Exercises for the course
Machine Learning for Data Science
Winter Semester 2022/23

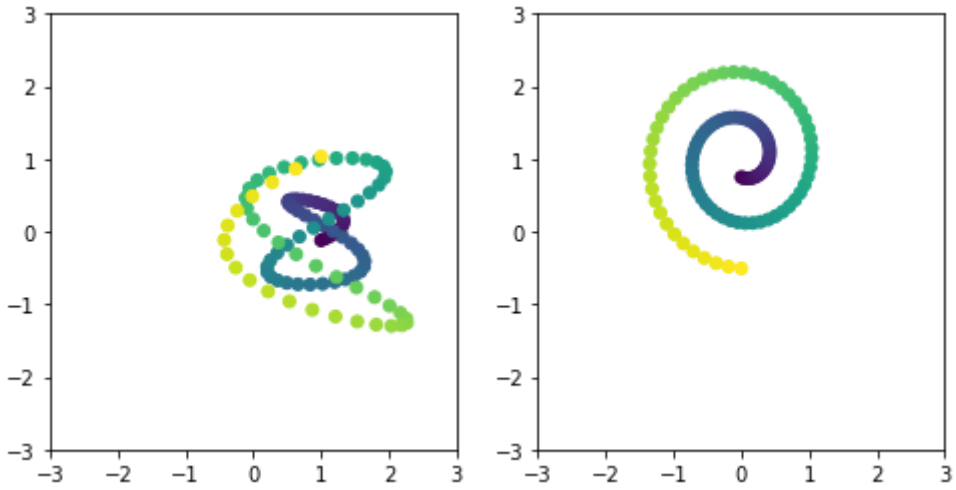
G. Montavon
Institute of Computer Science
Department of Mathematics and Computer Science
Freie Universität Berlin

Exercise Sheet 6 (programming part)

In this exercise, we consider canonical correlation analysis (CCA) on two simple problems, one in low dimensions and one in high dimensions. The goal is to implement the original CCA procedure, and the CCA variant for high-dimensional data, in order to handle both cases. The first dataset consists of two trajectories in two dimensions. The dataset is extracted and plotted below. The first data points are shown in dark blue, and the last ones are shown in yellow.

In [4]:

```
import numpy as np
import matplotlib
%matplotlib inline
from matplotlib import pyplot as plt
import utils

X,Y   = utils.getdata()
p1,p2 = utils.plotdata(X,Y)
```



For these two trajetories, that can be understood as two different modalities of the same data, we would like determine under which projections they appear maximally correlated.

Exercise 3: Implementing CCA (25 P)

As stated in the lecture, the CCA problem in its original form consists of maximizing the cross-correlation objective:

$$J(w_x, w_y) = w_x^\top C_{xy} w_y$$

subject to autocorrelation constraints $w_x^\top C_{xx} w_x = 1$ and $w_y^\top C_{yy} w_y = 1$. Using the method of Lagrange multipliers, this optimization problem can be reduced to finding the first eigenvector of the generalized eigenvalue problem:

$$\begin{bmatrix} 0 & C_{xy} \\ C_{yx} & 0 \end{bmatrix} \begin{bmatrix} w_x \\ w_y \end{bmatrix} = \lambda \begin{bmatrix} C_{xx} & 0 \\ 0 & C_{yy} \end{bmatrix} \begin{bmatrix} w_x \\ w_y \end{bmatrix}$$

Your first task is to write a function that solves this generalized eigenvalue problem. The function you need to implement receives two matrices `X` and `Y` of size `d1` × `N` and `d2` × `N` respectively. It should return two vectors of size `d1` and `d2` corresponding to the projections associated to the modalities `X` and `Y`. *(Hint: Note that the data matrices `X` and `Y` have not been centered yet.)*

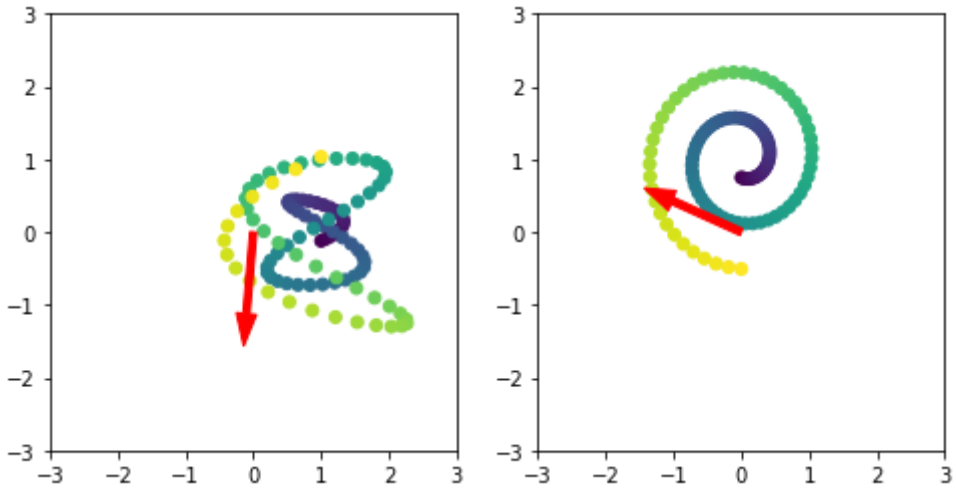
In [5]:

```
import numpy
import scipy.linalg

def CCA(X, Y):

    n = X.shape[1]
    dx = X.shape[0]
    dy = Y.shape[0]

    X = X - X.mean(axis=1, keepdims=True)
    Y = Y - Y.mean(axis=1, keepdims=True)

    C_xx = (X @ X.T) / n
    C_xy = (X @ Y.T) / n
    C_yx = C_xy.T
    C_yy = (Y @ Y.T) / n

    zeros_x = np.zeros((dx, dx))
    zeros_y = np.zeros((dy, dy))

    A = np.block([[zeros_y, C_xy],
                  [C_yx, zeros_x]])
    B = np.block([[C_xx, zeros_x],
                  [zeros_y, C_yy]])

    w, v = scipy.linalg.eigh(A, B)

    wx = v[:dx, -1]
    wy = v[dy:(dx+dy), -1]

    return wx, wy
```

The function can now be called with our dataset. The learned projection vectors w_x and w_y are plotted as red arrows.

In [6]:

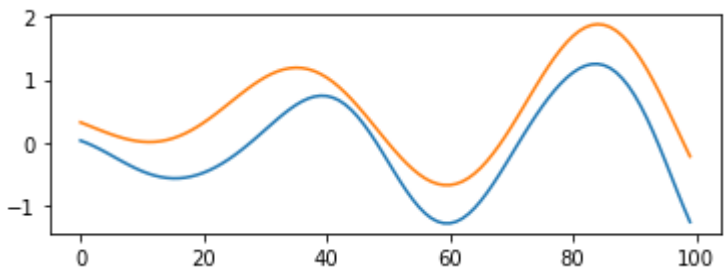
```
wx,wy = CCA(X,Y)

p1,p2 = utils.plotdata(X,Y)
p1.arrow(0,0,1*wx[0],1*wx[1],color='red',width=0.1)
p2.arrow(0,0,1*wy[0],1*wy[1],color='red',width=0.1)
plt.show()
```



In each modality, the arrow points in a specific direction (note that the optimal CCA directions are defined up to a sign flip of both w_x and w_y). Furthermore, we can verify CCA has learned a meaningful solution by projecting the data on it.

In [7]:

```
plt.figure(figsize=(6,2))
plt.plot(wx.dot(X))
plt.plot(wy.dot(Y))
plt.show()
```



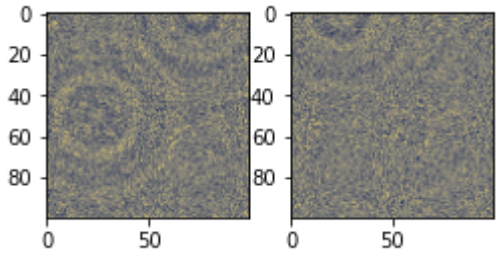
Clearly, the data is correlated in the projected space.

Exercise 4: Implementing CCA for High Dimensions (25 P)

In the second part of the exercise, we consider the case where the data is high dimensional (with $d \gg N$). Such high-dimensionality occurs for example, when input data are images. We consider the scenario where sources emit spatially, and two (noisy) receivers measure the spatial field at different locations. We would like to identify the signal that is common to the two measured locations, e.g. a given source emitting at a given frequency. We first load the data and show one example.

In [8]:

```
X,Y = utils.getHDdata()

utils.plotHDdata(X[:,0],Y[:,0])
plt.show()
```



Several sources can be perceived, however, there is a significant level of noise. Here again, we will use CCA to find subspaces where the two modalities are maximally correlated. In this example, because there are many more dimensions than there are data points, it is more advantageous to solve the alternate formulation of CCA in terms of the weightings α_x and α_y. Your task is to implement the latter CCA solver. Like the original CCA solver, it receives two data matrices of size $d1 \times N$ and $d2 \times N$ respectively as input, and should return the associate CCA directions (two vectors of respective sizes $d1$ and $d2$).

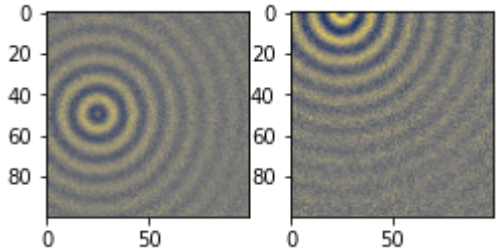
In [12]:

```
import copy

def CCA_HD(X,Y):

    n = X.shape[1]

    X_old = copy.deepcopy(X)
    Y_old = copy.deepcopy(Y)

    X = X - X.mean(axis=1, keepdims=True)
    Y = Y - Y.mean(axis=1, keepdims=True)

    Q_xx = (X.T @ X @ X.T @ X) / n # NxN
    Q_xy = (X.T @ X @ Y.T @ Y) / n # NxN
    Q_yx = Q_xy.T
    Q_yy = (Y.T @ Y @ Y.T @ Y) / n

    zeros_x = np.zeros((n, n))
    zeros_y = np.zeros((n, n))

    A = np.block([[zeros_y, Q_xy],
                  [Q_yx, zeros_x]])
    B = np.block([[Q_xx, zeros_x],
                  [zeros_y, Q_yy]])

    EPS = 0.001
    B = B + EPS * np.eye(2 * n)

    w, v = scipy.linalg.eigh(A, B)

    alpha_x = v[:n, -1].reshape((-1,1))
    alpha_y = v[n:, -1].reshape((-1,1))

    wx = (X_old @ alpha_x)[:,0]
    wy = (Y_old @ alpha_y)[:,0]

    return wx, wy
```

We now call the function we have implemented with a training sequence of 100 pairs of images. Because the returned solution is of same dimensions as the inputs, it can be rendered in a similar fashion.

In [13]:

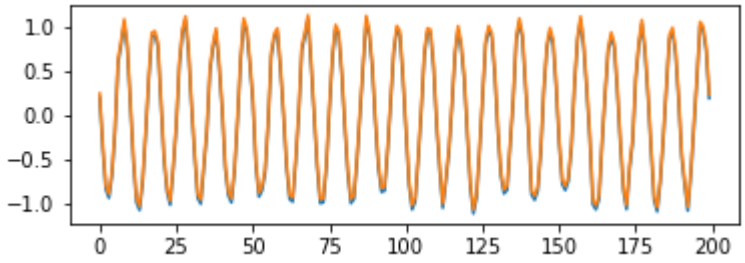
```
wx,wy = CCA_HD(X,Y)

utils.plotHDdata(wx,wy)
plt.show()
```



Here, we can clearly see a common factor that has been extracted between the two fields, specifically a point source emitting at a particular frequency. The sequence of image pairs can now be projected on these two filters:

In [14]:

```
plt.figure(figsize=(6,2))
plt.plot(wx.dot(X))
plt.plot(wy.dot(Y))
plt.show()
```



In [ ]:

```

```