SI231 - Matrix Computations, Fall 2020-21

Homework Set #5

Prof. Yue Qiu and Prof. Ziping Zhao

Name: Min Hongqi **Major:** EE

Student No.: 2020E8018482005 **E-mail:** minhq@sari.ac.cn

Acknowledgements:

1) Deadline: **2020-12-07 23:59:00**
2) Submit your homework at **Gradescope**. Homework #5 contains two parts, the theoretical part the and the programming part.
3) About the the theoretical part:
   (a) Submit your homework in **Homework 5** in gradescope. Make sure that you have correctly select pages for each problem. If not, you probably will get 0 point.
   (b) Your homework should be uploaded in the **PDF** format, and the naming format of the file is not specified.
   (c) No handwritten homework is accepted. You need to use LaTeX in principle.
   (d) Use the given template and give your solution in English. Solution in Chinese is not allowed.
4) About the programming part:
   (a) Submit your codes in **Homework 5 Programming part** in gradescope.
   (b) When handing in your homework in gradescope, package all your codes into your_student_id+hw4_code.zip and upload. In the package, you also need to include a file named README.txt/md to clearly identify the function of each file. (".zip" format package rather than ".rar, .7zip" should be uploaded, solution of the results should be named according to requirements.)
   (c) Make sure that your codes can run and are consistent with your solutions.
5) **Late Policy details can be found in the bulletin board of Blackboard.**

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. 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:

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

$\ell 2$ regularized LS:

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

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

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

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 $||\mathbf{Ax} - \mathbf{y}||_2^2$ 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)

2)

3) (a) Both L0 norm of $\mathbf{x_{LS}}$ two algorithms are 0.

(b) In the Majorization-Minimization method, we get the loss $||\mathbf{Ax} - \mathbf{y}||_2^2 = 2.9453e + 03$,
In the gradient decent method, we get the loss $||\mathbf{Ax} - \mathbf{y}||_2^2 = 2.9455e + 03$.

II. COMPUTATIONS OF SVD

Problem 2. 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. Fine 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 $m = 1, \ldots, 30$ and show which one is the desired result.

Algorithm 1: SVD Decomposition by $\mathbf{A}^T\mathbf{A}$

Input : Thin matrix A

1 Form $\mathbf{A}^T\mathbf{A}$.

2 Compute the eigenvalue decomposition $\mathbf{A}^T\mathbf{A}$=$\mathbf{V}^T\mathbf{\Lambda}\mathbf{V}$.

3 Let $\mathbf{\Sigma}$ be an m×n diagonal matrix with diagonal entries being the nonnegative square root of diagonal entries of $\mathbf{\Lambda}$.

4 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) Input $[V, D] = \mathrm{eig}(A)$ in matlab,

we get $D = \begin{bmatrix} -0.1778 & & & \\ & 0.1778i & & \\ & & -0.1778i & \\ & & & 0.1778 \end{bmatrix}$

so the eigenvalues are: -0.1778, 0.1778i, -0.1778i, 0.1778

Input $[U, S, V] = \mathrm{svd}(A)$ in matlab,

we get $S = \begin{bmatrix} 3 & & & \\ & 2 & & \\ & & 1 & \\ & & & 0.0002 \end{bmatrix}$

so the singular values are: 3, 2, 1, 0.0002

2) $B^T B = \begin{bmatrix} 10.25 & 10.25 \\ 10.25 & 20.25 \end{bmatrix}$

Input $[V, D] = \mathrm{eig}\left(B^T B\right)$ in matlab,

we get $V = \begin{bmatrix} -0.8481 & 0.5299 \\ 0.5299 & 0.8481 \end{bmatrix}$ $D = \begin{bmatrix} 3.8455 & \\ & 26.6545 \end{bmatrix}$

In step 3, we get $\Sigma = \begin{bmatrix} 1.9610 & 0 \\ 0 & 5.1628 \\ 0 & 0 \end{bmatrix}$

In put $[U, \Sigma] = qr(BV)$ in matlab,

we get $U = \begin{bmatrix} -0.7570 & -0.6364 & -0.1482 \\ 0.6484 & -0.7597 & -0.0494 \\ -0.0811 & -0.1335 & 0.9877 \end{bmatrix}$

In order to verify the correctness of SVD decomposition, calculate $\mathbf{U\Sigma V^T} = \begin{bmatrix} -0.4823 & -3.5732 \\ -3.1567 & -2.6525 \\ -0.2302 & -0.6686 \end{bmatrix}$

Obviously, the results are wrong.

Correction :

If we sort the eigenvalues of D in order from large to small in step 2, then the column vectors of V are going to changing the order, we will get: $V = \begin{bmatrix} 0.5299 & -0.8481 \\ 0.8481 & 0.5299 \end{bmatrix}$ $D = \begin{bmatrix} 26.6545 & \\ & 3.8455 \end{bmatrix}$

then sort the square root of diagonal entries of $\mathbf{\Lambda}$ in order from large to small, get $\Sigma = \begin{bmatrix} 5.1628 & 0 \\ 0 & 1.9610 \\ 0 & 0 \end{bmatrix}$

Continue with steps 4, we get $U = \begin{bmatrix} -0.6364 & 0.7570 & -0.1482 \\ -0.7597 & -0.6484 & -0.0494 \\ -0.1335 & 0.0811 & 0.9877 \end{bmatrix}$

and $\mathbf{U\Sigma V^T} = \begin{bmatrix} -3.0000 & -2.0000 \\ -1.0000 & -4.0000 \\ -0.5000 & -0.5000 \end{bmatrix}$

Although there is a minus sign difference between $\mathbf{U\Sigma V^T}$ and A, but the results can be considered right.

3) Plot two curves on a log scale for $m = 1, \ldots, 30$,

Because when $m = 10, 12, 13, 20, 22, 23, 25, 27$, the eigenvaluses calculated by Matlab of $\mathbf{A}^T\mathbf{A}$ will be negative, so it's nonsense to calculate the nonnegative square of diagonal entries of $\mathbf{\Lambda}$.

So the smallest singular value through SVD is the desired result.

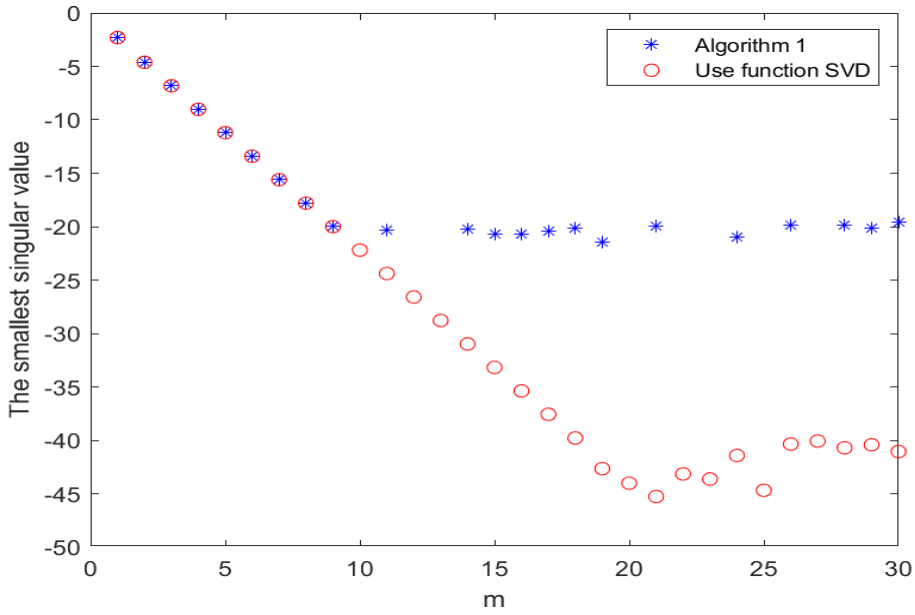


Figure 1: The smallest singular value through Algorithm 1 and SVD

III. Iterative methods.

Problem 3. 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, ..., N do
    update x^k to obtain x^{k+1} ;        // Use one of the three iteration methods
    if ||x^{k+1} - x^k||_2 < 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.

1)

2) From the Figure 2, set $eps = 1e-5$, we can see that when $w = 1$, we can achieve the highest convergence rate.

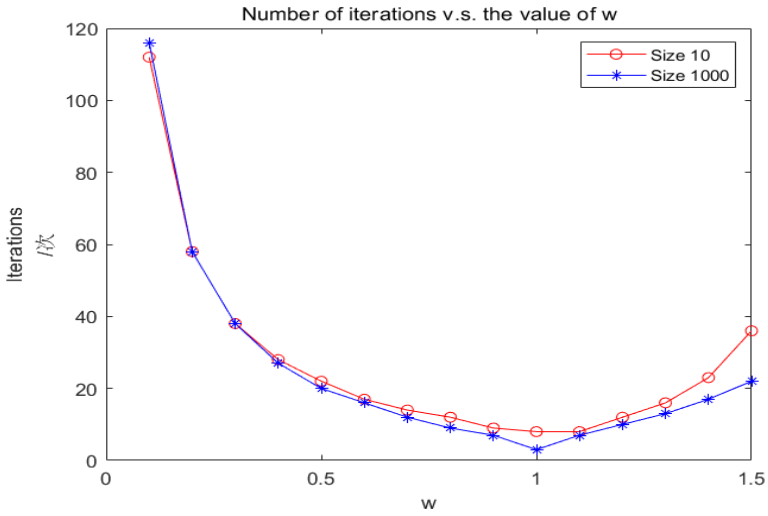


Figure 2: The smallest singular value through Algorithm 1 and SVD

3) From the Figure 3, set $eps = 1e-20$ and $w = 0.9$, then plot the curves on a log scale, we find convergence of Gauss-Seidel Iteration is faster than Jacobi.

When $n = 10$, the convergence of SOR is between Gauss-Seidel Iteration and Jacobi method,

When $n = 1000$, the convergence of SOR id the slowest.

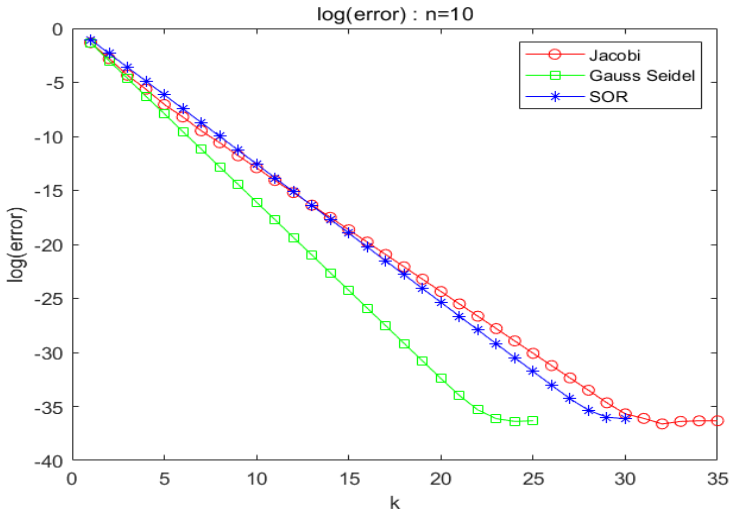


Figure 3: Analyze the convergence rate for n=10

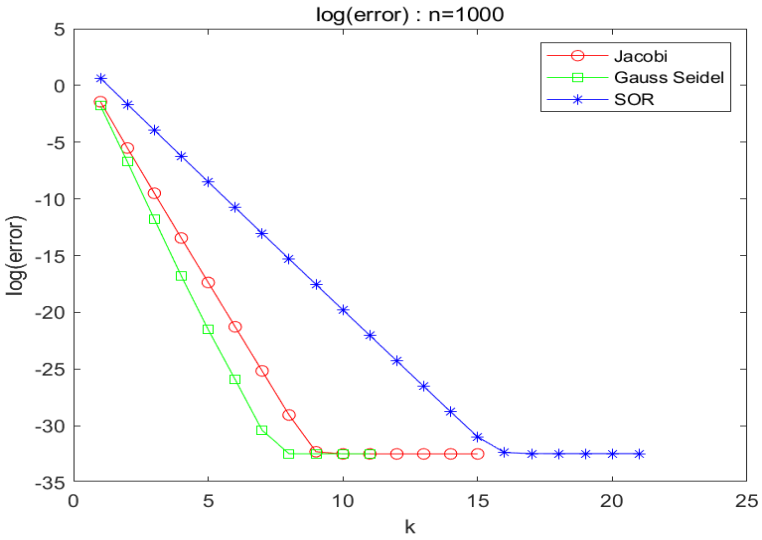


Figure 4: Analyze the convergence rate for n=1000

Give the final error $\|\mathbf{x}_{\text{true}} - \mathbf{x}^k\|_2$ for $k = N$.

N=10 :

a) Jacobi: $\|\mathbf{x}_{\text{true}} - \mathbf{x}^k\|_2 = 1.6725e - 16$

b) Gauss-Seidel: $\|\mathbf{x}_{\text{true}} - \mathbf{x}^k\|_2 = 1.6725e - 16$

c) SOR: $\|\mathbf{x}_{\text{true}} - \mathbf{x}^k\|_2 = 2.0266e - 16$

N=1000 :

a) Jacobi: $\|\mathbf{x}_{\text{true}} - \mathbf{x}^k\|_2 = 7.7727e - 15$

b) Gauss-Seidel: $\|\mathbf{x}_{\text{true}} - \mathbf{x}^k\|_2 = 7.8601e - 15$

c) SOR: $\|\mathbf{x}_{\text{true}} - \mathbf{x}^k\|_2 = 7.9653e - 15$

IV. Application of SVD

Problem 4. 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.

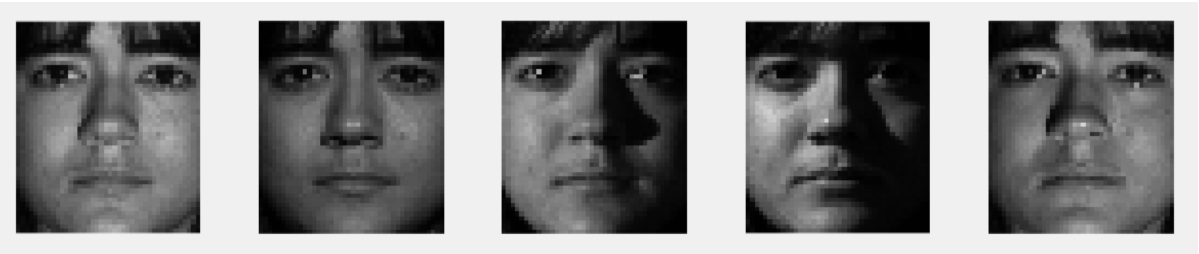


Figure 5: Face images of one individual under 5 different illumination conditions in the extended YaleB dataset. All images are frontal faces.

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

[1]you can use the images or the mat file in data1 folder

2) (**8 points**) Apply PCA in Algorithm 3 with $d = 10$ to the Training Set.
   a) Plot the mean face, the first eigenface, the second eigenface, the third eigenface and the tenth eigenface.
   b) Plot the sorted singular values.
   c) Project the Test Set onto the face subspace given by PCA, that is $\mathbf{Y}_{test} = \mathbf{U}^T(\mathbf{X}_{test} - \boldsymbol{\mu}\mathbf{1}^T)$. Compute the projected faces, that is $\text{Proj}(\mathbf{X}_{test}) = \boldsymbol{\mu}\mathbf{1}^T + \mathbf{U}\mathbf{Y}_{test}$. 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) Recommend to use truncated(thin) SVD for fast implementation.
5) Please **insert your figures in your PDF**.

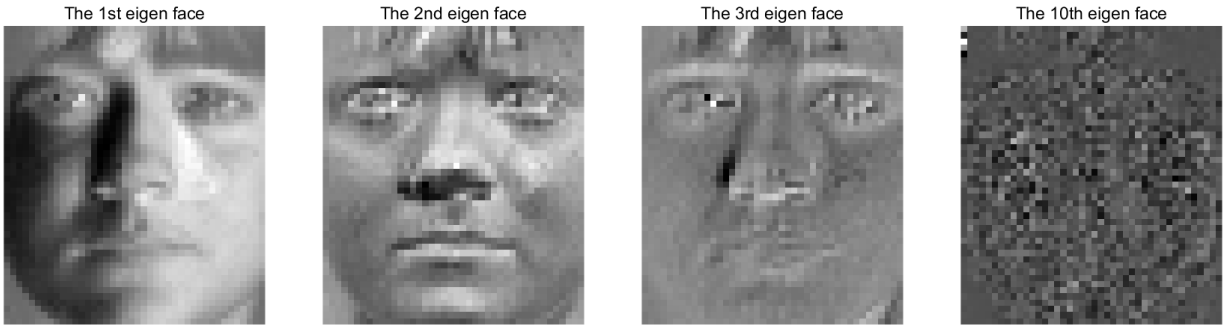
Solution.

[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 $x = [X_{11}, X_{21}, X_{31}, ..., X_{mn}]^T \in \mathbb{R}^{mn}$

1) The first eigenface resembles the mean face best. The bigger the singular value is, the eigenface is more similar to the meanface.

Figure 6: The mean face

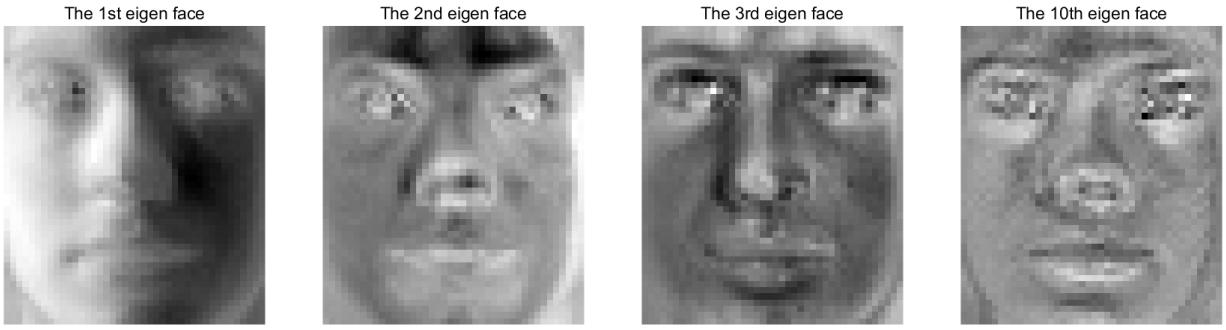


(a) The first eigen face (b) The second eigen face (c) The third eigen face (d) The tenth eigen face

Figure 7: Eigenfaces

2) a) The mean face and the eigenfaces of the training set.

Figure 8: The training set mean face



(a) The first eigen face (b) The second eigen face (c) The third eigen face (d) The forth eigen face

Figure 9: Eigenfaces

b) The the sorted singular values is showed in Figure 10.

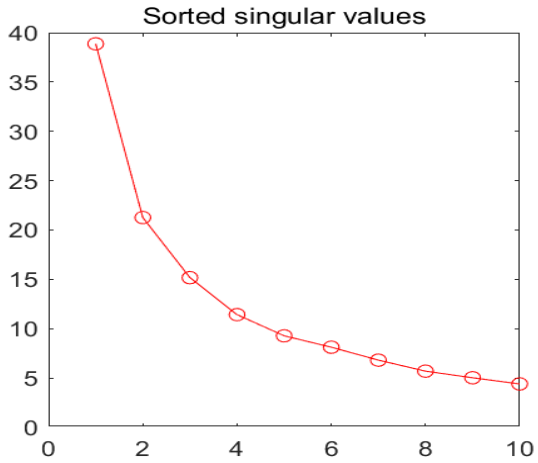
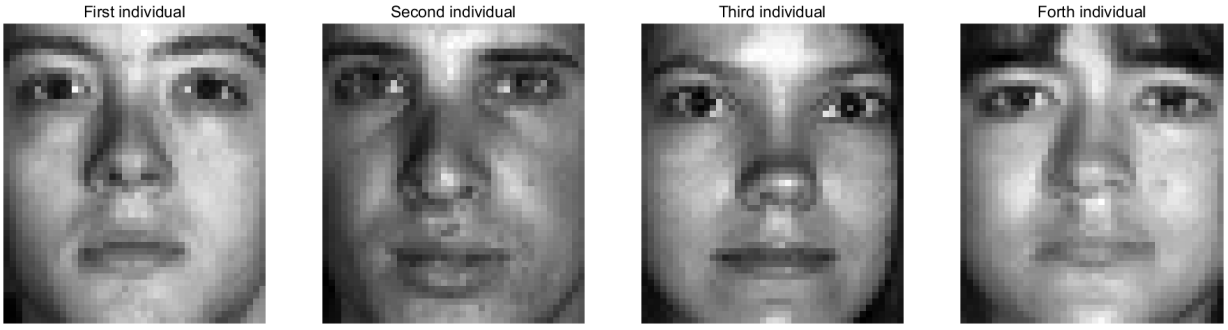


Figure 10: The training set mean face

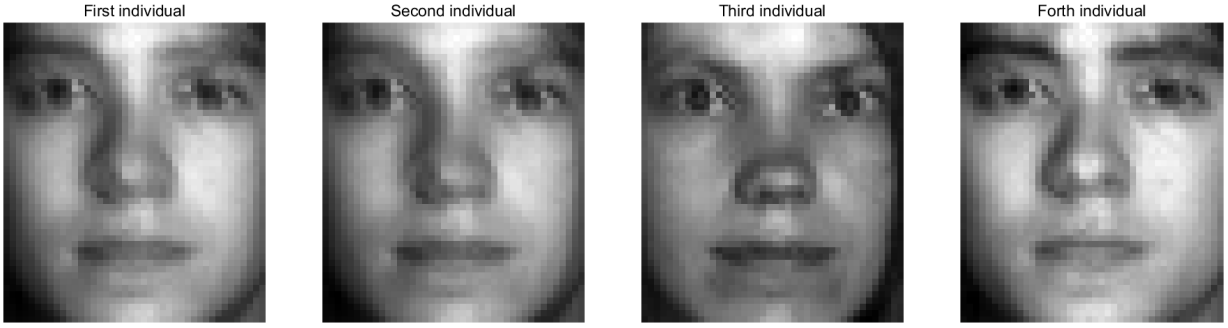
c) The projected faces for d=10 are as follows. It is clear that they are faces from four different individuals.



(a) The first eigen face (b) The second eigen face (c) The third eigen face (d) The forth eigen face

Figure 11: Eigenfaces

d=2 projected faces are more ambiguous. Because the biggest 10 singular values can perform most of the features of the faces, while the biggest 2 can't.



(a) The first eigen face (b) The second eigen face (c) The third eigen face (d) The forth eigen face

Figure 12: Eigenfaces

3) $d = 7$ gives the best recognition performance

Table II: Summary of errors

# eigenvectors	2	3	4	5	6	7	8
error	55%	45%	40%	25%	15%	0	0

Problem 5. 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

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

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

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 $\sigma_1(\mathbf{A}) \ge \sigma_2(\mathbf{A}) \ge \cdots \ge \sigma_K(\mathbf{A}) \ge 0$ and $K = \min\{D, N\}$):

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

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

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

the problem can be explicitly written as

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

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

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

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

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

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

τ	0.1	0.05	0.02	0.005
Compression rate				

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.

1) The solution to (7) is d = 2.

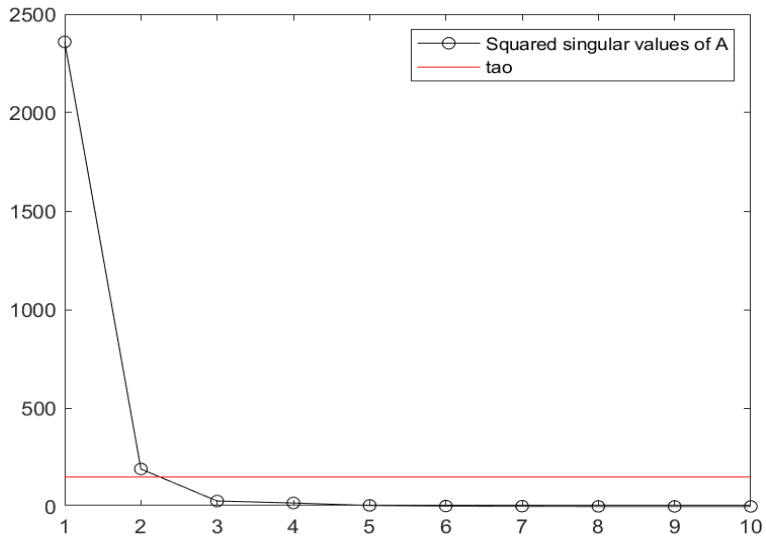


Figure 13: Squared singular values of A

2) N = 10, D = 2016, we can calculate the compression

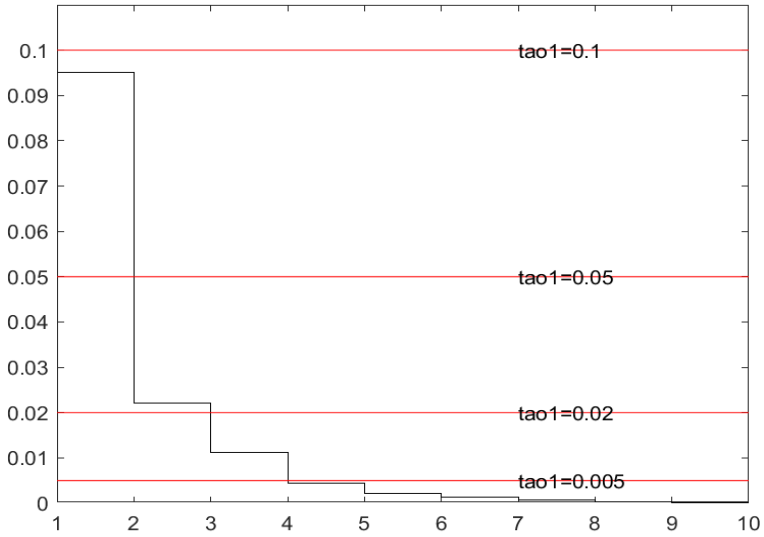


Figure 14: The function F

From the results above, we can get that $d^* = [1; 2; 3; 4]$

τ	0.1	0.05	0.02	0.005
d^*	1	2	3	4
Compression rate	10.05%	20.11%	30.16%	40.22%

Table IV: The compression rate with respect different threshold