Homework 3: application of block coordinate update method

Wenting Li liw14@rpi.edu

April 18, 2018

1. Nonnegative matrix factorization

The nonnegative matrix factorization (NMF) is to find two nonnegative matrices $W \in R^{m\times r}, H \in R^{n\times r}$ such that their product approximates a given nonnegative matrix $X \in R^{m\times n}$. The model is based on Gaussian noise.

$$\min_{W,H} \frac{1}{2}\|WH^T - X\|_F^2, \text{s.t.} W \in R_+^{m\times r}, H \in R_+^{n\times r}, \|W_j\|_2 \le 1, j = 1, \cdots, r. \tag{1}$$

Three solvers are developed as follows.

1.1 Projected Gradient (PG)

There are two steps of the projected gradient algorithm to update each variable. Let $f(W,H) = \frac{1}{2}\|WH^T - X\|_F^2$. The first step is to compute the gradient descent of each variable without considering their constrains, and the second step is to project the results of the first step to the feasible region. Specifically, we update W^{k+1}, H^{k+1} through the following rules:

$$W^{k+1} = \mathcal{P}_{\mathcal{W}}(W^k - \alpha^k \nabla_W f(W^k, H^k)) \tag{2}$$

$$H^{k+1} = \max((H^k - \alpha^k \nabla_H f(W^k, H^k)), 0) \tag{3}$$

$$\nabla_W f(W^k, H^k) = (WH^T - X)H \tag{4}$$

$$\nabla_H f(W^k, H^k)) = (WH^T - X)^T W \tag{5}$$

$$\mathcal{W} = \{W | W \in R_+^{m\times r}, \|W_j\|_2 \le 1, j = 1, \cdots, r \tag{6}$$

$$\mathcal{P}_{\mathcal{W}}(x) = \begin{cases} x & \text{if } x \in \mathcal{W} \\ \frac{x}{\|x\|_2} & \text{if } x \notin \mathcal{W} \end{cases} \tag{7}$$

$$\tag{8}$$

Since the Lipschitz constant L can be computed, we can set the step size $\alpha \le \frac{1}{L}$ and L is the spectrum norm of Hessian matrix of $f(W^k, H^k)$. Specifically, the step size is $\alpha \le 1/L = 1/\|\nabla^2 f(W^k, H^k)\|_2$, where

$$\nabla^2 f(W^k, H^k) = \begin{bmatrix} H^k(H^k)^T & 2H^k(W^k)^T - X^T \\ 2W^k(H^k)^T - X & (W^k)^T W^k \end{bmatrix} \tag{9}$$

Notice that actually from the experimental testings we found that if we consider variable W^k and H^k respectively and set the step size of W is $\alpha_w \le 1/L_w = 1/\|\nabla_W^2 f(W^k, H^k)\|_2 = 1/\|H^k(H^k)^T\|_2$ while the step size of H is $\alpha_h \le 1/\|\nabla_H^2 f(W^k, H^k)\|_2 = 1/\|(W^k)^T W^k\|_2$, and update the step size for each iteration, the optimization problem can also converge fast, as shown in Figure 1, thus we did not compute the huge Hessian matrix but applied α_w, α_h instead.

The implementation of the projecting the matrix W to the $\mathcal{W}$ space, we follow the (10). Let $W^+ = \max(W^k - \alpha^k \nabla_W f(W^k, H^k), 0)$, denoting that the normalization is only implemented only if the column of W^k is not in the unit circle, then we have

$$W^{k+1} = \mathcal{P}_{\mathcal{W}}(W^k - \alpha^k \nabla_W f(W^k, H^k)) \tag{10}$$

$$= \frac{W^+}{\max(W^+, 1)} \tag{11}$$

where the division is the elementmize operator.

The initialization W_0, H_0 is implemented by some random numbers, and the maximum iteration is M, rank r should be selected based on the dataset.

Algorithm 1 PG

```
1: Given parameters $M, X, r$
2: Initialization: $W_0, H_0, \alpha_w, \alpha_h,$
3: for $k = 0, 1, 2, \cdots, M$ do
4:     Set $\alpha_w = 1/\|H^k(H^k)^T\|_2,$
5:     $\alpha_h = 1/\|(W^k)^T W^k\|_2$
6:     Update $W^{k+1} = \frac{W^+}{\max(W^+,1)}$, where $W^+ = \max(W^k - \alpha^k \nabla_W f(W^k, H^k), 0)$
7:     Update $H^{k+1} = \max((H^k - \alpha_h \nabla_H f(W^k, H^k)), 0)$
8: return $W^{k+1}, H^{k+1}$
```

1.2 Alternating Minimization (AltMin1)

As function $f(W, H)$ is continuous and bounded in the feasible region, we know that the solution $\{W^k, H^k\}$ generated by the two block coordinate descent algorithm or the AltMin is the stationary point of the problem (1). Thus we can find the stationary point by solving two sub-problem (12), (13) at each iteration. We applied cvx to solve each sub optimization problem in algorithm 2.

The W, H sub problems are following,

$$W^{k+1} = \arg\min_W \frac{1}{2}\|W(H^k)^T - X\|_F^2, \text{s.t.} W \in R_+^{m\times r}, \|W_j\|_2 \le 1, j = 1, \cdots, r. \tag{12}$$

$$H^{k+1} = \arg\min_H \frac{1}{2}\|W^{k+1}H^T - X\|_F^2, \text{s.t.} H \in R_+^{n\times r} \tag{13}$$

Algorithm 2 AltMin

```
1: Given parameters $M, X, r$
2: Initialization: $W_0, H_0$
3: for $k = 0, 1, 2, \cdots, M$ do
4:     Update $W^{k+1} = \arg\min_W \frac{1}{2}\|W(H^k)^T - X\|_F^2$, s.t.$W \in R_+^{m\times r}$, $\|W_j\|_2 \le 1, j = 1, \cdots, r.$
5:     Update $H^{k+1} = \arg\min_H \frac{1}{2}\|W^{k+1}H^T - X\|_F^2$, s.t.$H \in R_+^{n\times r}$
6: return $W^{k+1}, H^{k+1}$
```

1.3 Alternating Proximal Gradient (APG)

The alternating proximal gardient algorithm is similar to PG, but the only difference is that when update the matrix H^{k+1} we apply the W^{k+1} instead of W^k, please see the details in algorithm 3.

There are two steps of the projected gradient algorithm to update each variable. Let $f(W,H) = \frac{1}{2}\|WH^T - X\|_F^2$. The first step is to compute the gradient descent of each variable without considering their constrains, and the second step is to project the results of the first step to the feasible region. Specifically, we update W^{k+1}, H^{k+1} through the following rules:

$$W^{k+1} = \mathcal{P}_{\mathcal{W}}(W^k - \alpha^k \nabla_W f(W^k, H^k)) \tag{14}$$

$$H^{k+1} = \max((H^k - \alpha^k \nabla_H f(W^{k+1}, H^k)), 0) \tag{15}$$

$$\nabla_W f(W^k, H^k) = (W^k (H^k)^T - X) H^k \tag{16}$$

$$\nabla_H f(W^k, H^k)) = (W^{k+1}(H^k)^T - X)^T W^{k+1} \tag{17}$$

$$\mathcal{W} = \{W | W \in R_+^{m\times r}, \|W_j\|_2 \le 1, j = 1, \cdots, r \tag{18}$$

$$\mathcal{P}_{\mathcal{W}}(x) = \begin{cases} x & \text{if } x \in \mathcal{W} \\ \frac{x}{\|x\|_2} & \text{if } x \notin \mathcal{W} \end{cases} \tag{19}$$

$$\tag{20}$$

The initialization W_0, H_0 is implemented by some random numbers, and the maximum iteration is M, rank r should be selected based on the dataset.

Algorithm 3 APG

```
1: **Given parameters** $M, X, r$
2: **Initialization:** $W_0, H_0, \alpha_w, \alpha_h$,
3: **for** $k = 0, 1, 2, \cdots, M$ **do**
4:     Set $\alpha_w = 1/\|H^k (H^k)^T\|_2$,
5:     Update $W^{k+1} = \frac{W^+}{\max(W^+, 1)}$, where $W^+ = \max(W^k - \alpha^k \nabla_W f(W^k, H^k), 0)$
6:     $\alpha_h = 1/\|(W^{k+1})^T W^{k+1}\|_2$
7:     Update $H^{k+1} = \max((H^k - \alpha_h \nabla_H f(W^{k+1}, H^k)), 0)$
8: **return** $W^{k+1}, H^{k+1}$
```

2. Comparison of the three solvers

In order to compare these three solvers, we solve the NMF problem through the ‘Swimmer’ dataset. We set the maximum iteration to be 500, rank $r = 17$, and compared the objective values of these three solvers in terms of iterations on the left of Figure 1. Here $iteration$ means to finish updating both W and H. The running time of these three solvers are 3.3 seconds, 3.25 seconds and 40734 seconds. In fact, the long running time of AltMin is due to the package of CVX.

We can observe that the AltMin algorithm has reached the optimal solution after 100 iterations while the other two algorithms are much slower, but for each iteration the AltMin solver needs many steps to solve the sub problem while the other two solvers only run one step. Actually after 500 iterations, the PG and APG reach their optimal solution. By comparing PG and APG, as shown on the right of the Figure 1, the APG converges faster than PG since this algorithm update H based on the new W at each iteration.

2.2.2 Visualize the columns of W

After solving the problem, we can visualize the learned parameter matrix W column by column. The 17 columns of W are reshaped into a 32 by 32 matrix and are showed as images in Figure 2. These are

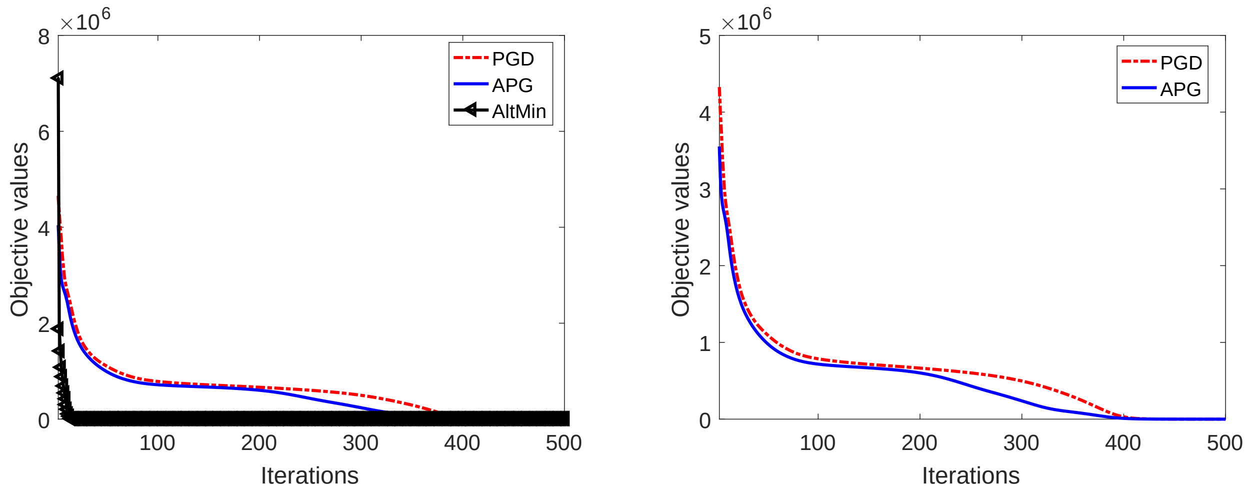

Figure 1: The objective values in terms of iteration k

the first 10 columns of W. As we can observe that the columns of W reflect the features of the input data, which are the limbs of the swimmers.

3.Robust PCA

Let X be composed of a spase matrix S and a low rank matrix L. The robust PCA to find S and L, given X. The problem can be formed as (21)

$$\min_{L,S} \|L\|_* + \lambda \|S\|_1, \text{s.t.} L + S = X \tag{21}$$

In order to use the alternating minimization algorithm, we can relax the (21) to (22).

$$\min_{L,S} \|L\|_* + \lambda \|S\|_1 + \frac{\beta}{2}\|L + S - X\|_F^2, \beta > 0 \tag{22}$$

$$\tag{23}$$

Then we can solve this problem by two algorithms: proximal gradient descent and alternating minimization algorithms as follow.

3.1 Proximal Gradient Descent(PGD)

Before explaining the sub-problem of L, we introduce the soft-thresholding operator $\mathcal{S}_\tau$ [R1] and a useful theory together with the proof. Given a matrix $X \in R^{m\times n}$ with a rank r, we have the singular value decomposition (SVD) of X in (24), and then the soft-thresholding operator $\mathcal{S}_\tau$ on matrix X is defined in (26).

$$X = U\Sigma V^* \tag{24}$$

$$\Sigma = \text{diag}(\delta_{i(1\le i\le r)}) \tag{25}$$

$$\mathcal{S}_\tau(X) = U\mathcal{S}_\tau(\Sigma)V^* \tag{26}$$

$$\mathcal{S}_\tau(\Sigma) = \text{diag}(\{\delta_i - \tau\}_+) 1 \le i \le r \tag{27}$$

where $U \in R^{m\times r}, V \in R^{n\times r}$ are orthonormal matrices, $*$ denotes the conjugate transpose and the singular value δ_i are positive, $\{\delta - \tau\}_+$ means the positive set of $\{\delta - \tau\}\forall i$.

Claim 1: The optimal solution of the problem $\tau\|X\|_* + \frac{1}{2}\|X - Y\|_F^2$ **is** $\mathcal{S}_\tau(Y)$.

Proof. As the function $\|X\|_* + \frac{1}{2}\|X-Y\|_F^2$ is convex, the optimal solution is unique and we need to show the $\bar{X} = \mathcal{S}_\tau(Y)$ is the solution.

By the first order optimal condition, we prove the optimal solution $\bar{X}$ should satisfy that

$$\mathbf{0} \in (\bar{X} - Y) + \tau\partial\|\bar{X}\|_* \tag{28}$$

Let the SVD of Y be $Y = U_0\Sigma_0V_0^* + U_1\Sigma_1V_1^*$, where the singular values of Σ_1 are less or equal to τ and that of Σ_0 are higher than τ, and $\bar{X} = \mathcal{S}_\tau(Y) = U_0(\Sigma_0 - \tau I)_+V_0^*$ Then we have

$$Y - \bar{X} = \tau(U_0V_0^* + W_0), W_0 = \frac{1}{\tau}(U_1\Sigma_1V_1^*) \tag{29}$$

thus $U_0W = 0, WV_0 = 0, \|W\|_2 \le 1$. As we know the subgradient of $\partial\|X\|_*$ should satisfy

$$\partial\|X\|_* = \{UV^* + W : UW = 0, WV = 0, \|W\|_2 \le 1\} \tag{30}$$

According to the definition of (30), we have $U_0V_0^* + W_0 \in \partial\|X\|_*$, thus $Y - \bar{X} - \partial\|X\|_* = 0$, and then optimal condition can be satisfied. The $\bar{X} = \mathcal{S}_\tau(Y)$ is the optimal solution. $\square$

Based on the claim 1, we can obtain the closed-form solution of the proximal gradient descent method at each iteration, we update S^{k+1} and L^{k+1} simultaneously by the two optimization problems. As the Lipschitz constant equals 2β (the spectrum norm of the Hessian matrix of $f(L^k, S^k)$ is 2β) , we can set the step size to be $\alpha \le \frac{1}{2\beta}$. Let $f(L^k, S^k) = \frac{\beta}{2}\|L + S - X\|_F^2$, define $Y^k = L^k - \alpha\nabla_L f(L^k, S^k) = \frac{1}{2}L^k + \frac{1}{2}(X - S^k)$, and SVD of $Y^k = U\Sigma V^*$. From the expression of Y^k, we can understand that Y^k is one point between the previous variable L^k and the its gradient value $X - S^k$.

$$\begin{align}
L^{k+1} &= \text{Prox}_{\frac{1}{2\beta}\|L\|_*}(L - Y^k) \tag{31}\\
&= \arg\min_L \frac{1}{2\beta}\|L\|_* + \frac{1}{2}\|L - Y^k\|_F^2 \tag{32}\\
&= \mathcal{S}_{\frac{1}{2\beta}}(Y^k) \tag{33}\\
&= U\mathcal{S}_{\frac{1}{2\beta}}(\Sigma)V^* \tag{34}
\end{align}$$

The solution of (33) holds due to the Claim 1. Then we update S sub-problem similarly. Let $Z^k = S^k - \frac{1}{2\beta}\nabla_S f(L^k, S^k) = \frac{1}{2}S^k + \frac{1}{2}(X - L^k)$

$$\begin{align}
S^{k+1} &= \text{Prox}_{\frac{\lambda}{2\beta}\|S\|_1}(S - Z^k) \tag{35}\\
&= \arg\min_S \frac{\lambda}{2\beta}\|S\|_1 + \frac{1}{2}\|S - Z^k\|_F^2 \tag{36}\\
&= \mathcal{S}_{\frac{\lambda}{2\beta}}(Z^k) \tag{37}
\end{align}$$

where the definition of $\mathcal{S}_\lambda(X)$ denotes to take the operator of $\mathcal{S}_\lambda(x)$ for each element x of X, where

$$\mathcal{S}_\lambda(x) = \begin{cases} x - \lambda & \text{if } x > \lambda \\ x + \lambda & \text{if } x < -\lambda \\ 0 & \text{otherwise} \end{cases} \tag{38}$$

Algorithm 4 PGD

```
1: Given parameters $X, \beta^0 = 0.01, \lambda, M, \beta_{max} = 10^5$
2: Initialization: $L_0, S_0$
3: for $k = 0, 1, 2, \cdots, M$ do
4:     We fixed the step size $\alpha = \frac{1}{2\beta}$, such that
5:     $Y^{k+1} = \frac{1}{2}L^k + \frac{1}{2}(X - S^k)$
6:     $Y^{k+1} = U\Sigma V^*$
7:     $Z^{k+1} = \frac{1}{2}S^k + \frac{1}{2}(X - L^k)$
8:     Update $L^{k+1} = U\mathcal{S}_{\frac{1}{2\beta}}(\Sigma)V^*$
9:     Update $S^{k+1} = \mathcal{S}_{\frac{\lambda}{2\beta}}(Z^{k+1})$
10:    Update $\beta^{k+1} = \min(\beta_{max}, 1.1\beta^k)$
11: return $L^{k+1}, S^{k+1}$
```

3.2 ALternating minimization (AltMin2)

For the alternating minimization algorithm, we need to solver two sub problems at each iteration. The one sub problem is to update the L by solving the (39) and the other sub problem is to update S by solving the (40). The update rule is still to follow (33) with fixing S^k and (35) with fixing L^k. For each sub problem, we choose the fixed step size $\alpha \leq \frac{1}{\eta\beta}, \eta > 1$ since the Lipschitz constant for L and S is β respectively.

$$L^{k+1} = \arg\min_{L} \|L\|_* + \frac{\beta}{2}\|L^k + S^k - X\|_F^2 \tag{39}$$

$$S^{k+1} = \arg\min_{S} \lambda\|S\|_1 + \frac{\beta}{2}\|L^{k+1} + S^k - X\|_F^2, \beta > 0 \tag{40}$$

Algorithm 5 Alternating minimization (AltMin2)

```
    Given parameters $X, \beta^0 = 0.01, \lambda, M, \eta = 2, \beta_{max} = 10^5$
2:  Initialization: $L_0, S_0$
    for $k = 0, 1, 2, \cdots, M$ do
4:      We fixed the step size $\alpha = \frac{1}{\eta\beta}$
        while not converge do
6:          $Y^{k+1} = L^k - \frac{1}{\eta}(L^k + S^k - X)$
            $Y^{k+1} = U\Sigma V^*$
8:          Update $L^{k+1} = U\mathcal{S}_{\frac{1}{\eta\beta}}(\Sigma)V^*$
        while not converge do
10:         $Z^{k+1} = S^k - \frac{1}{\eta}(L^{k+1} + S^k - X)$
            Update $S^{k+1} = \mathcal{S}_{\frac{\lambda}{\eta\beta}}(Z^{k+1})$
12:     Update $\beta^{k+1} = \min(\beta_{max}, 1.1\beta^k)$
    return $L^{k+1}, S^{k+1}$
```

The inner loop can also be computed by other solvers like cvx but we applied the approximation gradient descent to update L and S separately.

3.2.1 parameter adjustment

The two parameters λ, β have highly influence on the rank of L and the step size. When β is fixed, a smaller λ will push the rank of L to be lower, and otherwise the rank of L will be higher. As we want to separate the background from the image, we want the rank of L to be lower and thus the λ should be small. When fixing the λ, a large β will cause the step size to be small and a higher penalty on the constrain $L + S = X$. In order to have results within this feasible region, β should be large. Thus in order to ensure that the results are in the feasible region and the rank of L is low, we increase the β at each iteration from a small number, in this way, a small β push the rank of matrix L to be small and gradually a larger β constrains the L and S to the feasible region, otherwise a too large or too small fixed β can not ensure the separation of the low rank matrix and the sparse matrix.

4 Comparison of PGD and AltMin2

By testing on the Escalator video Dataset, we compared the performances of the two solvers. The parameter for these two solvers are as following: maximum iteration is 200, $\eta = 2$, $\lambda = 0.005$, β changes from 0.01 to 10^5 by increasing 1.1 times at each iteration.

4.1 Convergent rate comparison

The objective values of these two solvers are shown in Figure 3. The running time of 200 iterations for proximal gradient descent is 67.22 seconds , and that of Altmin is 832.36 seconds. From Figure 3, although the objective values of alternating minimization solver converges slightly faster, the objective values of the Alternating minimization solver at each iteration are computed by solving two sub optimal problems and this process often needs many iterations. Thus the total running time of alternating minimization solver is much larger than that of Proximal gradient descent method.

4.2 The images of the optimal L and S

By image playing the input datasets, we can observe that the datasets are the images of a period of video. For these images, most of the items are static, such as the ground and the time symbols, but only small parts are changing, for example, the people and the stairs of the elevator are moving. In this way, we can imply that the part of the data denoting the static items have a low rank while those representing the moving iterms are sparse. Therefore, by solving the robust PCA problem on this dataset, the low rank part L can be separated as the background from the moving parts S.

The goal of solving the robust PCA problem is to find the low rank L and the sparse matrix S. In order to obtain such matrices, we need to tune the penalty parameter β with iterations. At the beginning, we first fixed the $\lambda = 0.005$, and chosen a small number $\beta = 0.01$. As a result, we observed that the rank of L was very low at the beginning, starting from 4 even though L has the dimension of 20800 by 200. After each iteration, we increased the β with 1.1 times, and we can observe that the rank of L and the sparsity (the number of nonzeros) of matrix S increased gradually, and the objective values decreased, as shown in Table 1. Eventually the errors between $L + S$ and X is pushed to a small number by the large penalty β and the background and foreground are separated as shown in the Figure 3. After 170 iterations, the objective reached the optimal solution and the rank of L becomes 64. In Figure 4, we selectively showed 3 columns of the matrix S, L. The columns of matrix S, as shown in the first 3 images, demonstrated the moving people and stairs while the last 3 images are the columns of matrix L indicate the background.

Table 1: The variations of $L's$ rank and $S's$ sparsity and objective values with iterations

Iterations	Rank of L	Sparsity of S	objective
10	4	94685	6.37 $\times 10^5$
40	10	837234	3.79 $\times 10^4$
80	54	3698361	8.36 $\times 10^2$
160	64	4041988	4.08 $\times 10^{-1}$

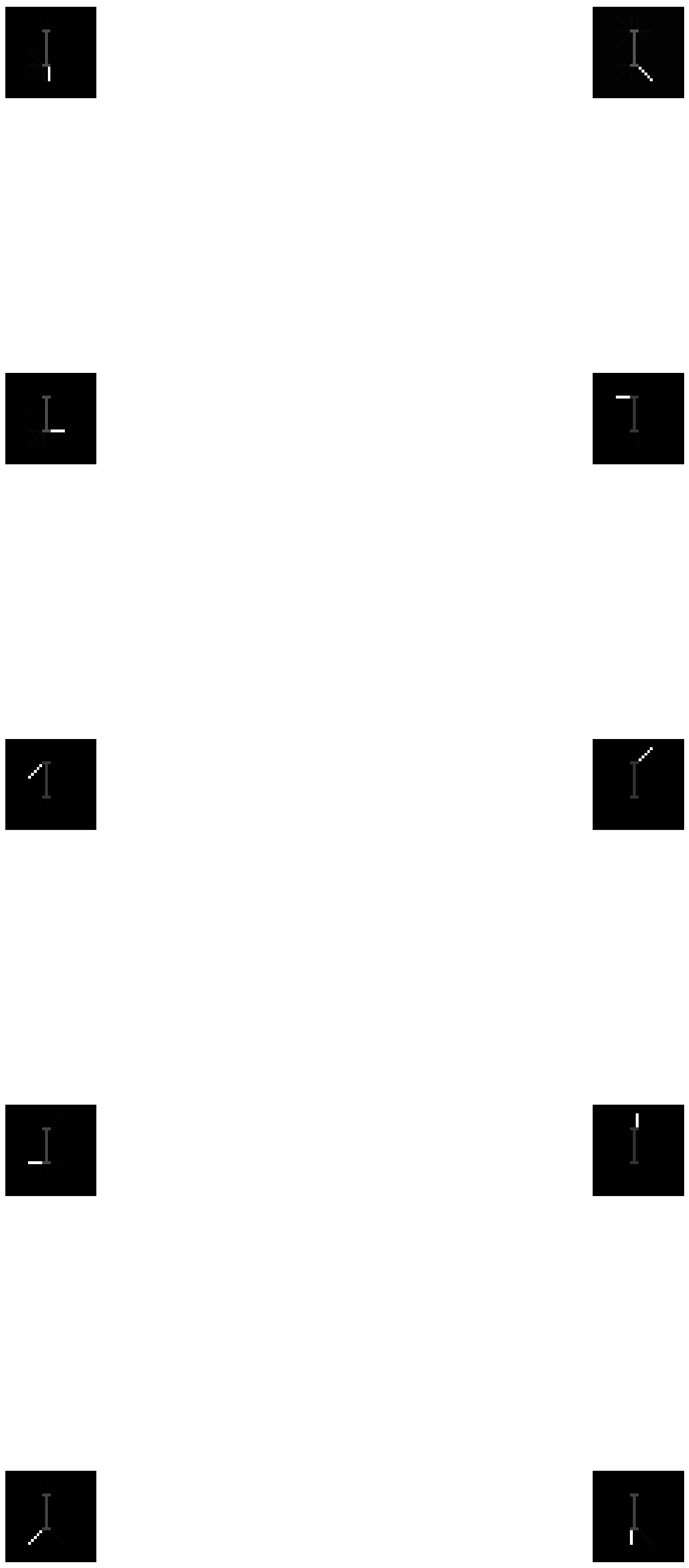

Figure 2: The 10 columns of matrix W

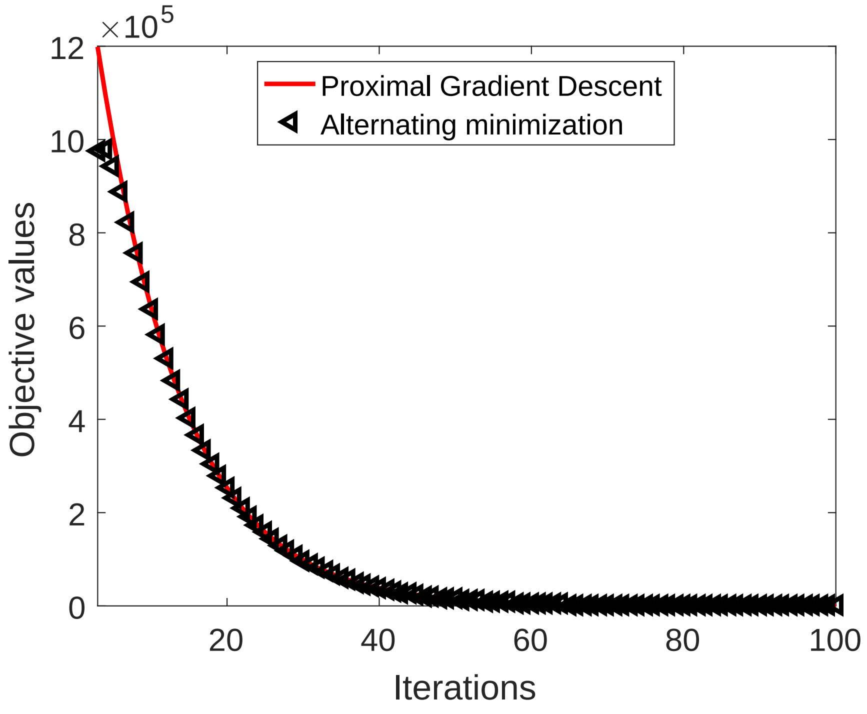

Figure 3: The objective values in terms of iterations

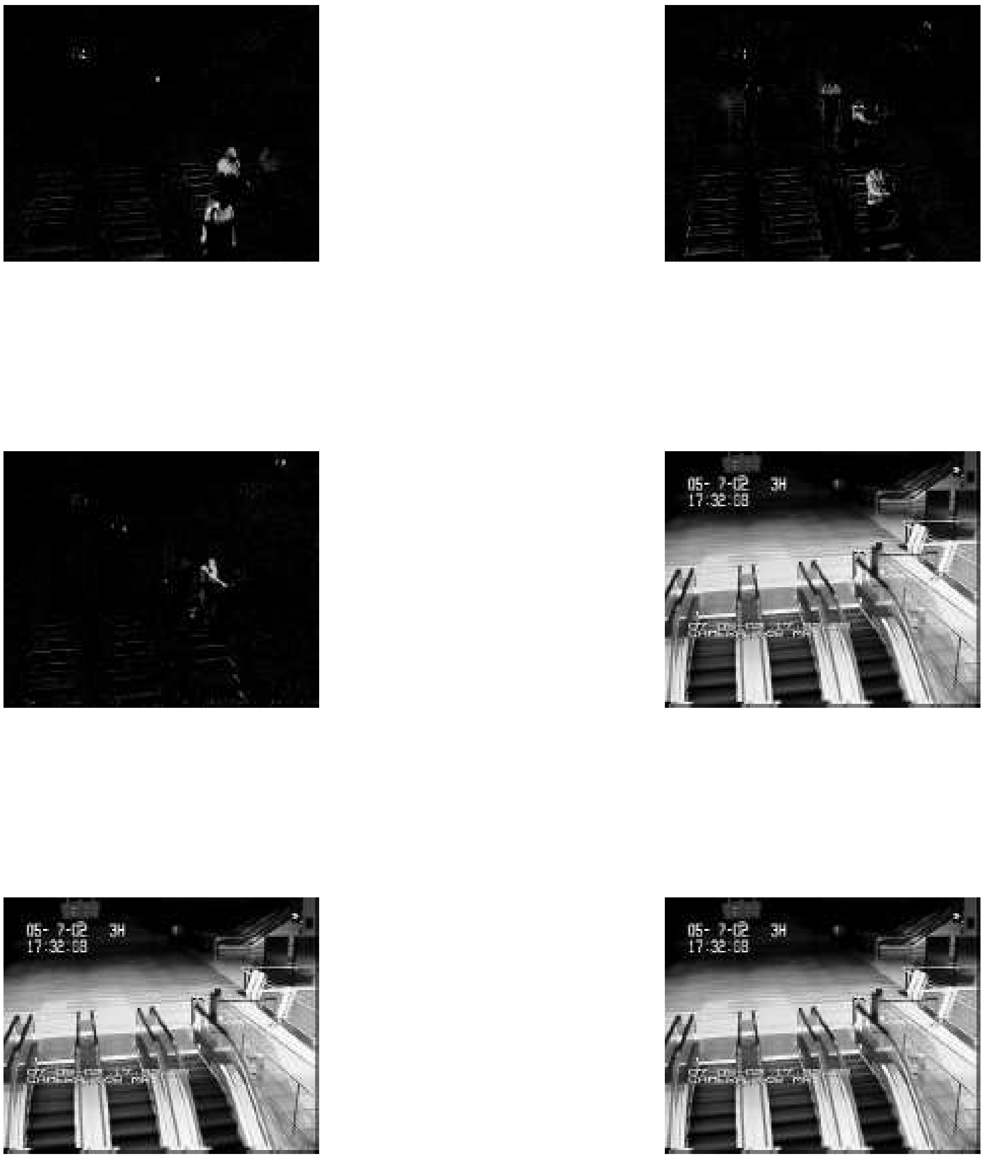

Figure 4: The image show of 3 columns of S and L

Appendix: Matlab code

PG

```
function [ W,H,Outs ] = Proj_grad( X, opts)
%
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The optimization problem is min_{W,H} 1/2 || WH' - X||^2
%                                  s.t. norm(Wj)<= 1; W \in R+, H \in R+
%parameters
% load('Swimmer.mat');
% X= reshape(Swimmer,[1024,256]);
maxit = 1000 ;
[m,n] = size(X); r =17;    eta=1;
% normalize and positive projection
nonnega_W = max(rand(m,r),0);
W0 = nonnega_W./max(nonnega_W,1);
H0 = max(rand(n,r),0);
if isfield(opts,'maxit') maxit = opts.maxit; end
if isfield(opts,'alpha') alpha = opts.alpha; end
if isfield(opts,'r') r = opts.r; end
if isfield(opts,'W0') W0 = opts.W0; end
if isfield(opts,'H0') H0 = opts.H0; end
% initialization
alpha_h =  1/(eta*norm(W0'*W0) );
alpha_w = 1/(eta*norm(H0'*H0) ) ;
W_hat = W0; H_hat = H0;
W=W0; H = H0;
grad_W = (W_hat*H_hat'-X)*H_hat;
grad_H = (W_hat*H_hat' -X)'*W_hat;
hist_f = zeros(maxit ,1);
start_time= tic;
k=1;
while (k< maxit+1)
     nonnega_W = max(0, W_hat - alpha_w * grad_W);
    W = nonnega_W./max(nonnega_W,1);
    H = max(0,H_hat - alpha_h * grad_H);
     f = 1/2 * norm(W*H' - X,'fro')^2  ;
    % save
     W_hat = W; H_hat = H; f_hat =f;
     hist_f(k) = f  ;
    % update gradient
     alpha_h = 1/(eta*norm(W_hat'*W_hat) );
     alpha_w = 1/(eta*norm(H_hat'*H_hat) );
     grad_W = (W_hat*H_hat'-X)*H_hat;
     grad_H = (W_hat*H_hat' -X)'*W_hat;
     if mod(k,300)==0
         fprintf('obj is %.4f\n',f);
     end
    k=k+1;
end
R_time = toc(start_time);
Outs.t = R_time; Outs.hist_f = hist_f;
Outs.k = k;
end
```

AliMin1

```
function [ W,H,Outs ] = Alt_min1_cvx( X, opts)
%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The optimization problem is min_{W,H} 1/2 || WH' - X||^2
%                                    s.t. norm(Wj)<= 1; W \in R+, H \in R+
% clc; clear ; close all;
% load('Swimmer.mat');
% X= reshape(Swimmer,[1024,256]);
%parameters
maxit = 100  ; r =17;
[m,n]=size(X);
W0= rand(m,r);H0=rand(n,r);
if isfield(opts,'maxit') maxit = opts.maxit; end
if isfield(opts,'r') r = opts.r; end
if isfield(opts,'W0') W0 = opts.W0; end
if isfield(opts,'H0') H0 = opts.H0; end
if isfield(opts, 'tol') tol = opts.tol; end
% initialization
W=W0; H = H0;
hist_f = [];
strat_time = tic;
for k = 1: maxit
    cvx_begin quiet
        if mod(k,2)==0
            variable W(m,r);
            nonnega_W = max(W,0);
            W = nonnega_W./max(nonnega_W,1);
        else
            variable H(n,r);
            H >= 0;
        end
        minimize(norm(W*H' - X,'fro'))   ;
    cvx_end
    f = 0.5*cvx_optval*cvx_optval;
    if ~isnan(cvx_optval)
        hist_f=[hist_f f];
    end
    if mod(k,2)==0
        fprintf('Iteration %d, obj is %.4f\n',k,f);
    end
end
R_time = toc(strat_time);
Outs.t = R_time; Outs.hist_f = hist_f;
Outs.k = k;
end
```

APG

```
function [ W,H,Outs ] = apg( X, opts)
%
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The optimization problem is min_{W,H} 1/2 || WH' - X||^2
%                            s.t. norm(Wj)<= 1; W \in R+, H \in R+
%parameters
% load('Swimmer.mat');
% X= reshape(Swimmer,[1024,256]);
maxit = 1000 ;    eta=1;
[m,n] = size(X); r =17;
% normalize and positive projection
nonnega_W = max(rand(m,r),0);
W0 = nonnega_W./max(nonnega_W,1);
H0 = max(rand(n,r),0);
if isfield(opts,'maxit') maxit = opts.maxit; end
if isfield(opts,'alpha') alpha = opts.alpha; end
if isfield(opts,'r') r = opts.r; end
if isfield(opts,'W0') W0 = opts.W0; end
if isfield(opts,'H0') H0 = opts.H0; end
% initialization
alpha_w = 1/(eta*norm(H0'*H0) ) ;
W_hat = W0; H_hat = H0;
W=W0; H = H0;
grad_W = (W_hat*H_hat'-X)*H_hat;
hist_f = zeros(maxit ,1);
start_time= tic;
k=1;
while (k< maxit+1)
    nonnega_W = max(0, W_hat - alpha_w * grad_W);
    W = nonnega_W./max(nonnega_W,1);
    grad_H = (W *H_hat' -X)'*W ;
    alpha_h = 1/(eta*norm(W'*W ) );
    H = max(0,H_hat - alpha_h * grad_H);
    f = 1/2 * norm(W*H' - X,'fro')^2  ;
    % save
    W_hat = W; H_hat = H; f_hat =f;
    hist_f(k) = f  ;
    % update gradient
    alpha_w = 1/(eta*norm(H_hat'*H_hat) );
    grad_W = (W_hat*H_hat'-X)*H_hat;
    if mod(k,300)==0
        fprintf('obj is %.4f\n',f);
    end
    k=k+1;
end
R_time = toc(start_time);
Outs.t = R_time; Outs.hist_f = hist_f;
Outs.k = k;
end
```

PGD

```
function [ L,S,Outs ] = q2_pgd2(X,opts)
%%%%%%%%%%%%%%%%% Robust PCA
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The optimization problem is min_{L,S} || L ||_* + lambda || S ||_1 +
    beta/2 || L+S - X||^2
maxit = 100  ; alpha = 0.5;
[m,n] = size(X);
lambda = 1/sqrt(m);beta=1/norm(X);  beta_max = 1e5;
L0= zeros(m,n);S0=zeros(m,n);
if isfield(opts,'maxit') maxit = opts.maxit; end
if isfield(opts,'alpha') alpha = opts.alpha; end
if isfield(opts,'beta') beta = opts.beta; end
if isfield(opts,'beta_max') beta_max = opts.beta_max; end
if isfield(opts,'lambda') lambda = opts.lambda; end
if isfield(opts,'L0') L0 = opts.L0; end
if isfield(opts,'S0') S0 = opts.S0; end
% initialization
L_hat = L0; S_hat = S0;
L=L0; S = S0;
grad =  (L_hat+S_hat-X);
% f_hat =   0.5* norm(L+S-X, 'fro')^2 ;
hist_f = zeros(maxit,1);
start_time  = tic;
for k =1:maxit
    Y  = L_hat - alpha*grad;
    Z  = S_hat - alpha*grad ;
     [U,D,V] = svd(Y,'econ');
    S = sign(Z).* max(abs(Z)-alpha*lambda/beta  ,0);
    L = U * diag(max(diag(D)-alpha /beta ,0))*V';
    f =  0.5* norm(L+S-X, 'fro')^2 ;
    % back tracking to update the alpha trace(grad'*(L-L_hat)) +trace(
        grad'*(S-S_hat)) +
%     while alpha > min_alpha && f>f_hat - 2*alpha * norm(grad,'fro')^2/
    norm(grad)% +1/(2*alpha)*(norm(L-Y,'fro')^2 + norm(S-Z,'fro')^2)
%         alpha =  max(eta * alpha, min_alpha) ;
%         Y  = L_hat - alpha*grad;
%           Z  = S_hat - alpha*grad ;
%           [U,D,V] = svd(Y,'econ');
%           S = sign(Z).* max(abs(Z)-alpha*lambda/beta ,0);
%           L = U * diag(max(diag(D)-alpha/beta ,0))*V';
%           f =  0.5* norm(L+S-X, 'fro')^2 ;
%      end
     hist_f(k)= f +lambda  * norm(S,1)/beta+ trace(sqrt(L'*L))/beta   ;
     L_hat = L; S_hat = S; %f_hat =f;
     % update gradient
     grad =   (L_hat+S_hat-X);
     rankL  = sum(diag(D)>alpha/beta);
     cardS = sum(sum(double(abs(S)>0)));
     if mod(k,10 )==0
         fprintf('obj is %.6e\n',hist_f(k));
         fprintf('The rank of L is %d \n', rankL );
         fprintf('The ||S||_0 is %d\n', cardS);
     end
     beta = min(beta * 1.1, beta_max);
end
R_time = toc(start_time);
Outs.t = R_time; Outs.hist_f = hist_f;
Outs.k = k; Outs.alpha = alpha;
end
```

AltMin2

```
% function [ W,H,Outs ] = Alt_min2 ( X, opts)
%
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%The optimization problem is min_{L,S} || L ||_* + \lambda ||S||_1 + %
    beta/2 ||L+S-X||^2
%parameters
         [m,n] = size(X);
         % initialization
         L0= zeros(m,n); S0=zeros(m,n);
         L=L0;   S = S0;
         L_hat = L0; S_hat = S0;
         opts.maxit = 200;    opts.alpha =0.5;
         opts.lambda = 0.005; opts.beta =0.01  ; opts.beta_max =1e5;
         if isfield(opts,'maxit') maxit = opts.maxit; end
         if isfield(opts,'lambda') lambda = opts.lambda; end
         if isfield(opts,'beta') beta = opts.beta; end
         if isfield(opts,'r') r = opts.r; end
         if isfield(opts,'W0') W0 = opts.W0; end
         if isfield(opts,'H0') H0 = opts.H0; end
         if isfield(opts, 'tol') tol = opts.tol; end
         % initialization
         hist_f = zeros(opts.maxit,1);
```

```
        strat_time = tic;
        for k=1:opts.maxit
                [  S,OutS ] = S_update1(X,L_hat,S_hat ,opts);
                [ L ,OutL ] = L_update1(X,L_hat,S,opts  );
                L_hat =L; S_hat =S;
                hist_f(k) =  0.5*norm(L+S-X, 'fro')^2 +opts.lambda *
                   norm(S,1)/opts.beta+ trace(sqrt(L'*L))/opts.beta  ;
                if mod(k,20)==0
                    fprintf('obj is %.4f\n',hist_f(k));
                end
                opts.beta = opts.beta*1.1;
        end
        R_time = toc(strat_time);
        Outs.t = R_time; Outs.hist_f = hist_f;
        Outs.k = k;
end
function [  S,Outs ] = S_update1(X,L,S ,opts)
maxit = 100  ;
[m,n] = size(X);    tol=1e-6;
lambda = 0.005;beta=1000;
if isfield(opts,'maxit') maxit = opts.maxit; end
if isfield(opts,'beta') beta = opts.beta; end
if isfield(opts,'eta') eta = opts.eta; end
if isfield(opts,'alpha') alpha = opts.alpha; end
if isfield(opts,'lambda') lambda = opts.lambda; end
if isfield(opts,'tol') tol = opts.tol; end
% initialization
S_hat = S;
L_hat = L;
grad =  (L +S_hat-X);
% grad =  (L +S -X);
f_hat =   0.5 * norm(L+S-X, 'fro')^2 ;%+lambda * norm(S,1)   ;
f_gap =Inf;
hist_f = zeros(maxit,1);
start_time  = tic;
k=1;
while (k < maxit) && (f_gap > tol)
    Z  = S_hat - alpha*grad ;
    S = sign(Z).* max(abs(Z)-alpha*lambda/beta  ,0);
    f =  0.5* norm(L+S-X, 'fro')^2 ;
    hist_f(k)= f + +lambda  * norm(S,1)/beta ;
    if k > 1
    f_gap = abs(hist_f(k)-hist_f(k-1));
    end
    S_hat = S;  f_hat = f;
    grad =  (L +S_hat-X);
    if mod(k,5)==0
        fprintf('Iteration %d of updating S, the objective is %.6f \n',
           k,hist_f(k));
    end
    k=k+1;
end
fprintf('S updates %d times \n', k);
R_time = toc(start_time);
Outs.t = R_time; Outs.hist_f = hist_f;
Outs.k = k;
end
function [ L,Outs] = L_update1(X,L,S,opts)
maxit = 100   ;     tol=1e-6;
[m,n] = size(X); beta=1000;
if isfield(opts,'maxit') maxit = opts.maxit; end
if isfield(opts,'beta') beta = opts.beta; end
if isfield(opts,'eta') eta = opts.eta; end
if isfield(opts,'alpha') alpha = opts.alpha; end
if isfield(opts,'tol') tol = opts.tol; end
% initialization
L_hat = L ;  S_hat =S;
grad =  (L_hat+S -X);
f_gap=Inf;
hist_f = zeros(maxit,1);
start_time  = tic;
k=1;
while k<( maxit) && (f_gap > tol)
    Y  = L_hat - alpha*grad;
    [U,D,V] = svd(Y,'econ');
    L = U * diag(max(diag(D)-alpha /beta ,0))*V';
    f =  0.5* norm(L+S-X, 'fro')^2 ;
    hist_f(k)=f+ trace(sqrt(L'*L))/beta    ;
    if k > 1
    f_gap = abs(hist_f(k)-hist_f(k-1));
    end
    L_hat = L;
    grad =   (L_hat+S -X);
    if mod(k,5)==0
        fprintf('Iteration %d of updating L, the objective is %.6f \n',
           k,hist_f(k));
    end
    k=k+1;
end
fprintf('L updates %d times \n', k);
R_time = toc(start_time);
Outs.t = R_time; Outs.hist_f = hist_f;
Outs.k = k;
end
```