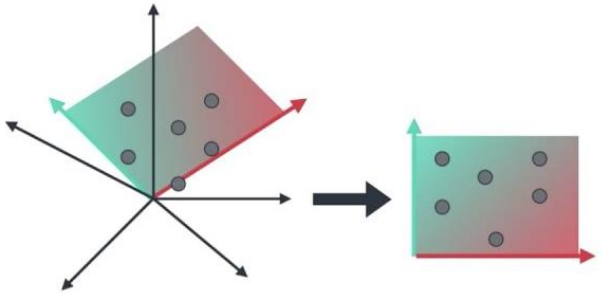
Computational Mathematics

ID	Name
19016660	مصطفى سيد أحمد طه أحمد
19016615	مروان محمد أحمد أبو العلا
19016365	محمد رزق أمين محمد
19016857	هيثم عطية السيد
19016856	هيثم أحمد شعبان أحمد
19015763	ساهر طارق أنور زايد أحمد
19016614	مروان مجدي مرزوق
19016850	هشام هيثم عبد الوهاب



Principal Component Analysis

PCA

History

- Principal component analysis (PCA) goes back to **Augustin-Louis Cauchy** but first proposed in statistics by **Karl Pearson** in **1901**.
- **Karl Pearson** formulated the analysis as finding “lines and planes of closest fit to systems of points in space”.
- PCA was briefly mentioned by **Fisher and MacKenzie** as more suitable than analysis of variance for the modelling of response data.
- **Hoteling** further developed PCA to its present stage.
- In the **1930s** the development of factor analysis (FA) was started by **Thurstone** and other psychologists. This needs mentioning here because FA is closely related to PCA and often the two methods are confused, and the two names are incorrectly used interchangeably.

History

- Since then, the utility of **PCA** has been rediscovered in many diverse scientific fields, resulting in an abundance of redundant terminology.
- **PCA** now goes under many names apart from those already mentioned:
  1. singular value decomposition (SVD) is used in numerical analysis.
  2. Karhunen-LoCve expansion is used in electrical engineering.
  3. Eigenvector analysis and characteristic vector analysis are often used in the physical sciences.
  4. In image analysis, the term Hotelling transformation is often used for a principal component projection.

Definition

- First of all, **PCA** is a method used to compress data in a certain way such that we can reduce the size of the certain file and keep a considerable amount of information.
- **P**rincipal **C**omponent **A**nalysis, or **PCA**, is a statistical procedure that allows you to summarize the information content in large data tables by means of a smaller set of “summary indices” that can be more easily visualized and analyzed.
- **PCA** is one of the most important dimensionality reduction techniques out there. So let`s say you have a huge table of data so big that it`s hard to process and you would like to make it into a smaller one while still keeping it as much of the information as possible so dimensionality reduction techniques what they do is, they reduce the number of columns.
- **P**rincipal **C**omponent **A**nalysis helps make data easier to explore and visualize. It is a simple non-parametric technique for extracting information from complex and confusing data sets.

Characteristics

- **PCA** is most commonly used when many of the variables are highly correlated with each other, and it is desirable to reduce their number to an independent set.
- One of the distinct advantages associated with the **P**rincipal **C**omponent **A**nalysis is that once patterns are found in the concerned data, compression of data is also supported.
- **P**rincipal **C**omponent **A**nalysis is sensitive to the relative scaling of the originally used variables.
- **P**rincipal **C**omponent **A**nalysis (**PCA**) is a technique used for identification of a smaller number of uncorrelated variables known as **principal components** from a larger set of data. The technique is widely used to emphasize variation and capture strong patterns in a data set.
- The number of **principal components** used in **P**rincipal **C**omponent **A**nalysis is less than or equal to the lesser number of observations.
- **P**rincipal **C**omponent **A**nalysis is focused on the maximum variance amount with the fewest number of **principal components**. Sometimes using only the first few **principal components** and ignoring the rest.

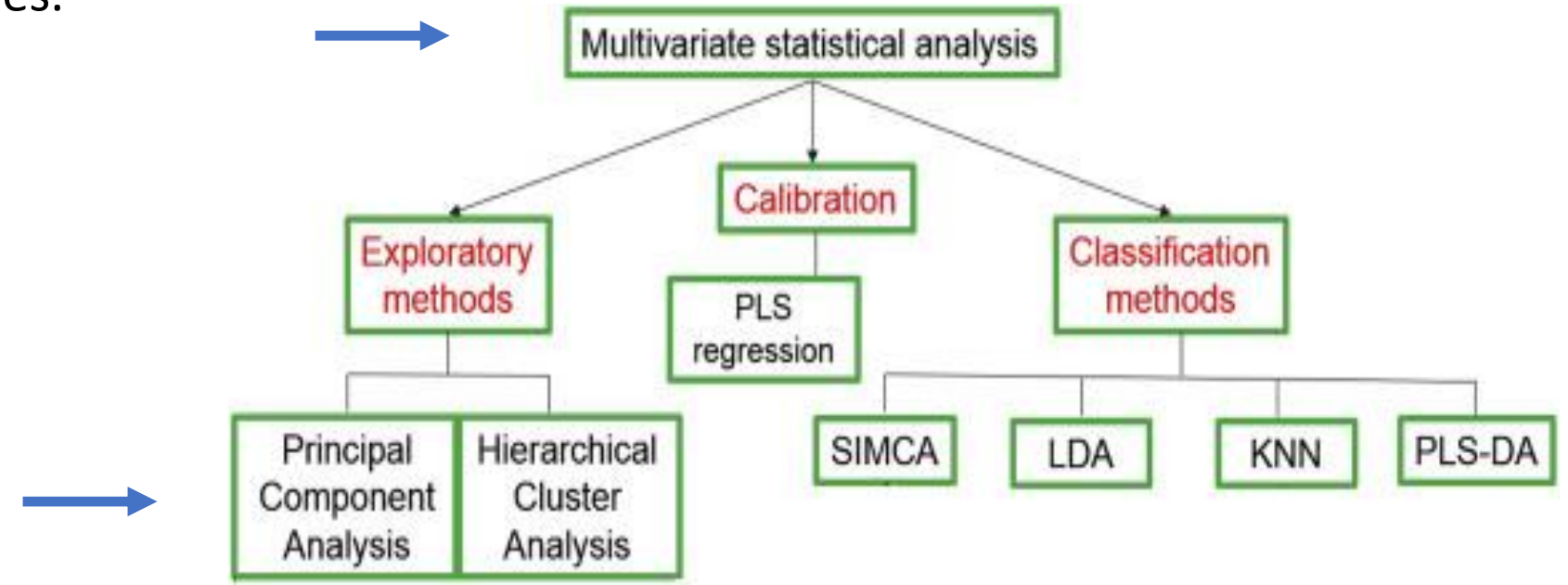
Applications

- **P**rincipal **C**omponent **A**nalysis is considered a useful statistical method and widely used in several fields and areas such as:

  1. Image Compression (which will be our main focus in this paper).
  2. Face Recognition.
  3. Neuroscience.
  4. Computer Graphics.
  5. Market Research.
  6. In industries where large data sets are used.
  7. Using **PCA** can help identify correlations between data points, such as whether there is a correlation between consumption of foods like frozen fish and crisp bread in Nordic countries.

Applications

- **PCA** is the mother method for **MVDA.**
- **PCA** forms the basis of **M**ulti**V**ariate **D**ata **A**nalysis based on projection methods. The most important use of **PCA** is to represent a multivariate data table as smaller set of variables (summary indices) in order to observe trends, jumps, clusters and outliers. This overview may uncover the relationships between observations and variables, and among the variables.



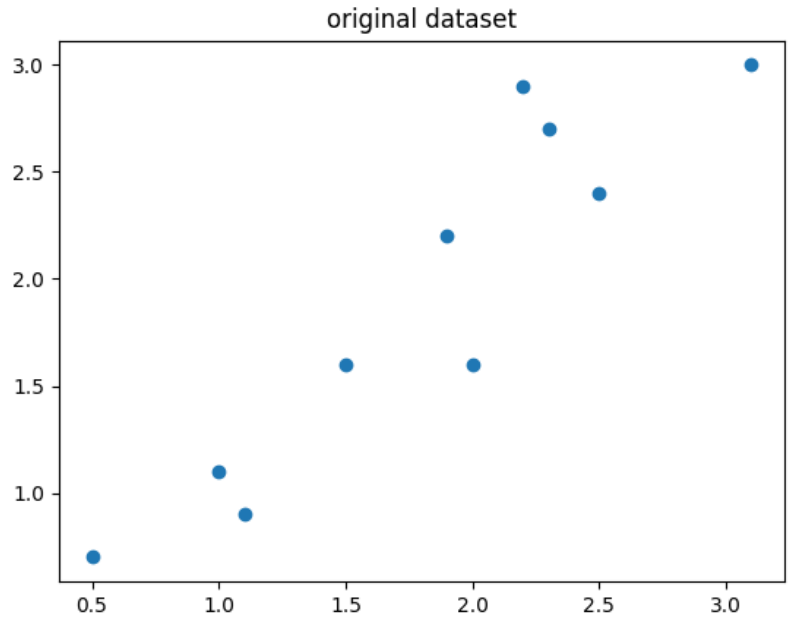
How PCA works

- Basic idea
  - Suppose we have a group of people, and we want to take a picture of them, each one can be considered as a 3D object (3 dimensions) and the picture is 2D, so we should choose the most suitable angle to make them all appear in the photo. So, it means that we reduced the dimensions from 3D to 2D while keeping as much information (their faces) as possible.

How PCA works

- If we have the following data set:

X	2.5	0.5	2.2	1.9	3.1	2.3	2.0	1.0	1.5	1.1
Y	2.4	0.7	2.9	2.2	3.0	2.7	1.6	1.1	1.6	0.9



How PCA works

- PCA consists of a few steps:
  1. Standardization.
  2. Computing the covariance matrix.
  3. Computing the eigenvectors and eigenvalues of the covariance matrix.
  4. Creating a feature vector.
  5. Recasting data.

How PCA works

1. Standardization.
   - The aim of this step is to standardize the range of the continuous initial variables so that each one of them contributes equally to the analysis.
   - When variables have a higher range of values they dominate variables with small range of values.
   - This step is done to put all of variables on a considerable scale such that the results are not to be biased heavily towards variables with much larger range.
   - This can be done by using this equation:

$$z = \frac{value \ - mean}{standard \ diviation}$$

How PCA works

1. Standardization.

X	2.5	0.5	2.2	1.9	3.1	2.3	2.0	1.0	1.5	1.1
Y	2.4	0.7	2.9	2.2	3.0	2.7	1.6	1.1	1.6	0.9

- Calculate the mean: $mean = \frac{\sum_{i=1}^{N} x_i}{N}$

X	2.5	0.5	2.2	1.9	3.1	2.3	2.0	1.0	1.5	1.1	Mean = 1.81
Y	2.4	0.7	2.9	2.2	3.0	2.7	1.6	1.1	1.6	0.9	Mean = 1.91

How PCA works

1. Standardization.

X	2.5	0.5	2.2	1.9	3.1	2.3	2.0	1.0	1.5	1.1
Y	2.4	0.7	2.9	2.2	3.0	2.7	1.6	1.1	1.6	0.9

- Calculate the standard deviation: $\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i - mean)^2}{N-1}}$

X	2.5	0.5	2.2	1.9	3.1	2.3	2.0	1.0	1.5	1.1	σ = 0.74
Y	2.4	0.7	2.9	2.2	3.0	2.7	1.6	1.1	1.6	0.9	σ = 0.80

How PCA works

1. Standardization.

X	2.5	0.5	2.2	1.9	3.1	2.3	2.0	1.0	1.5	1.1
Y	2.4	0.7	2.9	2.2	3.0	2.7	1.6	1.1	1.6	0.9

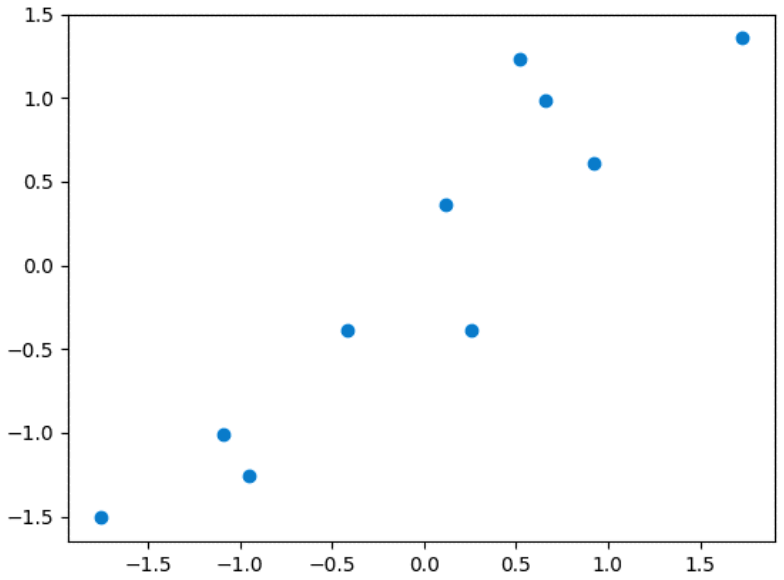
- From the equation $z = \frac{value - mean}{standard\ diviation}$

X	0.93	-1.77	0.53	0.12	1.74	0.66	0.26	-1.09	-0.42	-0.96
Y	0.61	-1.51	1.24	0.36	1.36	0.99	-0.39	-1.01	-0.39	-1.26

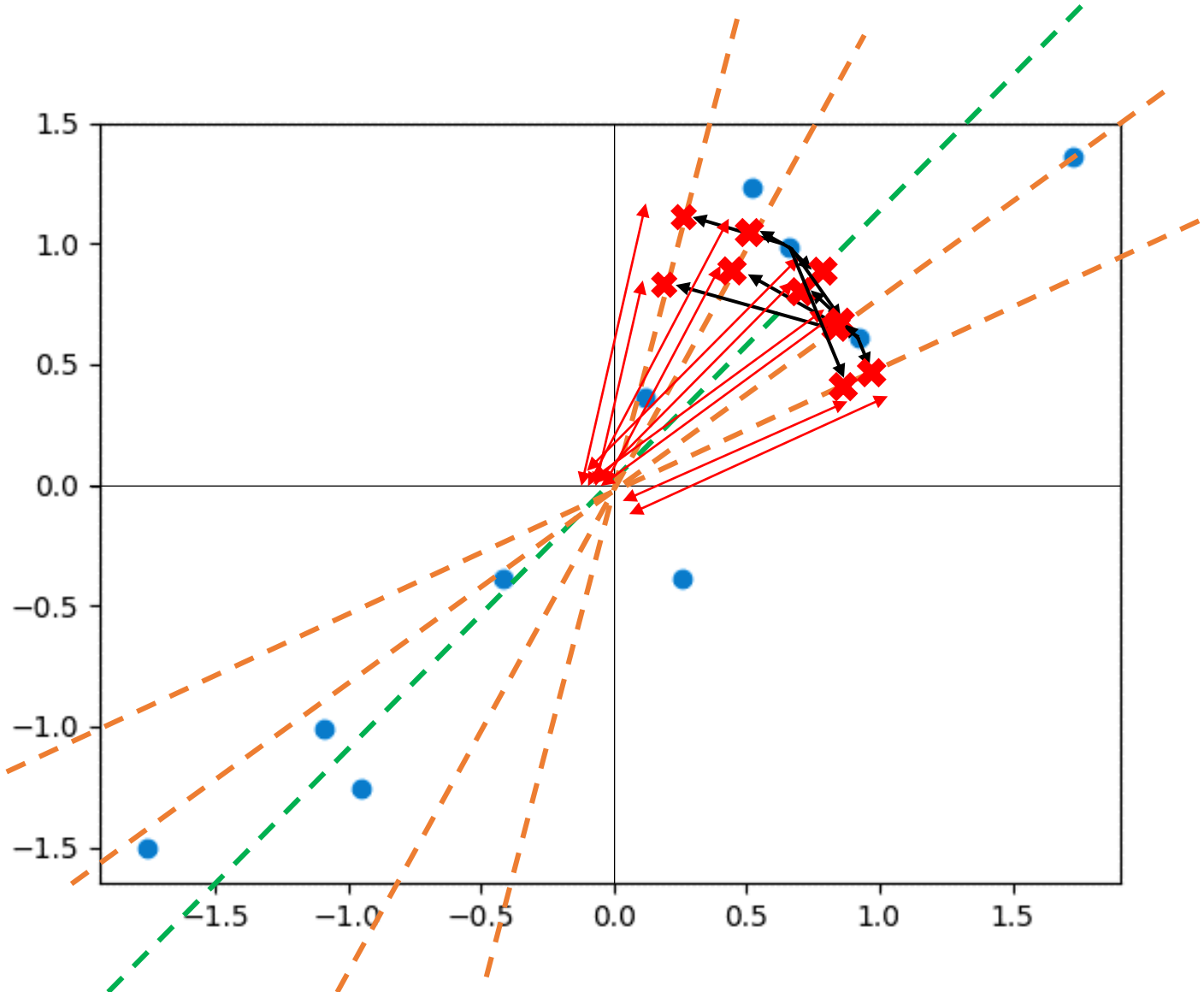
How PCA works

1. Standardization.
   - Standardized data set:

X	0.93	-1.77	0.53	0.12	1.74	0.66	0.26	-1.09	-0.42	-0.96
Y	0.61	-1.51	1.24	0.36	1.36	0.99	-0.39	-1.01	-0.39	-1.26



How PCA works



How PCA works

2. Computing the covariance matrix.

- This step is done to understand the relation between the variables to see whether they are highly correlated or not.
- If variables are highly correlated then they contain a lot of useless information that does not contribute much.
- The covariance matrix is a square matrix in which the number of rows and columns is the number of variables

$$\sum = \begin{bmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{bmatrix}$$

How PCA works

2. Computing the covariance matrix.

- Covariance of an element with itself is the variance of the element.

$$cov(x,x) = var(x) = \frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N-1}$$

- It also has the commutative property

$$cov(x,y) = cov(y,x) = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{N-1}$$

- For our standardized data set:

X	0.93	-1.77	0.53	0.12	1.74	0.66	0.26	-1.09	-0.42	-0.96
Y	0.61	-1.51	1.24	0.36	1.36	0.99	-0.39	-1.01	-0.39	-1.26

$$\Sigma = \begin{bmatrix} var(x) & cov(x,y) \\ cov(y,x) & var(y) \end{bmatrix} = \left(\frac{\boldsymbol{X}.\boldsymbol{X}^T}{N-1}\right)$$

Where $\boldsymbol{X}$ is the data set matrix (10×2 matrix), $\boldsymbol{X}^T$ is the transpose of matrix $\boldsymbol{X}$ and N is the number of elements

How PCA works

2. Computing the covariance matrix.

- Covariance of an element with itself is the variance of the element.

$$cov(x,x) = var(x) = \frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N-1}$$

- It also has the commutative property

$$cov(x,y) = cov(y,x) = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{N-1}$$

- For our standardized data set:

X	0.93	-1.77	0.53	0.12	1.74	0.66	0.26	-1.09	-0.42	-0.96
Y	0.61	-1.51	1.24	0.36	1.36	0.99	-0.39	-1.01	-0.39	-1.26

$$\Sigma = \begin{bmatrix} var(x) & cov(x,y) \\ cov(y,x) & var(y) \end{bmatrix} = \left(\frac{\boldsymbol{X}.\boldsymbol{X}^T}{N-1}\right) = \begin{bmatrix} 1.1233 & 1.037 \\ 1.037 & 1.1179 \end{bmatrix}$$

Where $\boldsymbol{X}$ is the data set matrix (10×2 matrix), $\boldsymbol{X}^T$ is the transpose of matrix $\boldsymbol{X}$ and N is the number of elements

How PCA works

3. Computing the eigenvectors and eigenvalues.

- Eigenvalues and eigenvectors are basically what we need to compute to determine the principal components (PCs) of the matrix.
- PCs are new variables that are constructed as linear combinations of the initial variables.
- These combinations are done to make the new variables uncorrelated and most of the information is compressed into the first components.
- PCA tries to put maximum possible information in the first component, then maximum remaining information in the second and so on.
- Organizing information in principal components will allow you to reduce dimensionality without losing much information.

How PCA works

3. Computing the eigenvectors and eigenvalues.
   - PCs represent the directions of the data that explain a maximal amount of variance.
   - The relationship between variance and information is that the larger the variance carried by a line, the larger the dispersion of the data points along it, and the larger the dispersion along a line, the more the information it has.

How PCA works

3. Computing the eigenvectors and eigenvalues.

- For our data set:

$$\Sigma = \begin{bmatrix} 1.1233 & 1.037 \\ 1.037 & 1.1179 \end{bmatrix}$$

- To find eigenvalues:

$$|\Sigma - \lambda \boldsymbol{I}| = 0$$

where λ is the eigenvalue and $\boldsymbol{I}$ is the identity matrix

$$\begin{vmatrix} 1.1233 - \lambda & 1.037 \\ 1.037 & 1.1179 - \lambda \end{vmatrix} = 0$$

$$\lambda = 2.1576 \qquad \lambda = 0.0836$$

How PCA works

3. Computing the eigenvectors and eigenvalues.

- For our data set:

$$\Sigma = \begin{bmatrix} 1.1233 & 1.037 \\ 1.037 & 1.1179 \end{bmatrix}$$

- To find eigenvectors:

$$(\Sigma - \lambda \boldsymbol{I})\boldsymbol{X} = 0$$

- For $\lambda = 2.1576$

$$\left[\begin{array}{cc|c} -1.0343 & 1.037 & 0 \\ 0 & 0 & 0 \end{array}\right]$$

$$-1.0343x_1 + 1.037x_2 = 0$$

$$x_1 = 1.0026x_2$$

How PCA works

3. Computing the eigenvectors and eigenvalues.

- For our data set:

$$\Sigma = \begin{bmatrix} 1.1233 & 1.037 \\ 1.037 & 1.1179 \end{bmatrix}$$

- To find eigenvectors:

$$(\Sigma - \lambda \boldsymbol{I})\boldsymbol{X} = 0$$

- For $\lambda = 0.0836$

$$\left[\begin{array}{cc|c} 1.0397 & 1.037 & 0 \\ 0 & 0 & 0 \end{array}\right]$$

$$1.0397x_1 + 1.037x_2 = 0$$

$$x_1 = -0.9974x_2$$

How PCA works

3. Computing the eigenvectors and eigenvalues.

- For $\lambda = 2.1576$
  - $\sqrt{1^2 + 1.0026^2} = 1.416$
  - $\begin{bmatrix} \frac{1.026}{1.416} \\ \frac{1}{1.416} \end{bmatrix} = \begin{bmatrix} 0.7246 \\ 0.7062 \end{bmatrix}$
  - So the eigenvector for $\lambda = 2.1576$ is

$$\begin{bmatrix} 0.7246 \\ 0.7062 \end{bmatrix}$$

- For $\lambda = 0.0836$
  - $\sqrt{1^2 + 0.9974^2} = 1.412$
  - $\begin{bmatrix} \frac{1}{1.412} \\ \frac{-0.9974}{1.412} \end{bmatrix} = \begin{bmatrix} 0.7082 \\ -0.7064 \end{bmatrix}$
  - So the eigenvector for $\lambda = 0.0836$ is

$$\begin{bmatrix} 0.7082 \\ -0.7064 \end{bmatrix}$$

How PCA works

4. Creating a feature vector.
   - In this step we choose which principal components to discard or not to discard any.
   - By ranking your eigenvectors in order of their eigenvalues, highest to lowest, you get the principal components in order of significance.
   - The feature vector is simply a matrix that has as columns the eigenvectors of the components that we decide to keep.
   - To compute the percentage of variance (information) accounted for by each component, we divide the eigenvalue of each component by the sum of eigenvalues.

How PCA works

4. Creating a feature vector.

- Continuing with our data set, the feature vector is:

$$\begin{bmatrix} 0.7246 & 0.7082 \\ 0.7062 & -0.7064 \end{bmatrix}$$

- Discarding the eigenvector of the smallest λ, the feature vector is:

$$\boldsymbol{F} = \begin{bmatrix} 0.7246 \\ 0.7062 \end{bmatrix}$$

- This reduces dimensionality by 1
- λ_1 is carrying 96.27% of the information

How PCA works

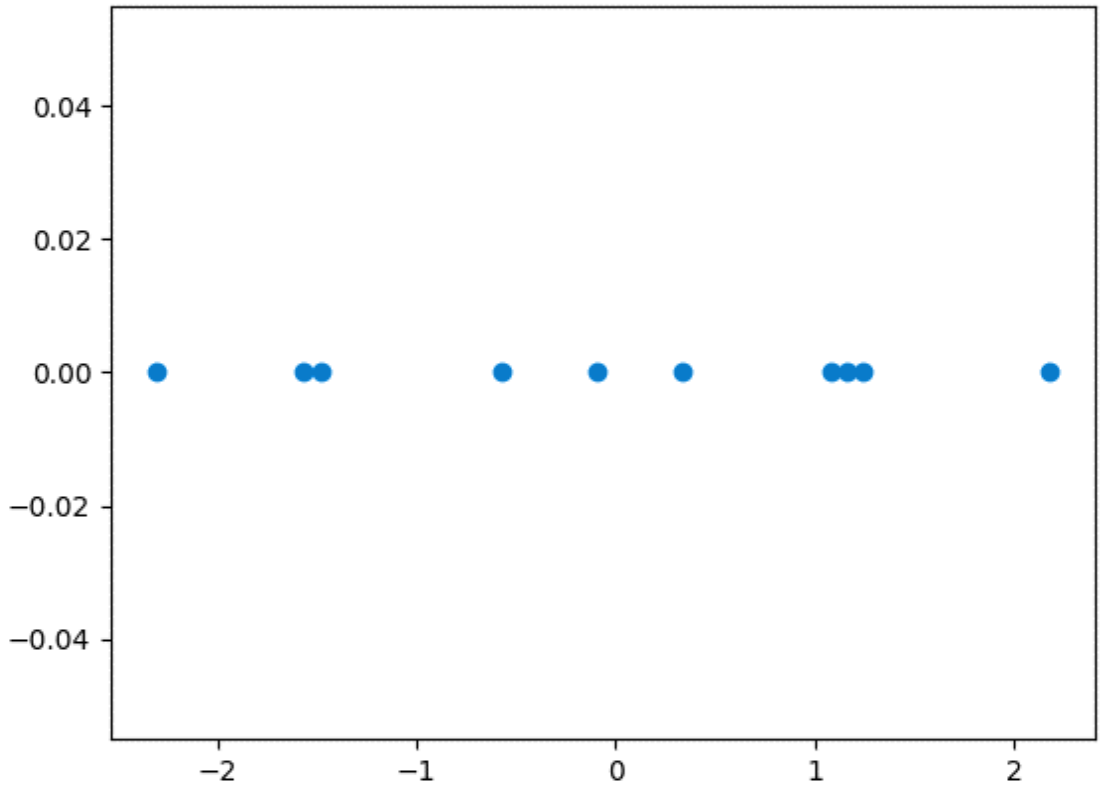
5. Recasting data.
   - The aim is to use the feature vector formed to reorient the data from the original axes to the ones represented by the PCs.
   - This can be done by multiplying the transpose of the original data set by the transpose of the feature vector.
   - For our example:

$$\boldsymbol{Z} = \boldsymbol{X}^{T} . \boldsymbol{F}$$

Where **X** is the original data set matrix and **F** is the feature vector.

How PCA works

5. Recasting data.



How PCA works

- A python code for PCA

```
import numpy as np
import matplotlib.pyplot as plt
from numpy import linalg as LA

input = np.array([[2.5 ,2.4],[0.5, 0.7],[2.2 ,2.9],[1.9 ,2.2],[3.1 ,3.0],
                  [2.3 ,2.7],[2 ,1.6],[1 ,1.1],[1.5, 1.6],[1.1 ,0.9]])

print(input)

plt.title("original dataset")
plt.scatter(input[:,0], input[:,1])
plt.show()

mean = np.mean(input, axis=0)
print("mean: ", mean)
standard_deviation = np.std(input, axis=0)
print("standard_deviation: ", standard_deviation)
```

```
adjusted_input = (input - mean)/standard_deviation
print("mean adjusted input: ", adjusted_input)
plt.scatter(adjusted_input[:, 0], adjusted_input[:, 1])
plt.show()
cov = np.cov(adjusted_input[:,0], adjusted_input[:,1])
print("cov=", cov)

w, v = LA.eig(cov)
print("Eigen Value")
print(w)
print("Eigen Vector")
print(v)

eigen_input = np.dot(adjusted_input, np.array(v.T[0]))
print(eigen_input.reshape(10,1))
plt.scatter(eigen_input, np.zeros_like(eigen_input)+0)
plt.show()
```

Image Compression using PCA

First: General idea of image compression

- There are many reasons for which we need image compression. Saving storage space is a main reason. This also helps optimize applications that use many images as smaller sized images load faster.
- Many techniques exist for image compression, and they can be characterized into two main groups: lossy and lossless compression.
- As the name implies, lossy methods cause a loss in data as a tradeoff for decreasing size. While lossless methods use different methods such as grouping similar pixels or sequences of data.
- We will focus on the first type for the purposes of this presentation

First: General idea of image compression_

- In the following figures we can visualize the basic idea of image compression. We can consider pixels that carry the same type on the same line (two or more of the same colour in sequence) to be data with high correlation. This means some of this data is redundant can be removed.

We may approximate the two red pixels to just one without changing the original sequence (data sample has low variance)

First: General idea of image compression

- In the second figure each pixel carries a different colour and follows a distinct sequence (low correlation). This means that this data is unique. Removing one of the colours will change the sequence and may change the data considerably.

We cannot remove any pixels without altering the original sequence (the data sample has high variance)

First: General idea of image compression

- So what we require now is a system to detect the parts which have multiple similar values (colours) in sequence and then reduce them to a lesser number, while keeping the values that do have minor relation to each other in sequence.

- This is where Principal Component Analysis comes in, and in the following section we will go into details how it can be applied in image compression, showcase an example and the results of compression.

How it is applied using PCA

- Let us consider a picture of dimensions a*b. We may be able to represent this picture as a matrix g(a*b). Each element represents the position of a pixel and carries the RGB coded value of its respective pixel position. It will be as follows:

$$g = \begin{matrix} g(1,1) & \cdots & g(1,b) \\ \vdots & \ddots & \vdots \\ g(a,1) & \cdots & g(a,b) \end{matrix}$$

How it is applied using PCA

- After carrying out standardization and obtaining the covariance matrix, we will find that the first picture ‘s column will yield a large covariance coefficient (dependent or similar). While in the second picture, its column will yield a small covariance coefficient (independent).
- This goes in line with what we want to achieve in order to carry out image compression.

How it is applied using PCA

- We can then obtain the eigenvector matrix and sort them in descending order according to their variance (eigenvalues).
- This will make the majority of the unique data carried in the first few eigenvectors, while the remaining ones contain mostly repeated data that can be discarded. Then we obtain the feature vector by discarding low variance data, and use it to get our new dataset.
- Image regeneration is then carried out and the compressed image is produced

Applying in MATLAB using stock image

- An example application of this is shown in this section, where a MATLAB code is used on an image to showcase the compression method and its effectiveness.
- Each step will be explained, and the results will be shown after explaining.
- The whole code is in the end of this presentation.

Applying in MATLAB using stock image

- The shown stock royalty free fruit image was used. Then for simplicity, it will be converted to grayscale.

```
I =imread('stockfruit.jpg');
g=double(rgb2gray(I));
figure, imshow(g,[]);
```

Applying in MATLAB using stock image

- In this code while standardizing the dataset, dividing by the standard deviation is not necessary.
- This is because the following steps (covariance and obtaining eigenvectors) do not change dramatically.
- Since we will multiply it back anyway when carrying out image regeneration, we may skip obtaining it. What we will have is technically not true PC's, but for the purposes of this application it will be sufficient.
- However the values must be zero-centered so we will subtract the mean regardless.

Applying in MATLAB using stock image

- Therefore calculating the mean of each pixel column and then subtracting it from the entirety of its respective column will readily standardize the data set.

```
meen=repmat(mean(g),size(g,1),1);%obtain mean for each column then
                                 %make it have the same amount of rows
G=(g-meen);  %centralization of data
```

Applying in MATLAB using stock image

- Now, we obtain the covariance matrix and then get the eigenvector matrix. Note that the eigenvector matrix should be sorted in descending order according to eigenvalue, and the function (eig) outputs them readily sorted but in ascending order. So we simply flip the eigenvector matrix from left to right.

```
gcov=cov(G);  %covariance of data matrix
[eigvec,eigval]=eig(gcov);   %eigenvectors and eigvenvalues of covariance matrix
eigvec=fliplr(eigvec);       %they are ascendingly arranged so we flip it to be descending
```

Applying in MATLAB using stock image

- We now obtain the feature vector by picking how many eigenvectors we want to keep, depending on level of compression needed. In this simulation we picked the values (10, 20, 50, 100, 200, all of them) just to showcase the result of carrying it out.

```
for i=[10 20 50 100 200 size(eigvec,1)];%iterations depending on amount of vectors kept
fv=eigvec(:,1:i');          %obtianing feature vector which contains kept eigenvectors
```

Applying in MATLAB using stock image

- Then the resolution of the output image is obtained by multiplying the feature vector by the transpose of the standardized data set.
- We multiply the transpose of the feature vector by the resolution to obtain the dimensions similar to the original image with the modified data set.
- Note that the result will be transposed simply to orient the image correctly, as it will be rotated and flipped.

```
resol=(fv)*(G');              %intermediate matrix carrying image size
newvals=(fv'*resol)';         %new values which will be used to regenerate the image
output=zeros(size(G));        %output matrix initialization
```

Applying in MATLAB using stock image

- Image regeneration now will take place by adding back the mean to each of the columns of the new compressed image. A zero matrix of the same size is used to hold the values, converted to the image number value datatype, and then plotted.

```
output=output+meen+newvals;%output image restored
output=uint8(output);%converting to image datatype
figure
imshow(output);
end;
```

MATLAB code compilation results

10 eigenvectors kept

20 eigenvectors kept

50 eigenvectors kept

100 eigenvectors kept

200 eigenvectors kept (from here the quality is more or less the same with only size differences)

Original Image

123 KB
141 KB
169 KB
189 KB
192 KB
244 KB

The sizes of each image
Showing that compression is a success
(top is 10 eigenvector image, bottom is whole image)

MATLAB code

```
I =imread('stockfruit.jpg');
g=double(rgb2gray(I));
figure, imshow(g,[]);
meen=repmat(mean(g),size(g,1),1);%obtain mean for each column then
                                 %make it have the same amount of rows
G=(g-meen);  %centralization of data
gcov=cov(G);  %covariance of data matrix
[eigvec,eigval]=eig(gcov);   %eigenvectors and eigvenvalues of covariance matrix
eigvec=fliplr(eigvec);       %they are ascendingly arranged so we flip it to be descending
for i=[10 20 50 100 200 size(eigvec,1)];%iterations depending on amount of vectors kept
fv=eigvec(:,1:i');          %obtianing feature vector which contains kept eigenvectors
resol=(fv)*(G');           %intermediate matrix carrying image size
newvals=(fv'*resol)';      %new values which will be used to regenerate the image
output=zeros(size(G));     %output matrix initialization
output=output+meen+newvals;%output image restored
output=uint8(output);%converting to image datatype
figure
imshow(output);
end;
```

The entire MATLAB code

Thank you…