SI231 - Matrix Computations, Fall 2020-21

Solution of Homework Set #5

Prof. Yue Qiu and Prof. Ziping Zhao

Name: aaa **Major:** Master in CS

Student No.: 2018123456 **E-mail:** aaa@shanghaitech.edu.cn

Study Guide

This homework concerns the following topics:

- **Regularization**, see Lecture 8 of Zhao, Lecture 20 of Qiu.
- **SVD computation**, see Lecture 7 of Zhao, Lecture 18 of Qiu.
- **Iterative methods**, see Lecture 2 of Zhao, Lecture 21 of Qiu.
- **Application of SVD, PCA**, see Lecture 7 of Zhao, Lecture 19 of Qiu.

I. Least square with regularization

Problem 1. (8 points + 8 points + 4 points) This problem is graded by Bing Jiang **(jiangbing@)**.

In this problem, we will learn how to solve LS when there is noise. for some $\lambda > 0$, where the term $\lambda||\mathbf{x}||_k^k$ is added to improve the system conditioning, thereby attempting to reduce noise sensitivity. Usually, we set $k = 1$ which is called $\ell 1$ regularization, or set $k = 2$ which is called $\ell 2$ regularization.

$\ell 1$ regularized LS:

$$\mathbf{x_{LS}} = \min_{\mathbf{X}\in\mathbb{R}^n} f(\mathbf{x}), \quad f(\mathbf{x}) = ||\mathbf{A}\mathbf{x} - \mathbf{y}||_2^2 + \lambda||\mathbf{x}||_1 \tag{1}$$

$\ell 2$ regularized LS:

$$\mathbf{x_{LS}} = \min_{\mathbf{X}\in\mathbb{R}^n} f(\mathbf{x}), \quad f(\mathbf{x}) = ||\mathbf{A}\mathbf{x} - \mathbf{y}||_2^2 + \lambda||\mathbf{x}||_2^2 \tag{2}$$

Write programs to solve the regularized least square problem, any programming language is suitable. where $\mathbf{A} \in \mathbb{R}^{m\times n}$ is a matrix representing the predefined data set with m data samples of n dimensions (m=1000, n=210), and $\mathbf{y} \in \mathbb{R}^m$ represents the labels. The data samples are provided in the "data.txt" file, and the labels are provided in the "label.txt" file, you are supposed to load the data before solving the problem. In this problem, we set $\lambda = 0.1$ for two algorithms.

1) (**8 points**) Solve $\ell 1$ regularized LS using Majorization-Minimization method, we set $c = 1e{+}5$ in this problem. The Majorization-Minimization method for solving problem updates $\mathbf{x}$ as

$$\mathbf{x^{(k+1)}} = \mathbf{soft}(\frac{1}{c}\mathbf{A^T}(\mathbf{y} - \mathbf{A}\mathbf{x^{(k)}}) + \mathbf{x^{(k)}}, \lambda/c),$$

where **soft** is called the soft-thresholding operator and is defined as follows: if $\mathbf{z} = \mathbf{soft}(\mathbf{x}, \sigma)$, then $z_i = sign(x_i)\max\{|x_i| - \sigma, 0\}$.

2) (**8 points**) Solve $\ell 2$ regularized LS using gradient descent method. The gradient descent method for solving problem updates $\mathbf{x}$ as

$$\mathbf{x} = \mathbf{x} - \gamma \cdot \nabla_{\mathbf{x}} f(\mathbf{x}),$$

where γ is the step size of the gradient decent methods. We suggest that you can set $\gamma = 1e - 5$.

3) (**4 points**) Compare two methods above.

   (a) compare L0 norm (the number of non-zero elements) of $\mathbf{x_{LS}}$ computed by above two algorithms;

   (b) Compare the loss $||\mathbf{A}\mathbf{x} - \mathbf{y}||_2^2$ for results $\mathbf{x} = \mathbf{x_{LS}}$ of above two algorithms.

Remarks:

- The solution of the two methods should be printed in files named "sol1.txt" and "sol2.txt" and submitted in gradescope. The format should be same as the input file (210 rows plain text, each rows is a dimension of the final solution).
- Make sure that your codes are executable and are consistent with your solutions.

Solution.

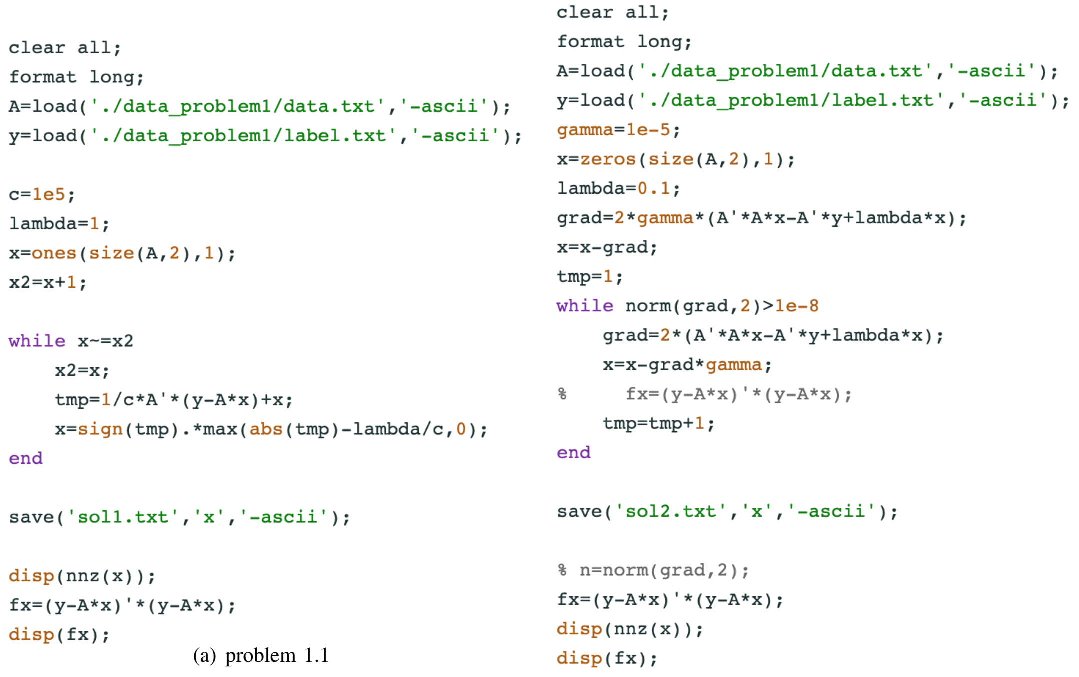
1) In this problem, you should solve ℓ1 regularized LS using Majorization-Minimization method, the iteration step is shown in the problem. In programming part, the iteration step is 4 points and soft function part is 4 points.

2) In this problem, you should solve ℓ2 regularized LS using gradient decent method. The terminal condition is an important point. You should make sure that the gradient is close to zero when iteration stops. To confirm the gradient is close to zero, you should make sure the norm of gradient is close to zero, and it should be the terminal condition in your code. in programming part, the iteration step is 4 points and the terminal condition is 4 points.

3) The difference between computed loss and reference loss lower than 10 is OK. 1 point for each result.

For ℓ2 regularized LS, loss: 2946.355819, L0 norm: 210.000000000

For ℓ1 regularized LS, loss: 2945.259669, L0 norm: 210.000000000



```
clear all;
format long;
A=load('./data_problem1/data.txt','-ascii');
y=load('./data_problem1/label.txt','-ascii');

c=1e5;
lambda=1;
x=ones(size(A,2),1);
x2=x+1;

while x~=x2
    x2=x;
    tmp=1/c*A'*(y-A*x)+x;
    x=sign(tmp).*max(abs(tmp)-lambda/c,0);
end

save('sol1.txt','x','-ascii');

disp(nnz(x));
fx=(y-A*x)'*(y-A*x);
disp(fx);
```

(a) problem 1.1

```
clear all;
format long;
A=load('./data_problem1/data.txt','-ascii');
y=load('./data_problem1/label.txt','-ascii');
gamma=1e-5;
x=zeros(size(A,2),1);
lambda=0.1;
grad=2*gamma*(A'*A*x-A'*y+lambda*x);
x=x-grad;
tmp=1;
while norm(grad,2)>1e-8
    grad=2*(A'*A*x-A'*y+lambda*x);
    x=x-grad*gamma;
%     fx=(y-A*x)'*(y-A*x);
    tmp=tmp+1;
end

save('sol2.txt','x','-ascii');

% n=norm(grad,2);
fx=(y-A*x)'*(y-A*x);
disp(nnz(x));
disp(fx);
```

(b) problem 1.2

Figure 1: The Matlab code for this problem

II. Computations of SVD

Problem 2. (4 points + 8 points + 8 points) This problem is graded by Zhicheng Wang **(wangzhch1@)**. In this problem you will see singular value stability and discover computation of SVD.

1) (**4 points**) Consider a 4×4 matrix

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \\ \frac{1}{6000} & 0 & 0 & 0 \end{bmatrix},$$

find its singular values and eigenvalues. In this sub-problem you can use eigenvalue decomposition ('eig') and singular value decomposition function ('svd') in Matlab or Python.

2) (**8 points**) Compute the SVD decomposition of a 3×2 matrix by Algorithm 1

$$\mathbf{B} = \begin{bmatrix} 3 & 2 \\ 1 & 4 \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}.$$

Are the results right? If not, can you get the correct answer through the results of Algorithm 1? The In this sub-problem you can use the eigenvalue decomposition ('eig') and QR decomposition function ('qr') in Matlab or Python.

3) (**8 points**) Consider an $m \times m$ upper triangular matrix with 0.1 on the main diagonal and 1 everywhere above the diagonal. Find the smallest singular value through the singular value decomposition function ('svd') and algorithm 1. You are required to plot two curves on a log scale for $m = 1, \dots, 30$ and show which one is the desired result.

```
Algorithm 1: SVD Decomposition by $\mathbf{A}^T\mathbf{A}$
Input  : Thin matrix A
Form $\mathbf{A}^T\mathbf{A}$.
Compute the eigenvalue decomposition $\mathbf{A}^T\mathbf{A}$=$\mathbf{V}^T\mathbf{\Lambda}\mathbf{V}$.
Let $\mathbf{\Sigma}$ be an m×n diagonal matrix with diagonal entries being the nonnegative square root of diagonal
  entries of $\mathbf{\Lambda}$.
Solve the system $\mathbf{U\Sigma} = \mathbf{AV}$for orthogonal matrix $\mathbf{U}$ (e.g., via QR factorization).
Output: $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$.
```

Remarks

1) Singular values cannot be negative.
2) Please show your result and insert figures in your PDF.

Solution.

1) Singular value is $\mathbf{\Sigma} = \begin{bmatrix} 3 \\ 2 \\ 1 \\ 1.67e-4 \end{bmatrix}$, and eigenvalue is $\lambda = \frac{1}{10}\begin{bmatrix} -1.78 \\ 1.78i \\ -1.78i \\ 1.78 \end{bmatrix}$.

2) Following the steps in Algorithm 1, we have, $\mathbf{U} = \begin{bmatrix} -0.7570 & -0.6364 & -0.1482 \\ 0.6484 & -0.7597 & 0.0494 \\ -0.0811 & -0.1335 & 0.9877 \end{bmatrix}$, $\mathbf{\Sigma} = \begin{bmatrix} 1.9610 & 0 \\ 0 & -5.1628 \\ 0 & 0 \end{bmatrix}$, $\mathbf{V} = \begin{bmatrix} -0.8481 & 0.5299 \\ 0.5299 & 0.8481 \end{bmatrix}$, and $\mathbf{U\Sigma} = \mathbf{AV}$. Since singular values cannot be negative, we rewrite $\mathbf{\Sigma}' = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{\Sigma} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 5.1628 & 0 \\ 0 & 1.9610 \\ 0 & 0 \end{bmatrix}$, $\mathbf{U}' = \mathbf{U} \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}^T = \begin{bmatrix} -0.6364 & -0.7570 & -0.1482 \\ -0.7597 & 0.6484 & -0.0494 \\ -0.13351 & -0.0811 & 0.9877 \end{bmatrix}$, and $\mathbf{V}' = \mathbf{V} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} -0.5299 & -0.8481 \\ -0.8481 & 0.5299 \end{bmatrix}$.

3) Figure as follow

Remarks: This problem contains 3 sub-problems.

1) In sub-problem 1), you are required to find singular values and eigenvalues of $\mathbf{A}$. Each result takes 2 points.

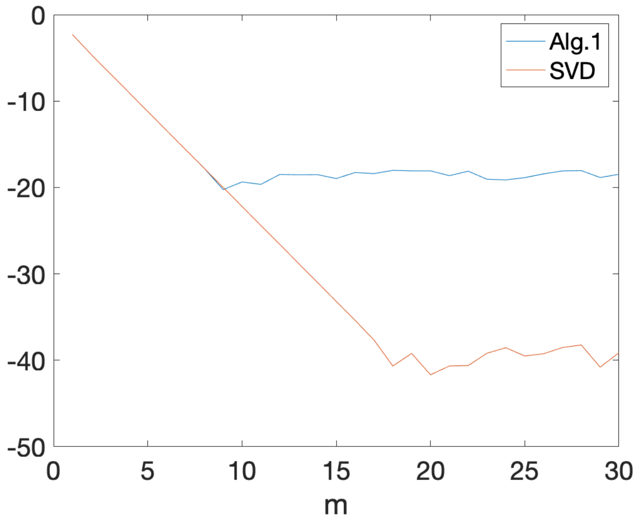


Figure 2: The solution to problem 2.3

2) In sub-problem 2), you are required to give result of Algorithm 1 and correct it. The result of Algorithm 1 takes 5 points, all singular values is negative takes 2 points and sorted singular values takes 1 point.
3) In sub-problem 3), you are required to give two results of Algorithm 1 and SVD function. Each result takes 4 points.

Grading policy:

1) In sub-problem 2), if $\mathbf{U}$ is not orthogonal, you will lose 2 point.
2) In sub-problem 2), if $\mathbf{U\Sigma V} \neq \boldsymbol{B}$, you will lose 2 point. And If it happens during correction, you will lose all points of correction.
3) In sub-problem 3), if forming matrix is wrong and you algorithm is right, you will lose 2 points. And if you not plot result with log scale, you will lose 1 point.

The Matlab code for solving this problem is given as follows:

```matlab
%%%%% Solution to Problem 5 %%%%%
%% load the data
clc;
clear;

%% Sub-problem 1

A = zeros(4);
A(1,2) = 1; A(2,3) = 2;
A(3,4) = 3; A(4,1)=1/6000;
svd(A)
eig(A)


%% Sub-problem 2
```

```

B = [3 2;1 4;0.5 0.5];
[v,lambda] =eig(B’*B);
[u,sigma] = qr(B*v);


%% Sub-problem 3

singular_min = zeros(1,30);
eig_min =zeros(1,30);

for m=1:30
    a=eye(m)*0.1;
    for i=1:m
        for j=i+1:m
            a(i,j)=1;
        end
    end
    a=a’;
    [v,t]=eig(a’*a);
    eig_min(m)=min(abs(sqrt(diag(t))));
    [u,sigma]=qr(a*v);
    singular_min(m)=min(svd(a));
end

m=1:30;
figure;semilogy(m,eig_min,m,singular_min);
legend(’Alg.1’,’SVD’);
```

III. Iterative methods.

Problem 3. (8 points + 8 points + 4 points) This problem is graded by Chenguang Zhang **(zhangchg@)**. In this problem, you will learn how to implement there iterative methods, Jacobi, Gauss-Seidel and SOR Methods to solve the linear equation $\mathbf{b} = \mathbf{A}\mathbf{x}$, and compare the convergence of these methods. "A1.txt", "A2.txt","b1.txt", "b2.txt","x1.txt", "x2.txt" is the data of the $\mathbf{b}, \mathbf{x}, \mathbf{A}$ with different size (10 and 1000 respectively).

Algorithm 2: Iterations method

```
Input: A starting point $\mathbf{x}^0$, maximum iterations $N$, error bound eps
Output: Solution $\mathbf{x}$ of $\mathbf{A}\mathbf{x} = \mathbf{b}$
for $k = 0, 1, 2, \ldots, N$ do
    update $\mathbf{x}^k$ to obtain $\mathbf{x}^{k+1}$ ;        // Use one of the three iteration methods
    if $\|\mathbf{x}^{k+1} - \mathbf{x}^k\|_2 < \text{eps}$ then
        break;
    end
end
```

Suppose $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{A} = [a_{ij}] \in \mathbb{R}^{n\times n}$ and $\mathbf{b} = (b_i) \in \mathbb{R}^n$.

Each iteration method uses different updates:

For Jacobi Iteration,

$$x_i^{k+1} = (b_i - \sum_{i \neq j} a_{ij} x_j^k)/a_{ii} \quad \text{for} \quad i = 1, \ldots, n,$$

For Gauss-Seidel Iteration,

$$x_i^{k+1} = (b_i - \sum_{j>i} a_{ij} x_j^{k+1} - \sum_{j<i} a_{ij} x_j^k)/a_{ii}, \quad \text{for} \quad i = 1, \ldots, n,$$

For SOR Iteration with relaxation factor ω,

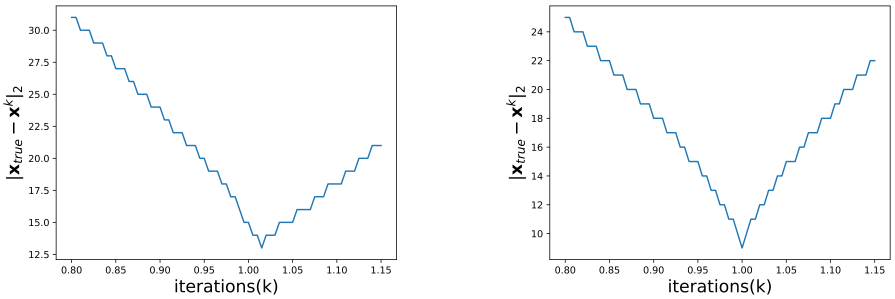
$$x_i^{k+1} = (1-\omega)x_i^k + \omega \left(b_i - \sum_{j>i} a_{ij} x_j^{k+1} - \sum_{j<i} a_{ij} x_j^k \right) /a_{ii}, \quad \text{for} \quad i = 1, \ldots, n.$$

Remarks:

1) (**8 points**) Implement the Jacobi Iteration and Gauss-Seidel Iteration methods.
2) (**8 points**) Implement the SOR Iteration method, tune the relaxation factor ω to achieve high convergence rate and plot the curve of number of iterations v.s. the value of ω.
3) (**4 points**) Plot the error $\|\mathbf{x}_{\text{true}} - \mathbf{x}^k\|_2$ for $k = 1, 2, \ldots, N$, analyze the convergence rate and the final error of three methods.

Note: The value of the “eps” in Terminal condition should not be too large!

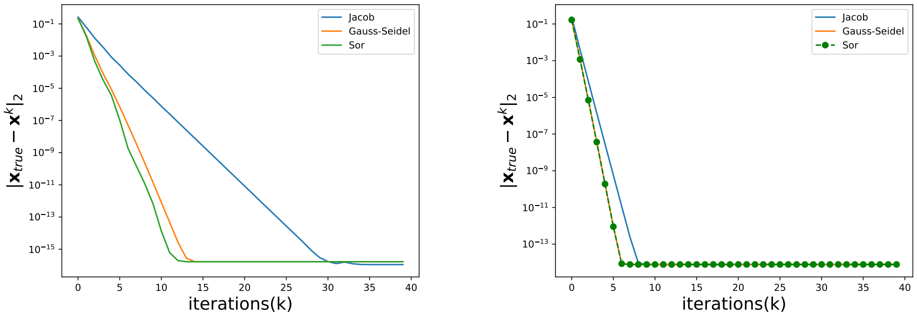
Solution.



(a) data1

(b) data2

Figure 3: iterations v.s. the value of ω



(a) data1

(b) data2

Figure 4: convergence analysis iterations v.s. $\|\mathbf{x}_{\text{true}} - \mathbf{x}^k\|_2$

1) Hand in the code of two method that can obtain the correct results within a reasonable number of iterations.
2) Hand in the code that can obtain the correct results within a reasonable number of iterations (about 10$\sim$ 30). Give out the value of ω and corresponding iteration numbers.
3) As can be seen the SOR iteration method shows fastest convergence rate and the Jacobi Iteration shows worst convergence rate. One of the interesting things is that when an algorithm iterates to a certain number of steps, the solution can not improve more. Because there is a numerical limit for iteration methods, so if we continue to iterate, the solution will not change, and there will always be an order of magnitude (E-15) distance between the solution and the true solution.

IV. Application of SVD

Problem 4. (4 points + 8 points + 8 points) This problem is graded by Xinyue Zhang (**zhangxy11@**).

In this problem you will see one of the application of SVD, what is doing face recognition on Yale B dataset. Here we will only use 4 individuals under 15 relatively brighter different illumination conditions.

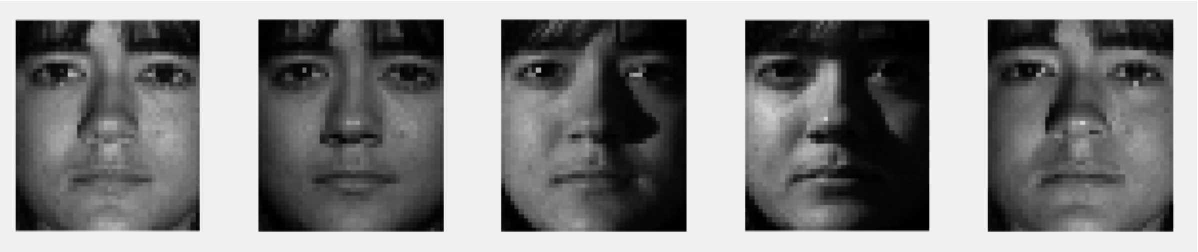


Figure 5: Face images of one individual under 5 different illumination conditions in the extended YaleB dataset. All images are frontal faces.

Face recognition is an area of computer vision in which low-dimensional linear models such as principal component analysis (PCA) and its variations have been popular tools for capturing the variability of face images. And it has been shown that PCA can be solved by SVD optimally. In the following questions you will see how Algorithm 3 can be used.

```
Algorithm 3: [$\boldsymbol{\mu}, \mathbf{U}, \mathbf{Y}$] = PCA_via_SVD($\mathbf{X}, d$)
Parameters:
$\mathbf{X}$:    $D \times N$ data matrix.
$d$:    Number of principal components.
Returned values:
$\boldsymbol{\mu}$:    Mean of the data.
$\mathbf{U}$:    Orthonormal basis for the subspace.
$\mathbf{Y}$:    Low-dimensional representation( or principal components).
Description:
Compute the SVD of the data matrix $\mathbf{X} - \boldsymbol{\mu}\mathbf{1}^T = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T$.
Sort the singular values in descending order, choose the first $d$ singular values and the corresponding $d$ left
  singular vectors $\mathbf{U}_d$.
Find the $d$ principal components $\mathbf{Y} = \mathbf{U}_d^T(\mathbf{X} - \boldsymbol{\mu}\mathbf{1}^T)$
```

The principal bases $\mathbf{U}$ estimated by PCA are also known as the *eigenfaces* in the computer vision literature. The first eigenface is the left singular vector corresponding to the largest singular value, the second eigenface is the left singular vector corresponding to the second largest singular value and so on.

1) (**4 points**) Apply PCA in Algorithm 3 with $d = 10$ to individual 4[1]. Plot the mean face $\boldsymbol{\mu}$, the first eigenface,

[1] images or the mat file in data1 folder

the second eigenface, the third eigenface and the tenth eigenface. Describe what you have observed.

2) (**8 points**) Apply PCA in Algorithm 3 with $d = 10$ to the Training Set.

   a) (**2 points**) Plot the mean face, the first eigenface, the second eigenface, the third eigenface and the tenth eigenface.

   b) (**2 points**) Plot the sorted singular values.

   c) (**4 points**) Project the Test Set onto the face subspace given by PCA, that is $\mathbf{Y}_{test} = \mathbf{U}_d^T(\mathbf{X}_{test} - \boldsymbol{\mu}\mathbf{1}^T)$. Compute the projected faces, that is $\text{Proj}(\mathbf{X}_{test}) = \boldsymbol{\mu}\mathbf{1}^T + \mathbf{U}_d\mathbf{Y}_{test}$. Then choose one projected face for each individual and plot them. And show those projected faces for $d = 2$ again.

3) (**8 points**) Classify these test faces using 1-nearest-neighbor, that is, label an image x as corresponding to individual i if its projected image y is closest to one of projected training image y_j of individual i[2]. Report the percentage of incorrectly classified face images for $d = 2, ..., 8$ and put them into the following table. Which value of d gives the best recognition performance?

Table I: Summary of errors

# eigenvectors	2	3	4	5	6	7	8
error							

Remarks:

1) Data are provided in two forms: images and mat data. You only need to use one of them.
2) 10 images for each individuals in the training data, and 5 images for each individuals in the test data. All images are of the same size 48×42.
3) In mat file **all_data.mat**, **data_train** and **data_test** are the matrices of training data and test data where each column is the column stack of an image[3], **Y_label_train** and **Y_label_test** are ground truth labels.
4) Plot the mean face or eigenface means that you should reshape the vector to the size of image and show it.
5) Recommend to use truncated(thin) SVD for fast implementation.
6) Please **insert your figures in your PDF**.

Solution.

3) The errors is as follows and d greater or equal to 7 give the best recognition performance.

Table II: Summary of errors

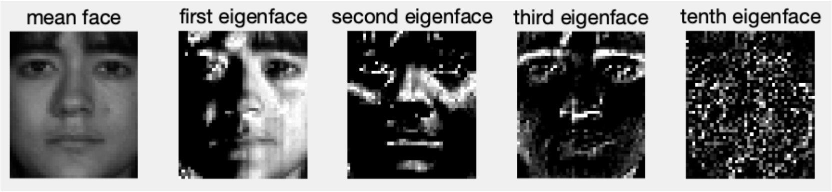
# eigenvectors	2	3	4	5	6	7	8
error	0.55	0.35	0.30	0.25	0.10	0	0

The Matlab code for solving this problem is given as follows:

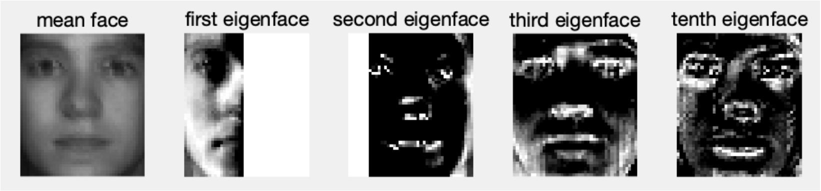
```matlab
%% 1)
load('data1/data1.mat');
```

[2] we use ℓ_2 norm as the distance measurement: $\|y - y_j\|_2$

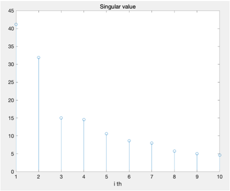
[3] the column stack of an image matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ is denoted by $x = [X_{11}, X_{21}, X_{31}, ..., X_{mn}]^T \in \mathbb{R}^{mn}$



(a) 1)

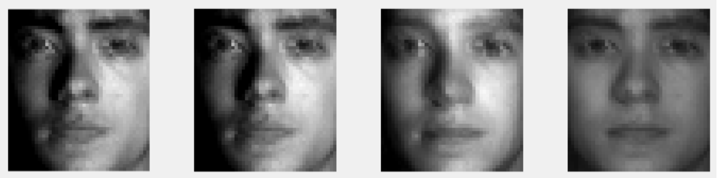


(b) 2)a



(c) 2)b

(d) 2)c $d = 10$



(e) 2)c $d = 2$

```
mu1 = mean(data1,2);
mean_face1 = reshape(mu1,192/4,168/4);
% figure;imshow(mean_face1);

[U1,S1,~] = svd(data1-mu1,'econ');
d = 10;
U1 = U1(:,1:d);

plot_option = 1;
if plot_option == 1
    id_list = [1,2,3,10];
    figure_name = {'first eigenface','second eigenface','third eigenface','tenth eigenface'};
    figure;
    subplot(1,5,1)
    imshow(mean_face1)
    title('mean face')
    for id = 1:length(id_list)
        temp = U1(:,id_list(id));
        subplot(1,5,id+1);
        show_img_vec(temp,192/4,168/4,1)
        title(figure_name{id})
    end
end

%% 2)
load(’all_data.mat’)
% a)
mu = mean(data_train,2);
mean_face = reshape(mu,192/4,168/4,1);
% figure;imshow(mean_face1);

[U,S,~] = svd(data_train-mu,’econ’);
d = 10;
U = U(:,1:d);

plot_option = 1;
if plot_option == 1
    id_list = [1,2,3,10];
    figure_name = {’first eigenface’,’second eigenface’,’third eigenface’,’tenth eigenface’};
    figure;
    subplot(1,5,1)
    imshow(mean_face)
    title(’mean face’)
    for id = 1:length(id_list)
        temp = U(:,id_list(id));
        subplot(1,5,id+1);
        show_img_vec(temp,192/4,168/4,1)
        title(figure_name{id})
    end
end

% b)
S = diag(S);
figure;
stem(S(1:10))
xlabel(’i th’)
title(’Singular value’)

% c)
Y_test = U’*(data_test-mu);
Proj_X_test = mu*ones(1,size(data_test,2)) + U*Y_test;

plot_option = 1;
```

```
if plot_option == 1
    figure;
    id_list = [4,9,14,20];
    for id = 1:4
        subplot(1,4,id);
        show_img_vec(Proj_X_test(:,id_list(id)),192/4,168/4,0)
    end
end
title(’d = 10’)

U2 = U(:,1:2);
Y_test2 = U2’*(data_test-mu);
Proj_X_test2 = mu*ones(1,size(data_test,2)) + U2*Y_test2;

plot_option = 1;
if plot_option == 1
    figure;
    id_list = [4,9,14,20];
    for id = 1:4
        subplot(1,4,id);
        show_img_vec(Proj_X_test2(:,id_list(id)),192/4,168/4,0)
    end
end
title(’d = 2’)

%% 3) classification
for d = 2:8
    Y_train = U(:,1:d)’*(data_train-mu);
    Y_test = U(:,1:d)’*(data_test-mu);
    Y_label_estimated = [];
    for i = 1:size(data_test,2)
        temp = Y_train - Y_test(:,i);
        temp_obj = vecnorm(temp,2,1);
        [ia,ib] = min(temp_obj);
        Y_label_estimated(i) = Y_label_train(ib);
    end

    fp_id = find(Y_label_estimated ~= Y_label_test);
    error(d-1) = length(fp_id)/size(data_test,2);
end

Table = [’d’,num2cell(2:8);’error’,num2cell(error)]

%% useful function
function show_img_vec(x,ia,ib,rescale)
x = reshape(x,ia,ib);
if rescale == 1
```

```
    x = x./max(x);
end
% figure;
imshow(x)
end
```

Grading policy:

1) Eigenface images should contain less face features as the singular values get smaller. At least the images should not be too black to observe anything. In sub-problem 1) each face image is 0.8 point. In sub-problem 2)a) each face image is 0.4 point.
2) In sub-problem 2)c), When $d = 2$ projected face for different individuals should be very similar since too much information are thrown away, but when it comes to $d = 10$ it is enough to distinguish different individuals. In particular, it has been shown that under certain idealized circumstances (such as Lambertian reflectance), images of the same face under varying illumination lie near an approximately nine-dimensional linear subspace known as the harmonic plane[4].
3) In sub-problem 3) every error takes 1 point, the observation takes 1 point.

[4] Basri, R., Jacobs, D. (2003). Lambertian reflection and linear subspaces. IEEE Transactions on Pattern Analysis and Machine Intelligence, 25(2), 218â233.

Problem 4. (8 points + 12 points) This problem is graded by Song Mao (**maosong@**).

In this problem, we will learn how to use SVD to compress, or reduce the dimension of the data. The real data is with low rank internal, but often corrupted, or contaminated by noise, which leads to the full rank of the data matrix $\mathbf{A} \in \mathbb{R}^{D\times N}$. To reduce the storage consumption of the real data (note that D is extremely large in practical), we seek to find a low rank approximation to $\mathbf{A}$, that is, we want to solve the problem

$$\min_{\mathbf{X}\in\mathbb{R}^{D\times N}} \quad \|\mathbf{X}-\mathbf{A}\|_F^2 \tag{3}$$

$$\text{s.t.} \quad \text{rank}(\mathbf{X}) \leq d \tag{4}$$

where $\|\cdot\|_F$ is the Frobenius norm, $d << D$ is unknown. By the Theorem of PCA via SVD, the above problem is equivalent to the following problem (suppose the eigenvalues of $\mathbf{A}$ are given by $\sigma_1(\mathbf{A}) \geq \sigma_2(\mathbf{A}) \geq \cdots \geq \sigma_K(\mathbf{A}) \geq 0$ and $K = \min\{D, N\}$):

$$\min_{d=1,\ldots,K} J(d) = \sum_{i=d+1}^{K} \sigma_i^2(\mathbf{A}) \tag{5}$$

However, this is not a good criterion, since the optimal solution is given by $d^* = \text{rank}(\mathbf{A})$ ($J(d^*) = 0$).

The problem of determining the optimal dimension d is in fact a "model selection" problem, which is due to the fact that choice of d balances the complexity of the model and the storage of the data. There are many model selection criterion, we will learn two of them.

1) The first way is to bound the residual of approximation, that is, suppose $\hat{\mathbf{X}}$ is the best $\text{rank}-d$ approximation of $\mathbf{A}$, given a threshold $\tau > 0$, we want to minimize the residual with respect to Frobinius-norm, that is

$$\min_{d=1,\ldots,K} \quad \|\mathbf{A}-\hat{\mathbf{X}}\|_F^2 \leq \tau \tag{6}$$

the problem can be explicitly written as

$$d^* = \min_{d=1,\ldots,K} \left\{ d \,\middle|\, \sum_{i=d+1}^{K} \sigma_i^2(\mathbf{A}) \leq \tau \right\} \tag{7}$$

2) Note that the first criterion (7) depends on specific problem (singular values of data matrix are not invariant with respect to linear transformations), it is hard to chose a reasonable threshold. To generalize our criterion, we consider the normalized version of (7), this new criterion, *a.k.a* **variance explained ratio** in machine learning, is given by

$$d^* = \min_{d=1,\ldots,K} \left\{ d \,\middle|\, \frac{\sum_{i=d+1}^{K} \sigma_i^2(\mathbf{A})}{\sum_{i=1}^{K} \sigma_i^2(\mathbf{A})} \leq \tau \right\} \tag{8}$$

Now, suppose the data matrix $\mathbf{A} \in \mathbb{R}^{D\times N}$ is the same as Problem 3 (data1/data1.mat). You are required to

1) (**8 points**) Plot the squared singular values of $\mathbf{A}$ along with the threshold $\tau = 150$, what is the solution to (7)?
2) (**12 points**, **6 points** for plot, **6 points** for table) Plot the figure of function

$$f(d) = \frac{\sum_{i=d+1}^{K} \sigma_i^2(\mathbf{A})}{\sum_{i=1}^{K} \sigma_i^2(\mathbf{A})}, \quad d = 1, \ldots, K \tag{9}$$

and fill the following table from the figure with respect to (8):

τ	0.1	0.05	0.02	0.005
Compression rate	0.0	0.0	0.0	0.0

Table III: The compression rate with respect different threshold

where the compression rate is defined as

$$\text{compression rate} = \frac{\#\{\text{entries in } \hat{\mathbf{X}}\}}{\#\{\text{entries in } \mathbf{A}\}} = \frac{d^*(D+N+1)}{DN} \tag{10}$$

Remarks

1) Please **insert your figures in your PDF**.
2) You can just mark the solution of problem 1) on your figure (**plot a vertical line with red color or mark the solution with marker**).
3) Please draw a continuous curve for problem 2), see *stairs* function in Matlab, *plt.step* function in Python for more details.
4) For simplicity, we omit some proof details, you may refer to *Generalized Principal Component Analysis, Section 2.1* for more details.

Solution.

The solution is given as follows:

1) To compute the solution to (7), we compute $\sum_{i=d+1}^{K} \sigma_i^2(\mathbf{A})$ for $d = 1, \ldots, K-1$:

$$\sum_{i=1+1}^{K} \sigma_i^2(\mathbf{A}) = 248.1159 \tag{11}$$

$$\sum_{i=2+1}^{K} \sigma_i^2(\mathbf{A}) = 57.3672 \tag{12}$$

thus the optimal solution is $d^* = 2$. The figures are shown as follows:

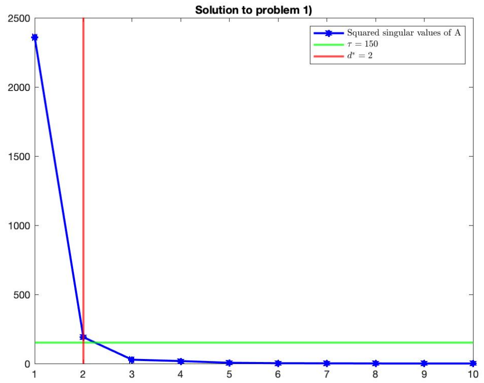


Figure 6: The solution to problem 1)

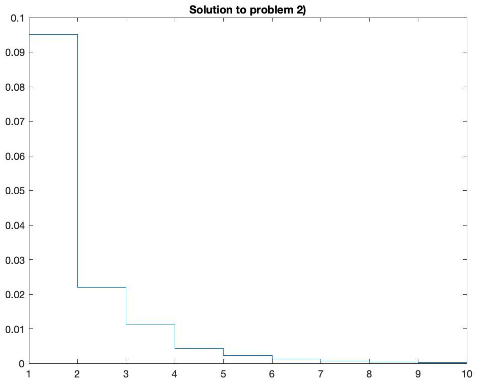


Figure 7: The solution to problem 2)

From the Figure 7, it's not hard to obtain the solution to the table: It's okay if you use different precision in the

τ	0.1	0.05	0.02	0.005
d^*	1	2	3	4
Compression rate	0.1005	0.2010	0.3015	0.4020

Table IV: The compression rate with respect different threshold

table.

The Matlab code for solving this problem is given as follows:

```
%%%%% Solution to Problem 5 %%%%%
%% load the data
clc;
clear;

% load the data
A = load(’data1.mat’);
A = A.data1;

%% data processing
[U, S, V] = svd(A);

% vector of singular values of A
sigma_A = diag(S);

% vector of squared singular values
squared_sigma_A = sigma_A .^2;

%% solution to problem 1)
tau = 150;

semilogy(squared_sigma_A, ’b-*’, ’LineWidth’, 2);
% it’s okay if you use the plot function
% plot(squared_sigma_A, ’b-*’, ’LineWidth’, 2);

hold on
yline(tau, ’g’, ’LineWidth’, 2);

% the optimal solution is given by d=2
%  (sigma_{d+1}^2(A) < tau)

% plot the optimal solution
d_star = 2;
xline(d_star, ’r’, ’LineWidth’, 2);

legend(’Squared singular values of A’, ’$\tau=150$’, ’$d^*=2$’, ’Interpreter’,’latex’);
title(’Solution to problem 1)’);

```

```
%% solution to problem 2)
% normalize the squared singular values
normalized_squared_sigma_A = squared_sigma_A / sum(squared_sigma_A);

% compute the cumsum of normalized squared singular values
accum_sum = cumsum(normalized_squared_sigma_A);

% plot the function f(d)
stairs(1.0 - accum_sum)
title(’Solution to problem 2)’)
```