# School of Computing
# National University of Singapore
# CS4243 Computer Vision and Pattern Recognition
# Semester 1, AY 2013/14

## Lab 5 : Principal Component Analysis

**Objectives:**

- Learn to use NumPy for numerical computations.
- Learn to write a Python program to do Principal Component Analysis.

**Preparation:**

- Create a folder in the PC with your name, e.g., `d:/myname`. This folder will be used as your working directory.
- Download the following files into your working directory: `pcaAY1314_v3.doc` & `md-data.txt`.

### Part 0. Initialisation

Set the working directory, e.g., `d:/myname`

```
>>> import os
>>> os.chdir("d:/myname")
```

Open `md-data.txt` using your favourite editor. You should see 30 rows of values. Each row contains 11 values corresponding to the 11 components of a data point. That is, these data points are 11-dimensional.

### Part 1. PCA

Write a program to perform PCA as follows. After writing the codes for each step, run the program to verify the results.

1. Import NumPy (linalg stands for "linear algebra"):

```
import numpy as np
import numpy.linalg as la
```

2. Read data in `md-data.txt` into a $d \times n$ matrix called `data`:

```
infile = open("md-data.txt")
data = np.genfromtxt(infile, delimiter=",")
data = np.matrix(data).T
infile.close()
```

   Print `data` to examine its content (see on your screen only, no need to print out hardcopy). It should contain 30 column vectors of 11 components each.

3. Get the number of dimensions $d$ and number of data points $n$:

```
d = data.shape[0]
n = data.shape[1]
```

   Print the values of $d$ and $n$. Are they correct?

4. Compute the mean of the data points:

```
mean = np.mean(data, 1)
```

   The function `np.mean` computes the mean along axis 1, i.e., horizontally. Print `mean` to verify that it is a column vector.

5. Subtract mean from data points:

```
sdata = data - mean
```

   This subtraction shifts the data points so that their centroid is at the origin. You can verify this by printing the mean of sdata:

```
print np.mean(sdata, 1)
```

   The values printed should be very close to 0.

6. Compute the covariance matrix of the shifted data points:

```
cov = np.cov(sdata, None, 1, 1)
```

   To call this function correctly, you need to indicate that the vectors in `sdata` are arranged in columns. Check up NumPy reference guide for the meanings of arguments.

7. Compute eigen-decomposition:

```
eigvalues, eigvectors = la.eig(cov)
index = np.argsort(eigvalues)
```

The function `la.eig` performs eigen-decomposition and returns two arrays. The first array contains the eigenvalues and the second contains the eigenvectors arranged in columns. The function `la.eig` does not necessarily return eigenvalues in sorted order.

The function `np.argsort` sorts the eigenvalues and returns an array of indices ordered in **increasing order** of eigenvalues.

**Part 2. Dimensionality Reduction**

Add the following codes to the program to perform dimensionality reduction.

1. Compute the ratio of **unaccounted variance** for each possible reduced dimension $l$ from 1 to $d$:

$$R = \frac{\sum_{j=l+1}^{d} \lambda_j}{\sum_{j=1}^{d} \lambda_j}$$

Note that in this formula, the eigenvalues are assumed to be sorted in **decreasing order**. For example, if $l = 3$, then only the first three largest eigenvectors are included, and the variances from $\lambda_4$ to $\lambda_d$ are not accounted for. So, the above formula measures the ratio of unaccounted variance over the total variance.

Be careful of the following:

a. The indices of python arrays and matrices **begin with 0** instead of 1. For example `eigvalues[0]` is the first eigenvalue in `eigvalues`. On the other hand, the indices in the above formula start with 1.

b. The function `np.argsort` sorts the eigenvalues in **increasing order** instead of decreasing order. So, `eigvalues[index[0]]` is the smallest eigenvalue $\lambda_d$. The following figure illustrates the relationship between the eigenvalues stored in `eigvalues` and those represented by $\lambda_j$ in the above formula:

| i | 0 | | | d-j | | | | | d-1 |
|---|---|---|---|---|---|---|---|---|---|
| `eigvalues[index[i]]` | $\lambda_d$ | | | $\lambda_j$ | | | | | $\lambda_1$ |

2. Plot a graph of $R$ vs. number of dimensions $l$ by doing the following:

   Add the following statement to your program:

   import matplotlib.pyplot as plt

   Then you can use plt.plot to plot the graph.

3. Based on the plotted graph, select a smaller number of dimensions $l$ that you think is enough for representing the data points.

4. Compose the matrix `Q` of eigenvectors from `eigvectors`. Arrange the eigenvectors in `Q` in **decreasing order** of eigenvalues. That is, `Q[:,0]` contains the eigenvector with the largest eigenvalue. Remember that `Q` must be a matrix instead of a 2D array.

5. Compute the $\mathbf{y}_i$ vectors of the data points in the eigenspace:

$$\mathbf{y}_i = \mathbf{Q}^{\mathrm{T}}(\mathbf{x}_i - \mathbf{m})$$

   Write this python statement without using explicit `for` loop to compute the $\mathbf{y}_i$ vector for each data point $\mathbf{x}_i$. This will let your program run faster.

6. Keep the first $l$ components of the $\mathbf{y}_i$ vectors, where $l$ is the number of dimensions you selected in Step 3. Call these reduced dimensional vectors $\hat{\mathbf{y}}_i$. Print $\hat{\mathbf{y}}_i$ to make sure that their dimensions are correct.

7. Create a reduced dimensional $\mathbf{Q}$ matrix, call it $\hat{\mathbf{Q}}$, by copying the first $l$ eigenvectors from the original $\mathbf{Q}$ to $\hat{\mathbf{Q}}$. Print $\hat{\mathbf{Q}}$ to make sure that it has the correct $l$ eigenvectors.

8. Compute estimates of the data points from $\hat{\mathbf{Q}}$ and $\hat{\mathbf{y}}_i$

$$\hat{\mathbf{x}}_i = \hat{\mathbf{Q}}\,\hat{\mathbf{y}}_i + \mathbf{m}$$

   Again, write this python statement without explicit `for` loop.

9. Use `la.norm` to compute the (square-root of the squared) difference between the original data points $\mathbf{x}_i$ and the estimated data points $\hat{\mathbf{x}}_i$ (summed over all data points). This difference should be small if $l$ dimensions are enough to represent the data points sufficiently.

10. Repeat steps 4 to 10 for $l$ ranging from 1 to d. Plot a graph with the la.norm values in step 10 along the vertical axis, and $l$ along the horizontal axis. Comment on the curve w.r.t. the curve plotted in step 3.

**Lab Report**

1. Your lab report should contain the program that you write and the following results:
    a. The values of d and n in Part1 Step 3.
    b. The mean vector of the input data points computed in Part 1 Step 4.
    c. The mean vector of the shifted data points computed in Part 1 Step 5.
    d. The eigenvalues and sorted indices computed in Part 1 Step 7.
    e. The graph of $R$ vs. $l$ plotted in Part 2 Step 3
    f. The $l$ value you selected in Part 2 Step 4
    g. The difference you computed in Part 2 Step 10 for the selected $l$ value.
    h. The graph of *la.norm* vs $l$ plotted in Part 2 Step 11

2. Submit the softcopy of your Python program to IVLE

    Please put your python program in a folder and submit the folder. Use the following convention to name your folder: MatriculationNumber_yourName_Lab#_Session. For example, if your matriculation number is A1234567B, and your name is Chow Yuen Fatt, for this lab, your file name should be A1234567B_ChowYuenFatt_Lab5_Wed630pm.