The linear algebra of Principal Component Analysis

(with Python examples)

Kamila Zdybał

Université libre de Bruxelles, kamila.zdybal@ulb.ac.be
kamilazdybal.github.io/science-docs, kamila.zdybal@gmail.com

Please feel free to contact me with any suggestions, corrections or comments.

Preface

Principal Component Analysis (PCA) is a dimensionality reduction technique in which a high-dimensional data set is projected on directions of decreasing importance, ordered by the amount of variance explained. The technique exploits the fact that the original basis to represent the data set might not be an *optimal* one and there might be a redundancy of dimensions. Once a new, orthogonal basis is found, the data can be transformed to that new basis and that can give certain advantages. The projected data has lower rank and is thus easier to analyze. This in turn can be helpful in a variety of problems such as extracting information, data compression or analysis of structures hidden in the data [1].

These notes are in a way a tutorial on PCA with a deeper focus on linear algebra and statistics aspects governing this method. The aim is to present a deeper understanding of several linear algebra concepts than online tutorials typically give. My goal was also to gather all important knowledge on PCA in one place.

The prerequisite for these notes is mainly familiarity with undergraduate level linear algebra. You should be comfortable with the concept of matrices and operations on matrices. You will profit more from these notes if you have an intuitive understanding of eigendecomposition. If you need a refresher, the excellent series on linear algebra by 3Blue1Brown [2] will be, in my opinion, everything you need to go through these notes. In addition, few concepts from statistics such as variance and covariance will be used.

Keywords

principal component analysis, data reduction, dimensionality reduction, linear algebra, Python

Contents

Nomenclature

$\mathbf{A}$	is a matrix
$\mathbf{A}^\top$	is a matrix transpose
$\mathbf{A}^{-1}$	is a matrix inverse
$\mathbf{a}$	is a vector
a	is a scalar
$\mathbf{a}_{:j}$	is the j^{th} column of a matrix $\mathbf{A}$, it is equivalent to $\mathbf{A}(:, j)$
$\mathbf{a}_{i:}$	is the i^{th} row of a matrix $\mathbf{A}$, it is equivalent to $\mathbf{A}(i, :)$
$a_{i,j}$	is an element from i^{th} row and j^{th} column of a matrix $\mathbf{A}$, it is equivalent to $\mathbf{A}(i, j)$

1 Data sets for PCA

The data set for performing PCA is a matrix $\mathbf{X} \in \mathbb{R}^{n \times Q}$. Each column of $\mathbf{X}$ represents all n observations of one variable. Each row of $\mathbf{X}$ corresponds to a single observation of all Q variables[1]. This structure can be seen in Fig. 1.

1.1 Data pre-processing

In data science, we are typically given a raw data set which is not centered and not scaled. Standardizing the data set might be a good idea before applying a data science technique. To motivate data scaling with an illustrative example, you might think of a set of variables from a single experiment, representing temperature in the units of $[K]$ and range from 300-1500 K and associated pressures in the units of $[atm]$ which range from 1-1.1 atm. If we did not scale the data, the largest *spread* or the *variance* would be found in temperature, since on purely numerical grounds, the range 300-1500 is more significant than the range 1-1.1.

In particular, *centering* allows to look at data as variations from some center. Graphically, centering shifts the center of the *cloud* of data points (which in general is multi-dimensional) to a new, selected center. One of the popular choices is to center each variable by subtracting the mean of this variable's observations - the center will be shifted to the origin[2]. Other centering that could be encountered[3] is subtracting the minimum of the variable's observations. Centering thus substitutes the original data set with:

$$\mathbf{X_c} = \mathbf{X} - \mathbf{C}. \tag{1}$$

Here, a matrix of centers, $\mathbf{C} \in \mathbb{R}^{n \times Q}$, is created, for instance by computing the mean of each column of $\mathbf{X}$. In that case, a single column of $\mathbf{C}$ is populated with the same value representing the mean of the corresponding column in $\mathbf{X}$. A specific center $\mathbf{c}_j$ is then subtracted from a corresponding column $\mathbf{x}_j$.

Scaling, on the other hand, allows us to cancel the effect of various ranges that the variables in the data set might have, and treat all variables with equal (or at least similar) importance. A centered and scaled data set can then be written as:

$$\mathbf{X_{cs}} = \mathbf{X_c}\mathbf{D}^{-1}. \tag{2}$$

In the above equation, $\mathbf{D} \in \mathbb{R}^{Q \times Q}$ is a diagonal matrix. Each entry on a diagonal is the scaling factor for the corresponding column in $\mathbf{X}$. Every column in $\mathbf{X_c}$ gets divided by a corresponding scale from the diagonal of matrix $\mathbf{D}$. One of the popular scaling methods used in literature is the *auto scaling*[4] in which each column is divided by the standard deviation of that column. After auto scaling, all columns have a standard deviation equal to unity. For notation simplicity, for the rest of this document, I assume that $\mathbf{X}$ represents already pre-processed data (in place of $\mathbf{X_{cs}}$).

[1] This is also the data format that is needed for the MATLAB® function `pca` [3] and for the Python function `sklearn.decomposition.PCA` [4].

[2] see Fig. 7-8.

[3] for instance in the *Min-Max* standardizing.

[4] also known as *standard scaling* or *z-score* (although *z-score* refers to a combination of centering by the mean value and scaling by a standard deviation).

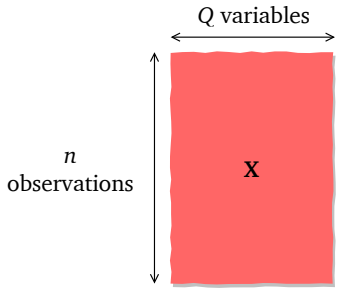


Figure 1: Structure of the data matrix for PCA.

2 Covariance matrix

Covariance between two random vectors, $\mathbf{x}$ and $\mathbf{y}$, is defined as:

$$\mathrm{cov}(\mathbf{x}, \mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}), \tag{3}$$

where n is the number of observations in a vector and $\bar{x}$ and $\bar{y}$ are mean values of $\mathbf{x}$ and $\mathbf{y}$ respectively. If we look at any data matrix, $\mathbf{X}$, as a composition of vectors formed by its columns, we may compute covariances of these vectors and store the result in another matrix, called a *covariance matrix*. This matrix is symmetric due to symmetry: $\mathrm{cov}(\mathbf{x}, \mathbf{y}) = \mathrm{cov}(\mathbf{y}, \mathbf{x})$. The off-diagonal elements have the meaning of covariance of two column vectors and the elements on the diagonal represent variance of each column, since $\mathrm{cov}(\mathbf{x}, \mathbf{x}) = \mathrm{var}(\mathbf{x})$ where:

$$\mathrm{var}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{\mathbf{x}})^2. \tag{4}$$

2.1 Construction

The starting point for performing PCA is to compute the covariance matrix from the data set $\mathbf{X}$. The covariance matrix, $\mathbf{S} \in \mathbb{R}^{Q \times Q}$, is given by:

$$\mathbf{S} = \frac{1}{n-1}\mathbf{X}^\top\mathbf{X}. \tag{5}$$

The similarity with the Eq. (3) might not be immediately evident. We will start by exploring the meaning of $\mathbf{X}^\top\mathbf{X}$. Let's look at the graphical representation of this matrix multiplication in Fig. 2. Any given column, say p (or k), of $\mathbf{X}$ represents all observations of a single p-th (or k-th) variable and can be viewed as vector with n elements. The same can be said about any row of a matrix $\mathbf{X}^\top$. Notice that the element at position (p,k) inside the covariance matrix has the interpretation of a dot product between a vector formed by the p-th row of a matrix $\mathbf{X}^\top$ and the k-th column of a matrix $\mathbf{X}$.

$$S_{p,k} = \frac{1}{n-1}(\mathbf{X}_p^\top \circ \mathbf{X}_k) = \frac{1}{n-1}\sum_{i=1}^{n} X_{p,i}X_{i,k} \tag{6}$$

Notice that as long as the columns of $\mathbf{X}$ have been centered, the off-diagonal elements of the covariance matrix have the mathematical meaning of covariance between two variables. In the case where we multiply row p with column p, we get a dot product of a vector with itself:

$$S_{p,p} = \frac{1}{n-1}(\mathbf{X}_p^\top \circ \mathbf{X}_p) = \frac{1}{n-1}\sum_{i=1}^{n} X_{p,i}X_{i,p} \tag{7}$$

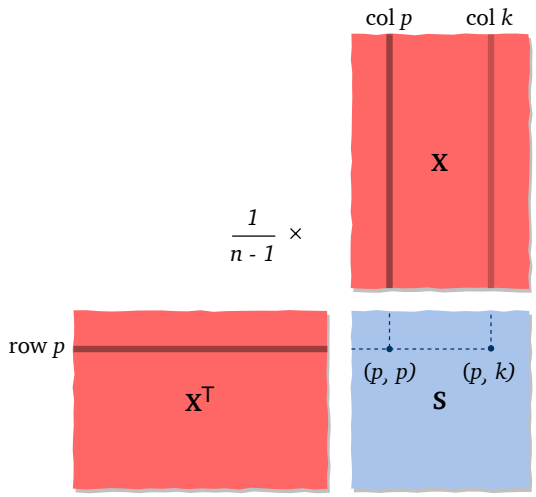


Figure 2: Covariance matrix **S** graphical interpretation.

This in turn looks similar to Eq. (4) and thus the diagonal elements of the covariance matrix represent the variance of a single variable.

In general, the dot product between two vectors $\mathbf{x}$ and $\mathbf{y}$ represents how much vector $\mathbf{x}$ lays in the direction of vector $\mathbf{y}$ (and vice versa) - and it is zero when two vectors are perpendicular to each other. This intuition can be carried to the covariance matrix. If any off-diagonal element is non-zero, say element at position (p,k) it means that some information about a p-th variable is carried by a k-th variable (and vice versa).

2.2 Covariance kernels

Interestingly, computing a dot product between two vectors is not the only way to *measure* covariance between variables. This might come as a surprise at first, but think for instance about different ways that we can measure distance. Euclidean distance is just *a* way and not *the* way - other distance metrics exist such as city block or Mahalanobis distance. Everything depends on your use case. The same story can be told about covariance. If up to this point in the tutorial I've managed to convince you that computing dot products is how we should populate the covariance matrix then you probably start to wonder: would there be any meaning in doing it differently? Well, here's a brain twister: as long as the resulting covariance matrix is symmetric, we can populate it however we want. In other words, we can generate a *rule* that takes two vectors and gives a scalar as a result. We often call that rule a *covariance function* or a *covariance kernel*. This might give us many advantages, as now we become chefs managing our variables in a data set and we specify how we would like our variables to correlate[5].

2.3 Properties of the covariance matrix

The covariance matrix is a very special matrix. It is worth pointing out some of its interesting properties that PCA makes an extensive use of. First, any matrix **S** constructed as $\mathbf{S} = \mathbf{C}^\top\mathbf{C}$ (where **C** is any real matrix) is square and symmetric. The reason for the symmetry can be seen looking at Fig. 2. Notice that the element (p,k) has the same value as the element (k,p) since it is a result of multiplying the same two vectors, just in different order. Notice also that there is a symmetry in: $\text{cov}(\mathbf{x},\mathbf{y}) = \text{cov}(\mathbf{y},\mathbf{x})$. Second, the eigenvalues of **S** are real and all are at least non-negative. Such matrix is called a *positive semidefinite* matrix. In a practical case that we are most interested in, this matrix has got only positive eigenvalues and we call it a *positive definite* matrix. Third, the eigenvectors of the covariance matrix are orthogonal. Now it gets even more interesting! You may already see some use-cases for such eigenvectors. One of which could be: they can form a new, Q-dimensional coordinate system – a *basis*. Hopefully I'm now on the right track to convincing you that such basis can be a very interesting basis.

[5]This concept is the basis for formulating Kernel PCA – a variation of the classical PCA presented here.

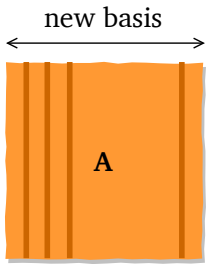


Figure 3: The orthonormal eigenvectors of the covariance matrix can form a Q-dimensional basis.

3 PCA workflow

As you may have already anticipated, the next step in PCA is to perform the eigendecomposition of the covariance matrix:

$$\text{eig}(\mathbf{S}) = [\mathbf{A}, \mathbf{\Lambda}], \tag{8}$$

where **A** is the matrix of eigenvectors and it is size $(Q \times Q)$. Each eigenvector defines the direction for the *principal components* (PCs). The principal components are orthogonal (due to properties discussed in 2.3) and therefore another important property holds: $\mathbf{A}^\top = \mathbf{A}^{-1}$ (proof[6]). The diagonal matrix $\mathbf{\Lambda}$ of size $(Q \times Q)$ is a matrix of the corresponding eigenvalues. Given the eigendecomposition of the matrix **S**, we may state that:

$$\mathbf{S} = \mathbf{A}\mathbf{\Lambda}\mathbf{A}^\top. \tag{9}$$

The principal components form a new basis in which we can represent our data set. We perform a transformation of the original data matrix **X** from the original space to the new space represented by the PCs. This transformation is achieved with the following matrix multiplication:

$$\mathbf{Z} = \mathbf{X}\mathbf{A}. \tag{10}$$

The new matrix **Z** is still our dataset **X** but represented in the basis associated with the matrix **A**. It is also called the *PC-scores* matrix, since one may think of every element in this matrix as a "score" that the corresponding element in **X** gets when represented in the new coordinate system after transformation. In the matrix multiplication from Eq. (10), every variable vector inside **X** gets transformed by the transformation matrix **A** and attains new scores in the basis associated with

[6]for orthogonal columns of **A** we have $\mathbf{A}^\top\mathbf{A} = \mathbf{I}$. Multiplying both sides from the right by $\mathbf{A}^{-1}$ we get $(\mathbf{A}^\top\mathbf{A})\mathbf{A}^{-1} = \mathbf{I}\mathbf{A}^{-1}$. Since matrix multiplication is associative, we may also perform: $\mathbf{A}^\top(\mathbf{A}\mathbf{A}^{-1}) = \mathbf{A}^{-1}$. From the definition of an inverse matrix $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$. Hence $\mathbf{A}^\top = \mathbf{A}^{-1}$.

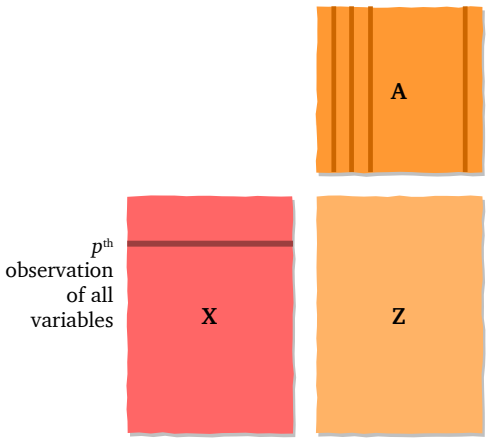


Figure 4: Data transformation to the new basis defined by **A**.

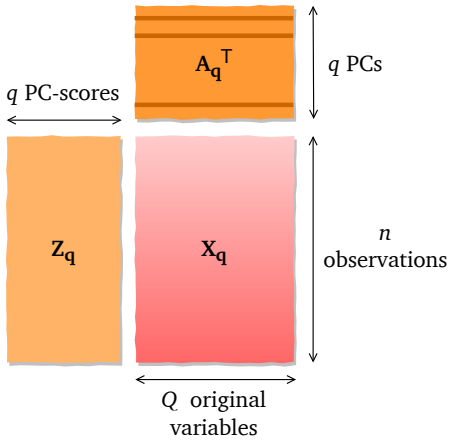


Figure 5: Data approximation with q first PCs.

A. The new representation of the old variable is now kept in the matrix **Z**.

We now approach the dimensionality reduction but first let's obtain the original data set back, given the PC-scores and the transformation matrix:

$$\mathbf{X} = \mathbf{Z}\mathbf{A}^\top. \tag{11}$$

The above equation is our route back to obtain the original data set in which the PC-scores are projected on the basis associated with a transposed eigenvectors matrix **A** (the fact that $\mathbf{A}^\top = \mathbf{A}^{-1}$ now appears in all of its usefulness). Suppose that we would like to find the approximation of the data matrix **X** with only q principal components (we project the PC-scores onto only q out of Q principal components). We shrink the transformation matrix **A** to the size $(Q \times q)$ (we only keep q columns of **A**). To match the matrix sizes we also need to shrink in size the PC-scores matrix which is originally of size $(n \times Q)$. We will denote these truncated matrices $\mathbf{A_q}$ and $\mathbf{Z_q}$ respectively. Projecting $\mathbf{Z_q}$ onto the basis $\mathbf{A_q}^\top$ will result in an an approximation of the original data set:

$$\mathbf{X_q} = \mathbf{Z_q}\mathbf{A_q}^\top. \tag{12}$$

How good such approximation is? It turns out that $\mathbf{X_q}$ is the closest rank-q approximation to **X** in terms of three popular norms: L^2-norm, Frobenius norm and trace norm. This is known as the Eckart-Young theorem [5].

4 Why eigenvectors?

In this section we come back to the Eq. (8) and answer the question: why are principal components the eigenvectors of a covariance matrix? To begin the understanding, let's look at Fig. 6. Principal Component Analysis aims to find a new, transformed data set **Z** such that if we computed a new covariance matrix as

$$\mathbf{S_Z} = \frac{1}{n-1}\mathbf{Z}^\top\mathbf{Z}, \tag{13}$$

the variances (the elements on the diagonal) are maximized and the covariances (the off-diagonal elements) are zero. This means that no more information about any column of **Z** is carried by any other column of **Z**. This removes the redundancy of information that could have been present in the original data set **X**; each column of **Z** now contributes to a "unique" piece of information that cannot be found in any other column of **Z**. In other words:

The goal of PCA is to diagonalize the new covariance matrix, $\mathbf{S_Z}$.

As a "template" for a diagonal matrix, PCA uses a diagonal matrix that is easily available (and which we have already produced!) - the matrix of eigenvalues $\mathbf{\Lambda}$. We will now ask ourselves: how do we need to construct **Z**, so that the matrix $\mathbf{S_Z} = \mathbf{\Lambda}$? We find **Z** through a series of transformations in which we combine Eq. (9) with Eq. (5):

$$\mathbf{A}\mathbf{\Lambda}\mathbf{A}^\top = \frac{1}{n-1}\mathbf{X}^\top\mathbf{X} \Big/ \mathbf{A}^\top \times \tag{14}$$

$$\mathbf{A}^\top\mathbf{A}\mathbf{\Lambda}\mathbf{A}^\top = \frac{1}{n-1}\mathbf{A}^\top\mathbf{X}^\top\mathbf{X} \Big/ \times \mathbf{A} \tag{15}$$

$$\mathbf{A}^\top\mathbf{A}\mathbf{\Lambda}\mathbf{A}^\top\mathbf{A} = \frac{1}{n-1}\mathbf{A}^\top\mathbf{X}^\top\mathbf{X}\mathbf{A} \tag{16}$$

$$\mathbf{I}\mathbf{\Lambda}\mathbf{I} = \frac{1}{n-1}\mathbf{A}^\top\mathbf{X}^\top\mathbf{X}\mathbf{A} \tag{17}$$

$$\mathbf{S_Z} = \mathbf{\Lambda} = \frac{1}{n-1}\mathbf{A}^\top\mathbf{X}^\top\mathbf{X}\mathbf{A} \tag{18}$$

It is thus visible, that if we chose $\mathbf{Z} = \mathbf{X}\mathbf{A}$, the product $\mathbf{Z}^\top\mathbf{Z}$ gives a diagonal matrix.[7]

There is one more great thing that PCA achieves: the elements on the diagonal of the new covariance matrix are maximized. PCA achieves both things at the same time: it diagonalizes the new covariance matrix and it makes the diagonal elements maximum possible. So in fact, we should say:

The goal of PCA is to diagonalize the new covariance matrix $\mathbf{S_Z}$ and maximize the elements on the diagonal.

Hopefully you could now see for yourself that the basis associated with the eigenvectors of the covariance matrix is a very interesting basis! It lets us diagonalize the product $\mathbf{Z}^\top\mathbf{Z}$ and it orders the PC-scores from the most to the least *informative* ones.

[7] Note, that if $\mathbf{Z} = \mathbf{X}\mathbf{A}$, then $\mathbf{Z}^\top = (\mathbf{X}\mathbf{A})^\top = \mathbf{A}^\top\mathbf{X}^\top$.

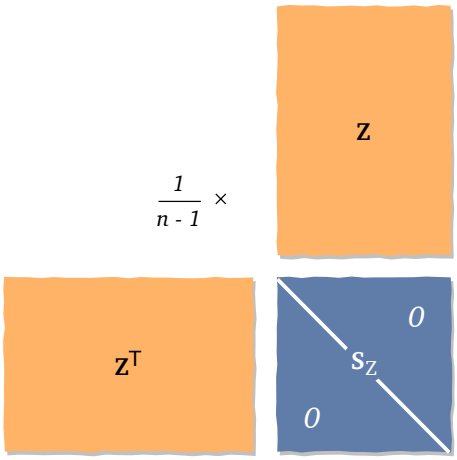


Figure 6: New transformed data set $\mathbf{Z}$ and its diagonal covariance matrix $\mathbf{S_Z}$.

Appendix

This tutorial was generated from a Jupyter notebook that can be accessed here.

A Visualizing PCA with Python

A.1 Plotting the workflow

We will now go on to visualizing on an artificial 2D data set every step of PCA. We will use the `PCA` function from a Python library `sklearn.decompositions`. The full code can be accessed in the GitHub repository. Here, we will only recall elements of that code that are for performing PCA. We create the `Dataset` (the equivalent of $\mathbf{X}$) as follows:

```
import numpy as np
Np = 100
x = np.linspace(3, 6, Np)
y = 0.8*x + 1*np.random.rand(Np)
Dataset = np.column_stack((x, y))
```

Let's assume that the first column of this data set are realizations of the first variable x and the second column are realizations of the second variable y.

The two variables x and y span two dimensional space but the data set exhibits a low-dimensional structure which is easily visible to the eye just by looking at the Fig. 7. We already see that the data seems to be spread along some linear function. Perhaps changing the basis to a basis associated with this linear function will be a more effective representation by our data set? Note here also, that for multidimensional data sets "seeing" such data structures is no longer possible (as it still is in 2-D or 3D). We need to rely on the dimensionality reduction technique that we chose to find this low-dimensional structure for us.

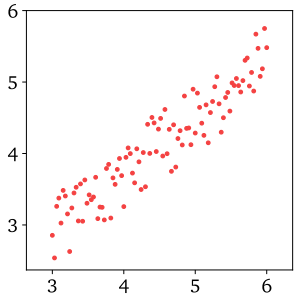


Figure 7: Raw data set.

We center the data set, which simply moves the center of the cloud of points to the origin of the coordinate system. If necessary, data set would also be scaled to allow for even comparison of the two variables.

```
Dataset_mean = np.mean(Dataset, axis=0)
Dataset_proc = Dataset - Dataset_mean
```

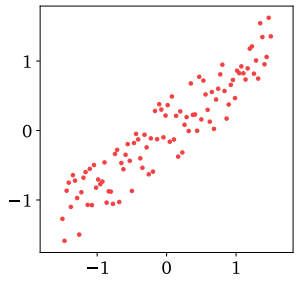


Figure 8: Data set centered.

Next, PCA is performed on the dataset and the eigenvectors (the Principal Components) `PCs` are found, with the corresponding eigenvalues `eigvals`.

```
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(Dataset)

eigvals = pca.explained_variance_ratio_
PCs = pca.components_
PCscores = pca.transform(Dataset)
```

In the above code, we create an object `pca` of class `PCA`. We train the model with our `Dataset` using the `fit` function.

The eigenvectors are plotted on the data set in Fig. 9. Their lengths are proportional to their corresponding eigenvalue. Notice that PCA was able to find the direction of the largest variance in the data marked by the direction of the first, longest Principal Component. The second PC is perpendicular to the first one.

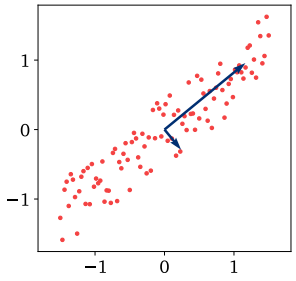


Figure 9: Data set with principal components.

We also compute the transformed data set PCscores represented in the basis associated with the obtained PCs.

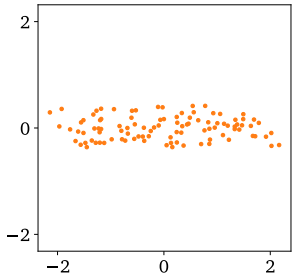


Figure 10: PC-scores.

Next, the PC-scores can be projected on the first PC, to reduce the dimensionality - from two dimensions to one. The data representation from Fig. 11 can be viewed as the "scores" each data point would attain it represented on 1-dimensional structure associated with the first Principal Component.

```
q = 1
Dataset_projected = np.dot(Dataset_proc,
np.transpose(pca.components_[:q,:]))
```

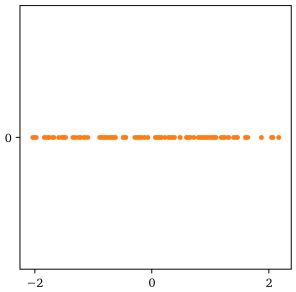


Figure 11: Data projection on lower dimension.

We reconstruct the original data set from the reduced space. This represents going back from the 1-dimensional space to the original dimensions. The mean of the data set is added back to undo the data centering.

```
Dataset_approx =
np.dot(pca.transform(Dataset)[:,:q],
pca.components_[:q,:]) + Dataset_mean
```

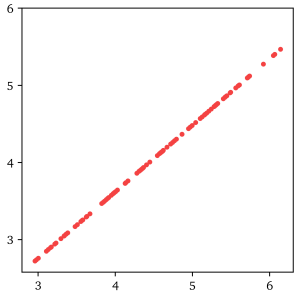
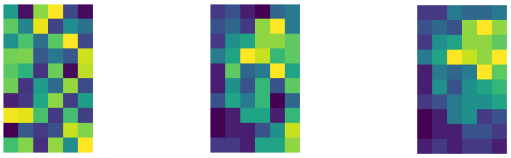


Figure 12: Data approximation with $q = 1$.

A.2 Low-rank approximations

We perform PCA on three artificially generated matrices of size (10×6): a **random** matrix which is populated by random floats in the range 0-1 and a **semi-structured** and **structured** matrices whose elements are also in the range 0-1 but were populated so that there is an increasing level of structure that was judged visually. These matrices are presented in Fig. 13.



(a) (b) (c)

Figure 13: Original data matrices: (a) random matrix, (b) semi-structured matrix, (c) structured matrix.

The visual judgment of the level of imposed "structure" is quite objective in this exercise but the aim was to group the elements of high numerical value (most yellow) in a single region of the matrix and elements of low numerical value (most purple) in other regions of the matrix.

The level of the imposed structure can also be observed quantitatively after performing PCA from the eigenvalue distribution, presented in Fig. 14. The structured matrix has got the strongest decaying behaviour which suggest that the matrix can be approximated by relatively low number of modes and hence exhibits the strongest low-rank structure. The first PC is expected to carry 80% of the total variance in the structured data matrix.

We reconstruct the original data matrices using a certain number q of PC-scores and corresponding PCs. Using the Matlab notation we may write the approximation as:

$$\mathbf{D}_{\text{app}} = \text{PC-scores}(:, 1:q) \cdot \text{PCs}^{\top}(1:q, :) + \mathbf{D}_{\text{mean}} \tag{19}$$

which is equivalent to Eq. (12) and to the Python approximation presented in Sec.A.1.

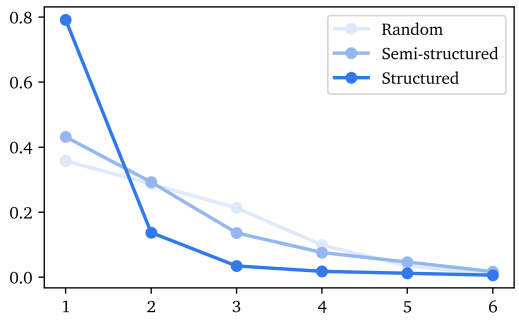


Figure 14: Eigenvalue distribution after performing PCA on original data matrices.

The above multiplication is presented in three cases in Fig. 15. We can see a rank-1 approximation of the original matrices using the 1^{st} Principal Component found by PCA. The vectors (10×1) represent the PC-scores and the vectors (1×6) represent the PCs.

What can be seen visually is that for the semi-structured and structured matrix the low numerical value region is clearly separated. For the random matrix, only few regions of lowest and highest numerical values are recovered in the rank-1 approximation.

It is worth noticing here that indeed the obtained matrices are necessarily rank-1, since they are formed as a linear combination of a single vector. This can be seen in two ways: either you may take the vector of PC-scores (10×1) and assume that it forms every column of the (10×6) matrix through multiplying it by the corresponding element from the PC vector. Or, you may assume that the PC is a vector that forms every row of the (10×6) matrix when multiplied by the corresponding element from the PC-scores vector. In either case, the full matrix (10×6) becomes a linear combination of a single vector - hence, it is rank-1.

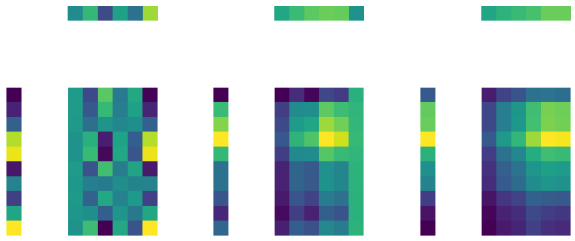


Figure 15: Reconstruction with 1^{st} Principal Component of: (a) random matrix, (b) semi-structured matrix, (c) structured matrix.

In Fig. 16 we present a rank-2 approximation, where we maintained two first PCs. Again, the matrices (10×2) represent the PC-scores and the vectors (2×6) represent the PCs.

In the semi-structured and structured matrix, the single matrix elements with highest numerical values (yellow) are already recovered in their actual positions. In the random matrix this is still not the case with rank 2-approximation.

Following the analogous reasoning as for Fig. 15 we may notice that the matrix reconstructed with two PCs is a rank-2 matrix.

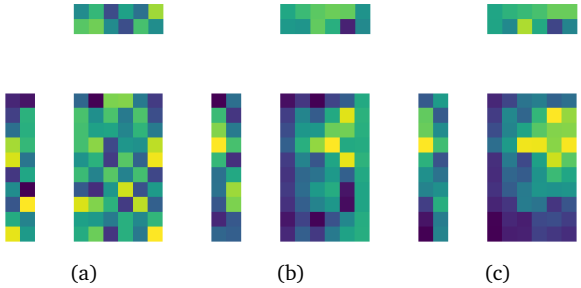


Figure 16: Reconstruction with 2 Principal Components of: (a) random matrix, (b) semi-structured matrix, (c) structured matrix.

The original data matrices are not completely retrieved until all 6 Principal Components and 6 PC-scores are taken into account. In Fig. 17 we obtain the final data matrices of rank-6.

The PC-scores are low-dimensional representations of the original data matrix and we return to the original dimensions by the transformation from Eq. (12). In the case of taking all 6 PCs, the PC-scores $\mathbf{Z_q} = \mathbf{Z}$ and the Eq. (19) becomes the Eq. (11), rather than the approximation from Eq. (12).

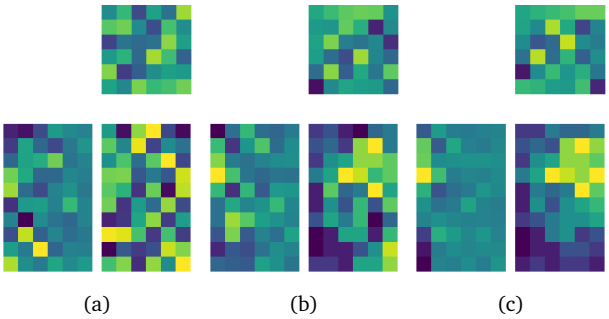


Figure 17: Reconstruction with 6 Principal Components of: (a) random matrix, (b) semi-structured matrix, (c) structured matrix.

Indeed, the multiplications from Fig. 17 show full PC-scores matrices transformed to the original 6 dimensions using the inverse (transposition) of the basis matrix made from PCs. The resulting matrix has to be the original data matrix.

A.3 Local PCA

PCA can also be applied on portions of the entire data set, in *local clusters*. This can have certain advantages. Firstly, the reconstruction of the data set from low-rank approximations in the local clusters can allow for further reduction in dimensionality of a non-linear data set. This is due to the fact that clustering creates portions that are locally linear (or at least that are close to being linear). Secondly, the interpretation of Principal Components in local clusters can have more physical meaning, since they become better suited to represent that specific cluster.

Below is an example of the general idea of performing Local PCA. A data set composed of two distinct clouds of data can be first partitioned (for instance by techniques such as K-Means clustering) and then PCA is performed separately on both portions of the entire data set. What happens in practice is that the same PCA workflow is applied but on a subset of the data. Notice that the found PCs have different directions.

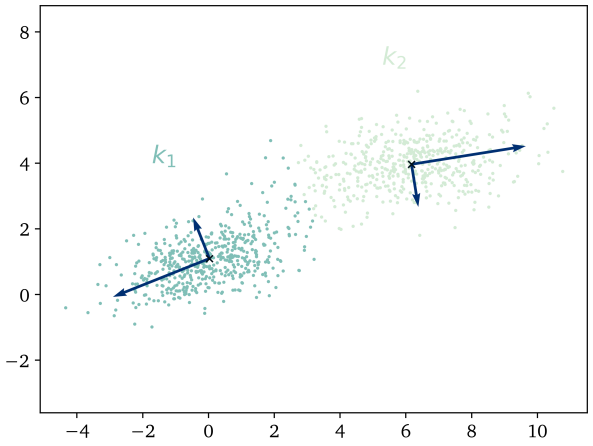


Figure 18: Local PCA.

In the second example presented in the figure below we have a data set with a non-linear behaviour. Performing PCA on the entire data set will result in two PCs that will not represent very accurately the direction of variance of the most "bent" region at the top of the figure. You can observe that after dividing the data into three clusters the local PCs adjust to the direction of variance in each cluster.

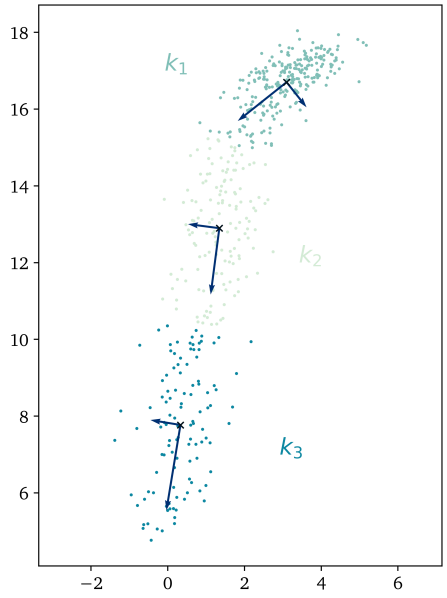


Figure 19: Local PCA.

The reconstruction from the Local PCA is slightly more involved since now we have to account for contributions from each cluster separately.

B Questions to pause and ponder

To give you a chance for further exposure to PCA, here are a few questions for you to think about. Take a piece of paper and a pencil, or use your favourite programming language to explore these.

1. What happens if you did PCA on a rank-deficient matrix? Would it still find all the Principal Components? And if yes, what's the meaning of that...? What would the eigenvalues look like?
2. What happens if the columns of the original data matrix $\mathbf{X}$ are *very close* to being linearly dependent? What I mean by that is, let's assume that matrix $\mathbf{X}$ is still of full-rank, but each column is approximately a linear combination of some other column.
3. What happens if you do PCA on a *ring* data set like the one in Fig. 20?

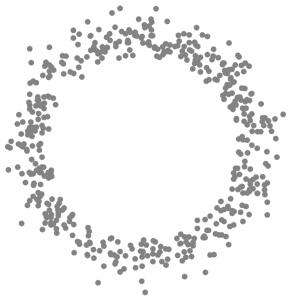


Figure 20: Ring data set.

4. What would the original data set $\mathbf{X}$ have to be so that the PCs found are the standard basis vectors?

References

[1] Herve Abdi, Lynne J. Williams, *Principal component analysis*, 2010

[2] 3Blue1Brown, *Essence of linear algebra*

[3] https://nl.mathworks.com/help/stats/pca.html

[4] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[5] C. Eckart, G. Young, *The approximation of one matrix by another of lower rank*, 1936, Psychometrika, 1 (3) 211-218

[6] Ian T. Jolliffe, *Principal Component Analysis*, Second Edition, 1986

[7] Gilbert Strang, *Introduction to Linear Algebra*, Fifth Edition, 2016

[8] Jonathon Shlens, *A Tutorial on Principal Component Analysis*, 2016, https://arxiv.org/abs/1404.1100

[9] http://people.sju.edu/ pklingsb/dot.cov.pdf

[10] J. Edward Jackson, *A User's Guide To Principal Components*, 1991

[11] Lindsay I. Smith, *A tutorial on Principal Component Analysis*, 2002

[12] Cosma Shalizi, Course on statistics: *The Truth about Principal Components and Factor Analysis*, 2009 https://www.stat.cmu.edu/ cshalizi/350/