Programming task 3

Signal Compression and Reconstruction
with
Principal Component Analysis (PCA)
using the
Generalized Hebbian Algorithm

Principal Component Analysis

In this task we want to compress a signal by projecting the original signal data blockwise on the principal components. The number of principal components used, determines the compression ratio. By storing (or transmitting) the reduced number of projected data we have reduced the storage need (or the transmitting capacity need). In order to get the original signal back, we reconstruct the signal by blockwise inverting the projection. The resulting error between the original and the reconstructed signal is minimal in the least square sense, provided that we have used the real principal components as projection basis. This transformation to a new set of coordinates is also called the "Karhunen-Loeve transformation". It is well known that the principal components are the eigenvectors of the correlation matrix of the original signal. In this task we want to find the principal components with the use of a Neural Network. The network consists of a single layer with linear neurons and the training is an unsupervised algorithm, the "Generalized Hebbian Algorithm". The number of neurons is the same as the number of principal components we wish to use in the compression.

a) In this section we will use an artificial signal with three coordinates, where there is mutual dependence between the coordinates. (Note that, if there is no dependency between the coordinates there will be no redundant information and concequently we will not be able to compress the signal without losing relevant information.)
Consider a vector containing three coordinates where each is represented as

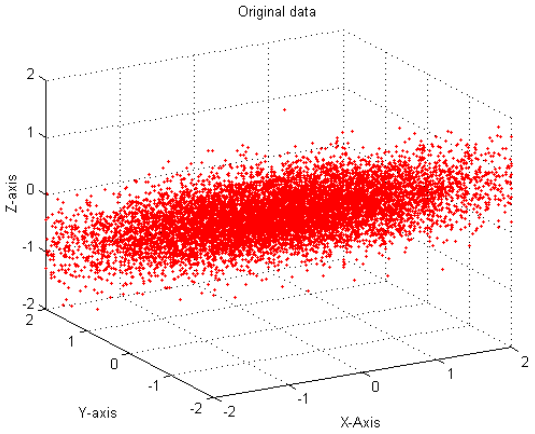
$$\begin{cases} x_1 &= & \varepsilon_1 \\ x_2 &= & -0.5x_1 + 0.5\varepsilon_2 \\ x_3 &= & 0.4x_1 + 0.2x_2 + 0.1\varepsilon_3 \end{cases}$$

where $\varepsilon_1, \varepsilon_2, \varepsilon_3$ are three independent Gaussianly distributed zero mean random variables with unit variance. One can see that the variables have dependencies on each other.

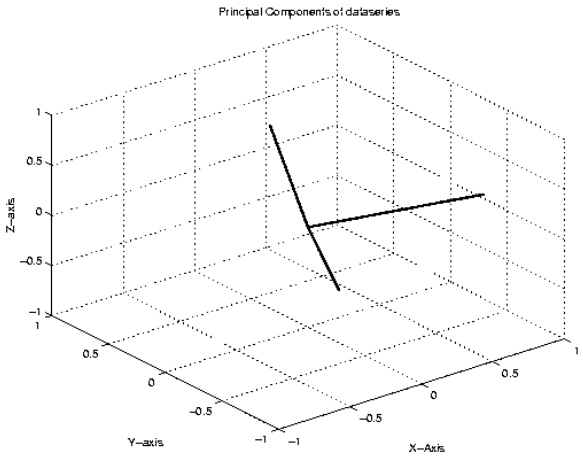
- Calculate, by hand, the (3-by-3) correlation matrix $R_{xx} = E\{xx^T\}$, where $E\{\cdot\}$ is the expectation operation.
- Use matlab to calculate the eigenvectors of the calculated correlation matrix (function: eig.m).
- Generate 5000 independent variables, $\varepsilon_1, \varepsilon_2, \varepsilon_3$, and create an realization of the signal

$$\mathbf{X}(n) = \begin{bmatrix} x_1(n) \\ x_2(n) \\ x_3(n) \end{bmatrix} = \begin{bmatrix} x_1(1)\ x_1(2) \ldots\ x_1(5000) \\ x_2(1)\ x_2(2) \ldots\ x_2(5000) \\ x_3(1)\ x_3(2) \ldots\ x_3(5000) \end{bmatrix} = [\mathbf{x}_1\ \mathbf{x}_2\ \ldots\ \mathbf{x}_{5000}].$$

An example of such an realization can be seen in the figure below, where each point in the 3-dimensional space corresponds to a vector in the realization above.



The principal components of the signal can be seen in the following figure:



- Use the Generalized Hebbian algorithm to find the principal components from the realization made before. That is, estimate the principal components from the data using an one layer linear Neural Network.
- Make a plot similar to the one above where both the eigenvectors (calculated) and the estimated principal components are shown. They should be similar to each other. The estimated principal components are the final weight vectors of the neural network. The largest principal component is estimated with the highest accuracy and the rest of the components are estimated with decreasing accuracy.

OBS: The estimated principal components can have different sign than the eigenvectors of the correlation matrix.

b) In this section we will use the principal components in order to compress a picture. The following picture is taken from a National park in Croatia by Dr. Xiao-Jiao Tao.

The picture have (280-by-420) pixels and each pixel is a real value in [0 1], (of course, we have finite precision in matlab). It can be acquired from the course web-site. The picture can be viewed by using the function call imshow.m from the image toolbox in matlab. It can be loaded as an ordinary matlab data file by the command: load picture1 (made for matlab v.4) and the image is represented by a matrix G.

- Divide the picture in blocks of size (10-by-20), this will give 28x21 (= 588) blocks. In order to create the input vectors, each block should be written as (200-by-1) vectors, where each column of the blocks has been added to the previous one. In this way we will have 588 input vectors of size 200.

In order to compress the signal we will project all these input vectors on *m* number of principal components estimated from these inputs. The principal components are estimated from the picture

using the Generalized Hebbian algorithm. In order to reconstruct the picture we just invert the projection.

- Use the input vectors and train a neural network to find the first 150 principal components. Use some stopping criteria.
- Project the inputs on these principal components in order to create the compressed signal.
  OBS: The compressed vectors will be of size (150-by-1) and the number of them is the same as the number of input vectors.
- Invert the projection in order to reconstruct the image. Use the pseudo-inverse of the weight matrix. Rearrange the reconstructed vectors back to blocks so that they can build up the picture.
- Make a plot of the reconstructed picture. Is the compression ratio 3:4 to high? Does the reconstructed picture look similar to the original one?
- If the picture looks very much distorted use more principal components, this will reduce the compression ratio but also maintain more of the information in the picture.

Remark: In conventional compression standards they use other projections than the principal components even though this one is optimal. The reason is that the computational cost of finding them is quite high. The cosine transform matrix is used in the MPEG2 standard as the projection matrix. This method will give performance which is close to the one using principal components.