Principal Component Pursuit By Alternating Directions

Report

MIS-4 End semester Project

Submitted by:

Team-5

MEKAPATI SPANDANA REDDY - CB.EN.U4AIE20038

PALETI NIKHIL CHOWDARY - CB.EN.U4AIE20046

PRANAV UNNINIKRISHNAN – CB.EN.U4AIE20053

ROHAN SANJEEV – CB.EN.U4AIE20059

Under the supervision of

Prof.(Dr.) V. Sowmya

AMRITA SCHOOL OF ENGINEERING

04 July 2022

Contents

Chapter 1

INSTALLING ROS

1.1 Overview

Let $X \in R^{p \times n}$ be a data matrix, PCA looks for the following decomposition:

$$X = L + E$$

where

L is of low rank (at most rank "k")

error matrix E has a small Frobenius norm,

Mathematically PCA looks to solve the optimization problem:

$$\min_{L \in R^{p \times n}} ||X - L||_2$$

subject to $rank(L) \leq k$

1.2 Principal Directions

First let's see the role of SVD in finding principal directions and then let's see a numerical example to compute principal directions.

1.2.1 SVD

Given a data matrix $X \in R^{p \times n}$, SVD decomposes the matrix into U, Σ, V^T as shown in Fig:1.1.

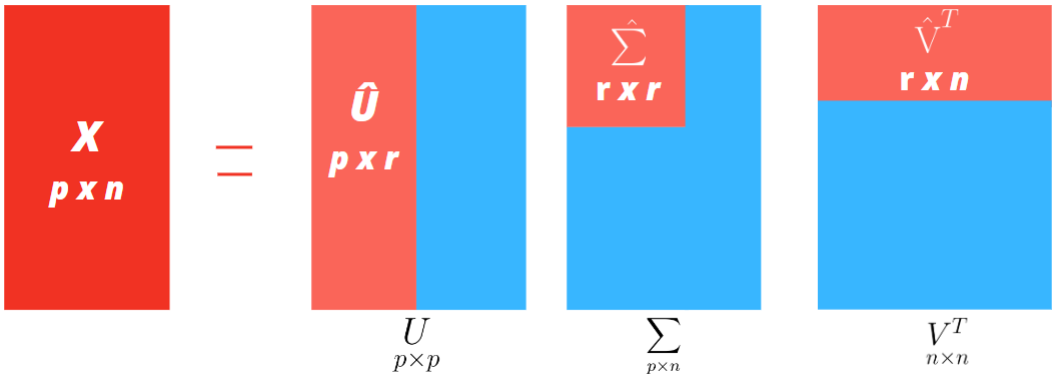


Figure 1.1: SVD

The rows of V^T or the columns of V from the SVD of centered data matrix contain the principal directions.

To understand why, lets take an example:

Let $X \in R^{p\times 2}$ be a data matrix

$$X = \begin{bmatrix} | & | \\ x_1 & x_2 \\ | & | \end{bmatrix}$$

For the given data matrix, we can construct the correlation matrix of columns of data matrix as follows:

$$X^T X = \begin{bmatrix} x_1^T x_1 & x_1^T x_2 \\ x_2^T x_1 & x_2^T x_2 \end{bmatrix}$$

Eigenvectors of correlation matrix:

$$X^T X = V\Sigma U^T U\Sigma V^T (X = U\Sigma V^T)$$

$$X^T X = V\Sigma^2 V^T (U^T U = I)$$

$$X^T XV = V\Sigma^2$$

The columns of V or the rows of V^T of the SVD decomposition are also the eigenvectors of the correlation matrix $X^T X$. These eigenvectors point in the direction of maximum variance and that directions are the principal directions.

1.2.2 Numerical example

Let us construct a data matrix $X \in R^{183\times 2}$ which contains mortality and fertility rate of 183 countries. First we center our data matrix to have zero mean by:

$$X = X - mean(X)$$

country	mortality	fertility
Afghanistan	3.79	1.70
Albania	-1.70	-1.07
Algeria	-0.64	-0.07
...	...	...
Yemen	2.27	1.11
Zambia	3.13	2.15
Zimbabwe	2.65	0.90

183 rows × 2 columns

Figure 1.2: X centered

the centered data matrix is shown in Fig:1.2.

SVD on the centered data matrix gives us $U \in R^{183\times 183}, \Sigma \in R^{183\times 2} and V^T \in R^{2\times 2}$.

As noted before, the rows of V^T contain the principal directions.

$$V^T = \begin{bmatrix} -0.93 & -0.37 \\ -0.37 & 0.93 \end{bmatrix}$$

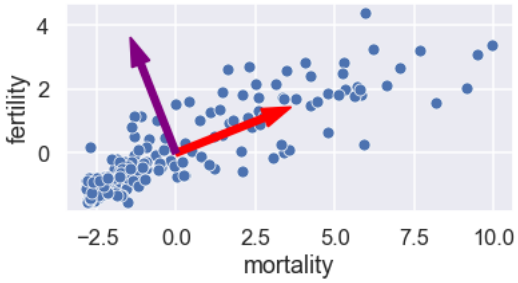


Figure 1.3: data with principal directions

Fig: 1.3 shows the plot of the data along with the principal directions which we calculated. The red line shows the first principal direction and the purple line shows the second principal direction.

1.3 Principal Component Analysis

If X has a dimension of 'n' then , it can be said that X will have 'n' principal directions. To reduce the dimension of X to 'k' dimensions, the data is projected onto a first 'k' principal directions.This is how PCA helps in dimensionality reduction.

To illustrate PCA, we will take the previous example and the reduce the dataset from 2 dimensions to 1.

From the previous example we can recall that from the SVD computed, the matrix V^T contained two principal directions,To remove a direction we simply drop the last row.

$$V^T = \begin{bmatrix} -0.93 & -0.37 \end{bmatrix}$$

Finally, we project the original data points X onto the single principal direction.

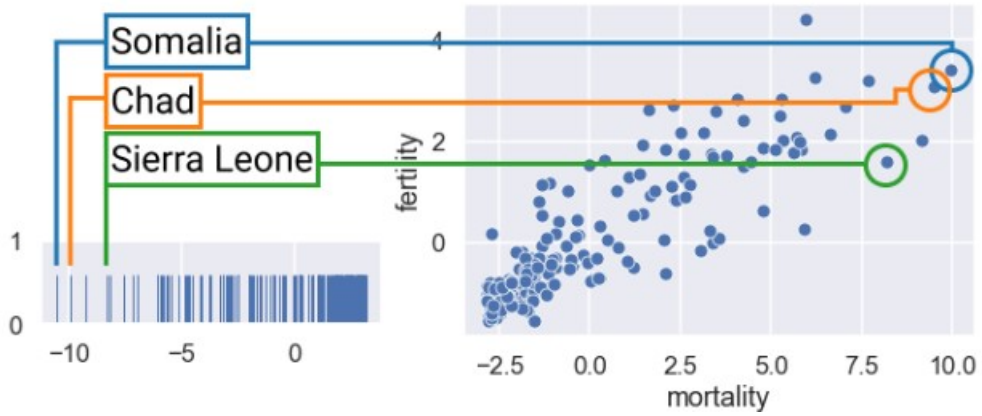


Figure 1.4: data with principal directions

Fig: 1.8 shows that the that points that appear together in the two-dimensional data (scatter plot) also appear together in the one-dimensional data (rug plot). Hence the dimension of the original data has been reduced while preserving the patterns in the data.

In practice, we follow a simpler method to compute the reduced dimension X which is obtained from the following equivalence:

$$X = U\Sigma V^T$$

$$XV = U\Sigma V^T V(\text{Right-multiply by } V)$$

$$XV = U\Sigma$$

This means that instead of removing the rows from V^T and then computing XV we can remove columns from $U\Sigma$ to obtain the desired dimension.

1.4 Rank-k Approximation

A matrix X has rank k if any vector(row/column) in the matrix can be represented as the linear combination of k rank-one matrices, and cannot be written as a liner combination of k 1 or fewer rank-one vectors(rows/columns).
A low-rank approximation provides a compressed version of the matrix. If X represents an image, the dimensions, m and n are typically in the order of 100s. With images, a modest value of k is usually enough to achieve approximations that look a lot like the original image.
Here we project the points in X to the top-k principal components. We already know that the first k right singular vectors contain the dominant principal directions.

we know SVD gives the following decomposition:

$$SVD(X) = U\Sigma V^T$$

where,

$$X \in R^{m\times n}, U \in R^{m\times m}, \Sigma \in R^{m\times n}, V \in R^{n\times n}$$

To project our data into top k principal components, we first split U, Σ *and* V as follows:

$$U = (U_L|U_R), \Sigma = \begin{pmatrix} \Sigma_{TL} & 0 \\ 0 & \Sigma_{BR} \end{pmatrix}, V = (V_L|V_R)$$

where,

$$U_L \in R^{m\times k}, V_L \in R^{n\times k}, \Sigma_{TL} \in R^{k\times k}$$

Then,

$$X \approx U_L \Sigma_{TL} V_L^T$$

X is again a $m \times n$ matrix as a result of matrix multiplications of dimensions $m \times k * k \times k * k \times n$

1.5 Effect of outliers in data

In this section let's explore the effect of outliers in data on PCA

1.5.1 Without outliers

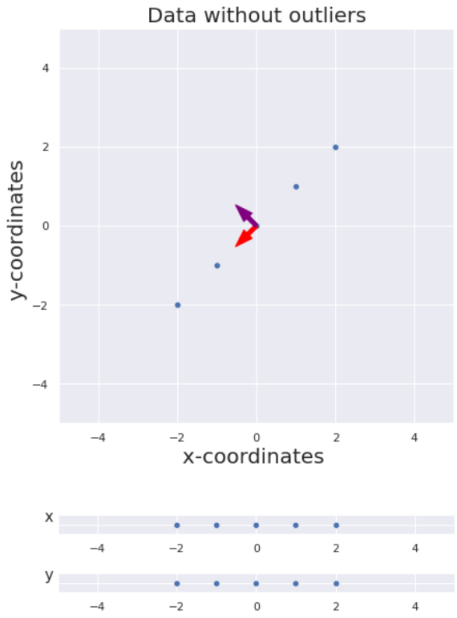


Figure 1.5: data without ouliers

Fig 1.5 shows the data taken and it doesn't have any outlier. there is a linear relation between the x-coordinate values and y-coordinate values. Performing PCA on this data, the output is presented in 1.6

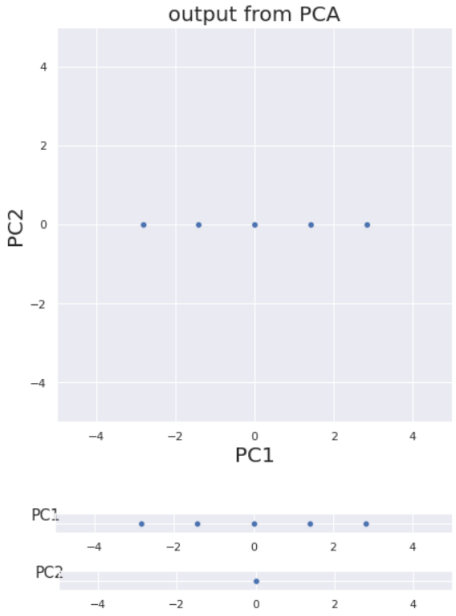


Figure 1.6: PCA without outliers

Since we have a linear relation between our two variables, all our data is captured by the first principal component PC1, the second principal component holds no information about our data.

1.5.2 With outliers

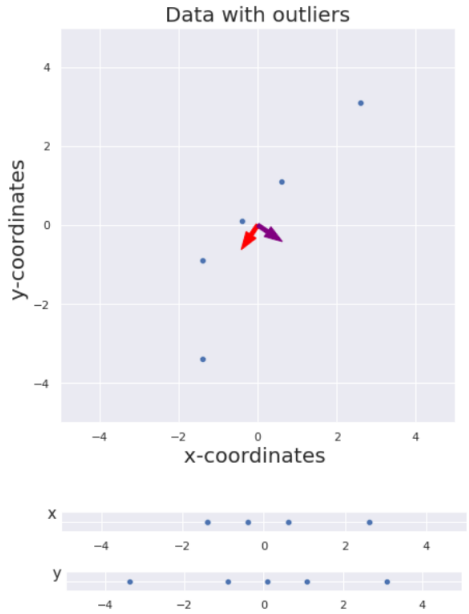


Figure 1.7: data with ouliers

Fig 1.7 shows the data taken with an outlier. Performing PCA on this data, the output is presented in 1.8

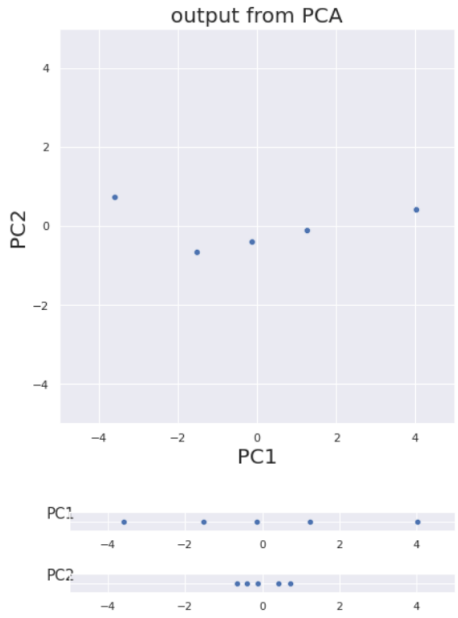


Figure 1.8: PCA with ouliers

1.8 shows the failure of PCA on data with outliers. we can see that with a single outlier, theres a lot of information added to the second principal component PCA2.
In the following chapter let's look at Robust Principal Component analysis which is not sensitive to outliers.

Chapter 2

MAKING ROBOT

2.1 Overview

Let $X \in R^{p \times n}$ be a data matrix, RPCA looks for the following decomposition:

$$X = L + S$$

where

L is of low rank (at most rank "k")

S is a sparse matrix

Mathematically RPCA looks to solve the optimization problem:

$$\min_{L,S} \text{rank(L)} + ||S||_0$$

$$\text{subject to } L + S = X$$

where

$$||S||_0 := \#\{S_{i,j} \neq 0\}$$

(The number of non zero values in S)

2.2 Convex test

Non-Convex problems are generally very hard to solve because a direct minimization might lead to a poor local minima. Convex relaxation aims to solve these by approximating these functions using convex functions and in a convex function, any local minimum is the global minimum.

We say that f is convex on S if

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y)$$

$$\text{for all } x, y \in S \text{ and all } \lambda \in [0,1]$$

2.2.1 Convexity of Rank

Let's test the convexity of rank() by taking an example,

Let X = $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$, Y= $\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$ and λ =0.5

$$rank(0.5 * X + 0.5 * Y) = 2$$

$$0.5 * rank(X) + 0.5 * rank(Y) = 1$$

as $2 > 1$, the inequality fails.

Rank() is non-convex

2.2.2 Convexity of L_0 norm

Let's test the convexity of L_0 norm by taking an example,

Let x = $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$, y = $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$, for any $\lambda \in (0,1)$

$$||\lambda x + (1-\lambda)y)||_0 = 2$$

$$\|\lambda f(x) + (1-\lambda) f(y)\|_0 = 1$$

as $2 > 1$, the inequality fails.

L_0 norm is non-convex

Since both terms in our objective function are non-convex, our problem is hard to solve. Convex relaxation helps us solve the problem which is discussed in the section below.

2.3 Convex relaxation

2.3.1 Relaxing Rank

$$\textbf{rank}(L) := \#\{\sigma_i(L) \neq 0\} \implies \|L\|_* = \sum_i \sigma_i(L)$$

We can define rank to be the number of non zero singular values of a Matrix. As we saw in the previous section rank is non-convex. We relax rank() with nuclear norm which is the sum of the singular values of a matrix.

convex test of nuclear norm

Let's test the convexity of nuclear norm by taking an example,

Let x = $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$, y = $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $\lambda = 0.5$

$$||\lambda x + (1-\lambda)y)||_* = 0.7071$$

$$\|\lambda f(x) + (1-\lambda) f(y)\|_* = 1$$

as $0.7071 \leq 1$, the inequality holds.

Nuclear norm is convex

2.3.2 relaxing L_0 norm

$$\|S\|_0 := \#\left\{S_{ij} \neq 0\right\} \implies \|S\|_1$$

We can define L_0 norm to be the number of non zero values of a Matrix. As we saw in the previous section L_0 norm is non-convex. We relax L_0 norm with L_1 norm which is the sum of all the absolute values in a matrix.

convex test of $L1$ norm

Let's test the convexity of $L1$ norm by taking an example,

Let x = $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$, y = $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$, for any $\lambda \in (0,1)$

$$||\lambda x + (1-\lambda)y)||_1 = 1$$

$$\|\lambda f(x) + (1-\lambda)f(y)\|_1 = 1$$

as $1 = 1$, the inequality holds.

L_1 norm is convex

New objective function:

$$\min_{L,S} \|L\|_* + \lambda ||S||_1$$

$$\text{subject to } L + S = X.$$

here $\lambda = 1/\sqrt{\max(n,m)}$ where m,n are dimensions of X

2.4 L_1 Norm and Sparsity

This norm is defined as the summation of absolute values of a vector's all components. If a vector is [x,y] , its L_1 norm is

$$|x| + |y|$$

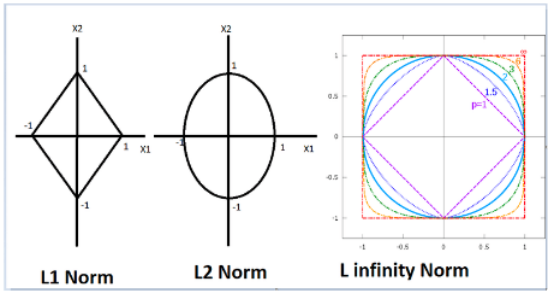


Figure 2.1: Different p-order norms

Fig2.1 shows the different p-order norms equal to 1. From this we can observe that the L-1 Norm has pointy spikes at sparse points. Hence we use the L-1 Norm to induce sparsity.

For example by looking at Fig 2.2 we can see that on gradually expanding the value of L1 and L2 norms, we get a solution when it touches a point in the norm. On comparing

both the norms, we observe that when using the L1 norm, the spiky sparse points touch the line first. Since these points are sparse, we arrive at a sparse solution for the line shown in the figure.

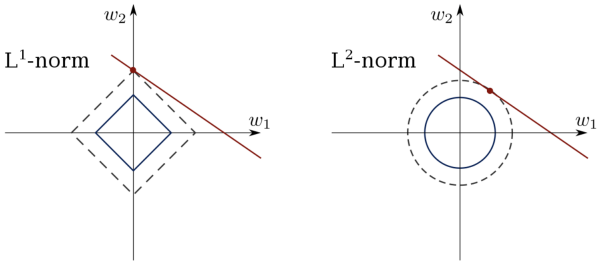


Figure 2.2: Comparison of L1 norm and L2 norm

2.5 Vector and Matrix Derivatives

2.5.1 Inner Product

Vectors

Let,

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Taking partial derivative of $a^T b$ w.r.t b we get,

$$\frac{\partial}{\partial \mathbf{b}} \mathbf{a}^t \mathbf{b} = \frac{\partial}{\partial \mathbf{b}} (a_1 b_1 + a_2 b_2 + a_3 b_3)$$

$$\frac{\partial}{\partial \mathbf{b}} \mathbf{a}^t \mathbf{b} = \begin{bmatrix} \frac{\partial}{\partial b_1}(a_1 b_1) \\ \frac{\partial}{\partial b_2}(a_2 b_2) \\ \frac{\partial}{\partial b_3}(a_3 b_3) \end{bmatrix}$$

$$\frac{\partial}{\partial \mathbf{b}} \mathbf{a}^t \mathbf{b} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \mathbf{a}$$

Matrices

For 2 matrices A and B, the inner product can be computed by converting the matrices into a vector(column stacking) and taking the dot product or by taking the trace of A^tB.

Let,

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

And,

$$B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

Computing A^TB

$$A^TB = \begin{bmatrix} a_{11}b_{11} + a_{21}b_{21} & a_{11}b_{12} + a_{21}b_{22} \\ a_{12}b_{11} + a_{22}b_{21} & a_{12}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Now we take the trace

$$Tr(A^TB) = a_{11}b_{11} + a_{21}b_{21} + a_{12}b_{12} + a_{22}b_{22}$$

On taking partial derivative w.r.t B we get,

$$\frac{\partial \langle A,B \rangle}{\partial B} = \begin{bmatrix} \frac{\partial}{\partial b_{11}} a_{11}b_{11} & \frac{\partial}{\partial b_{12}} a_{12}b_{12} \\ \frac{\partial}{\partial b_{21}} a_{21}b_{21} & \frac{\partial}{\partial b_{22}} a_{22}b_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = A$$

2.5.2 L_1 Norm

Vectors

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}, \|a\|_1 = \sum_i |a_i| = |a_1| + |a_2| + |a_3|$$

$$\frac{\partial \|\mathbf{a}\|_1}{\partial \mathbf{a}} = \begin{bmatrix} \frac{\partial |a_1|}{\partial a_1} \\ \frac{\partial |a_2|}{\partial a_2} \\ \frac{\partial |a_3|}{\partial a_3} \end{bmatrix} = \begin{bmatrix} sign(a_1) \\ sign(a_2) \\ sign(a_3) \end{bmatrix} = sign(\mathbf{a}) \tag{2.1}$$

Matrices

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \|A\|_1 = \sum_{ij} |a_{ij}| = |a_{11}| + |a_{12}| + |a_{21}| + |a_{22}|$$

$$\frac{\partial \|\mathbf{A}\|_1}{\partial \mathbf{A}} = \begin{bmatrix} \frac{\partial \|a\|_{11}}{\partial a_{11}} & \frac{\partial \|a\|_{12}}{\partial a_{12}} \\ \frac{\partial \|a\|_{21}}{\partial a_{21}} & \frac{\partial \|a\|_{22}}{\partial a_{22}} \end{bmatrix}$$

$$\frac{\partial \|\mathbf{A}\|_1}{\partial \mathbf{A}} = \begin{bmatrix} sign(a_{11}) & sign(a_{12}) \\ sign(a_{21}) & sign(a_{22}) \end{bmatrix} = sign(\mathbf{A})$$

2.5.3 Frobenius Norm (L_2 norm of Matrices)

$$A = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}, B = \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix}$$

$$\|A+B\|_F^2 = \sum_{i,j} (A+B)_{i,j}^2 = (a_1+b_1)^2 + (a_2+b_2)^2 + (a_3+b_3)^2 + (a_4+b_4)^2$$

$$\frac{\partial \|A+B\|_F^2}{\partial B} = \begin{bmatrix} \frac{\partial (a_1+b_1)^2}{\partial b_1} & \frac{\partial (a_2+b_2)^2}{\partial b_2} \\ \frac{\partial (a_3+b_3)^2}{\partial b_3} & \frac{\partial (a_4+b_4)^2}{\partial b_4} \end{bmatrix}$$

$$\frac{\partial \|A+B\|_F^2}{\partial B} = \begin{bmatrix} 2(a_1+b_1) & 2(a_2+b_2) \\ 2(a_3+b_3) & 2(a_4+b_4) \end{bmatrix}$$

$$2(A+B)$$

2.6 Algorithm

2.6.1 Lagrangian and Augmented Lagrangian

We construct the Lagrangian function from the new objective function formulated in Section 2.3.2.

Objective Function:

$$\min_{L,S} \|L\|_* + \lambda \|S\|_1$$

$$subject\ to$$

$$X = L + S$$

The Lagrangian Function:

$$\mathscr{L}(L,S,Y) = \|L\|_* + \lambda \|S\|_1 - \langle Y, X - L - S \rangle$$

Here , Y is the Lagrangian multiplier. $Y \in R^{m \times n}$

The Augmented Lagrangian Function:

$$\mathscr{L}(L,S,Y) = \|L\|_* + \lambda \|S\|_1 - \langle Y, X - L - S \rangle + \frac{\mu}{2} \|X - L - S\|_F^2$$

Here μ is the augmented Lagrangian multiplier.

2.6.2 Shrinkage and SVT operators

Shrinkage operator

If we have an optimization problem of the form:

$$\min_s f(s) = \|s\|_1 + \frac{1}{2\tau} \|s - x\|_2^2$$

Then the solution to the problem is given by:

$$s^* = x - \tau sign(x)$$

$$s^* = S_\tau(x) = \begin{cases} x - \tau & \text{if } x > 0, x > \tau \\ x + \tau & \text{if } x < 0, |x| > \tau \\ 0 & \text{if } |x| \leq \tau \end{cases}$$

This equation can be also written as:

$$S_\tau(x) = sign(x) max(|x - \tau|, 0)$$

Singular Value Thresholding operator

If we have an optimization problem of the form:

$$\underset{X}{argmin} \frac{1}{2} \|X - Y\|_F^2 + \lambda \|X\|_*$$

Then the solution to the problem is given by SVT theorem as:

$$\mathbf{X}^* = SVT_\lambda(Y) := U[\Sigma - \lambda I]_+ V^T$$

Where $U\Sigma V^T = SVD(Y)$ and $[.]_+ = max(., 0)$.

2.6.3 Updating Variables

L Update

For updating L, first we consider the terms w.r.t L by omitting the constant terms (independent of L) in EQN 2.6.1

$$\mathscr{L}(L) = ||L||_* - \langle Y, -L \rangle + \frac{\mu}{2} ||X - L - S||_F^2$$

The L update can be written as:

$$L^{k+1} = \underset{L}{argmin} ||L||_* + \left\langle Y^k, L \right\rangle + \frac{\mu}{2} \left|\left| X - L - S^k \right|\right|_F^2$$

Considering the function:

$$\underset{L}{argmin} f(L) = ||L||_* + \left\langle Y^k, L \right\rangle + \frac{\mu}{2} \left|\left| X - L - S^k \right|\right|_F^2$$

Taking first order optimality:

$$0 = \partial(||L||_*) + Y^k + \mu(L - X + S^k)$$

Now let's take another function similar to the problem which we saw in SVT theorem:

$$L^{k+1} = \underset{L}{argmin} ||L||_* + \frac{\mu}{2} \left|\left| X - L - S^k + \frac{Y^k}{\mu} \right|\right|_F^2$$

Taking first order optimality for this function:

$$0 = \partial(||L||_*) + Y^k + \mu(L - X + S^k)$$

The first order optimality is the same for both the functions. Since we already know the solution of the second function through SVT theorem, we consider that solution for updating L.

$$L^{k+1} = SVT_{\frac{1}{\mu}}(X - S^k + \frac{Y^k}{\mu})$$

S Update

For updating S, first we consider the terms w.r.t S by omitting the constant terms (independent of L) in EQN 2.6.1

$$\mathscr{L}(S) = \lambda \, ||S||_1 - \langle Y, -S \rangle + \frac{\mu}{2} \, ||X - L - S||_F^2$$

The S update can be written as:

$$S^{k+1} = \underset{S}{argmin} \; \lambda \, ||S||_1 + \left\langle Y^k, S \right\rangle + \frac{\mu}{2} \left|\left| X - L^{k+1} - S \right|\right|_F^2$$

Considering the function:

$$\underset{S}{argmin} \; f(S) = \; \lambda \, ||S||_1 + \left\langle Y^k, S \right\rangle + \frac{\mu}{2} \left|\left| X - L^{k+1} - S \right|\right|_F^2$$

Taking first order optimality:

$$0 = \lambda \partial(||S||_1) + Y^k + \mu(-X + L^{k+1} + S)$$

The S update can now be written as:

$$S^{k+1} = \frac{\lambda}{\mu}(sign(S)) + \frac{Y^k}{\mu} + (X - L^{k+1})$$

The S update using the Shrinkage Operator:

$$S^{k+1} = \boldsymbol{S}_{\frac{\lambda}{\mu}}(X - L^{k+1} + \frac{Y}{\mu})$$

Y Update

The gradient of $\mathscr{L}$ in EQN 2.6.1 w.r.t Y:

$$\frac{\partial \mathscr{L}}{\partial Y} = X - L - S$$

$$Y^{k+1} = Y^k + \mu(X - L^{k+1} - S^{k+1})$$

Here μ is the step size.

2.6.4 Final Algorithm

Algorithm 1 Principal Component Pursuit by Alternating Directions

Require: $S_0 = Y_0 = 0, \mu > 0$

while not converged **do**

compute $L^{k+1} = SVT_{\frac{1}{\mu}}(X - S^k + \frac{Y^k}{\mu})$

compute $S^{k+1} = S_{\frac{\lambda}{\mu}}(X - L^{k+1} + \frac{Y}{\mu})$

compute $Y^{k+1} = Y^k + \mu(X - L^{k+1} - S^{k+1})$

end while

return L, S

2.6.5 Effect of outliers in data

In the previous chapter while discussing PCA, we have seen that PCA is sensitive to outliers. Now let's see the effect of outliers on RPCA.

RPCA without outliers

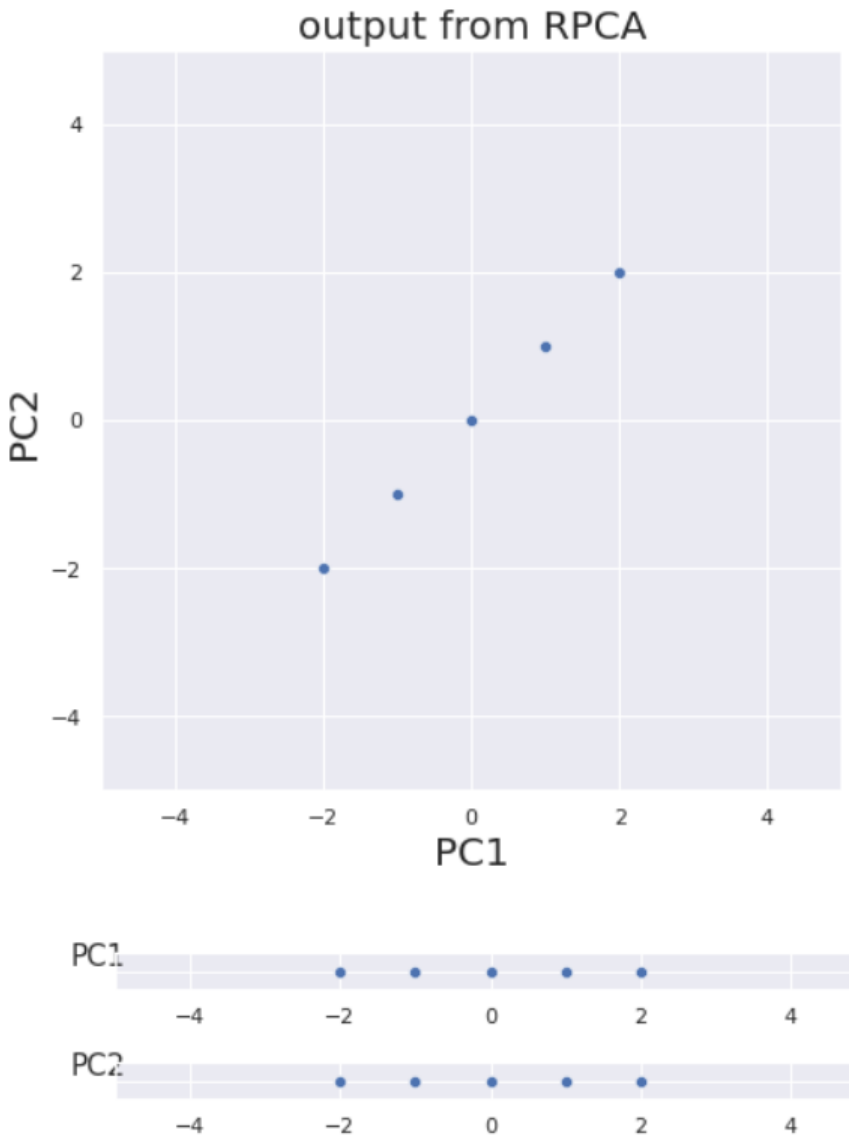


Figure 2.3: RPCA without outliers

RPCA with outliers

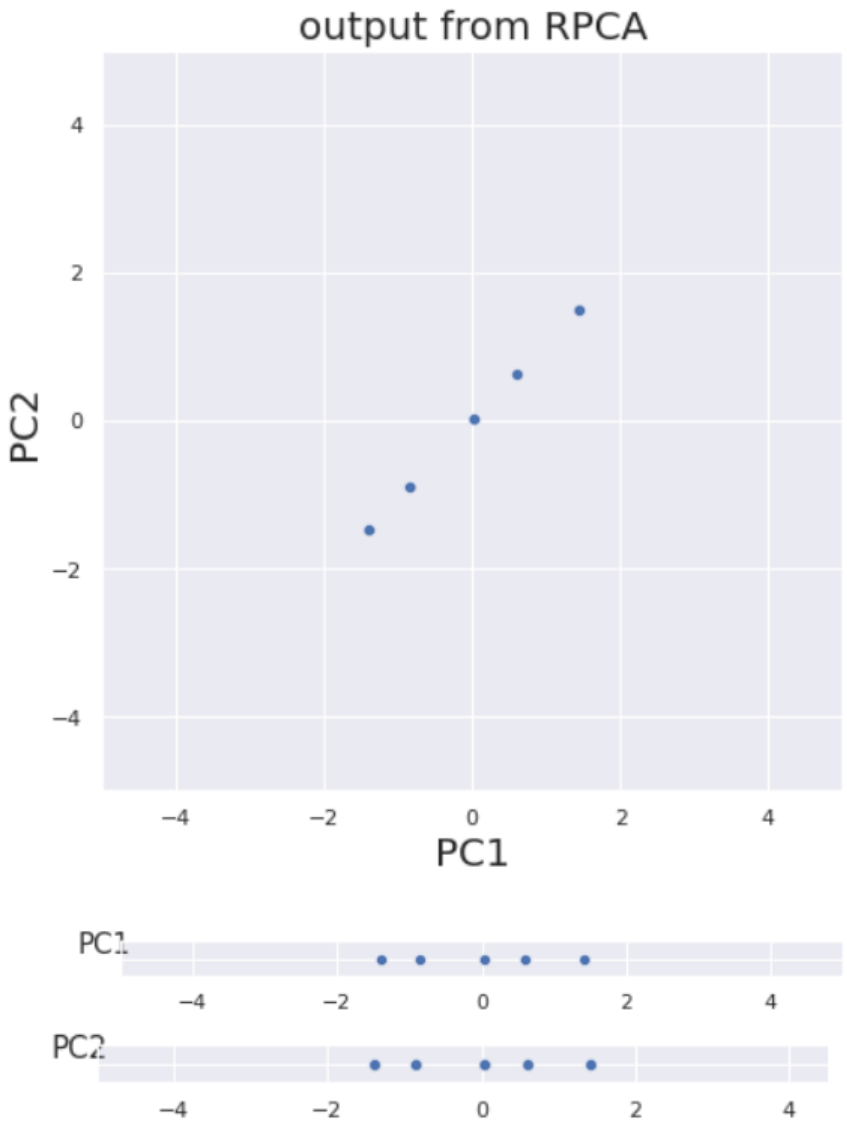


Figure 2.4: RPCA with outliers

From Fig 2.3 and Fig 2.4 we can observe that the outliers do not affect the output. This solves the problem seen in Fig 1.8

Chapter 3

CONNECTING THE COMPONENT'S

In this section we perform PCA and RPCA on the Yale B Database to remove noise from images. Further, additional noise is added to the data to test the performance of the algorithm. We try to decompose the image to try and split it into a low rank matrix and a sparse matrix.

The low rank matrix (L) will contain the dominant data in the images and the sparse matrix (S) will contain the noise. We try to achieve this using PCA and RPCA via Principal Component Pursuit.

Further, outliers(noise) will be added to test the performance of the algorithms.

3.1 Yale B database

The Yale B database contains 2414 frontal-face images with size 192×168 over 38 subjects and about 64 images per subject. A sample set of images is shown in Fig3.1

Figure 3.1: Dataset

For our implementation, we will be considering 2 individuals that is 64 and 62 images each since the image size is (192×168).

3.2 Principal Component Analysis

3.2.1 Original Dataset

Figure 3.2: Basic Reconstruction using PCA

Figure 3.3: Basic Reconstruction using PCA

3.2.2 Dataset with Salt and Pepper Noise

Figure 3.4: Salt and Pepper Reconstruction using PCA

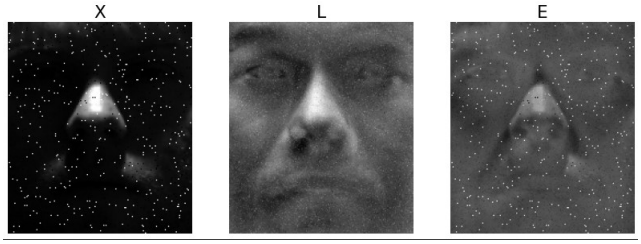


Figure 3.5: Salt and Pepper Reconstruction using PCA

3.2.3 Dataset with Salt and Pepper Noise and Patches

Figure 3.6: Salt, Pepper and Patch Reconstruction using PCA

Figure 3.7: Salt, Pepper and Patch Reconstruction using PCA

3.3 RPCA via Principal Component Pursuit by Alternating Directions

3.3.1 Original Dataset

Figure 3.8: Basic Reconstruction using RPCA

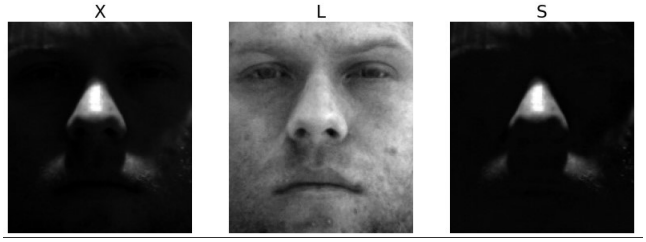


Figure 3.9: Basic Reconstruction using RPCA

3.3.2 Dataset with Salt and Pepper Noise

Figure 3.10: Salt and Pepper Reconstruction using RPCA

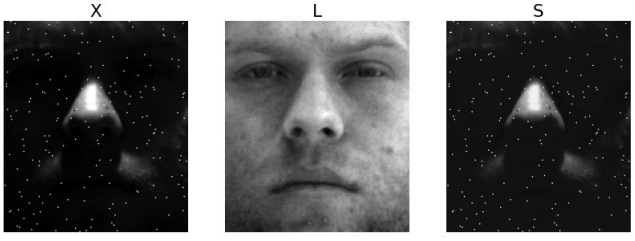


Figure 3.11: Salt and Pepper Reconstruction using RPCA

3.3.3 Dataset with Salt and Pepper Noise and Patches

Figure 3.12: Salt, Pepper and Patch Reconstruction using PCA

Figure 3.13: Salt, Pepper and Patch Reconstruction using PCA

Chapter 4

PROGRAMMING THE ROBOT

Chapter 5

STIMULATION

Conclusion

In this report PCA and RPCA were explored. We have seen that PCA is a very good technique for data compressing and dimensionality reduction. This technique also excels in other domains like facial recognition, computer vision, etc. Singular Value Decomposition also plays a key role in PCA. The problem with PCA arised when outliers are introduced.

To solve this problem RPCA via Principal Component Pursuit by Alternating Directions was applied to check whether outliers were affecting the algorithm or not.

By observing the results seen in 3.2 and 3.3, we can see that both the algorithms are good in removing noise. But upon the introduction outliers(in the form of noise), it is clearly evident that RPCA is a very good technique that tackles the problem of outliers after adding noises like salt and pepper and patches.

There are many other algorithms out there to implement RPCA such as RPCA via the inexact ALM method apart from the algorithm presented in this report.

References

1. http://databookuw.com/
2. https://www.youtube.com/watch?v=yDpz0PqULXQ&list=PLMrJAkhIeNNRHP5UA-gIimsXLQ
3. https://arxiv.org/pdf/0912.3599.pdf
4. https://www.samlau.me/test-textbook/ch/19/pca_svd.html
5. https://www.cs.utexas.edu/users/flame/laff/alaff/chapter02-best-approximatio
   html
6. https://yao-lab.github.io/2020.csic5011/slides/Lecture05_SDP.pdf
7. http://www.optimization-online.org/DB_FILE/2009/11/2447.pdf
8. https://angms.science/doc/LA/SVT_operator.pdf
9. https://angms.science/doc/LA/SVT_operator2.pdf
10. https://www.mat.univie.ac.at/~herman/papers/convexity.pdf
11. https://www.princeton.edu/~aaa/Public/Teaching/ORF523/S16/ORF523_
    S16_Lec2_gh.pdf
12. https://en.wikipedia.org/wiki/Matrix_norm

Appendix

5.1 Code

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [7]:
```
import seaborn as sns
```

In [3]:
```
cm_path = 'child_mortality_0_5_year_olds_dying_per_1000_born.csv'
fe_path = 'children_per_woman_total_fertility.csv'
cm = pd.read_csv(cm_path).set_index('country')['2017'].to_frame()/10
fe = pd.read_csv(fe_path).set_index('country')['2017'].to_frame()
child_data = cm.merge(fe, left_index=True, right_index=True).dropna()
child_data.columns = ['mortality', 'fertility']
child_data.head()
```

Out[3]:

country	mortality	fertility
Afghanistan	6.470	4.48
Angola	8.040	5.62
Albania	0.902	1.71
United Arab Emirates	0.768	1.73
Argentina	1.040	2.28

In [4]:
```
cntr_child = child_data - child_data.mean(axis=0)
cntr_child
```

Out[4]:

country	mortality	fertility
Afghanistan	3.535167	1.726828
Angola	5.105167	2.866828
Albania	-2.032833	-1.043172
United Arab Emirates	-2.166833	-1.023172
Argentina	-1.894833	-0.473172
...	...	...
Samoa	-1.304833	1.176828
Yemen	2.565167	1.136828
South Africa	0.595167	-0.323172
Zambia	3.005167	2.176828
Zimbabwe	1.995167	0.926828

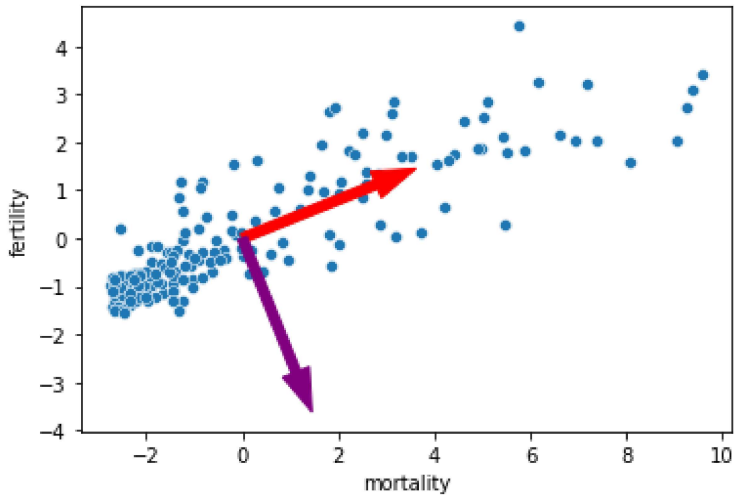
186 rows × 2 columns

In [5]:
```
svd = np.linalg.svd
```

In [6]:
```
U, S, Vt = svd(cntr_child, full_matrices=False)
Vt
```

Out[6]:
```
array([[-0.92818863, -0.37211002],
       [ 0.37211002, -0.92818863]])
```

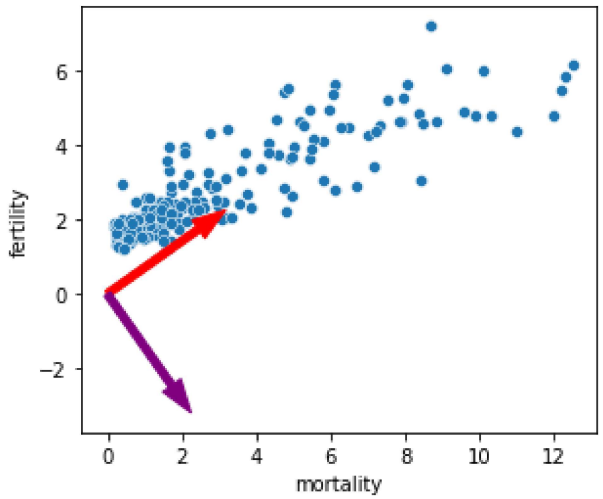
In [8]:
```
sns.scatterplot(data=cntr_child, x='mortality', y='fertility')
plt.arrow(0, 0, -3 * Vt[0, 0], -3 * Vt[0, 1], width=0.2, color='red')
plt.arrow(0, 0, 3 * Vt[1, 0], 3 * Vt[1, 1], width=0.2, color='purple')
plt.gca().set_aspect(1)
```



In [9]:
```
U, S, Vt = svd(child_data, full_matrices=False)
Vt
```

Out[9]:
```
array([[-0.81663257, -0.57715791],
       [ 0.57715791, -0.81663257]])
```

In [10]:
```
sns.scatterplot(data=child_data, x='mortality', y='fertility')
plt.arrow(0, 0, -3 * Vt[0, 0], -3 * Vt[0, 1], width=0.2, color='red')
plt.arrow(0, 0, 3 * Vt[1, 0], 3 * Vt[1, 1], width=0.2, color='purple')
plt.gca().set_aspect(1)
```


fertility
mortality
6
4
2
0
−2
0
2
4
6
8
10
12

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
def make_fig(X, Vt, title):
    plt.rcParams['figure.figsize'] = [7, 7]
    sns.set_theme()
    gfg = sns.scatterplot(x=X[:,0], y=X[:,1])
    plt.arrow(0, 0, -0.3 * Vt[0, 0], -0.3 * Vt[0, 1], width=0.1, color='red')
    plt.arrow(0, 0, 0.3 * Vt[1, 0], 0.3 * Vt[1, 1], width=0.1, color='purple')
    gfg.set_ylim(-5, 5)
    gfg.set_xlim(-5, 5)
    gfg.set_xlabel("x-coordinates ", fontsize = 20)
    gfg.set_ylabel("y-coordinates ", fontsize = 20)
    gfg.set_title(title, fontsize = 20)
    plt.gca().set_aspect(1)
```

In [3]:
```
def make_fig_projected(X, title):
    sns.set_theme()
    gfg = sns.scatterplot(x=X[:,0], y=X[:,1])
    gfg.set_ylim(-5, 5)
    gfg.set_xlim(-5, 5)
    gfg.set_xlabel("PC1", fontsize = 20)
    gfg.set_ylabel("PC2", fontsize = 20)
    gfg.set_title(title, fontsize = 20)
    plt.gca().set_aspect(1)
```

In [4]:
```
def line_plot(data, title):
    sns.set_theme()
    gfg = sns.scatterplot(x=data, y=np.array([0,0,0,0,0]))
    gfg.set_ylim(-0.25, 0.25)
    gfg.set_xlim(-5, 5)
    gfg.set_ylabel(title, fontsize = 15, rotation=0)
    plt.gca().set_aspect(1)
    gfg.yaxis.labelpad = 0
    gfg.set(yticklabels=[]);
```

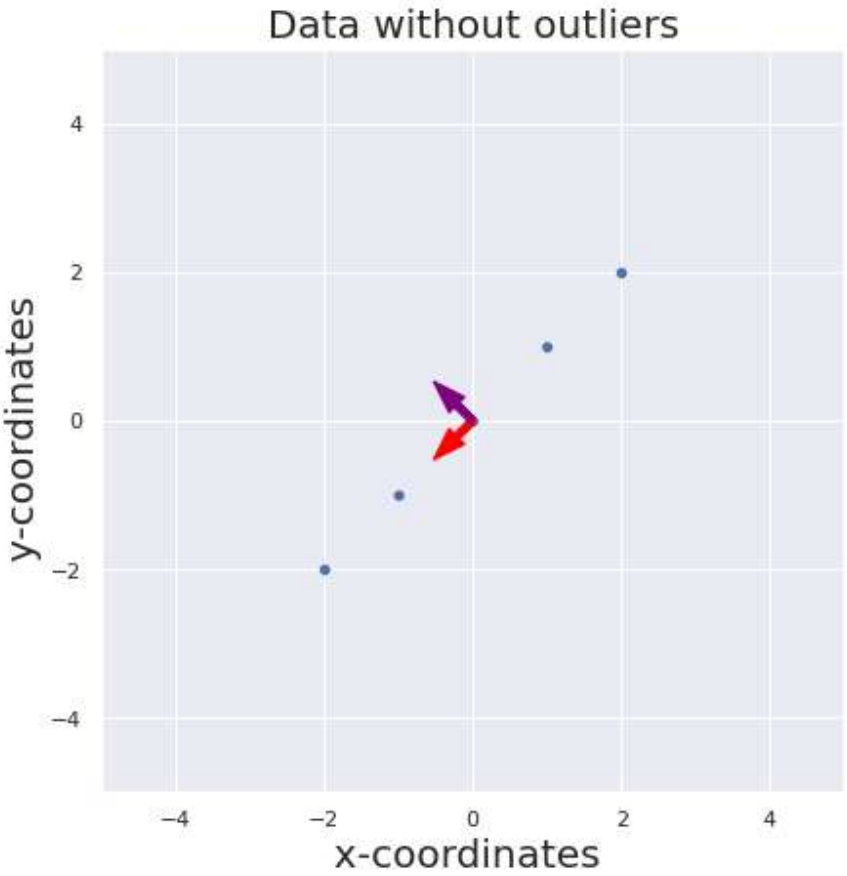
In [5]:
```
x = np.array([1,2,3,4,5]).reshape(-1,1)
y = np.array([1,2,3,4,5]).reshape(-1,1)
X = np.concatenate((x, y), axis=1)

x_o = np.array([1,2,3,1,5]).reshape(-1,1)
y_o = np.array([1,2,3,-1.5,5]).reshape(-1,1)
X_outliers = np.concatenate((x_o, y_o), axis=1)
```

In [6]:
```
X = X - np.mean(X, axis=0)
U, S, Vt = np.linalg.svd(X, full_matrices = False)
Vt
```

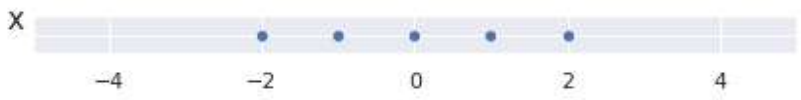
Out[6]:
```
array([[ 0.70710678,  0.70710678],
       [-0.70710678,  0.70710678]])
```

In [7]:
```
projected_data = X @ Vt.T
```

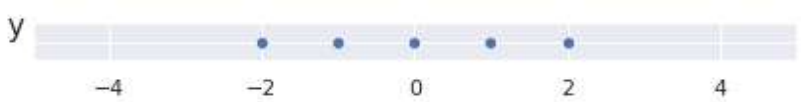
In [8]:
```
make_fig(X, Vt, "Data without outliers")
```



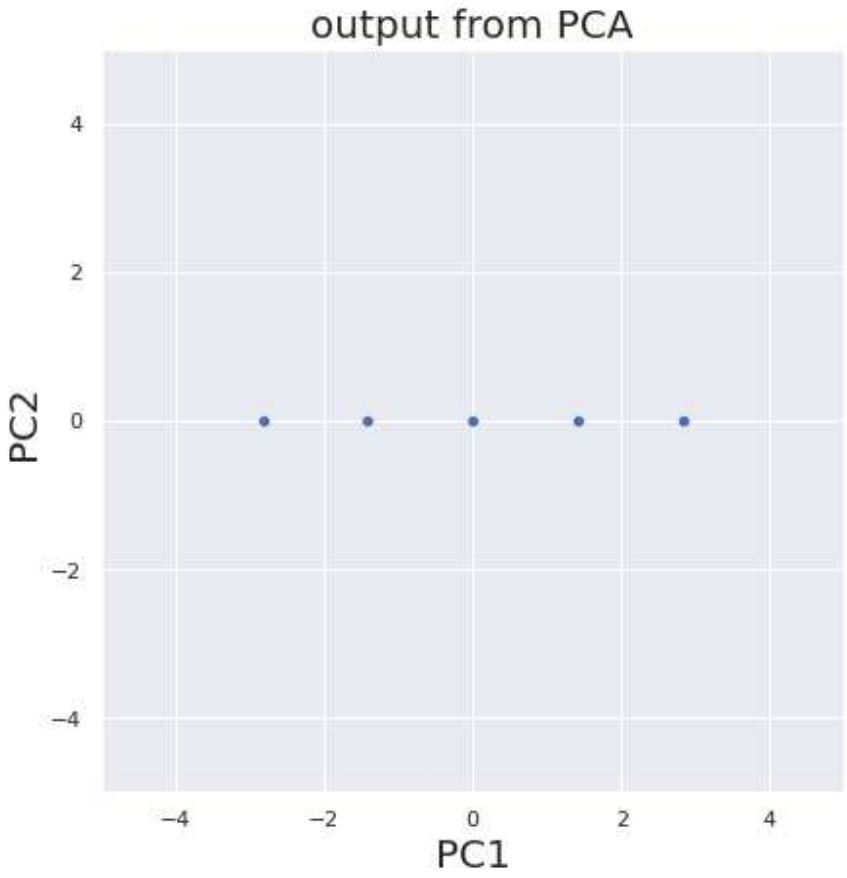
In [9]:
```
line_plot(X[:,0], "x")
```



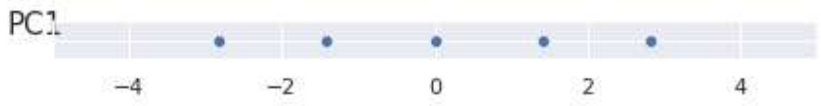
In [10]:
```
line_plot(X[:,1], "y")
```



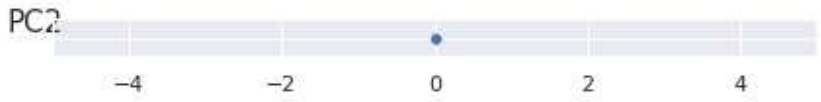
In [11]:
```
make_fig_projected(projected_data, "output from PCA")
```



In [12]:
```
line_plot(projected_data[:,0], "PC1")
```



In [13]:
```
line_plot(projected_data[:,1], "PC2")
```



In [14]:
```
X_outliers = X_outliers - np.mean(X_outliers, axis=0)
U2, S2, Vt2 = np.linalg.svd(X_outliers, full_matrices = False)
Vt2
```

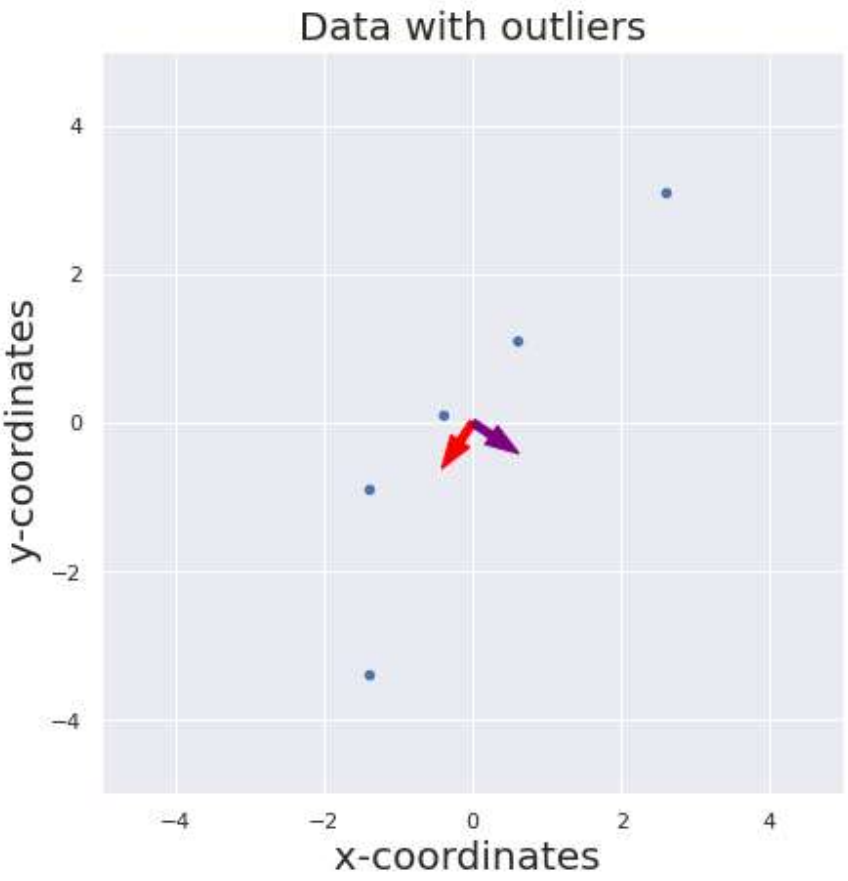
Out[14]:
```
array([[ 0.55771991,  0.83002922],
       [ 0.83002922, -0.55771991]])
```

In [15]:
```
projected_data = X_outliers @ Vt2.T
projected_data
```

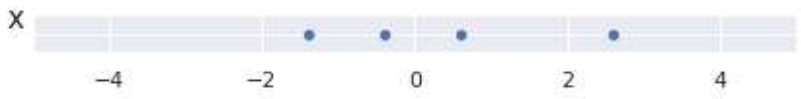
Out[15]:
```
array([[-1.52783417, -0.66009298],
       [-0.14008504, -0.38778368],
       [ 1.24766408, -0.11547437],
```

```
       [-3.60290721,  0.7342068 ],
       [ 4.02316234,  0.42914423]])
```

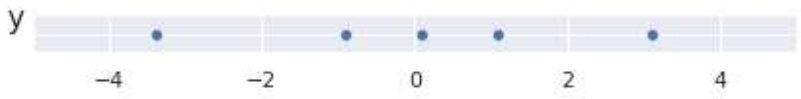
In [16]:
```
make_fig(X_outliers, Vt2, "Data with outliers")
```



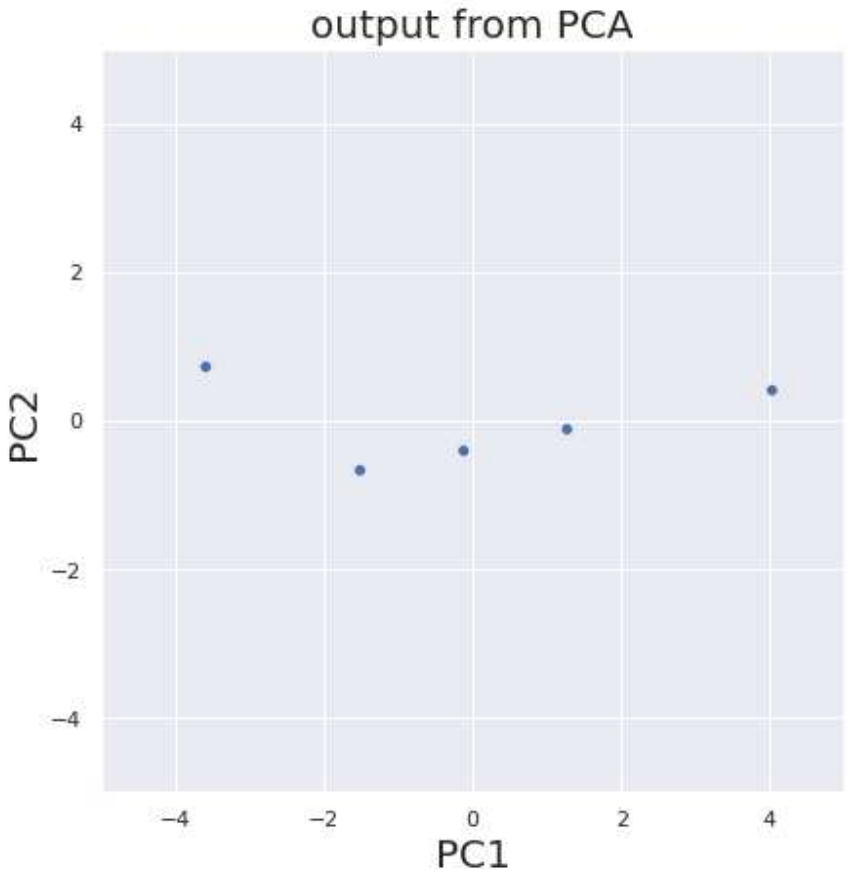
In [17]:
```
line_plot(X_outliers[:,0], "x")
```



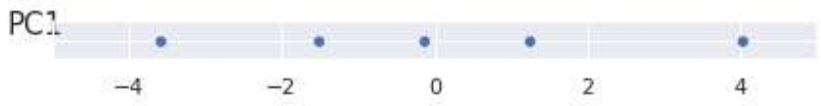
In [18]:
```
line_plot(X_outliers[:,1], "y")
```



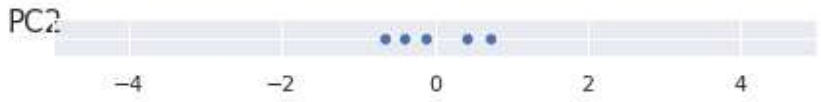
In [19]:
```
make_fig_projected(projected_data, "output from PCA")
```



In [20]:
```
line_plot(projected_data[:,0], "PC1")
```



In [21]:
```
line_plot(projected_data[:,1], "PC2")
```



RPCA

In [22]:
```
def shrink(X,tau):
    Y = np.abs(X)-tau
    return np.sign(X) * np.maximum(Y,np.zeros_like(Y))
def SVT(X,tau):
    U,S,VT = np.linalg.svd(X,full_matrices=0)
    out = U @ np.diag(shrink(S,tau)) @ VT
    return out
def RPCA(X):
    n1,n2 = X.shape
    mu = n1*n2/(4*np.sum(np.abs(X.reshape(-1))))
    lambd = 1/np.sqrt(np.maximum(n1,n2))
```

```
    thresh = 10**(-7) * np.linalg.norm(X)

    S = np.zeros_like(X)
    Y = np.zeros_like(X)
    L = np.zeros_like(X)
    count = 0
    while (np.linalg.norm(X-L-S) > thresh) and (count < 1000):
        L = SVT(X-S+(1/mu)*Y,1/mu)
        S = shrink(X-L+(1/mu)*Y,lambd/mu)
        Y = Y + mu*(X-L-S)
        count += 1
    return L,S
```

In [23]:

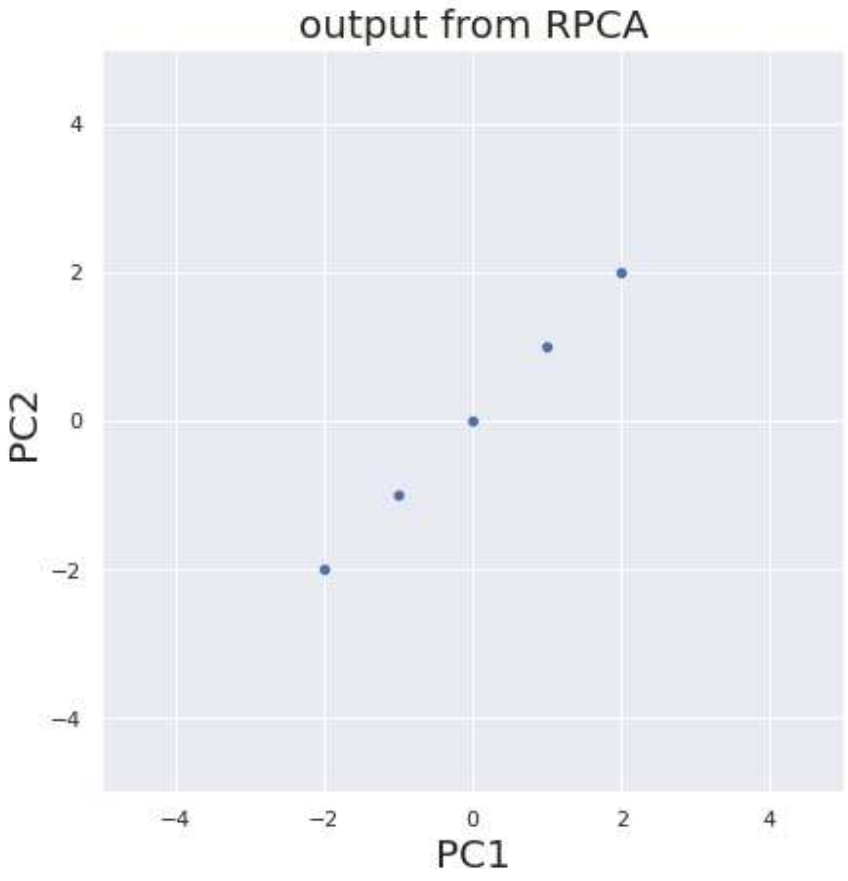
```
L,S = RPCA(X)
L
```

Out[23]:

```
array([[-2., -2.],
       [-1., -1.],
       [ 0.,  0.],
       [ 1.,  1.],
       [ 2.,  2.]])
```

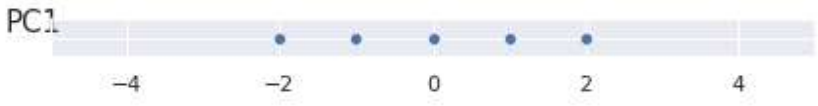
In [24]:

```
make_fig_projected(L, "output from RPCA")
```

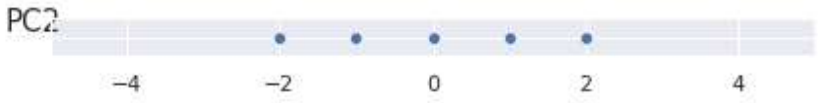


In [25]:

```
line_plot(L[:,0], "PC1")
```



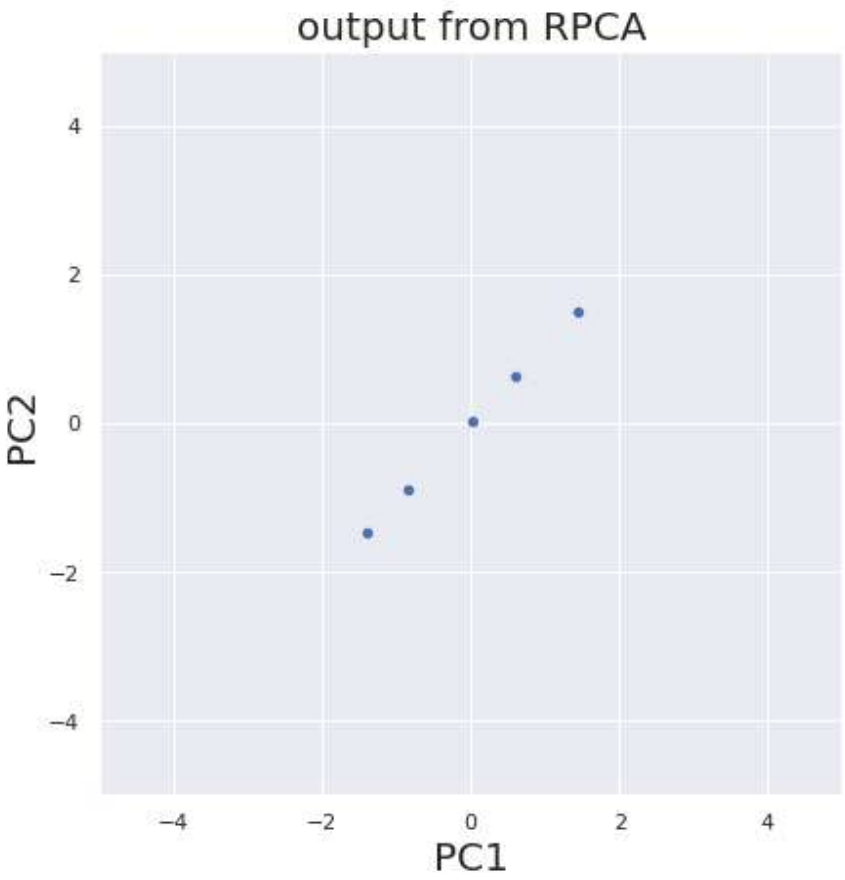
In [26]:
```
line_plot(L[:,0], "PC2")
```



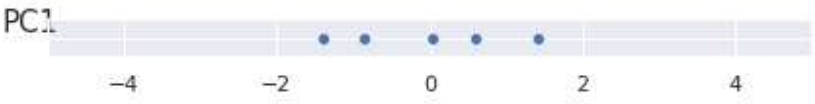
In [27]:
```
L2,S2 = RPCA(X_outliers)
L2
```

Out[27]:
```
array([[-0.85941598, -0.8999998 ],
       [ 0.03292945,  0.03448446],
       [ 0.60000049,  0.62833405],
       [-1.4       , -1.46611158],
       [ 1.42757837,  1.49499227]])
```

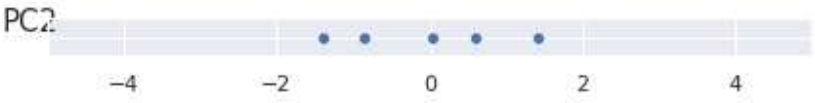
In [28]:
```
make_fig_projected(L2, "output from RPCA")
```



In [29]:
```
line_plot(L2[:,0], "PC1")
```



In [30]:
```
line_plot(L2[:,0], "PC2")
```



In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import os
import scipy.io

plt.rcParams['figure.figsize'] = [14, 14]
plt.rcParams.update({'font.size': 18})

mat = scipy.io.loadmat("/mnt/c/Users/nikhi/Documents/sem4/mis/project/allFaces.mat")
faces = mat['faces']
nfaces = mat['nfaces'].reshape(-1)
```

In [3]:
```
def add_noise(img):

    # Getting the dimensions of the image
    row , col = img.shape

    # Randomly pick some pixels in the
    # image for coloring them white
    # Pick a random number between 300 and 10000
    number_of_pixels = np.random.randint(300, 10000)*5
    for i in range(number_of_pixels):

        # Pick a random y coordinate
        y_coord= np.random.randint(0, row - 1)

        # Pick a random x coordinate
        x_coord= np.random.randint(0, col - 1)

        # Color that pixel to white
        img[y_coord][x_coord] = 255

    # Randomly pick some pixels in
    # the image for coloring them black
    # Pick a random number between 300 and 10000
    number_of_pixels = np.random.randint(300 , 10000)*5
    for i in range(number_of_pixels):

        # Pick a random y coordinate
        y_coord= np.random.randint(0, row - 1)

        # Pick a random x coordinate
        x_coord= np.random.randint(0, col - 1)

        # Color that pixel to black
        img[y_coord][x_coord] = 0

    return img
```

In [4]:
```
def patch_noise(data):
    no_of_pixels , no_of_images = data.shape
    patch_size = 40
    for i in range(no_of_images):
        image = data[:, i]
        image = image.reshape(168, 192)
```

```
        row_start = np.random.randint(0, image.shape[0]-patch_size)
        row_end = row_start + patch_size
        col_start = np.random.randint(0, image.shape[1]-patch_size)
        col_end = col_start + patch_size
        image[row_start:row_end, col_start:col_end] = 0
        data[:,i] = image.reshape(-1)

    return data
```

In [5]:

```
def plt_singular_values(data):
    average_face = data.mean(axis=0)
    centered = data - average_face
    U, S, Vt = np.linalg.svd(data, full_matrices = False)
    fig, ax = plt.subplots()
    ax.plot(S)
```

In [6]:

```
def compute_rank_k_approximation(data, k):
    average_face = data.mean(axis=0)
    centered = data - average_face
    U, S, Vt = np.linalg.svd(data, full_matrices = False)
    approx = (U @ np.diag(S))[:, :k] @ Vt[:k]
    L = approx + average_face
    E = data - L
    return L,E
```

In [7]:

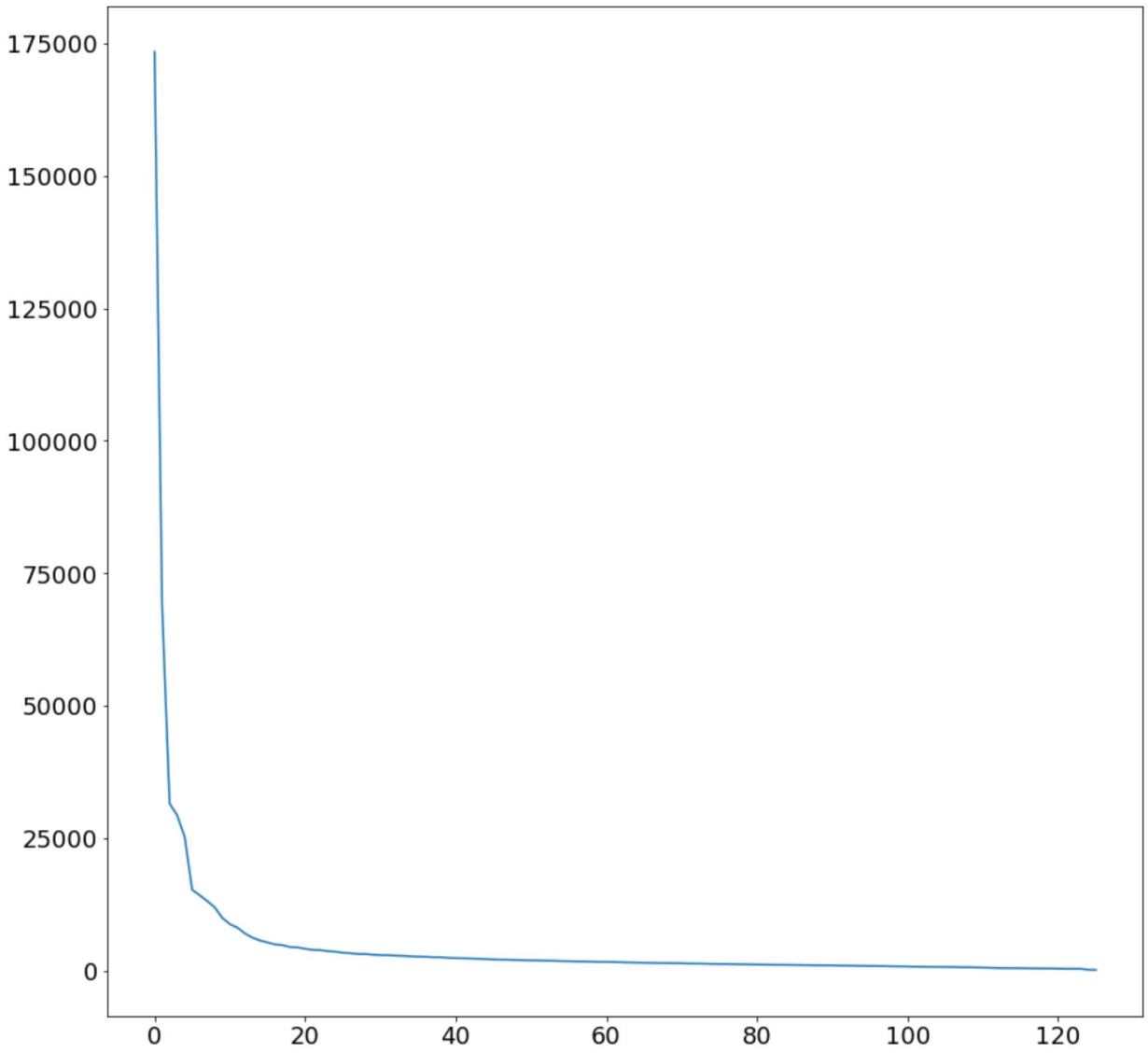
```
def my_plot(X,L,S, k):
    fig,axs = plt.subplots(1,3)
    axs = axs.reshape(-1)
    axs[0].imshow(np.reshape(X[:,k-1],(168,192)).T,cmap='gray')
    axs[0].set_title('X')
    axs[1].imshow(np.reshape(L[:,k-1],(168,192)).T,cmap='gray')
    axs[1].set_title('L')
    axs[2].imshow(np.reshape(S[:,k-1],(168,192)).T,cmap='gray')
    axs[2].set_title('E')
    for ax in axs:
        ax.axis('off')
```

In [8]:

```
X = faces[:,:nfaces[0]+nfaces[1]]
```

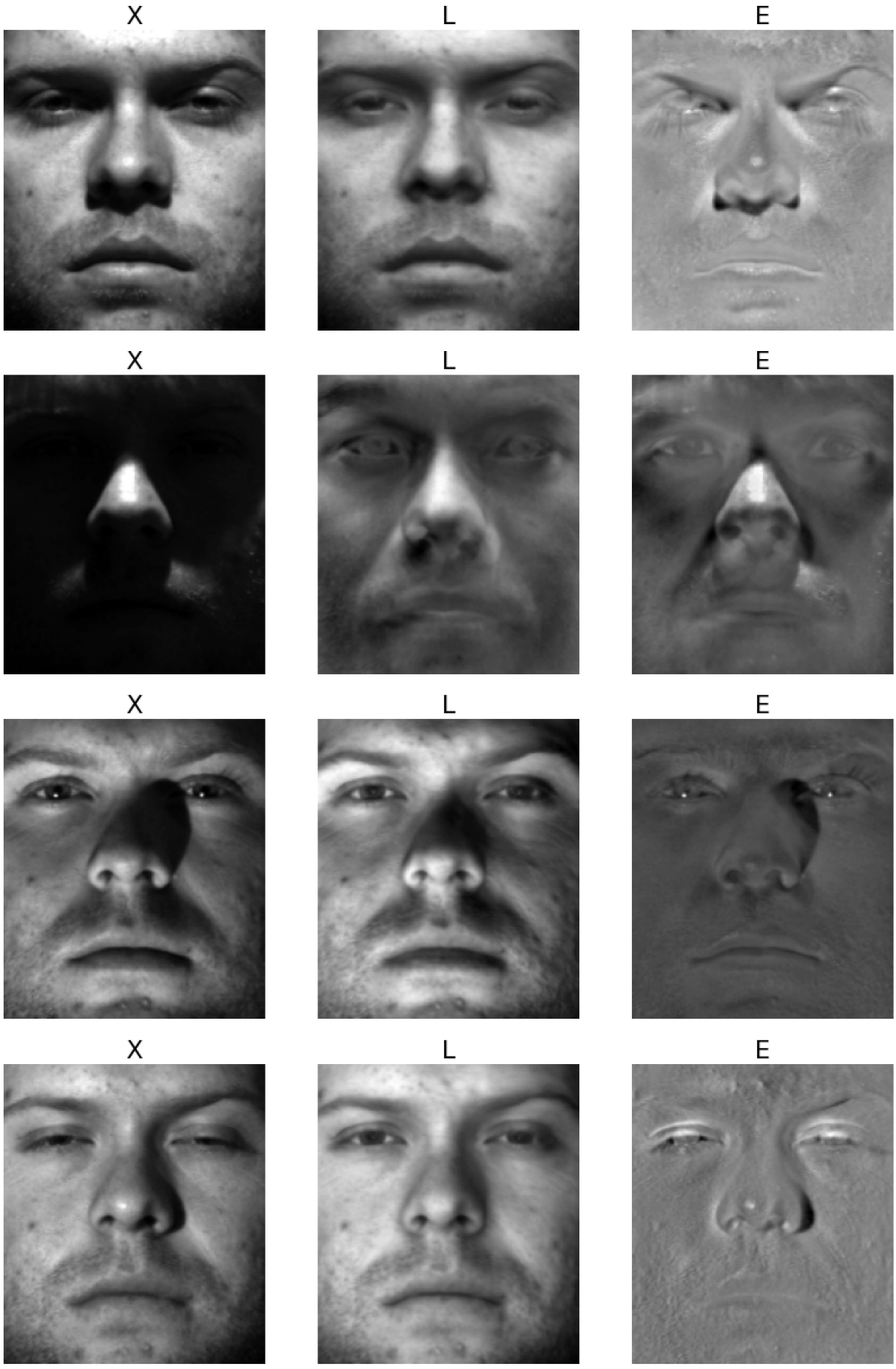
In [9]:

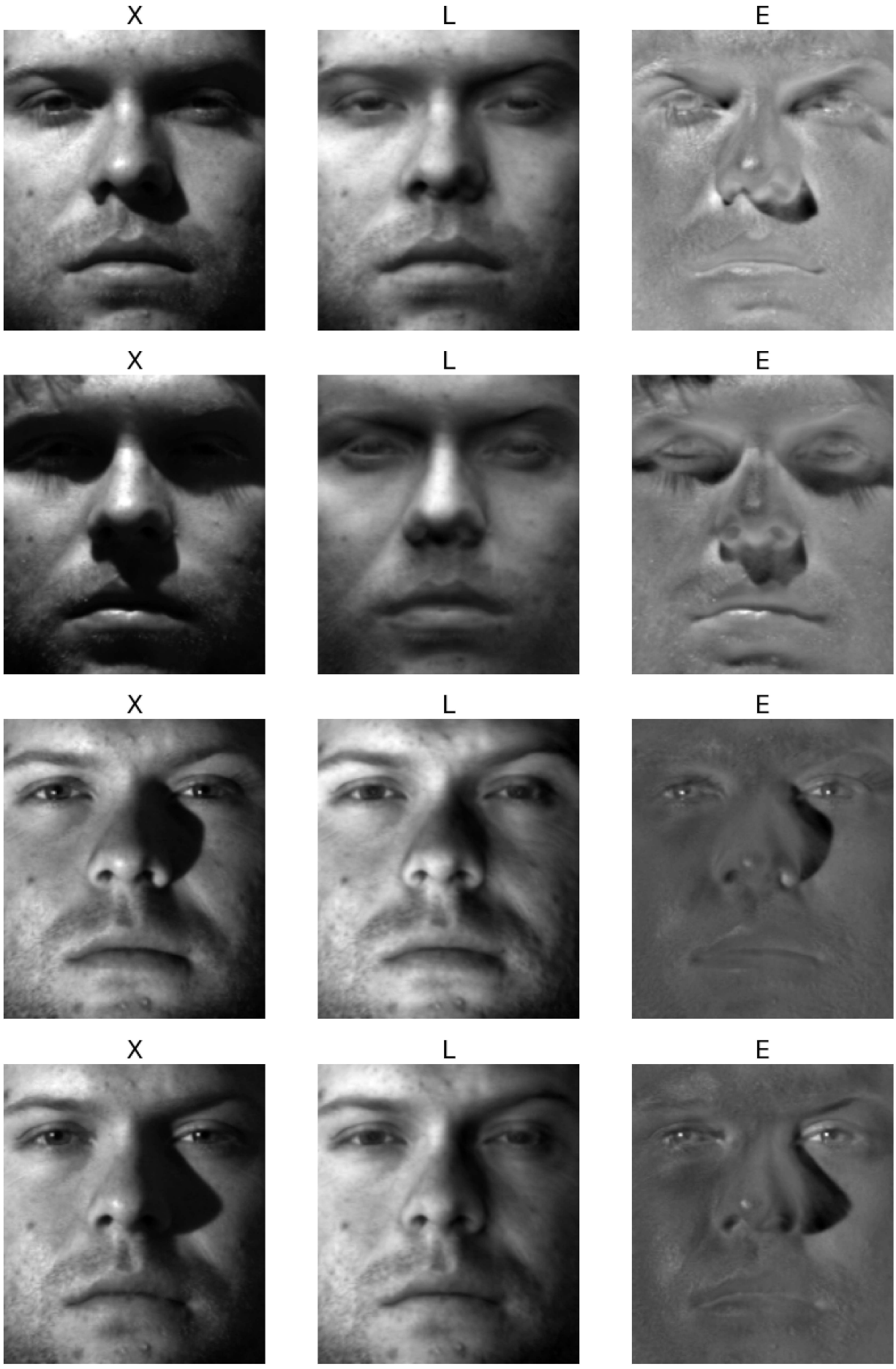
```
plt_singular_values(X)
```

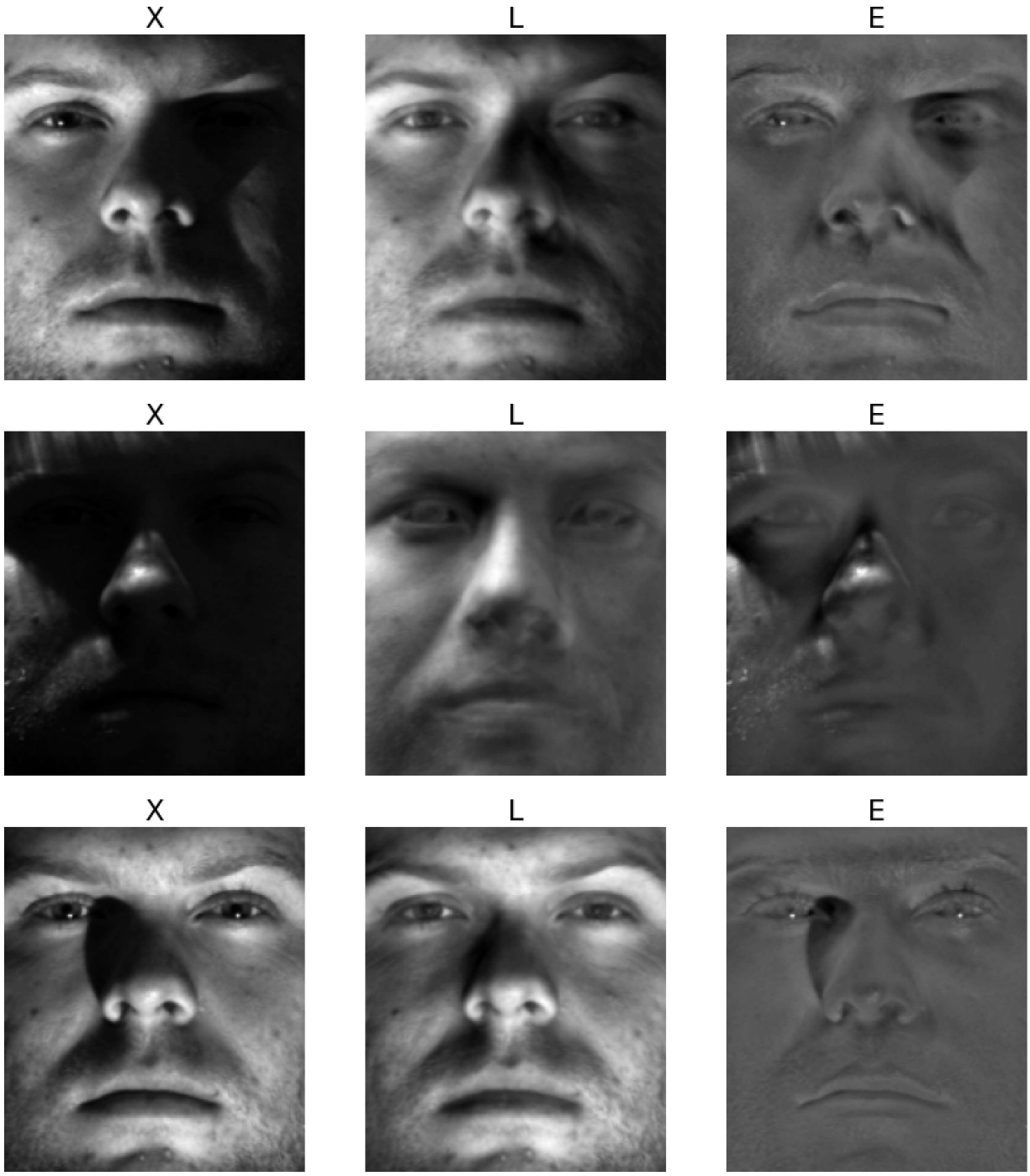


In [10]:
```
L,S = compute_rank_k_approximation(X, 10)
```

In [11]:
```
inds = (3,4,14,15,17,18,19,20,21,32,43)

for i in inds:
    my_plot(X,L,S, i)
```

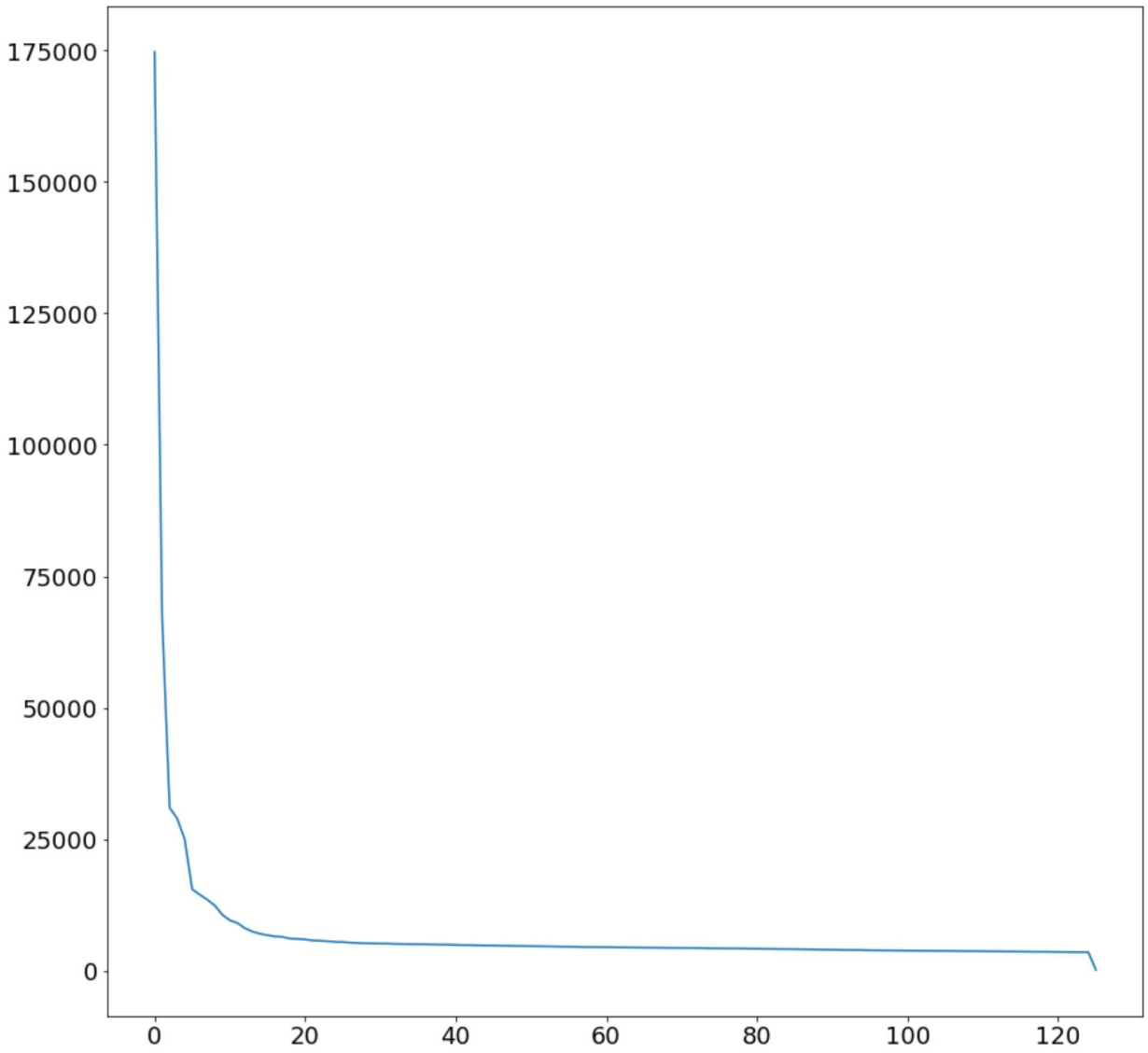

X
L
E
X
L
E
X
L
E
X
L
E


X
L
E
X
L
E
X
L
E
X
L
E



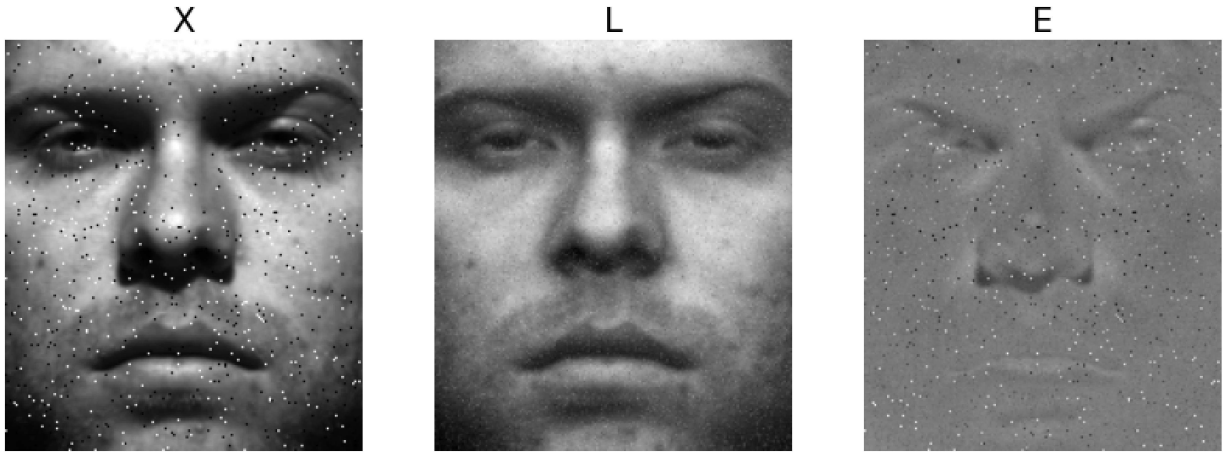
In [12]:
```
X2 = add_noise(X)
```

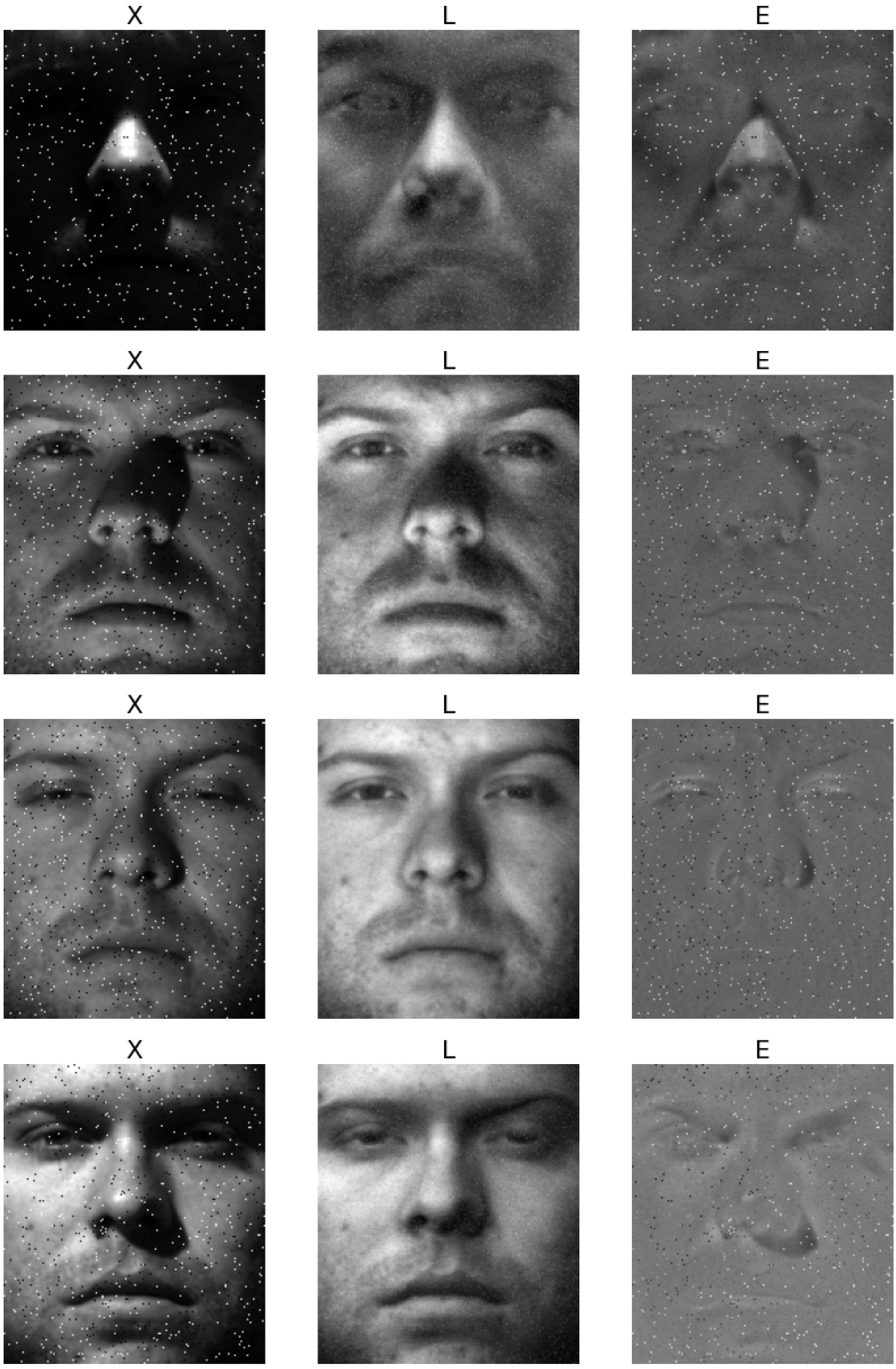
In [13]:
```
plt_singular_values(X2)
```

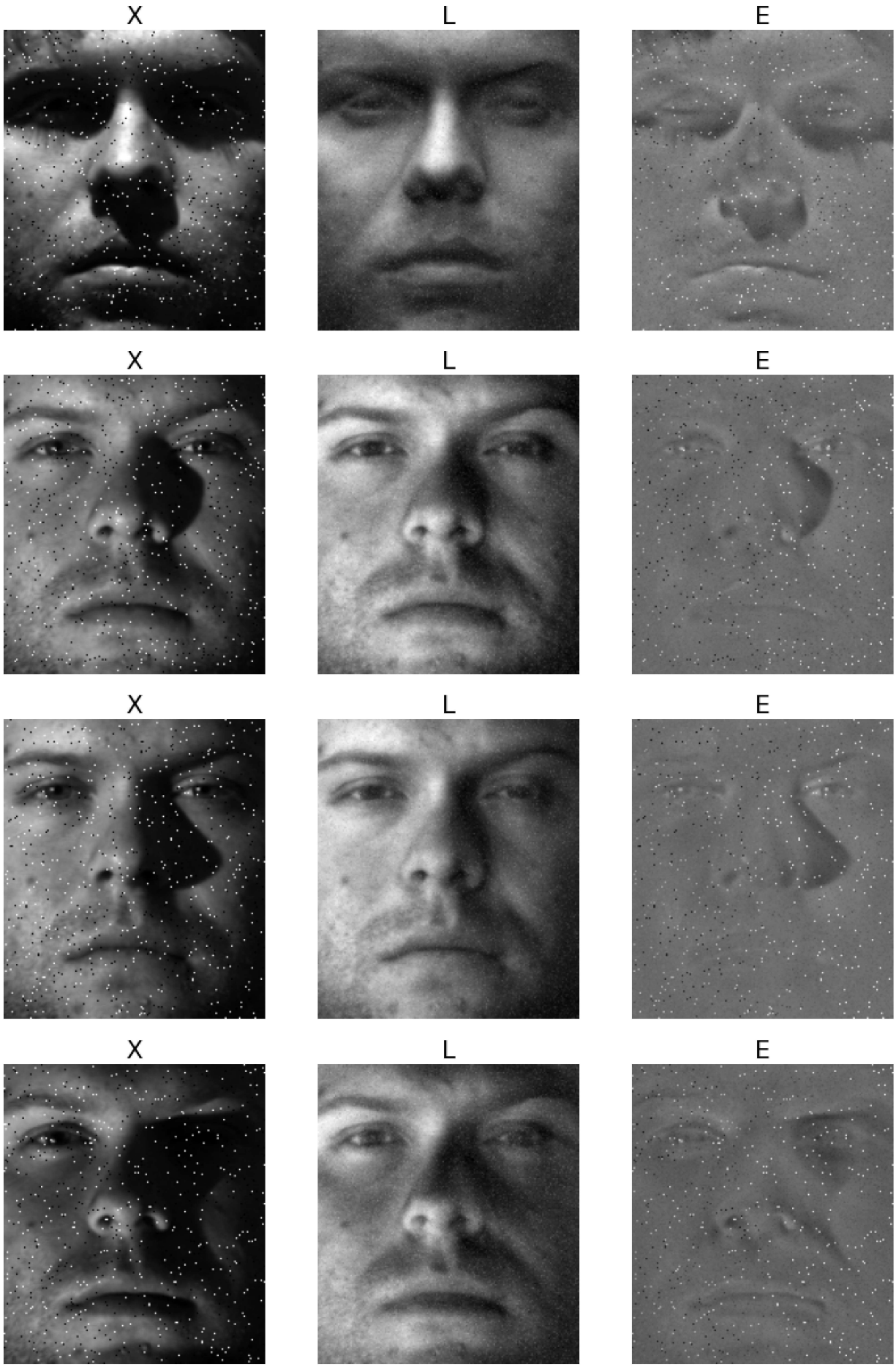


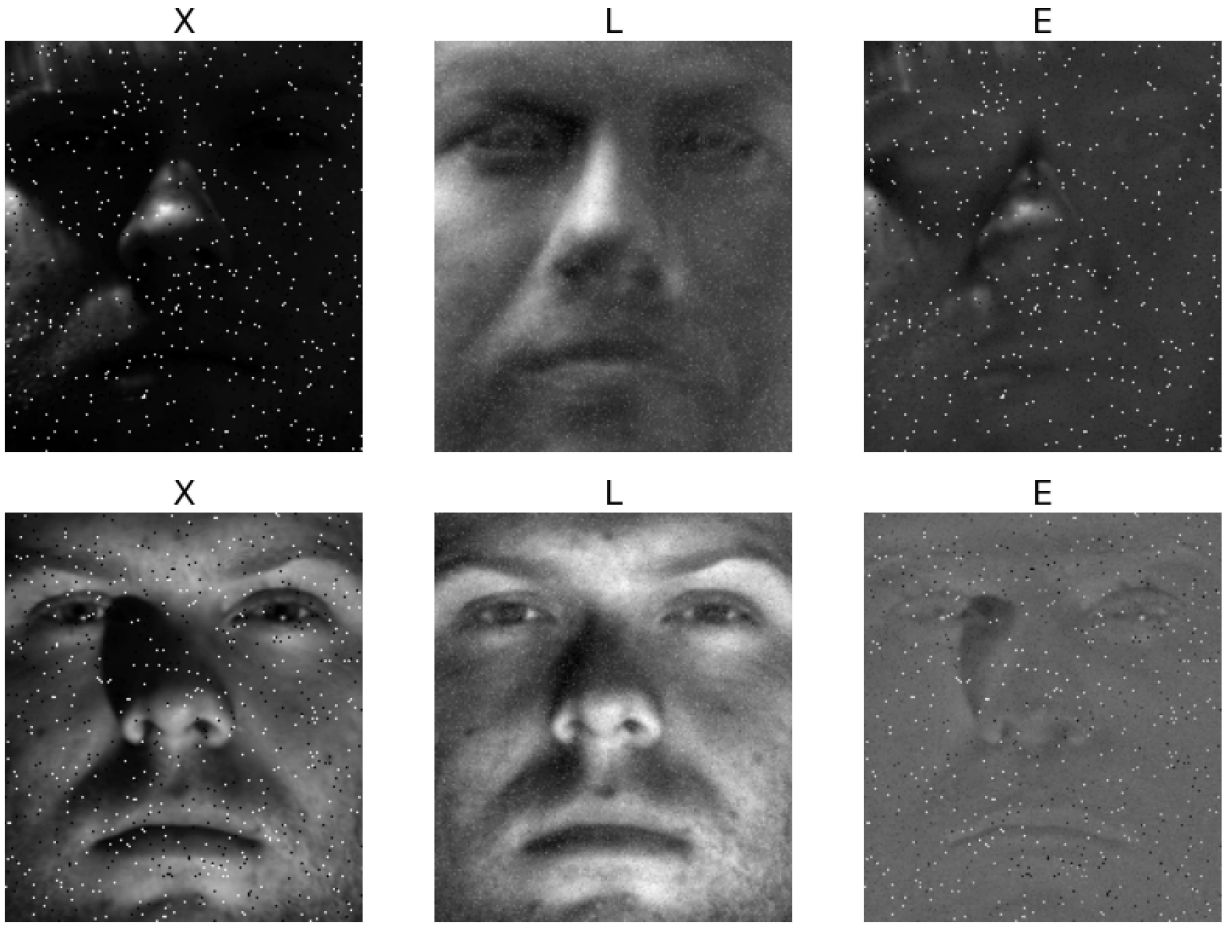
In [14]:
```
L2,S2 = compute_rank_k_approximation(X2, 10)
```

In [15]:
```
for i in inds:
    my_plot(X2,L2,S2, i)
```



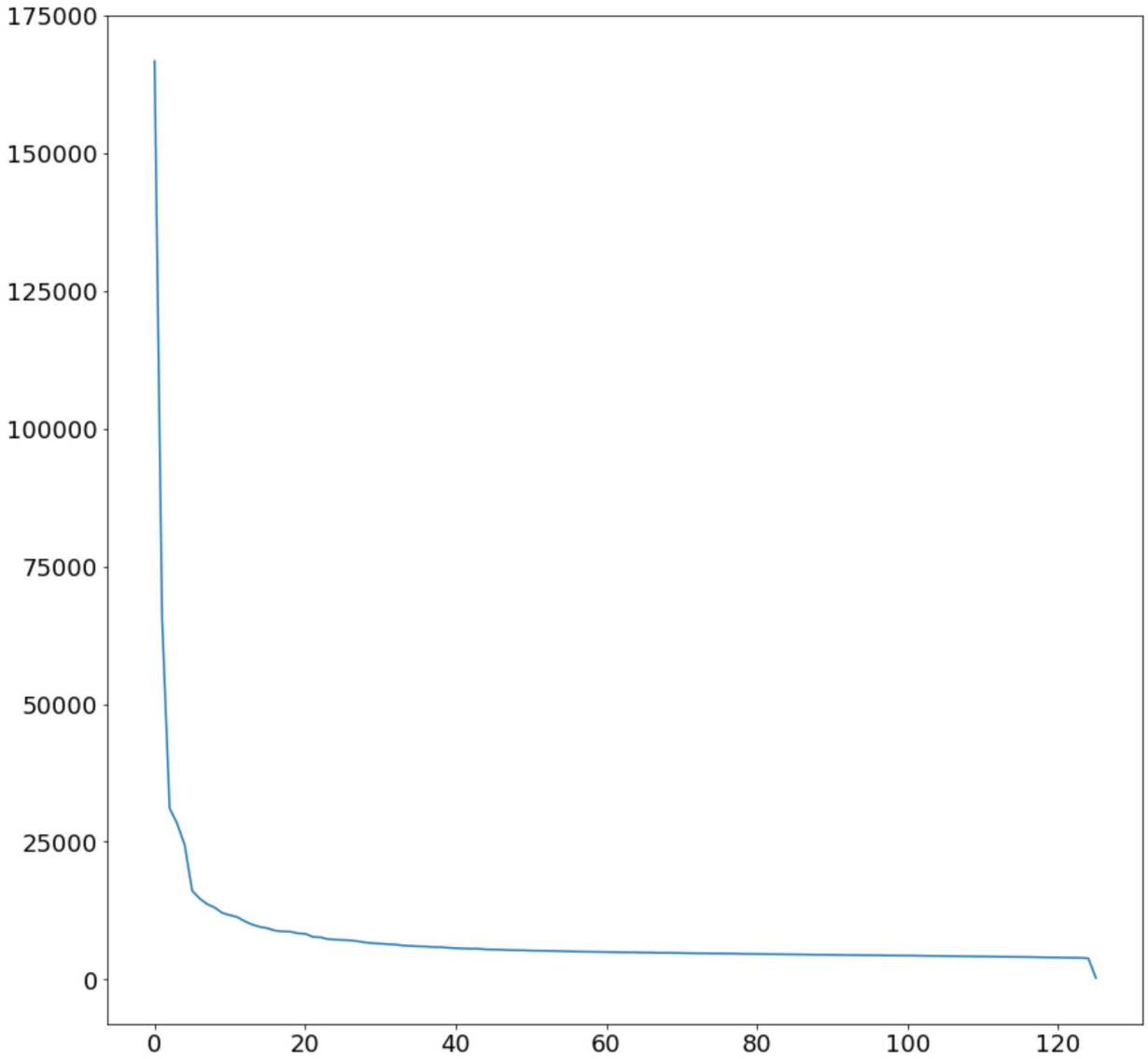

X
L
E
X
L
E
X
L
E
X
L
E


X
L
E
X
L
E
X
L
E
X
L
E



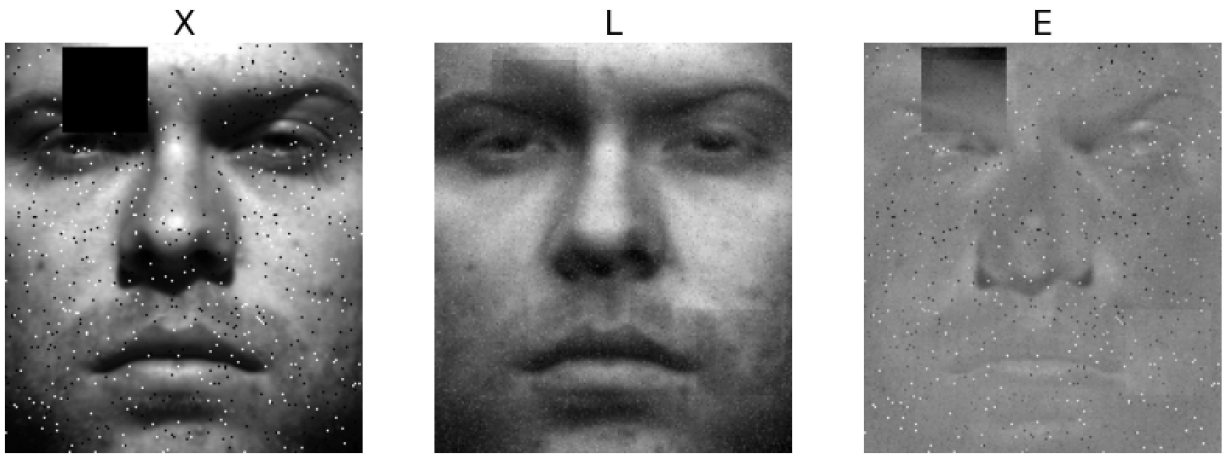
In [16]:
```
X3 = patch_noise(X)
```

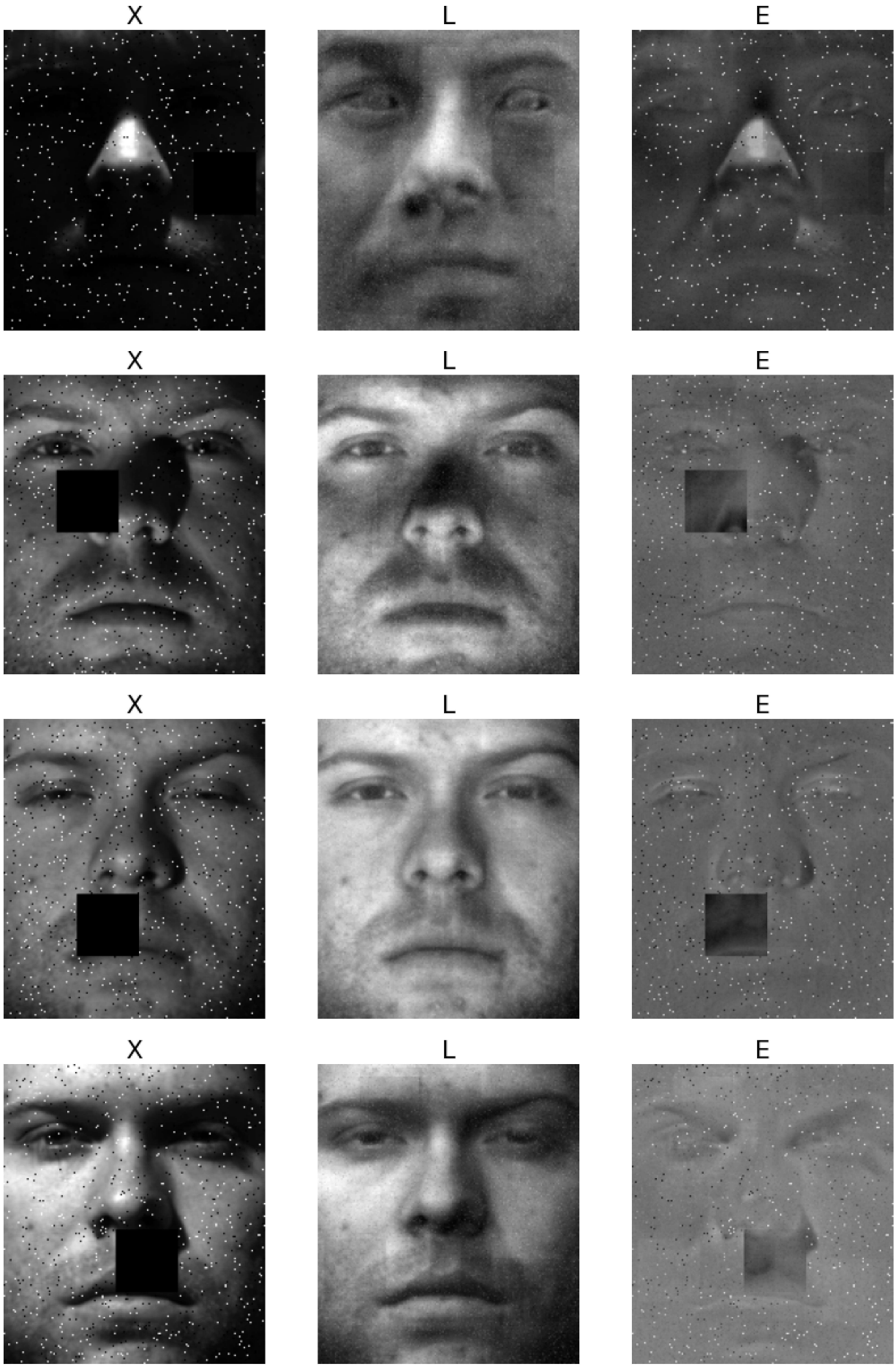
In [17]:
```
plt_singular_values(X3)
```

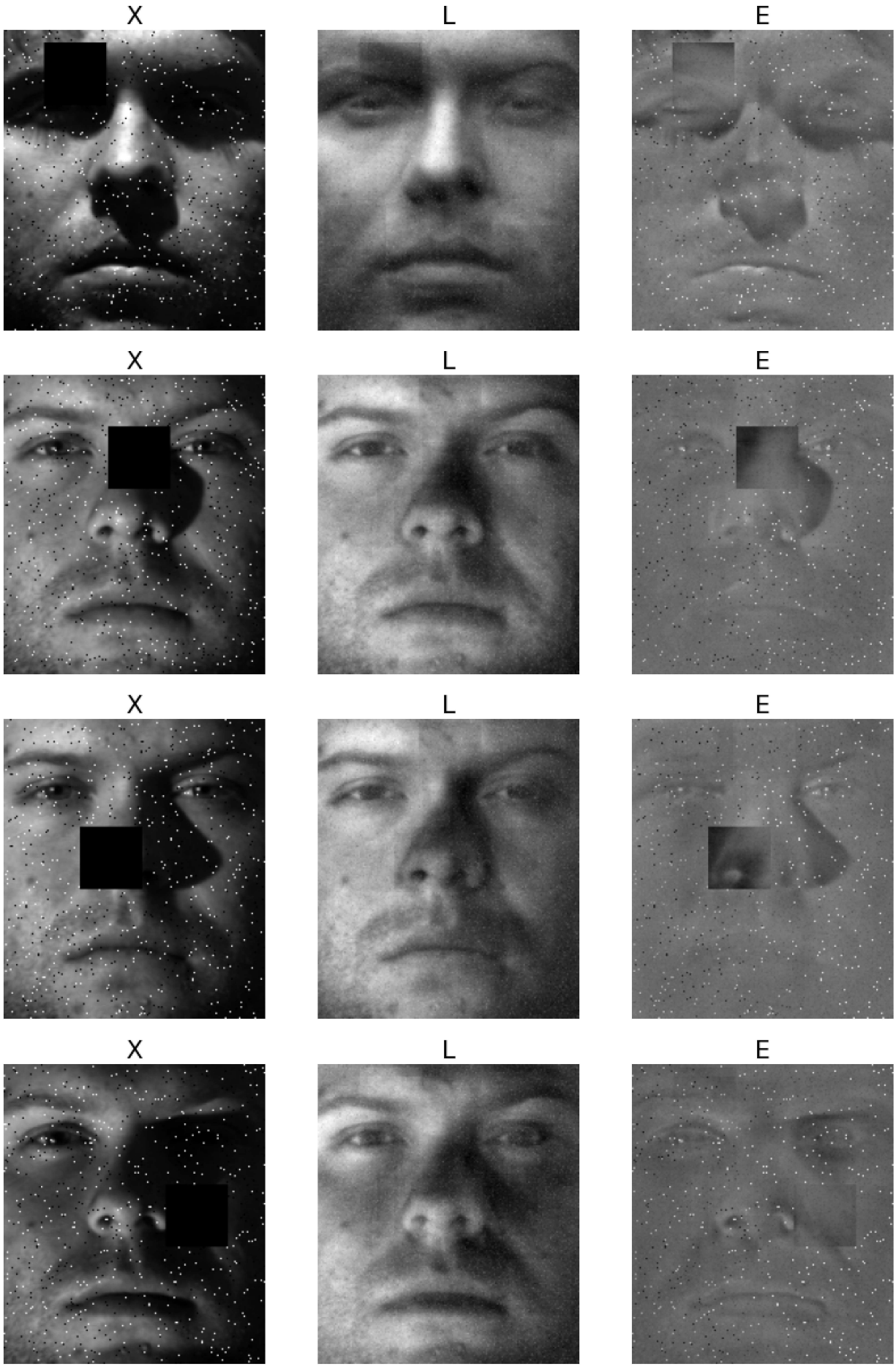


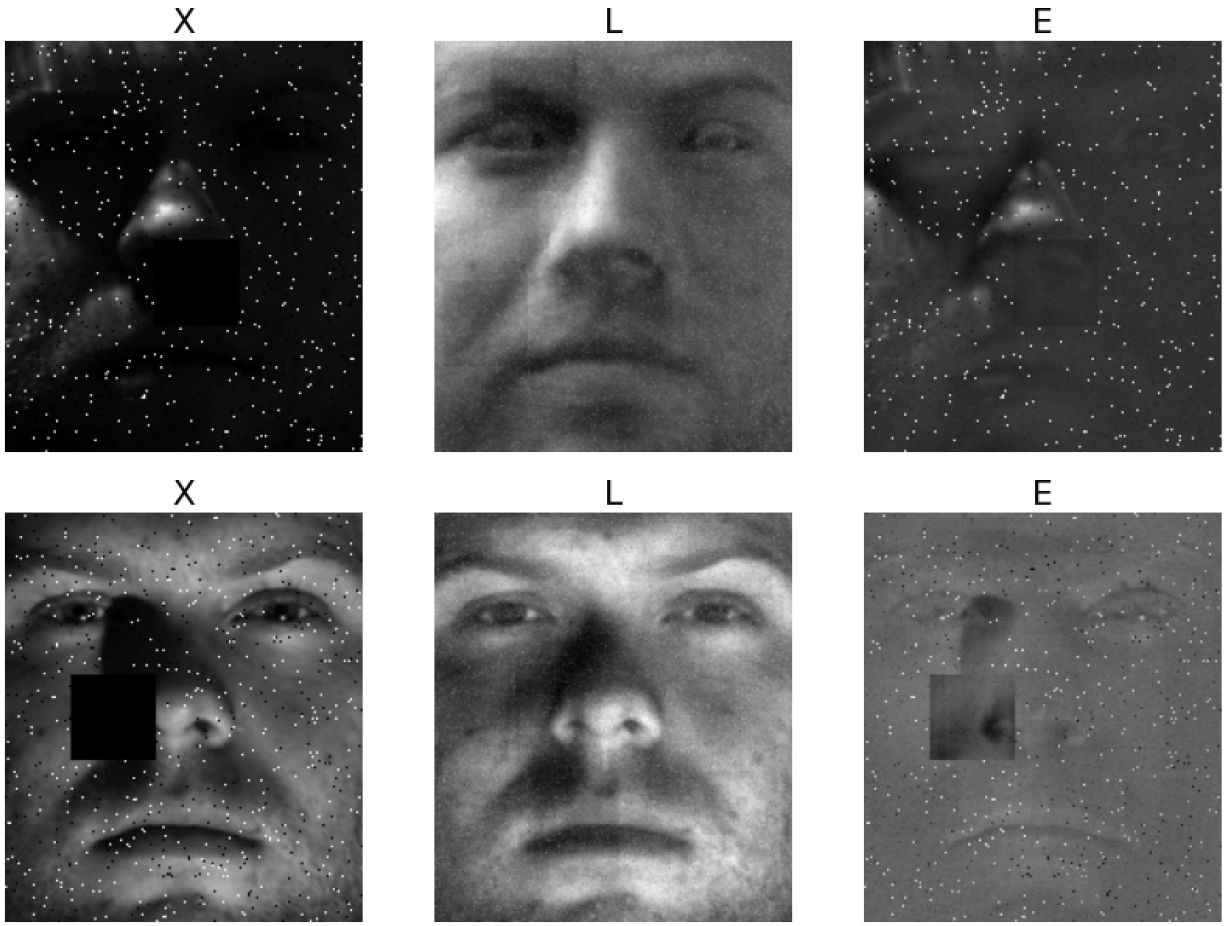
In [18]:
```
L3,S3 = compute_rank_k_approximation(X3, 10)
```

In [19]:
```
for i in inds:
    my_plot(X3,L3,S3, i)
```




X
L
E
X
L
E
X
L
E
X
L
E


X
L
E
X
L
E
X
L
E
X
L
E


X
L
E
X
L
E

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import os
import scipy.io

plt.rcParams['figure.figsize'] = [14, 14]
plt.rcParams.update({'font.size': 18})

mat = scipy.io.loadmat("/mnt/c/Users/nikhi/Documents/sem4/mis/project/allFaces.mat")
faces = mat['faces']
nfaces = mat['nfaces'].reshape(-1)
```

In [2]:
```
def add_noise(img):

    # Getting the dimensions of the image
    row , col = img.shape

    # Randomly pick some pixels in the
    # image for coloring them white
    # Pick a random number between 300 and 10000
    number_of_pixels = np.random.randint(300, 10000)*5
    for i in range(number_of_pixels):

        # Pick a random y coordinate
        y_coord= np.random.randint(0, row - 1)

        # Pick a random x coordinate
        x_coord= np.random.randint(0, col - 1)

        # Color that pixel to white
        img[y_coord][x_coord] = 255

    # Randomly pick some pixels in
    # the image for coloring them black
    # Pick a random number between 300 and 10000
    number_of_pixels = np.random.randint(300 , 10000)*5
    for i in range(number_of_pixels):

        # Pick a random y coordinate
        y_coord= np.random.randint(0, row - 1)

        # Pick a random x coordinate
        x_coord= np.random.randint(0, col - 1)

        # Color that pixel to black
        img[y_coord][x_coord] = 0

    return img
```

In [3]:
```
def patch_noise(data):
    no_of_pixels , no_of_images = data.shape
    patch_size = 40
    for i in range(no_of_images):
        image = data[:, i]
        image = image.reshape(168, 192)
```

```
        row_start = np.random.randint(0, image.shape[0]-patch_size)
        row_end = row_start + patch_size
        col_start = np.random.randint(0, image.shape[1]-patch_size)
        col_end = col_start + patch_size
        image[row_start:row_end, col_start:col_end] = 0
        data[:,i] = image.reshape(-1)

    return data
```

In [4]:

```
## Function Definitions

def shrink(X,tau):
    Y = np.abs(X)-tau
    return np.sign(X) * np.maximum(Y,np.zeros_like(Y))
def SVT(X,tau):
    U,S,VT = np.linalg.svd(X,full_matrices=0)
    out = U @ np.diag(shrink(S,tau)) @ VT
    return out
def RPCA(X):
    n1,n2 = X.shape
    mu = n1*n2/(4*np.sum(np.abs(X.reshape(-1))))
    lambd = 1/np.sqrt(np.maximum(n1,n2))
    thresh = 10**(-7) * np.linalg.norm(X)

    S = np.zeros_like(X)
    Y = np.zeros_like(X)
    L = np.zeros_like(X)
    count = 0
    while (np.linalg.norm(X-L-S) > thresh) and (count < 1000):
        L = SVT(X-S+(1/mu)*Y,1/mu)
        S = shrink(X-L+(1/mu)*Y,lambd/mu)
        Y = Y + mu*(X-L-S)
        count += 1
    return L,S
```

In [6]:

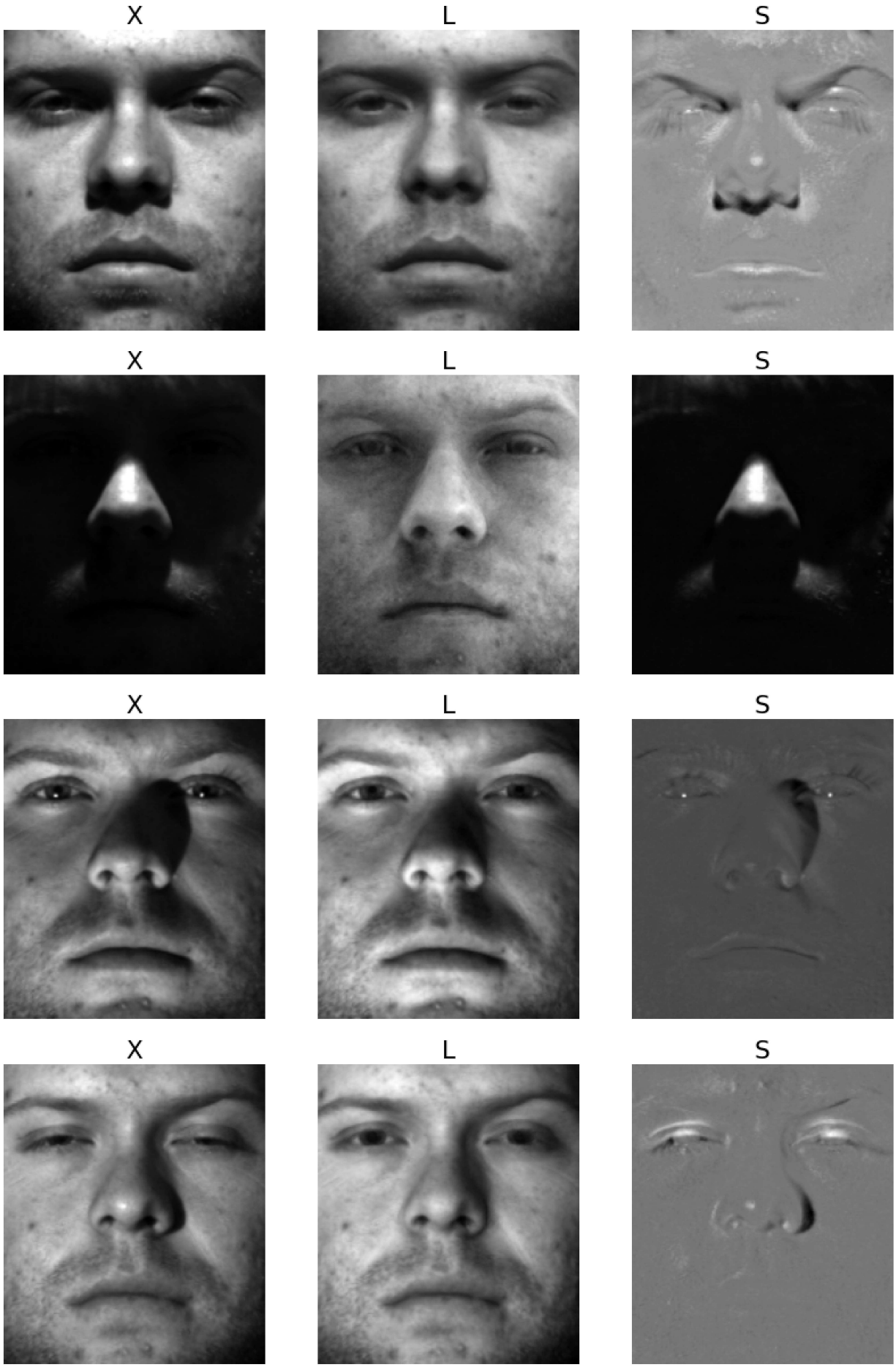
```
def my_plot(X,L,S, k):
    fig,axs = plt.subplots(1,3)
    axs = axs.reshape(-1)
    axs[0].imshow(np.reshape(X[:,k-1],(168,192)).T,cmap='gray')
    axs[0].set_title('X')
    axs[1].imshow(np.reshape(L[:,k-1],(168,192)).T,cmap='gray')
    axs[1].set_title('L')
    axs[2].imshow(np.reshape(S[:,k-1],(168,192)).T,cmap='gray')
    axs[2].set_title('S')
    for ax in axs:
        ax.axis('off')
```

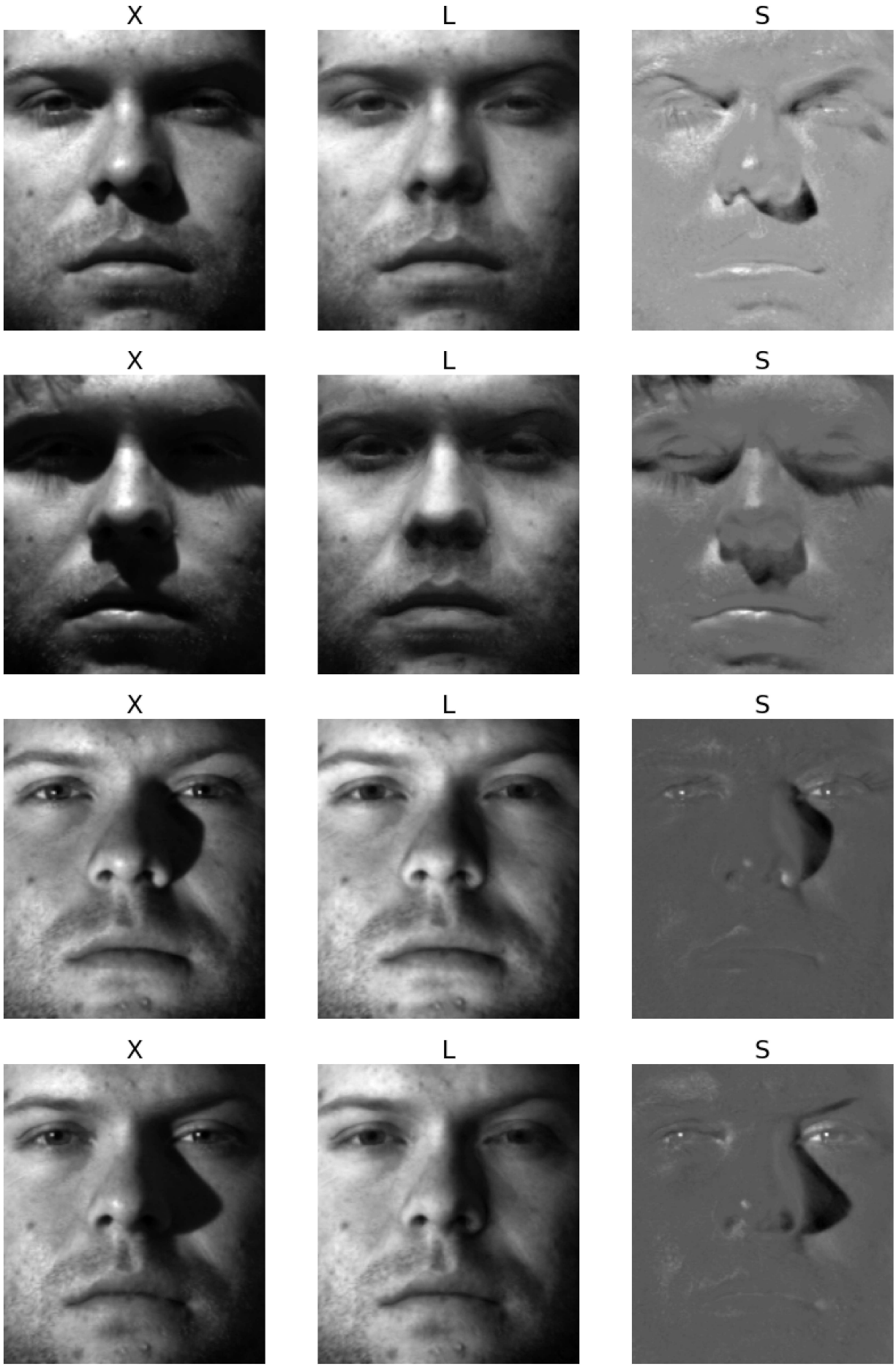
In [5]:

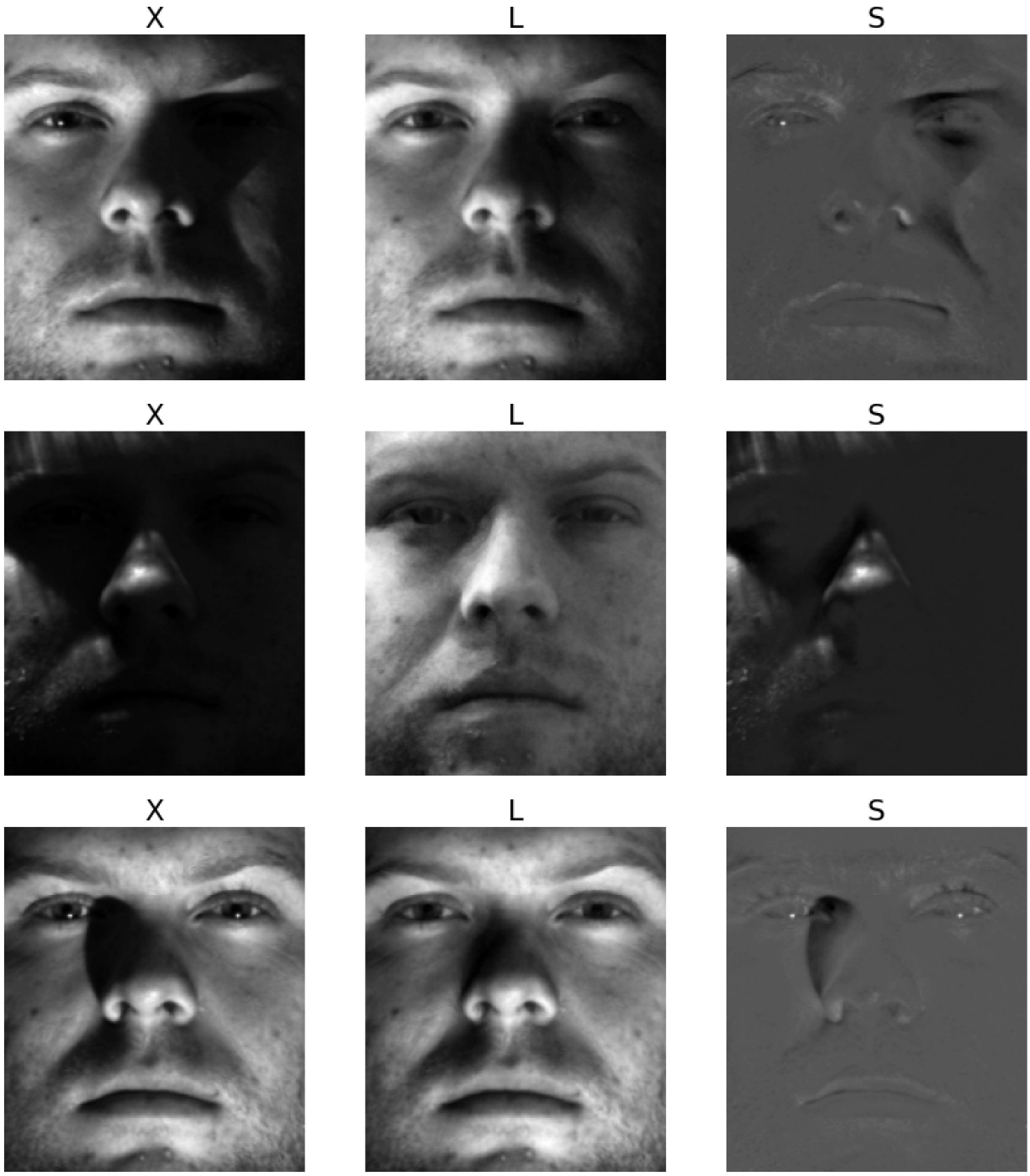
```
X = faces[:,:nfaces[0]+nfaces[1]]
L,S = RPCA(X)
```

In [7]:

```
inds = (3,4,14,15,17,18,19,20,21,32,43)

for i in inds:
    my_plot(X,L,S, i)
```


X
L
S
X
L
S
X
L
S
X
L
S

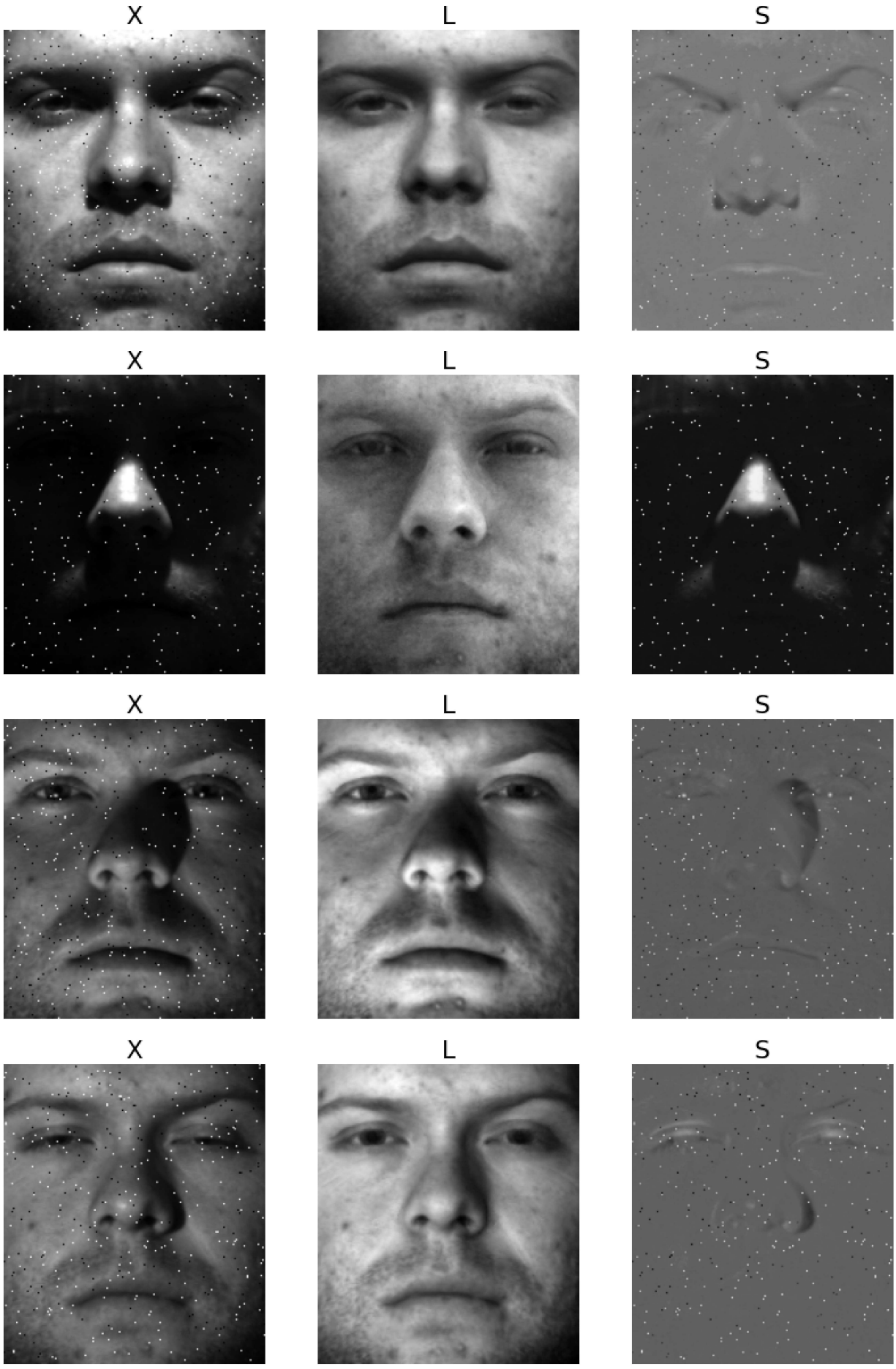

X
L
S
X
L
S
X
L
S
X
L
S

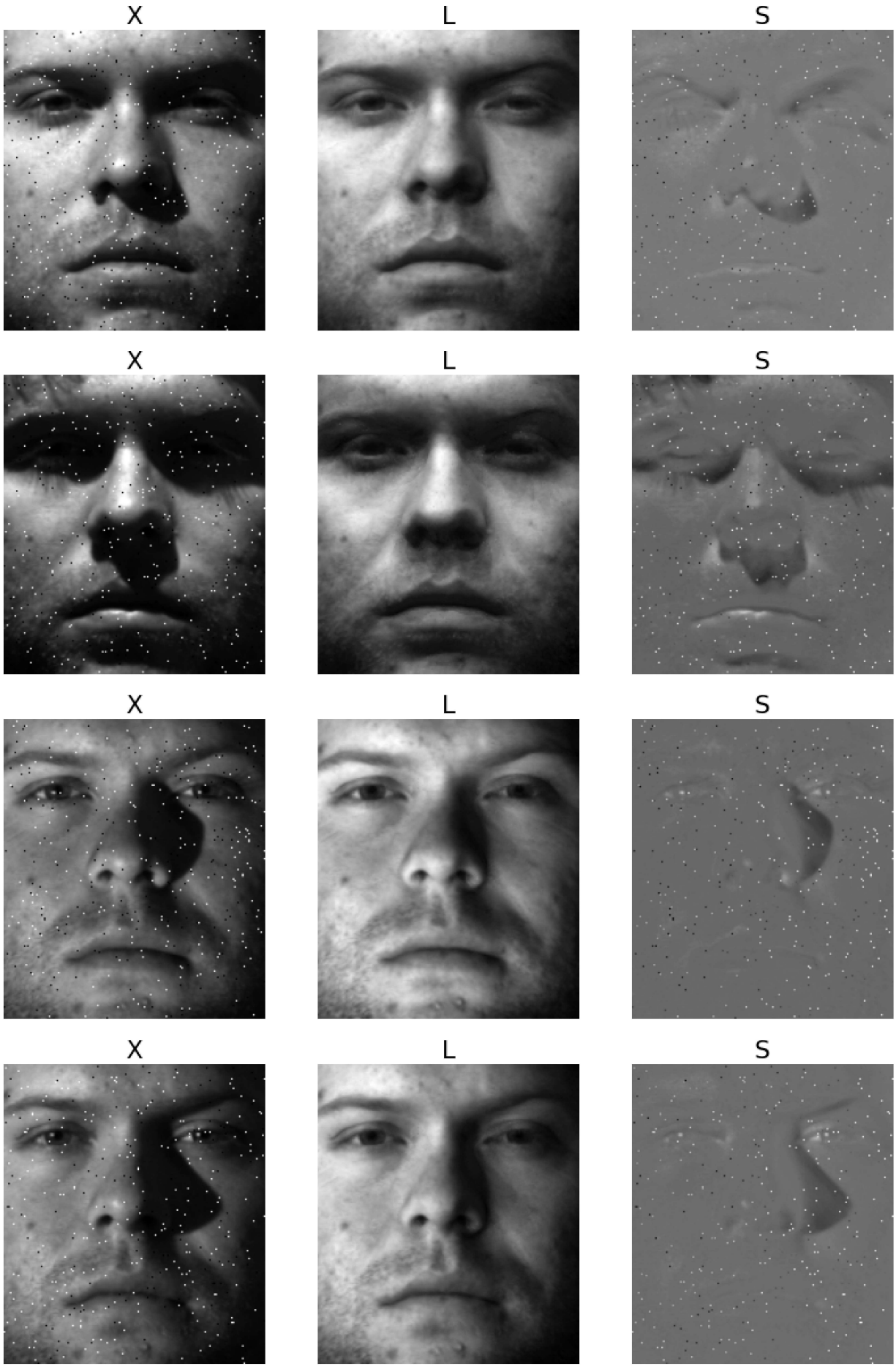


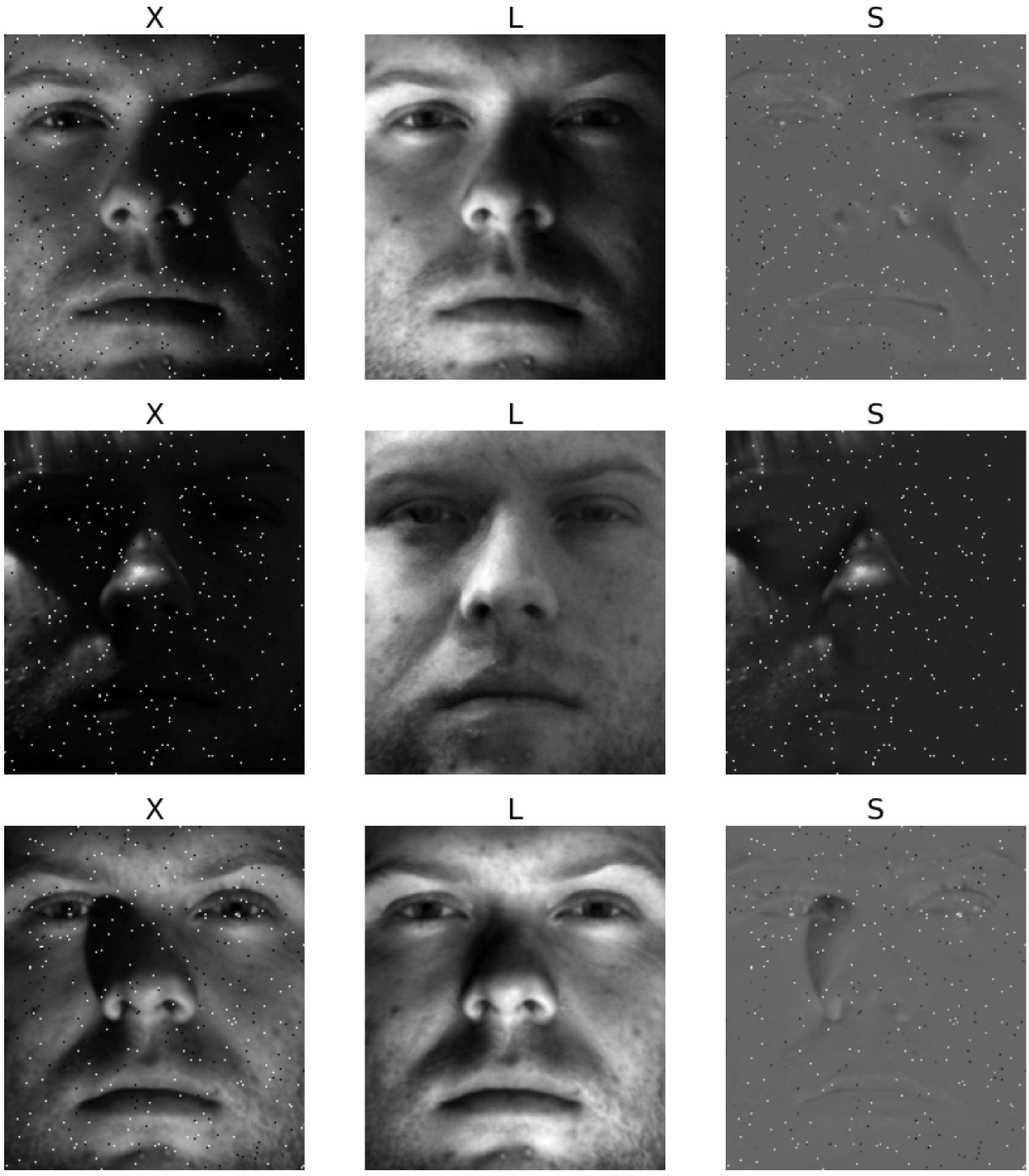
In [8]:
```
X2 = add_noise(X)
```

In [9]:
```
L2,S2 = RPCA(X2)
```

In [10]:
```
for i in inds:
    my_plot(X2,L2,S2, i)
```


X
L
S
X
L
S
X
L
S
X
L
S

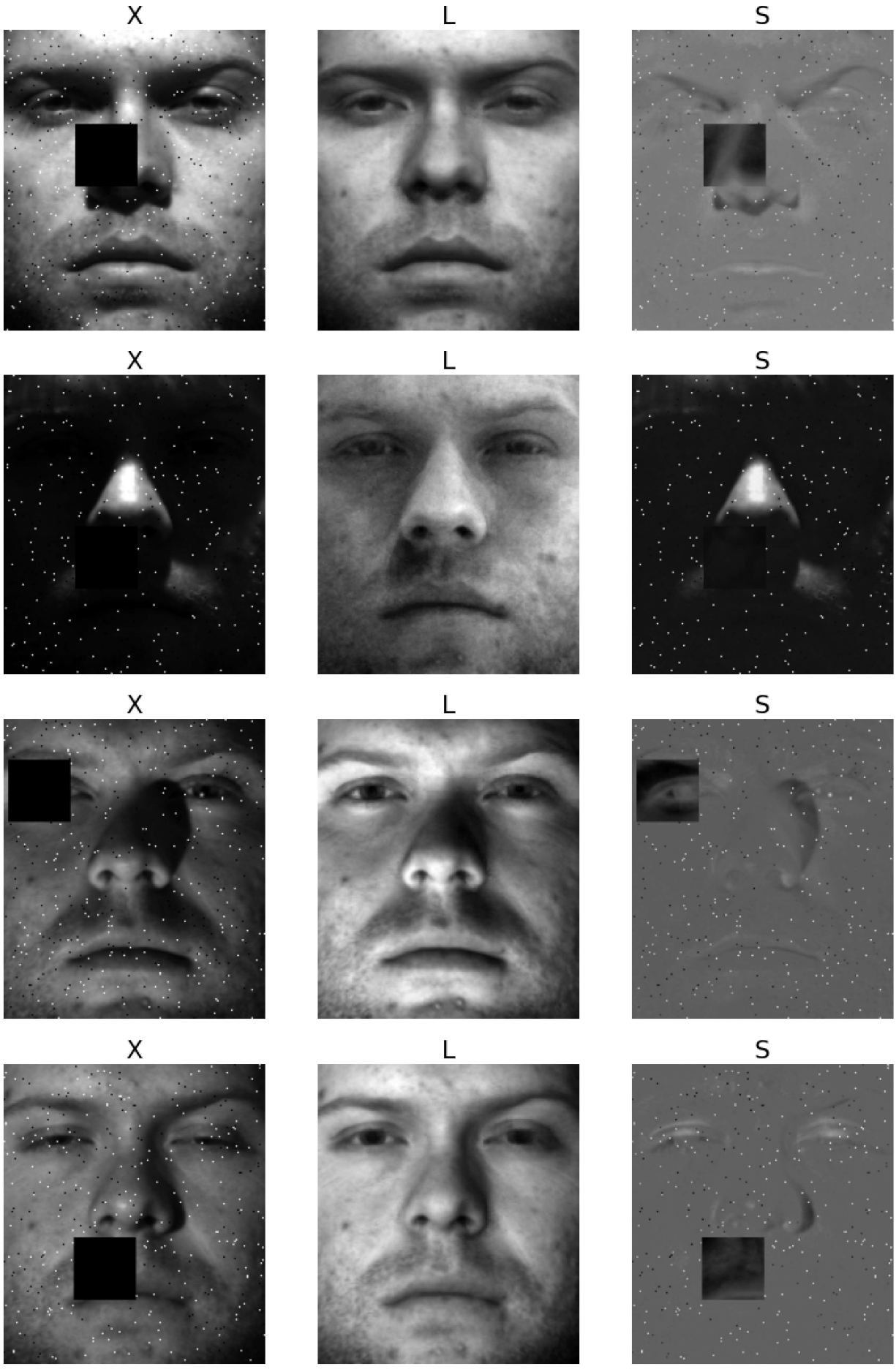

X
L
S
X
L
S
X
L
S
X
L
S

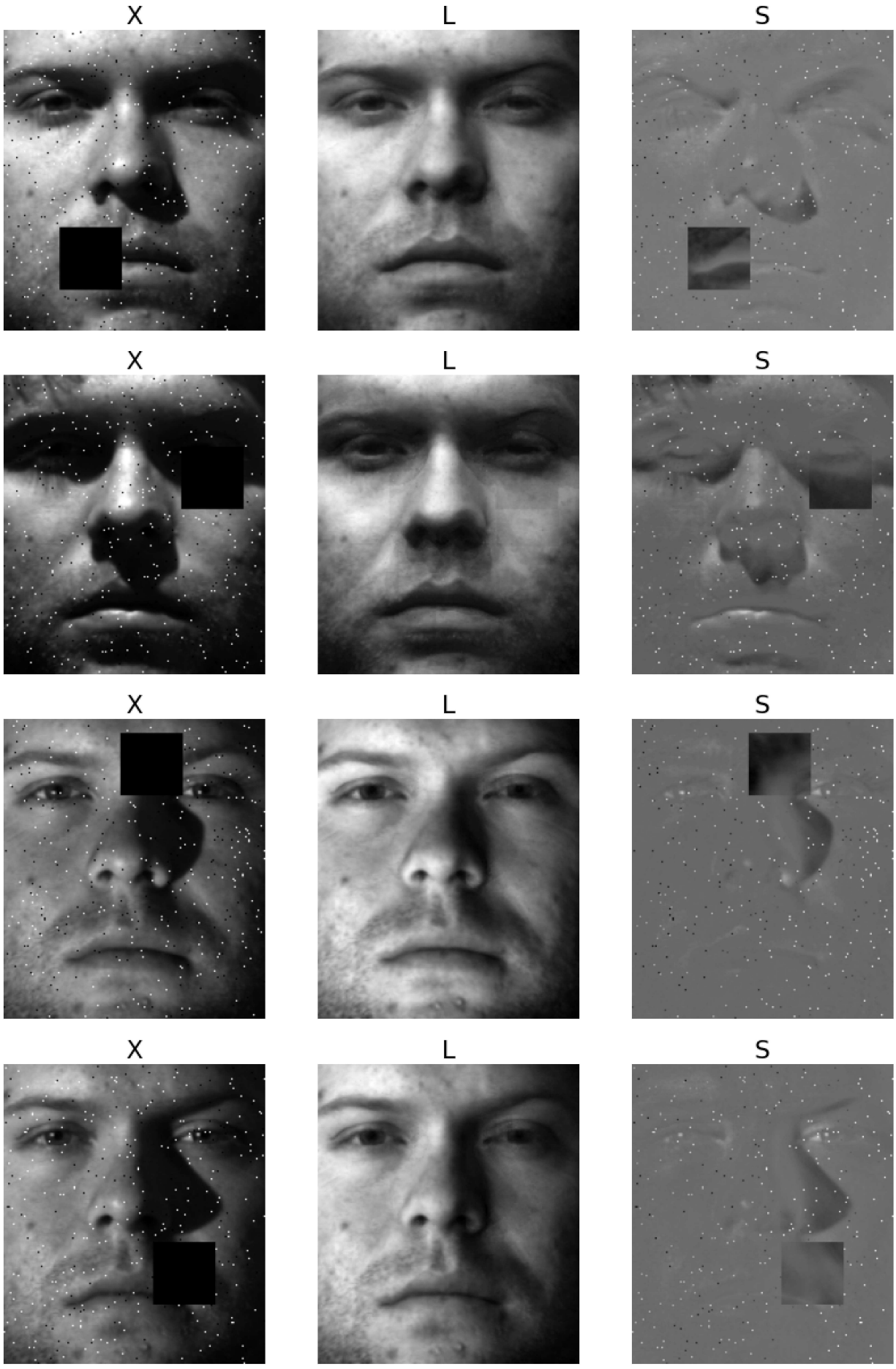


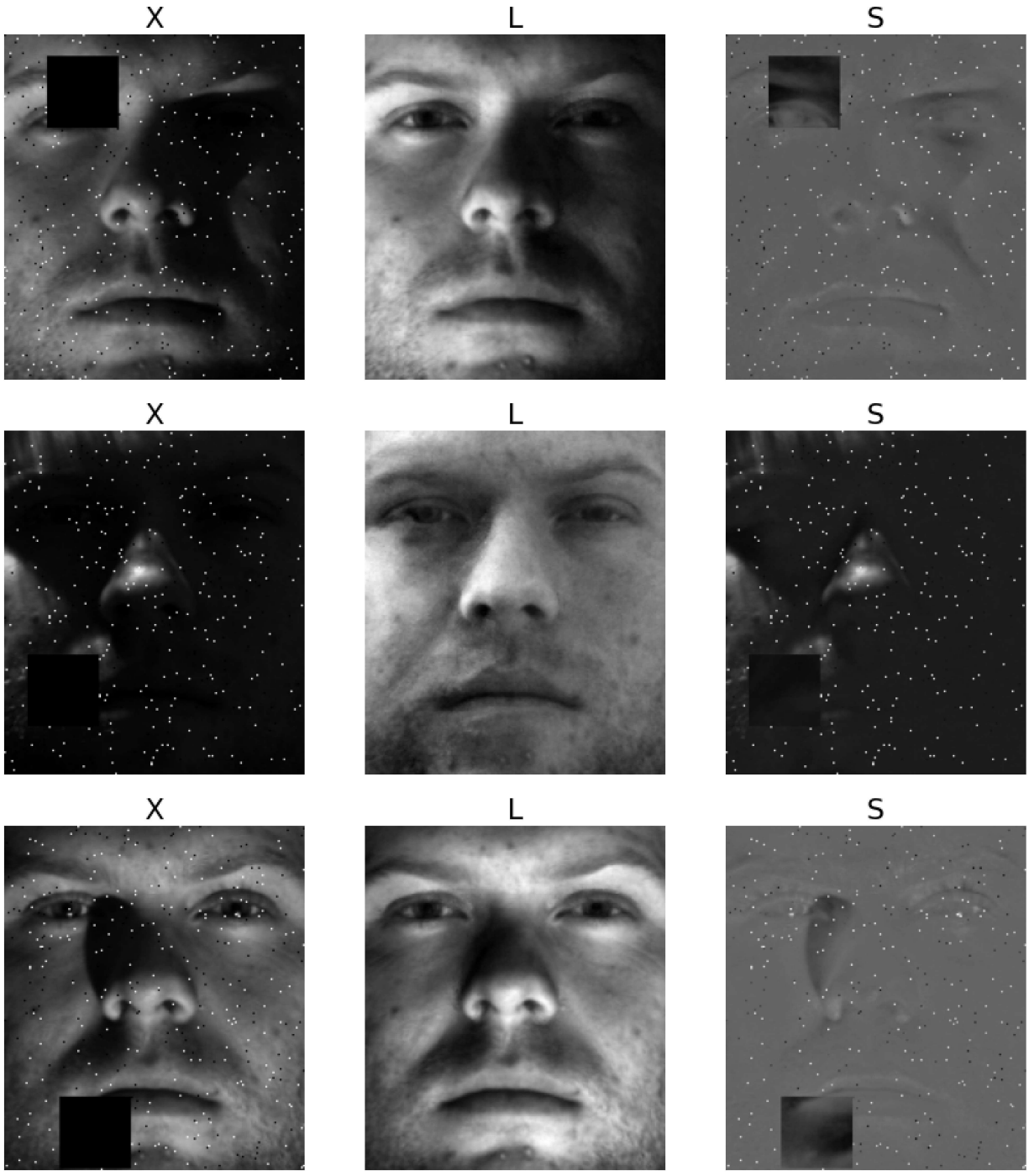
In [11]:
```
X3 = patch_noise(X)
```

In [12]:
```
L3,S3 = RPCA(X3)
```

In [13]:
```
for i in inds:
    my_plot(X3,L3,S3, i)
```


X
L
S
X
L
S
X
L
S
X
L
S


X
L
S
X
L
S
X
L
S
X
L
S



In [ ]: