Bruxelles, January 2019

The linear algebra of Principal Component Analysis

(with Python examples)

Kamila Zdybał

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

Preface

These are notes on **Principal Component Analysis** (PCA), a dimensionality reduction technique in which a large data set is reduced to maintain only the directions of the largest variance.

These notes are in a way a tutorial on PCA with a deeper focus on linear algebra and statistics aspects that govern this method, compared to other PCA tutorials that I have so far found on the web. I have an aim that the reader will both gain a deep and intuitive understanding as well as an appreciation for PCA after going through these notes. This requires a deep and intuitive understanding of several linear algebra concepts such as the eigendecomposition or matrix operations.

Some initial level of linear algebra knowledge and intuition is hence required. I highly recommend a very pleasant to watch course by 3Blue1Brown YouTube channel, which, to my belief, will be everything you need to master about linear algebra to gain a great intuition behind PCA.

This document is still in preparation. Please feel free to contact me with any suggestions, corrections or comments.

Keywords

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

Contents

1 Nomenclature 1

2 Motivation for data reduction 1

3 Data sets for PCA 1

4 Data pre-processing 2

5 Covariance matrix 2
- 5.1 Construction 2
- 5.2 Properties 3

6 PCA workflow 3

7 Why eigenvectors? 3

8 How it all links together 4

9 Python examples 4
- 9.1 Plotting the workflow 4
- 9.2 Low-rank approximations 6

A Highlights of statistics 7

1 Nomenclature

$\boldsymbol{A}$	is a matrix
$\boldsymbol{A}^T$	denotes a matrix transpose
$A_{:,j}$	is a vector formed by the j^{th} column of a matrix $\boldsymbol{A}$, it is equivalent to $\boldsymbol{A}(:,j)$
$A_{i,:}$	is a vector formed by the i^{th} row of a matrix $\boldsymbol{A}$, it is equivalent to $\boldsymbol{A}(i,:)$
$a_{i,j}$	is an element from i^{th} row and j^{th} column of a matrix $\boldsymbol{A}$, it is equivalent to $\boldsymbol{A}(i,j)$

2 Motivation for data reduction

To motivate why studying techniques such as PCA and learning to apply them on real data sets is important, there are several questions which stimulated the development of data reduction and data decomposition techniques:

1. Can we send less data but preserve maximum information contained by the data (data compression)?
2. Can we predict what the outcome of another observation will be?
3. Can we build a model in a purely data-driven way - based on data that we have collected (e.g. from experiments or numerics)?
4. Can we make sense of a large, multi-dimensional data set (which cannot be plotted graphically)?
5. Can we extract low-dimensional features that underlay the data?

All of these questions are not necessarily answerd by PCA. Different applications might work well with different data reduction techniques.

3 Data sets for PCA

For the rest of this document, we assume that the data set for performing PCA is structured as follows: each column represents one variable (feature) V_j observations and the rows correspond to all variable's observations (samples) O_i, e.g. at different positions in space, or in time. Let $i \in \langle 1, n \rangle$ and $j \in \langle 1, Q \rangle$. The raw data matrix $\boldsymbol{X}$ is hence size $(n \times Q)$.[1] This structure can be seen in Fig.1.
One of the strengths of PCA is that it can be performed on a non-time-resolved data set, namely the matrix $\boldsymbol{X}$ can be collected at a single time snapshot and be only spatially-resolved.

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

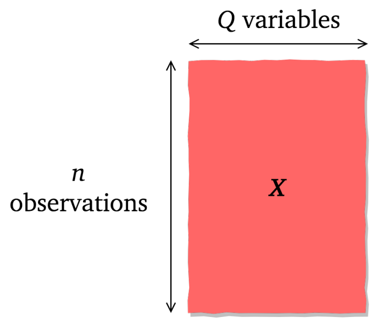

Figure 1: Data matrix for PCA.

4 Data pre-processing

In data science we are typically given a raw data set which is not centered and not scaled.
Data centering allows to look at it as variations from some center. Typically we might center each variable by subtracting the mean of this variable's observations (realizations) - we would then look at the data as variations from the mean. This would substitute the original data set with:

$$\boldsymbol{X'} = \boldsymbol{X} - \text{mean}(\boldsymbol{X}) \tag{1}$$

Such centering will shift the "cloud" of data points (which in general is multi-dimensional) to the origin. However, other values for centering are possible.
Scaling, on the other hand, allow us to cancel the effect of unit that a variable might have and treat all variables with equal importance.

$$\boldsymbol{X''} = \boldsymbol{X'}\boldsymbol{D}^{-1} \tag{2}$$

In the above equation, the matrix $\boldsymbol{D}$ is a diagonal matrix whose entries are the corresponding scalings. Hence, every column of the matrix $\boldsymbol{X'}$ gets divided by a corresponding scale from the diagonal of the matrix $\boldsymbol{D}$.
You might for instance think about 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.
For simplicity, for the rest of this document we assume that $\boldsymbol{X}$ represents pre-processed data (in place of $\boldsymbol{X'}$ or $\boldsymbol{X''}$).

5 Covariance matrix

The covariance between two random vectors X and Y is defined as:

$$\text{cov}(X,Y) = \frac{1}{N}\sum_{i=1}^{N}(X_i - \bar{X})(Y_i - \bar{Y}) \tag{3}$$

where N is the number of samples. If we look at any data matrix as a composition of vectors formed by its columns, we may compute the covariances of these vectors and store the result in another matrix, called a *covariance matrix*. This matrix is symmetric, due to symmetry $\text{cov}(X,Y) = \text{cov}(Y,X)$. The off-diagonal elements have the meaning of covariance of two random vectors and the elements on the diagonal are a variance, since $\text{cov}(X,X) = \text{var}(X)$.

5.1 Construction

The starting point for performing PCA is to compute a covariance matrix from the data set. The covariance matrix is given by:

$$\boldsymbol{S} = \frac{1}{n-1}\boldsymbol{X}^T\boldsymbol{X} \tag{4}$$

and is therefore size $(Q \times Q)$. Notice the similarity with the eq.(3).
We will start by exploring the meaning of $\boldsymbol{X}^T\boldsymbol{X}$. Let's look at the graphical representation of this matrix multiplication in Figure 2.
Any given column, say p or k, of the data matrix $\boldsymbol{X}$ represents all measurements of a single variable V_p (or V_k) and can be viewed as a single n-dimensional vector. The same can be said about any row of a matrix $\boldsymbol{X}^T$.

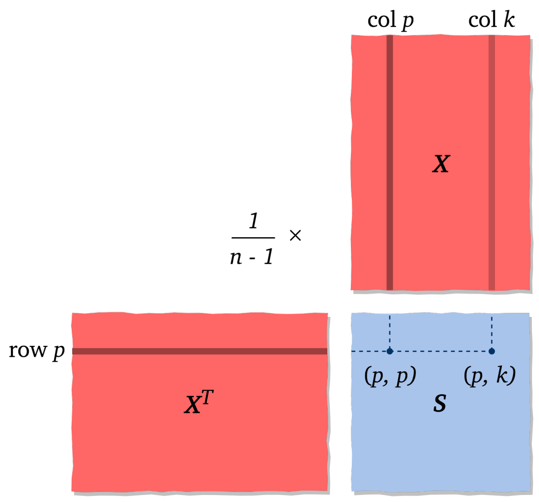

Figure 2: Covariance matrix $\boldsymbol{S}$ graphical interpretation.

Notice that an element at position (p,k) inside the matrix $\boldsymbol{S}$ has the interpretation of a dot product between a vector formed by the p-th row of a matrix $\boldsymbol{X}^T$ and a k-th column of a matrix $\boldsymbol{X}$, with the $\frac{1}{n-1}$ factor out front to which we come back in the Appendix A.

$$s_{p,k} = \frac{1}{n-1}\text{dot}(\boldsymbol{X}^T(p,:), \boldsymbol{X}(:,k)) \tag{5}$$

In a special case, where we multiply the row p with the column p, we get a dot product of a vector with itself.

$$s_{p,p} = \frac{1}{n-1}\text{dot}(\boldsymbol{X}^T(p,:), \boldsymbol{X}(:,p)) \tag{6}$$

In general, the dot product between two vectors X and Y represents how much vector X lays in the direction of vector Y (and vice versa) - and it is zero when two vectors are perpendicular to each other. This intuition can be carried to our covariance matrix $\boldsymbol{S}$. If any off-diagonal element is non-zero, say element at position (p,k) this means that some information about variable V_p is carried by a variable V_k (and vice versa).
In other words, every element in the covariance matrix is computed as:

$$s_{i,j} = \frac{1}{n-1}\sum_{k=1}^{n} V_i(k)V_j(k) \tag{7}$$

5.2 Properties

The covariance matrix is a very special matrix and it is worth pointing out some of its interesting properties that the Principal Component Analysis makes use of.
First of all, a matrix constructed as: $\boldsymbol{S} = \boldsymbol{C}^T\boldsymbol{C}$ (where $\boldsymbol{C}$ is any real matrix) is square and symmetric (the reason for this is easy to see from Fig.2). Apart from that, the eigenvalues of such matrix 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.
Another important property is that the eigenvectors of the covariance matrix are orthogonal, which is a very special thing indeed. You may already see some interesting uses for such eigenvectors. One of which could be: they can form a new coordinate system - a *basis*. The question thus remains: can such basis be any interesting basis?

6 PCA workflow

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

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

The matrix $\boldsymbol{A}$ is a matrix of eigenvectors and it is size $(Q \times Q)$. Each eigenvector is called a *principal component* (PC). The principal components are orthogonal to each other, and therefore the following important property holds: $\boldsymbol{A}^T = \boldsymbol{A}^{-1}$ (proof[2]).
The diagonal matrix $\boldsymbol{\Lambda}$ of size $(Q \times Q)$ is a matrix of the corresponding eigenvalues.
Given the eigendecomposition of matrix $\boldsymbol{S}$, we may state that:

$$\boldsymbol{S} = \boldsymbol{A}\boldsymbol{\Lambda}\boldsymbol{A}^T \tag{9}$$

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

$$\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{A} \tag{10}$$

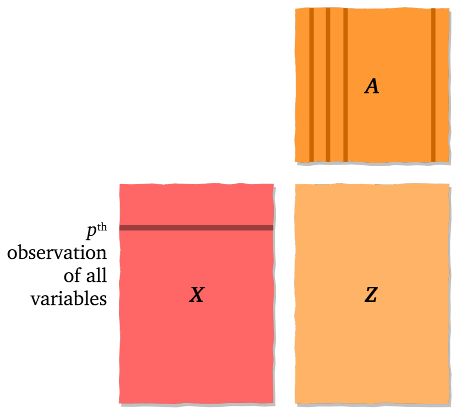

Figure 3: Data transformation to a new basis.

[2]Proof: for orthogonal columns of $\boldsymbol{A}$ we have $\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}$ Multiplying both sides by $\boldsymbol{A}^{-1}$ we get $(\boldsymbol{A}^T\boldsymbol{A})\boldsymbol{A}^{-1} = \boldsymbol{I}\boldsymbol{A}^{-1}$. Since matrix multiplication is associative, we may also perform: $\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^{-1}) = \boldsymbol{A}^{-1}$. From definition of an inverse matrix, $\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{I}$, hence: $\boldsymbol{A}^T = \boldsymbol{A}^{-1}$.

The new matrix $\boldsymbol{Z}$ is still our dataset $\boldsymbol{X}$ but represented in the basis associated with the matrix $\boldsymbol{A}$. 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 $\boldsymbol{X}$ gets when represented in the new coordinate system after transformation.
In the matrix multiplication from eq.(10), every variable vector inside $\boldsymbol{X}$ gets transformed by the transformation matrix $\boldsymbol{A}$ and attains new scores in the basis associated with $\boldsymbol{A}$. The new representation of the old variable is now kept in the matrix $\boldsymbol{Z}$.
We now approach the dimensionality reduction but first let's obtain the original data set back, given the PC-scores and the transformation matrix:

$$\boldsymbol{X} = \boldsymbol{Z}\boldsymbol{A}^T \tag{11}$$

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 $\boldsymbol{A}$ (recall that $\boldsymbol{A}^T = \boldsymbol{A}^{-1}$).
Suppose that we would like to find the approximation of the data matrix $\boldsymbol{X}$ with only q principal components (we project the PC-scores onto only q out of Q principal components).
We shrink the transformation matrix $\boldsymbol{A}$ to be of size $(Q \times q)$ (we only keep q principal components). To match the matrix sizes we also need to shrink in size the PC-scores matrix which originally is size $(n \times Q)$ - the same size as the data matrix $\boldsymbol{X}$. We will denote these truncated matrices $\boldsymbol{A_q}$ and $\boldsymbol{Z_q}$ respectively.
Projecting $\boldsymbol{Z_q}$ onto the basis $\boldsymbol{A_q^T}$ will result in an an approximation of the original data set:

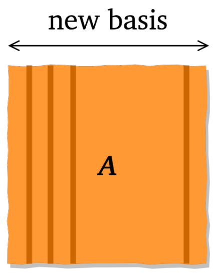

Figure 4: Eigenvalues of the covariance matrix form a new basis.

$$\boldsymbol{X_q} = \boldsymbol{Z_q}\boldsymbol{A_q^T} \tag{12}$$

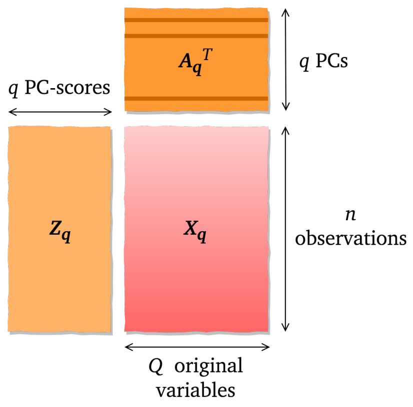

Figure 5: Data approximation with q-PCs.

7 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?

The goal of PCA in terms 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 $\boldsymbol{Z}$ such that if we computed a new covariance matrix in such a way:

$$\boldsymbol{S_Z} = \frac{1}{n-1}\boldsymbol{Z}^T\boldsymbol{Z} \tag{13}$$

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 $\boldsymbol{Z}$ is carried by any other column of $\boldsymbol{Z}$. This removes the redundancy of information that could have been present in the original data set $\boldsymbol{X}$; each column of $\boldsymbol{Z}$ now contributes to a "unique" piece of information that cannot be found in any other column of $\boldsymbol{Z}$.
In other words, what we want to achieve with PCA is to diagonalize this new covariance matrix $\boldsymbol{S_Z}$. As a "template" PCA uses a diagonal matrix that is easily available (and which we have already produced), and which is guaranteed to be diagonal - the matrix of eigenvalues $\boldsymbol{\Lambda}$.
We will now ask ourselves: how do we need to construct $\boldsymbol{Z}$, so that the matrix $\boldsymbol{S_Z} = \boldsymbol{\Lambda}$?
We find $\boldsymbol{Z}$ through a series of transformations in which we combine eq.(9) with eq.(4):

$$\boldsymbol{A\Lambda A}^T = \frac{1}{n-1}\boldsymbol{X}^T\boldsymbol{X} \;\Big/ \boldsymbol{A}^T\times \tag{14}$$

$$\boldsymbol{A}^T\boldsymbol{A\Lambda A}^T = \frac{1}{n-1}\boldsymbol{A}^T\boldsymbol{X}^T\boldsymbol{X} \;\Big/ \times\boldsymbol{A} \tag{15}$$

$$\boldsymbol{A}^T\boldsymbol{A\Lambda A}^T\boldsymbol{A} = \frac{1}{n-1}\boldsymbol{A}^T\boldsymbol{X}^T\boldsymbol{XA} \tag{16}$$

$$\boldsymbol{I\Lambda I} = \frac{1}{n-1}\boldsymbol{A}^T\boldsymbol{X}^T\boldsymbol{XA} \tag{17}$$

$$\boldsymbol{S_Z} = \boldsymbol{\Lambda} = \frac{1}{n-1}\boldsymbol{A}^T\boldsymbol{X}^T\boldsymbol{XA} \tag{18}$$

It is thus visible, that if we chose $\boldsymbol{Z} = \boldsymbol{XA}$, the product $\boldsymbol{Z}^T\boldsymbol{Z}$ give a diagonal matrix.[3]

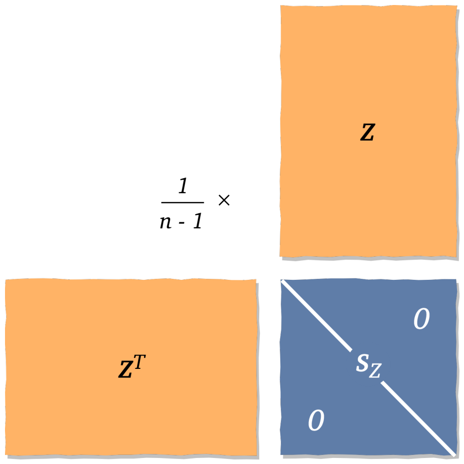

Figure 6: New transformed data set $\boldsymbol{Z}$ and its diagonal covariance matrix $\boldsymbol{S_Z}$.

There is one more thing owed explaining. We mentioned before that the elements on the diagonal are maximized, so is this also achieved by taking the eigenvalues matrix as a template? It turns out that the answer is yes. PCA achieves both things at the same time: it diagonalizes the new covariance matrix and it makes the diagonal elements maximum possible. Pretty neat, no?

[3]Note here that if $\boldsymbol{Z} = \boldsymbol{XA}$, then $\boldsymbol{Z}^T = (\boldsymbol{XA})^T = \boldsymbol{A}^T\boldsymbol{X}^T$.

We can now answer the previously posed question: the basis associated with the eigenvectors of the covariance matrix is a very interesting basis! It lets us diagonalize the product $\boldsymbol{Z}^T\boldsymbol{Z}$ and it orders the PC-scores from the most to the least "informative" ones.

8 How it all links together

Hopefully by now the steps of PCA begin to look connected. In fact, at least to my eye, PCA technique seems like a "series of fortunate events" that makes a very effective usage of a few popular concepts from linear algebra.

9 Python examples

9.1 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 $\boldsymbol{X}$) as follows:

```python
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.

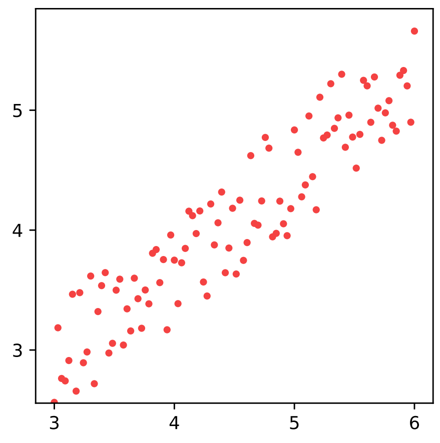

Figure 7: Raw data set.

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
```

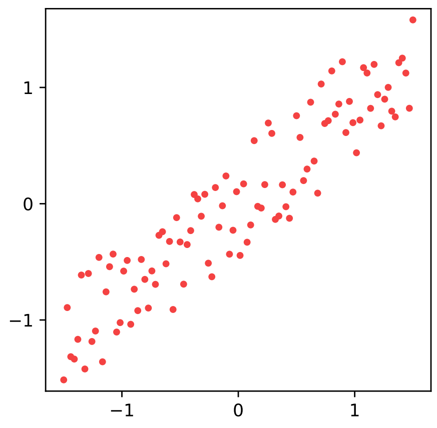

Figure 8: Data set centered.

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.

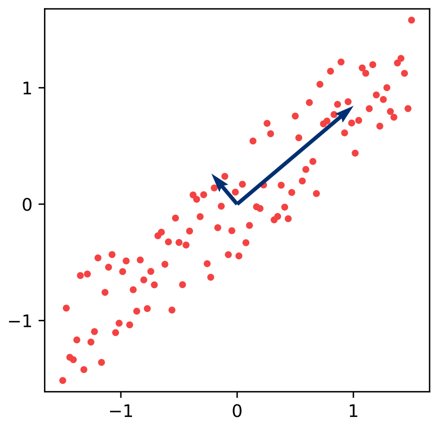

Figure 9: Data set with principal components.

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

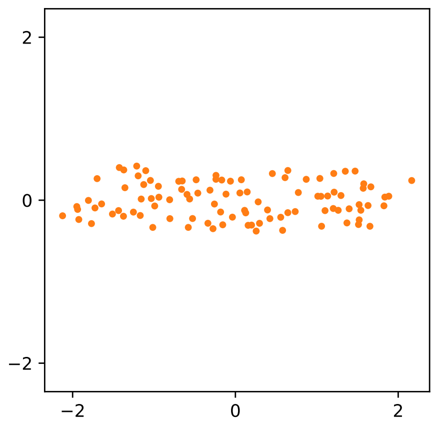

Figure 10: PC-scores.

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,:]))
```

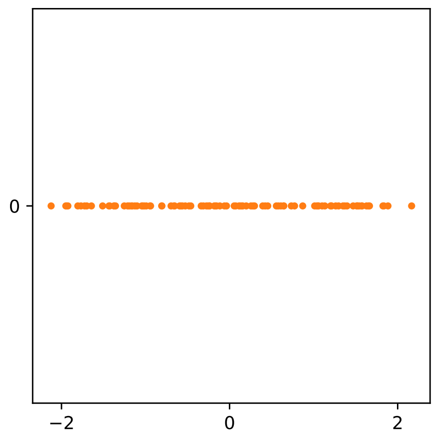

Figure 11: Data projection on low dimension.

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
```

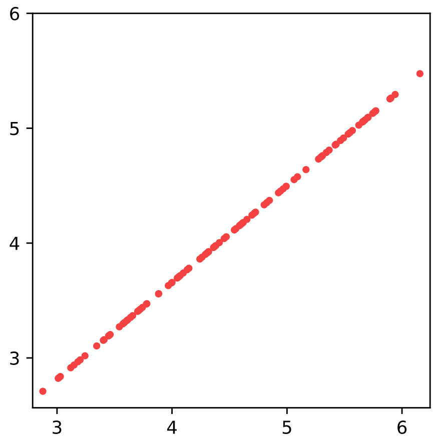

Figure 12: Data approximation with $q = 1$.

9.2 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 by the user and have an increasing level of structure that was judged visually. These matrices are presented in Fig.13.

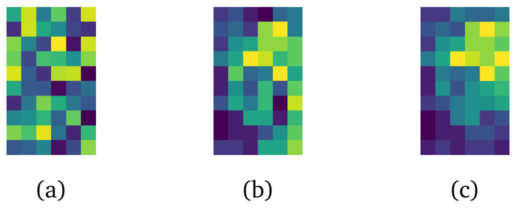

Figure 13: Original data matrices: (a) random matrix, (b) semi-structured matrix, (c) structured matrix.

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 (most purple) value 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.

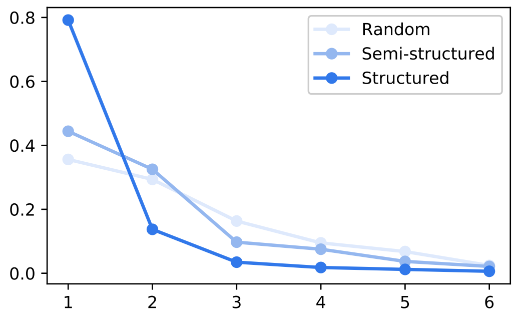

Figure 14: Eigenvalue distribution after performing PCA on original data matrices.

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:

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

which is equivalent to eq.(12) and to the Python approximation presented in Sec.9.1.
The above multiplication is presented in three cases in Fig.15. We can see a rank-1 approximation of the original matrices using the 1st 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.

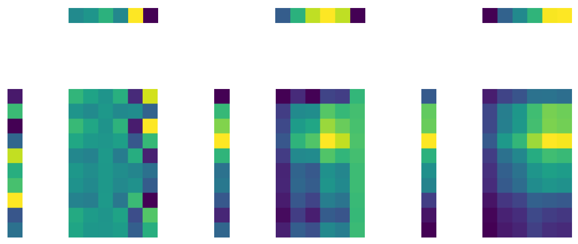
(a) (b) (c)

Figure 15: Reconstruction with 1st 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.

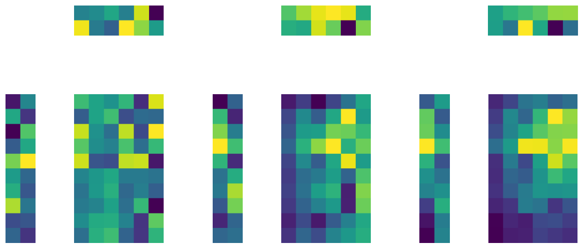
(a) (b) (c)

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 $\boldsymbol{Z_q} = \boldsymbol{Z}$ and the eq.(19) becomes the eq.(11), rather than the approximation from eq.(12).

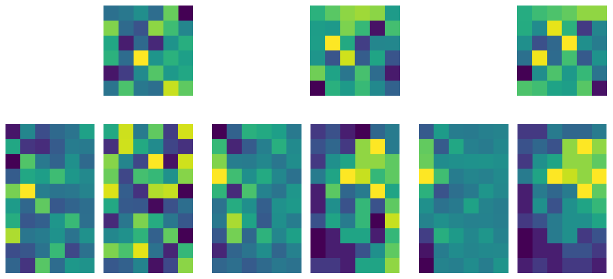
(a) (b) (c)

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 Highlights of statistics

The intuition behind $\frac{1}{1-n}$.

References

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

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

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

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

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

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

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