# TP 4 – AOS1
# PCA
# Corrigé

## 1 Python warm up: PCA by hand

(1) Generate a dataset with the following instruction

```
X = np.random.multivariate_normal([1, 3], [[2, 1], [1, 2]], 100)
```

How many samples are generated? How many features? What is the underlying distribution of samples in $X$?

```
In [1]: import matplotlib.pyplot as plt
        import numpy as np
        import scipy.linalg as linalg
        X = np.random.multivariate_normal([1, 3], [[2, 1], [1, 2]], 100)
        X.shape
```

```
Out [1]: (100, 2)
```

There are 100 samples and 2 features. Samples are drawn according to $\mathcal{N}\left(\begin{pmatrix}1\\3\end{pmatrix}, \begin{pmatrix}2 & 1\\1 & 2\end{pmatrix}\right)$.

(2) Verify the relation that exists between singular values and eigenvalues using a matrix $X$. To use the functions provided by the `scipy` library, use the following command:

```
import scipy.linalg as linalg
```

and look at the functions `linalg.eig`, `linalg.eigh`, `linalg.eigvals`, `linalg.eigvalsh`, `linalg.svd` `linalg.svdvals`

```
In [2]: X = np.random.normal(size=(6, 2))
        print(linalg.eigvalsh(X.T @ X))
```

```
Out [2]: [0.64178977 7.47512647]
```

```
In [3]: print(linalg.svdvals(X)**2)
```

```
Out [3]: [7.47512647 0.64178977]
```

Nonzero eigenvalues of $X^T X$ (or $XX^T$) are squared singular values of $X$.

③ Compute the principal directions and principal components by hand using the unbiased variance–covariance estimator. Verify that they coincide with the ones computed by `scikit-learn`.

```
In [4]: n, p = 100, 15
        X = np.random.normal(size=(n, p))
        X0 = X - X.mean(axis=0)
        V = 1/(n-1) * X0.T @ X0
        vp, U = linalg.eigh(V)
        print(vp)
```

```
Out [4]: [0.39346284 0.46841904 0.54507063 0.56325593 0.72062481
          ↪ 0.78908287
          0.87556462 0.92644902 0.96237418 1.15910771 1.18983685
           ↪ 1.34312761
          1.37139569 1.62307881 1.82253541]
```

```
In [5]: Xpca = X0 @ U
        print(n / (n-1) * Xpca.std(axis=0)**2)
```

```
Out [5]: [0.39346284 0.46841904 0.54507063 0.56325593 0.72062481
          ↪ 0.78908287
          0.87556462 0.92644902 0.96237418 1.15910771 1.18983685
           ↪ 1.34312761
          1.37139569 1.62307881 1.82253541]
```

```
In [6]: from sklearn.decomposition import PCA
        pca = PCA()
        pca.fit(X)
```

```
Out [6]: PCA()
```

```
In [7]: print(pca.explained_variance_)
```

```
Out [7]: [1.82253541 1.62307881 1.37139569 1.34312761 1.18983685
          ↪ 1.15910771
          0.96237418 0.92644902 0.87556462 0.78908287 0.72062481
           ↪ 0.56325593
          0.54507063 0.46841904 0.39346284]
```

We have the same eigenvalues (in reverse order).

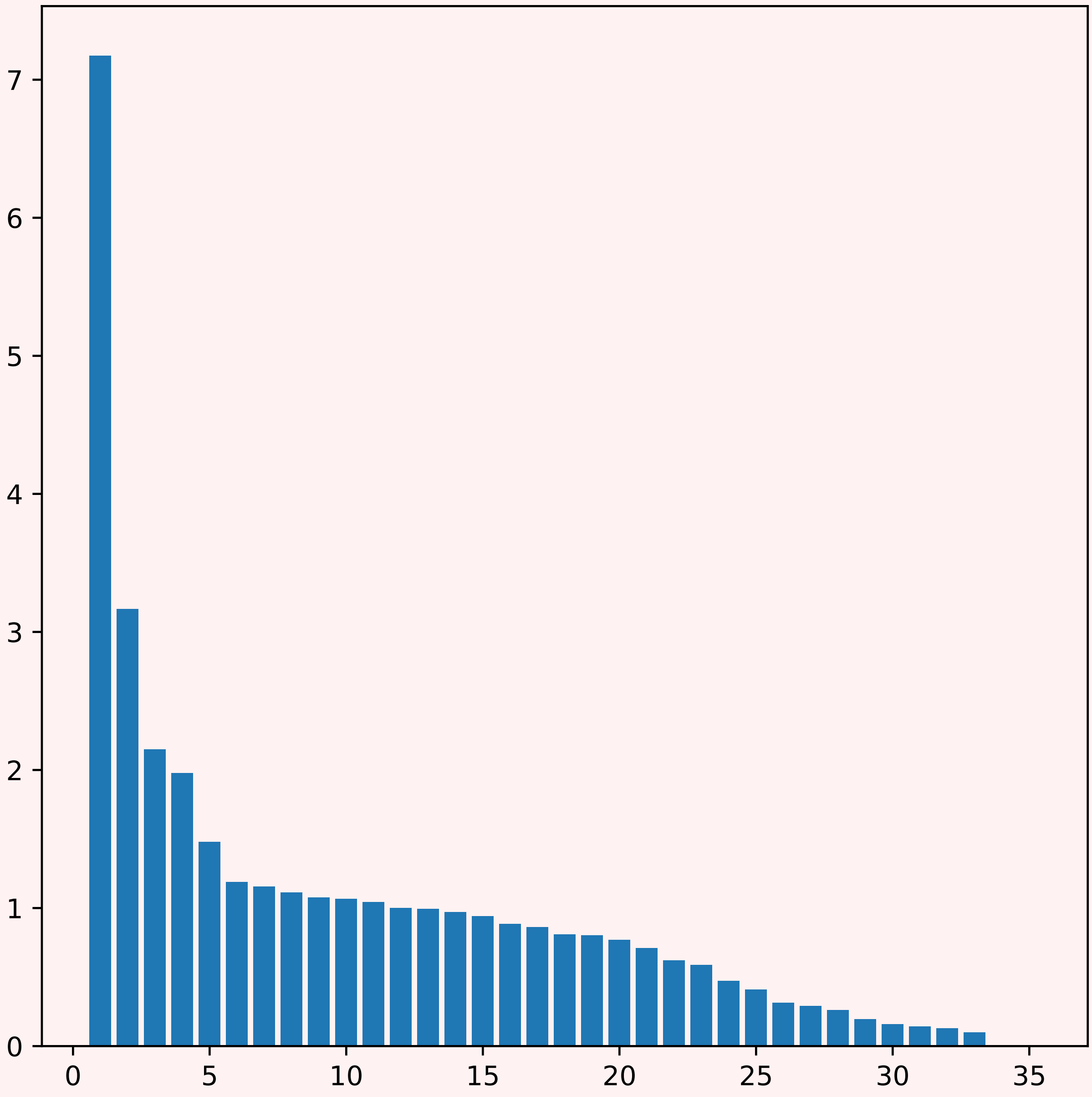
## 2 PCA for dimension reduction

In this section, we use the `house_prices` regression dataset. To load it use

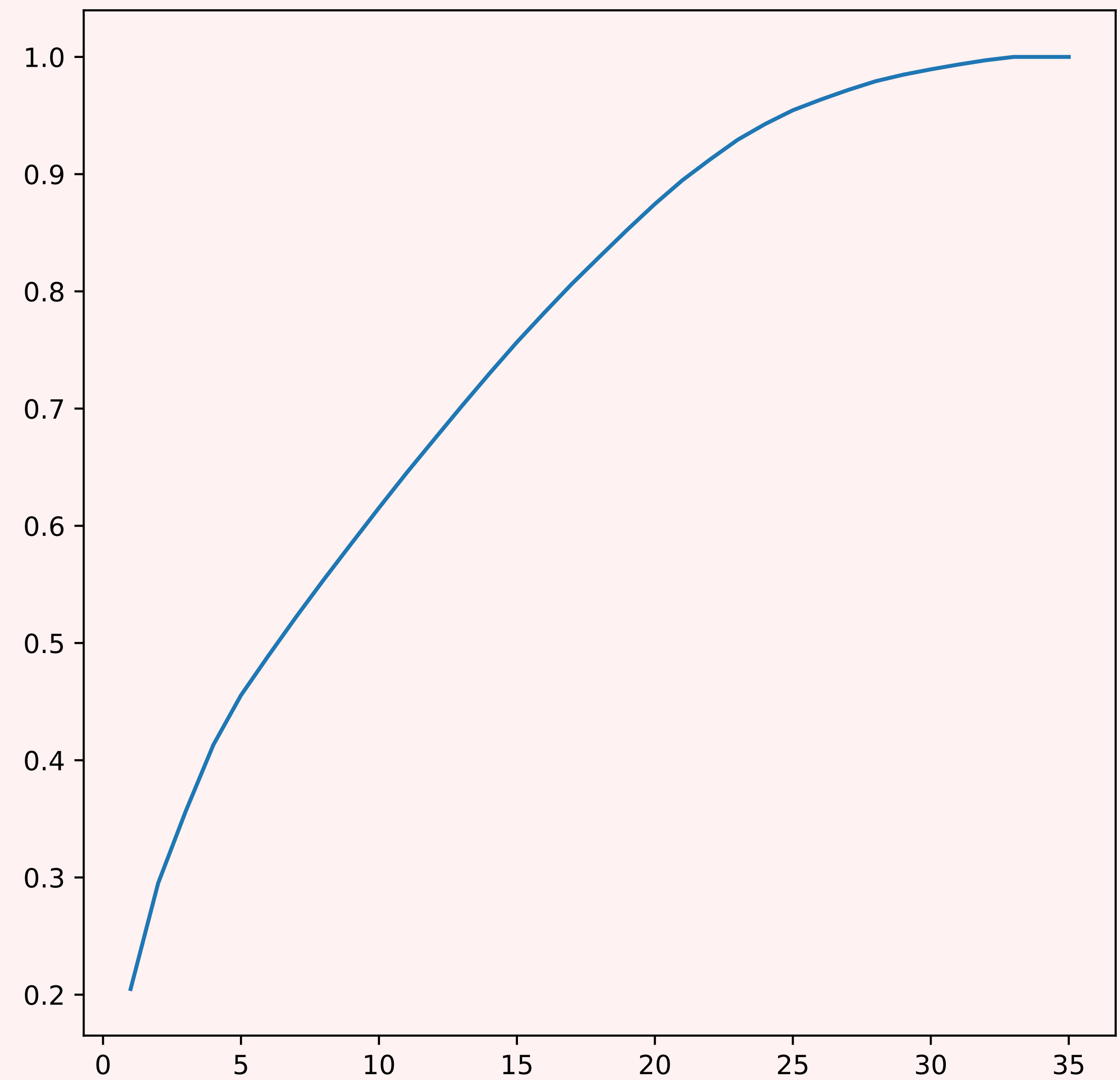
```python
from sklearn.datasets import fetch_openml
housing = fetch_openml(name="house_prices", as_frame=True)
X = housing.frame
X = X.loc[:, ~X.isna().any() & (X.dtypes.astype(str).isin(["float64", "int64"]))]
y = housing.target
```

(4) Perform a PCA on this dataset and study how many number of principal components should be retained from the two empirical methods seen in class.

```
In  [8]: from sklearn.preprocessing import StandardScaler
         from sklearn.datasets import fetch_openml
         housing = fetch_openml(name="house_prices", as_frame=True,
          ↪  parser="auto")
         X = housing.frame
         X = X.loc[:, ~X.isna().any() &
          ↪  (X.dtypes.astype(str).isin(["float64", "int64"]))]
         X = StandardScaler().fit_transform(X)
         y = housing.target
         pca = PCA()
         pca.fit(X)

Out [8]: PCA()

In  [9]: plt.figure()
         plt.bar(range(1, X.shape[1]+1), pca.explained_variance_)

Out [9]: <BarContainer object of 35 artists>

In  [10]: plt.show()
```




```
In [11]: plt.figure()
         plt.plot(range(1, X.shape[1]+1),
          ↪   np.cumsum(pca.explained_variance_ratio_))
         plt.show()
```




The scree plot suggests to choose 5 components. The cumulative sum of explained variance ratio is telling us that to retain 80% of the explained variance, $\approx 17$ components are required.

5 Describe the following code. What is it supposed to be doing? Adapt it to determine the optimal number of principal components for the regression task at hand.

```
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

pca = PCA()
lin = LinearRegression()
```

```python
pca_lin = Pipeline([("scale", StandardScaler()), ("pca", pca), ("lin", lin)])
clf = GridSearchCV(
    estimator=pca_lin,
    cv=10,
    param_grid=dict(pca__n_components=range(1, X.shape[1] + 1)),
)
clf.fit(X, y)
```

This snippet of code is defining a pipeline consisting of a PCA followed by a linear regression on the housing data. The optimal number of components is then computed by cross-validation with 10 folds.

In [12]:

```python
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

pca = PCA()
lin = LinearRegression()
pca_lin = Pipeline([("scale", StandardScaler()), ("pca", pca),
 ↪  ("lin", lin)])
clf = GridSearchCV(
    estimator=pca_lin,
    cv=10,
    param_grid=dict(pca__n_components=range(1, X.shape[1] + 1)),
)
clf.fit(X, y)
```

Out [12]:

```
GridSearchCV(cv=10,
             estimator=Pipeline(steps=[('scale',
              ↪  StandardScaler()),
                                       ('pca', PCA()),
                                       ('lin',
                                        ↪  LinearRegression())]),
             param_grid={'pca__n_components': range(1, 36)})
```

In [13]:

```python
print(clf.best_params_)
```

Out [13]:

```
{'pca__n_components': 33}
```

The best number of principal components using the mean square error and a 10–CV is then $\approx 30$ (it might vary)

# 3 Problem: band reduction in multispectral images

A multispectral image is an image that has several components. For example, a color image has 3 components: red, green and blue and each pixel can be viewed as a vector in $\mathbb{R}^3$. More generally a multispectral image of size $N \times M$ with $P$ spectral bands can be stored as a $N \times M \times P$ array. There are $N \times M$ pixels living in $\mathbb{R}^p$.

When the number of spectral bands $P$ is too large, it is desirable to somehow reduce that number ultimately to 3 for viewing purposes. This process is called band reduction.

Propose a method using the PCA performing a band reduction to 3 bands and use it on the provided multispectral image.

Some multispectral images are available on the internet to test your band reduction algorithm. See for example the following website

- `http://lesun.weebly.com/hyperspectral-data-set.html`

Most of them are available as a Matlab data file (.mat files). It can be loaded with `scipy` with the following function

```
scipy.io.loadmat
```

You will probably have to reshape arrays. It can be done with the `reshape` method. For example, an array of size $6 \times 6 \times 3$ can be "linearized" using reshape

```
X_lin = X.reshape((-1, 3))
```

the $-1$ is automatically inferred from the number of elements in the array. The array is then reshaped into an array of size $36 \times 3$.

It might be handy to be able to rescale the data when it has to belong the some specific range. `scikit-learn` has several rescalers available. For example

```
from sklearn.preprocessing import MinMaxScaler
```

rescales the data between 0 and 1.

`matplotlib` can display images with the function

```
plt.imshow
```

Beware of the type of the array (float or integers)!