)
  Flist[l]=(norm(W,type = "F")^2+norm(H,type = "F")^2)/2+lambda*sum(abs(S))
}
plot(Flist,xlab = "iteration",ylab="objective function value")
```




```
plot(tl,Flist,xlab="system running time",ylab="objective function value")
```



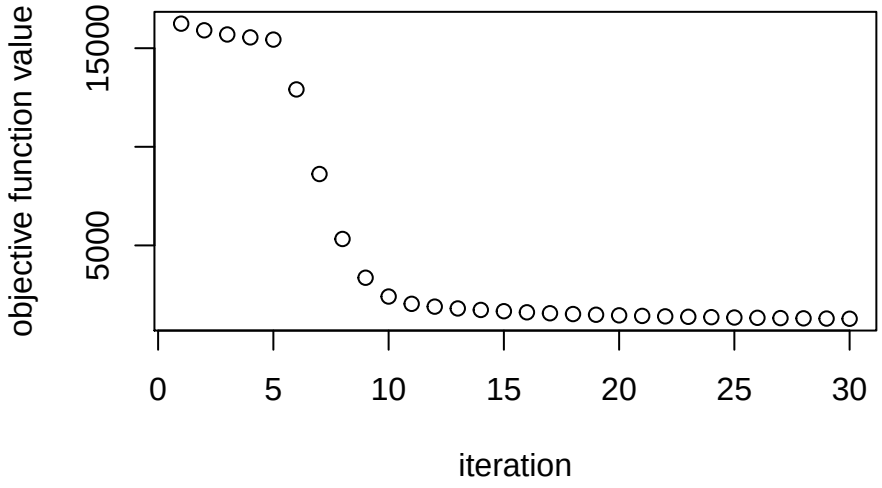

```
plot(tlist[-1],Flist,xlab="Accumulated system running time",ylab="objective function value")
```



the better one is showed below

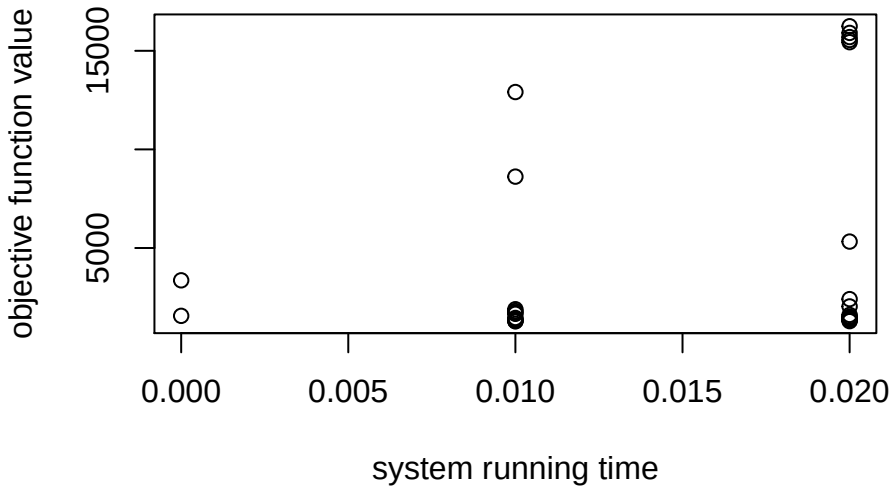
```
lambda=1
eta=0.001
k=10
b=1
#initialize W,H,S,Y

W=matrix(numeric(m*n)+0.1,m,k)
H=matrix(numeric(m*n)+0.1,n,k)
S=matrix(numeric(m*n)+0.1,m,n)
Y=matrix(numeric(m*n),m,n)

tlist=numeric(30)
tl=numeric(30)
Flist=numeric(30)

A=function(W,H,S,Y){
  W=Wupdate(W)
  H=Hupdate(H)
  S=Supdate(S)
  Y=Yupdate(Y)
}


for (l in 1:30){
  tl[l]=system.time(A(W,H,S,Y))[3]
  tlist[l+1]=tlist[l]+tl[l]
  W=Wupdate(W)
  H=Hupdate(H)
  S=Supdate(S)
  Y=Yupdate(Y)
  Flist[l]=(norm(W,type = "F")^2+norm(H,type = "F")^2)/2+lambda*sum(abs(S))
```

```
}
plot(Flist,xlab = "iteration",ylab="objective function value")
```

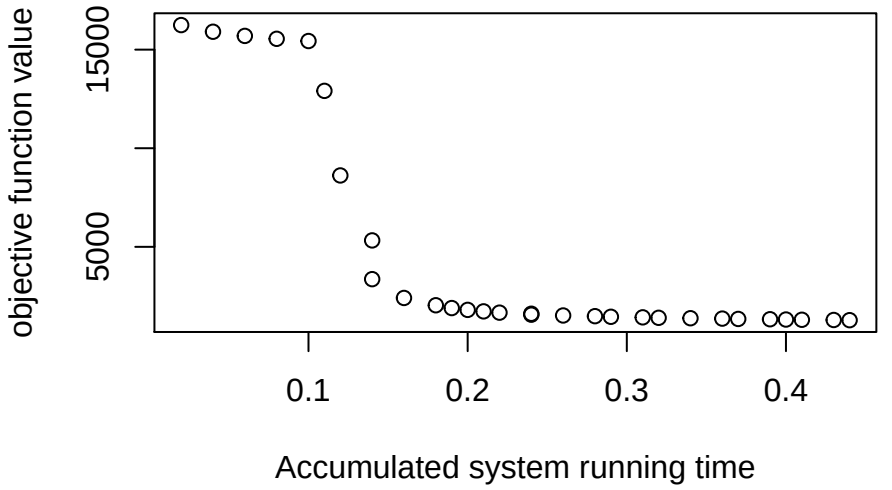



```
plot(Flist[-c(1:10)],xlab = "iteration",ylab="objective function value",main="the last 20 iteration")
```


```
plot(tl,Flist,xlab="system running time",ylab="objective function value")
```




```
plot(tlist[-1],Flist,xlab="Accumulated system running time",ylab="objective function value")
```

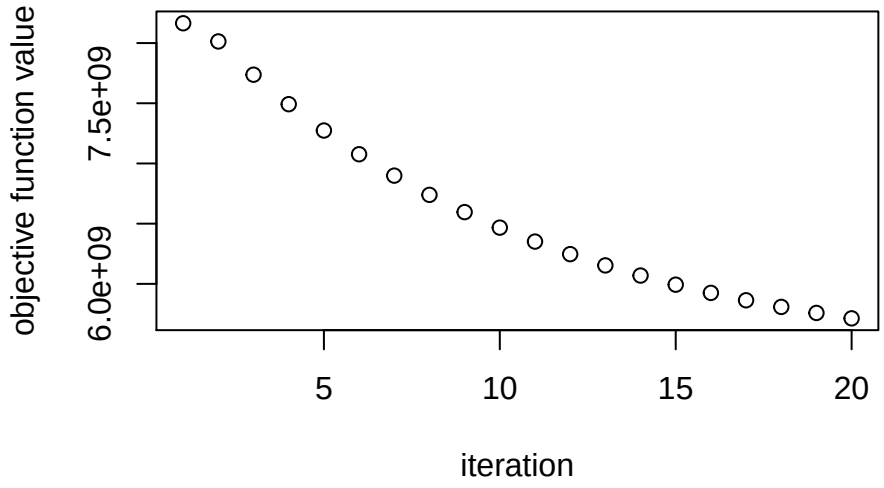


For every iteration, the system running time is almost the same.

4.(5 pt) Test your algorithm on a real dataset "mnist" (choose a parameter λ which you think gives you better result). Report the final objective function value and discuss your findings.

```
M=as.matrix(read.table("G:\\mnist"))
m=nrow(M)
n=ncol(M)
lambda=5
```

```
eta=10^(-5)
k=10
b=1
#initialize W,H,S,Y

W=matrix(numeric(m*n)+0.001,m,k)
H=matrix(numeric(m*n)+0.001,n,k)
S=matrix(numeric(m*n),m,n)
Y=matrix(numeric(m*n),m,n)

time=20

Flist=numeric(time)

for (l in 1:time){
  W=Wupdate(W)
  H=Hupdate(H)
  S=Supdate(S)
  Y=Yupdate(Y)
  Flist[l]=(norm(W,type = "F")^2+norm(H,type = "F")^2)/2+lambda*sum(abs(S))
}
plot(Flist,xlab = "iteration",ylab="objective function value")
```




```
min(Flist)
```

```
## [1] 5713445336
```

```
Flist[time]
```

```
## [1] 5713445336
```

I found that my algorithm is time comsuming. It is mainly because the "Supdate" function. it need to go to every elements in the large matrix to change its single value. If i can do it parallelly, it will be much quicker.