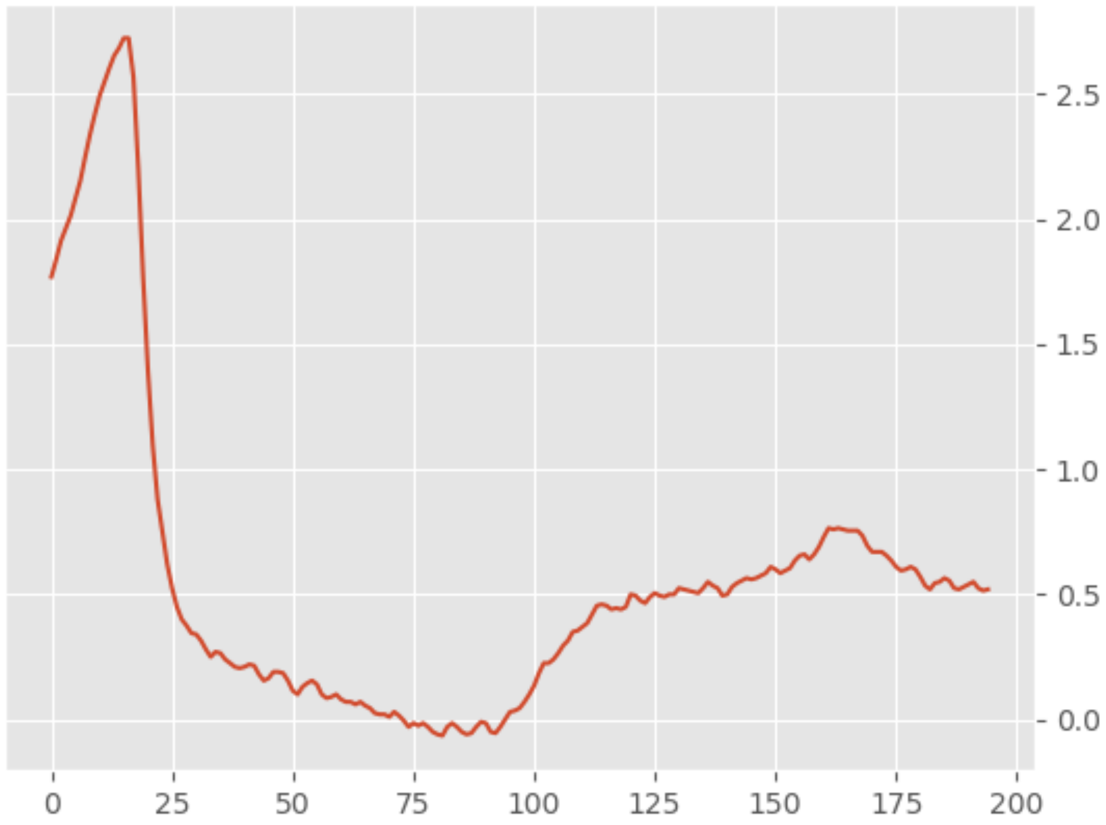
NUS-ISS
Intelligent Sensing and Sense Making

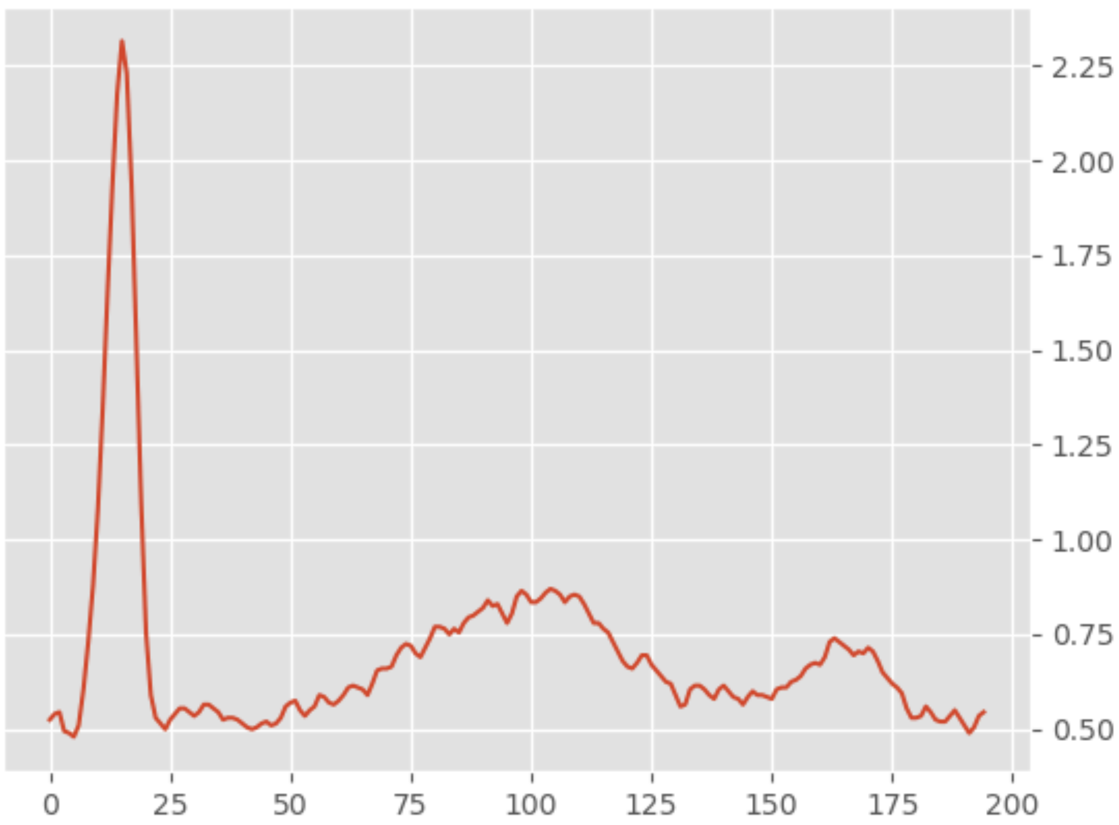
Dynamic Time Warping

by Dr. Tan Jen Hong

Problem

- How similar are these two signals?
- In which manners are they similar?



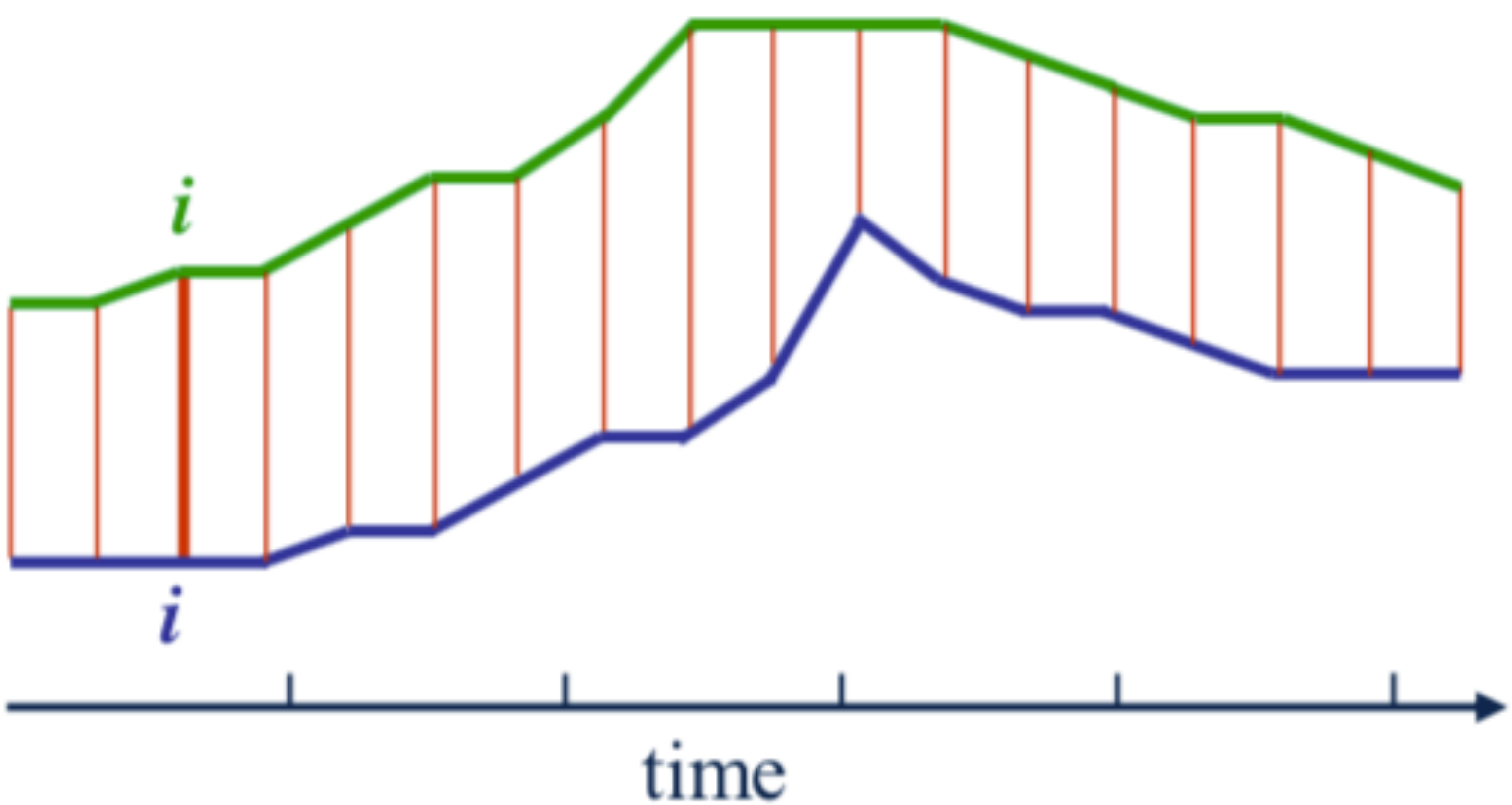


Can we try ...

Euclidean, Manhattan .....?

- We can measure the similarity of two signals by calculating the distance between the i-th point on one signal and the i-th point on another signal

- Simple concept, but could not capture the similarity in shape



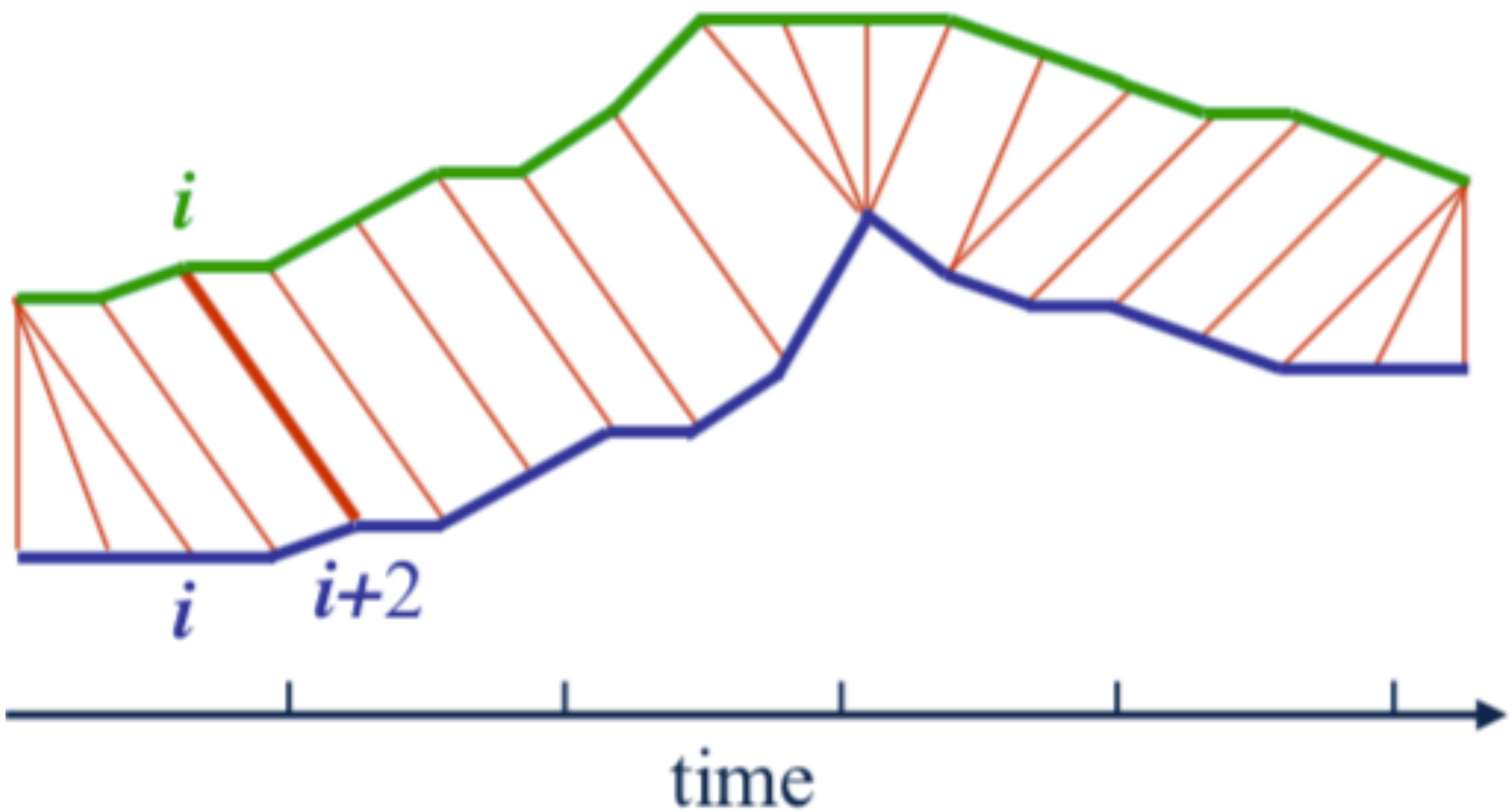
Source: "Dynamic Time Warping Algorithm", by Elena Tsiporkova

How about ...

non-linear alignment .....?

- Elastic alignment between points of two signals produces a better, more intuitive similarity measure

- Allow similar shapes to match even if they are out of phase



Source: "Dynamic Time Warping Algorithm", by Elena Tsiporkova

Distance

Another term to say 'similiarity'

- Consider two distinct signals

$$\mathbf{x} = [x_1, x_2, \ldots, x_i, \ldots x_m]$$
$$\mathbf{y} = [y_1, y_2, \ldots, y_j, \ldots y_n]$$

- The distance between the two signals is defined as

$$d_s(\mathbf{x}, \mathbf{y})$$

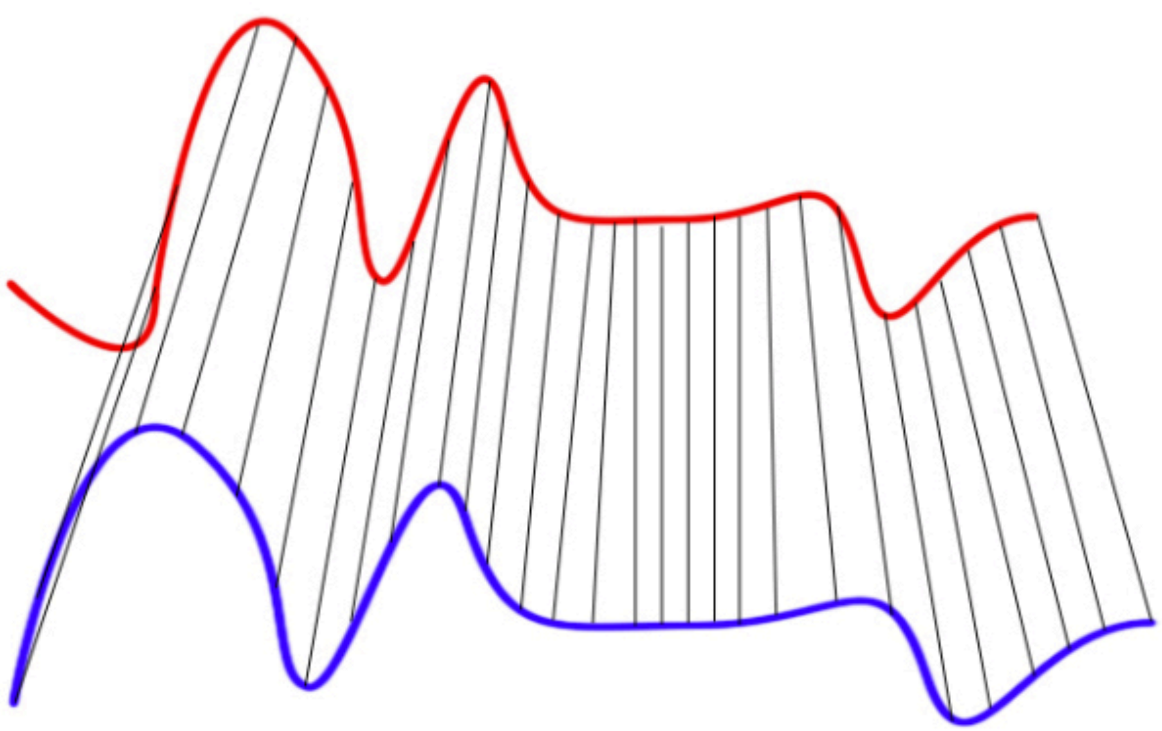
- Euclidean distance between two signals:

$$d_s(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_i d_s(x_i, y_i)^2}$$

- Problem: two signals must be of same length!

Dynamic Time Warping
DTW

- An algorithm to measure similarity between two temporal sequences (signal), which may vary in speed
- DTW calculates an optimal match between two given sequences
- Sequences are warped along time dimension to determine similarity independent of variations in time
- DTW produces warping path, which enables alignment between two signals

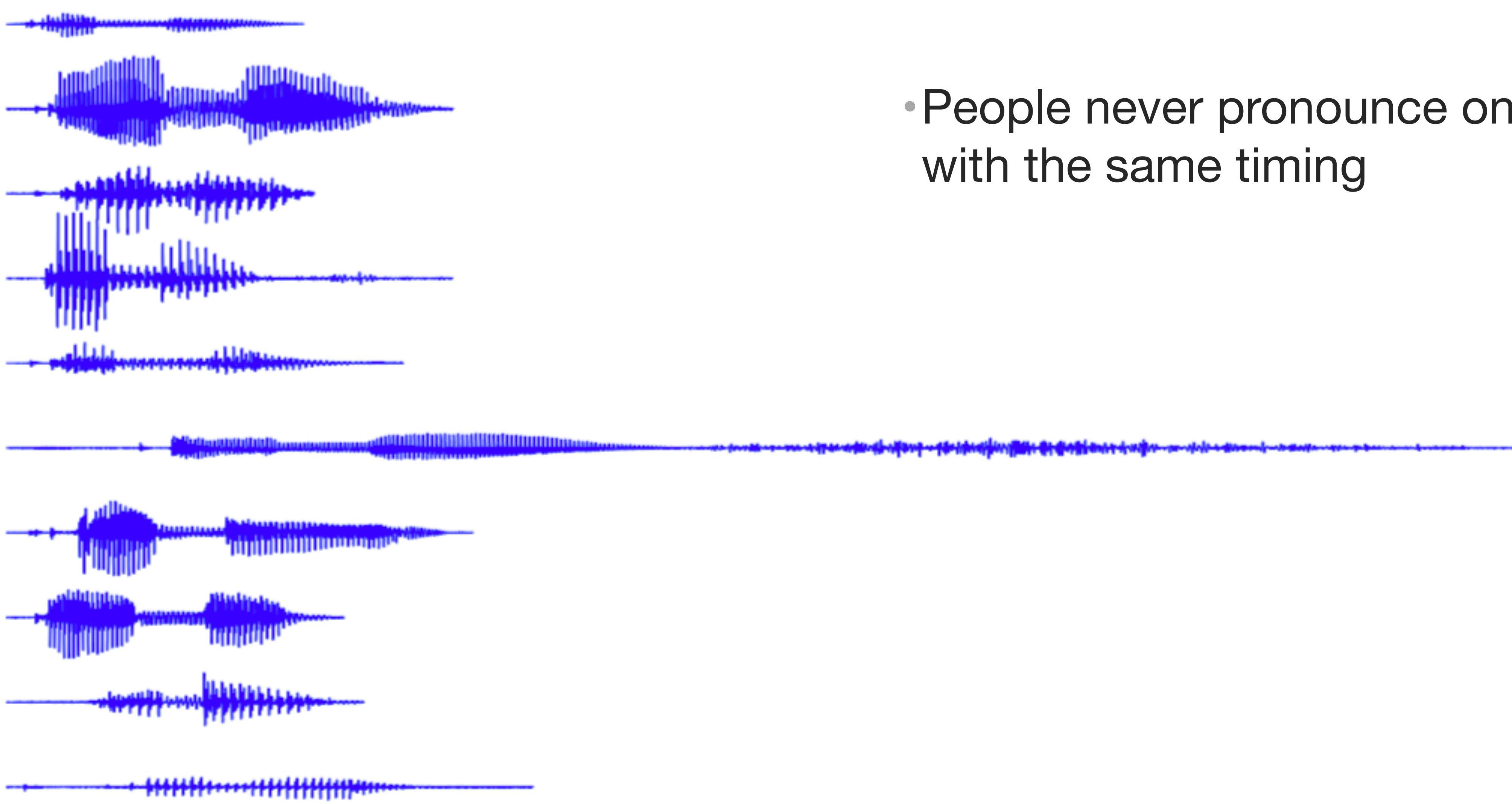


Source: https://th.wikipedia.org/wiki/Dynamic_time_warping#/media/File:Euclidean_vs_DTW.jpg

Dynamic Time Warping

Usage

"timing"



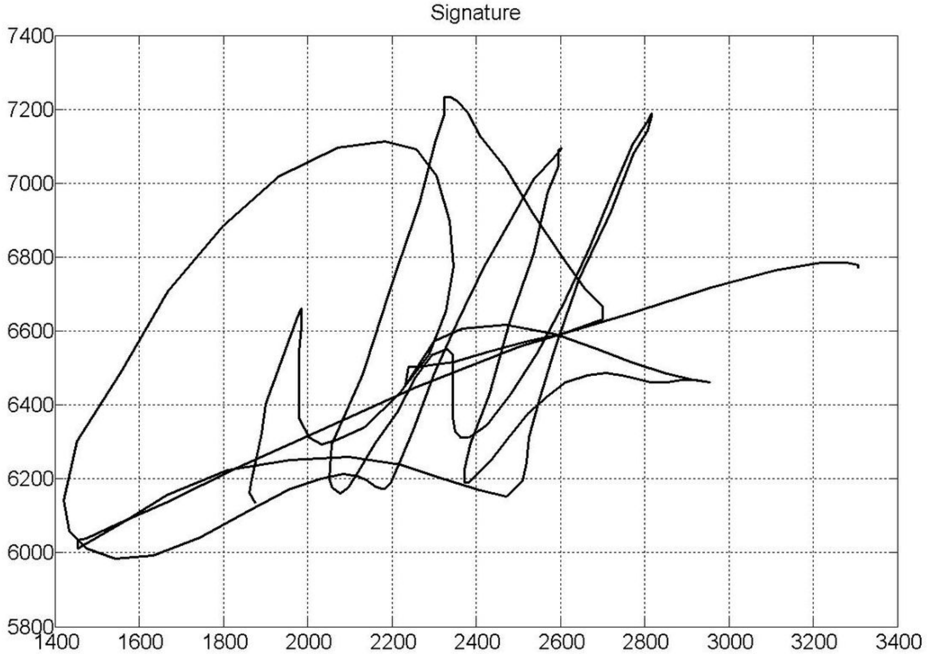
Source: "Speech Recognition - Intro and DTW", by Jan Černocky

- Commonly used in speech recognition
- Individual never pronounces one word twice in exact way
- People never pronounce one word with the same timing

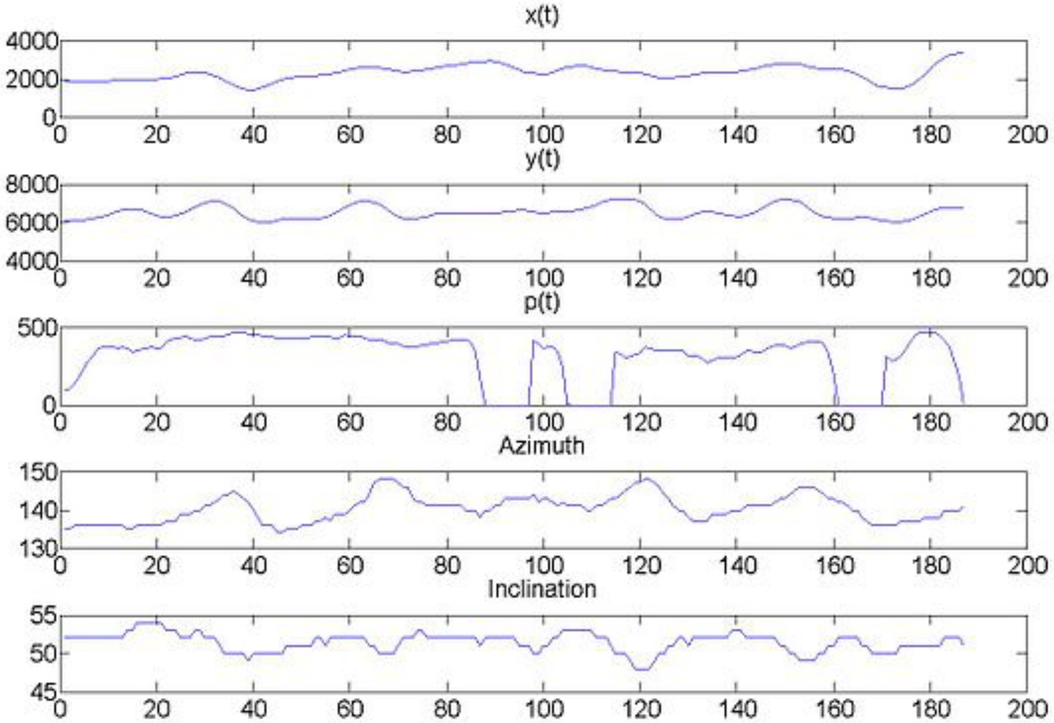
Dynamic Time Warping

Dynamic signature recognition

- Users sign their signature on digital tablet
- Dynamic information captured:
  - x position
  - y position
  - pressure
  - azimuth
  - inclination
- Use DTW to check / match signature



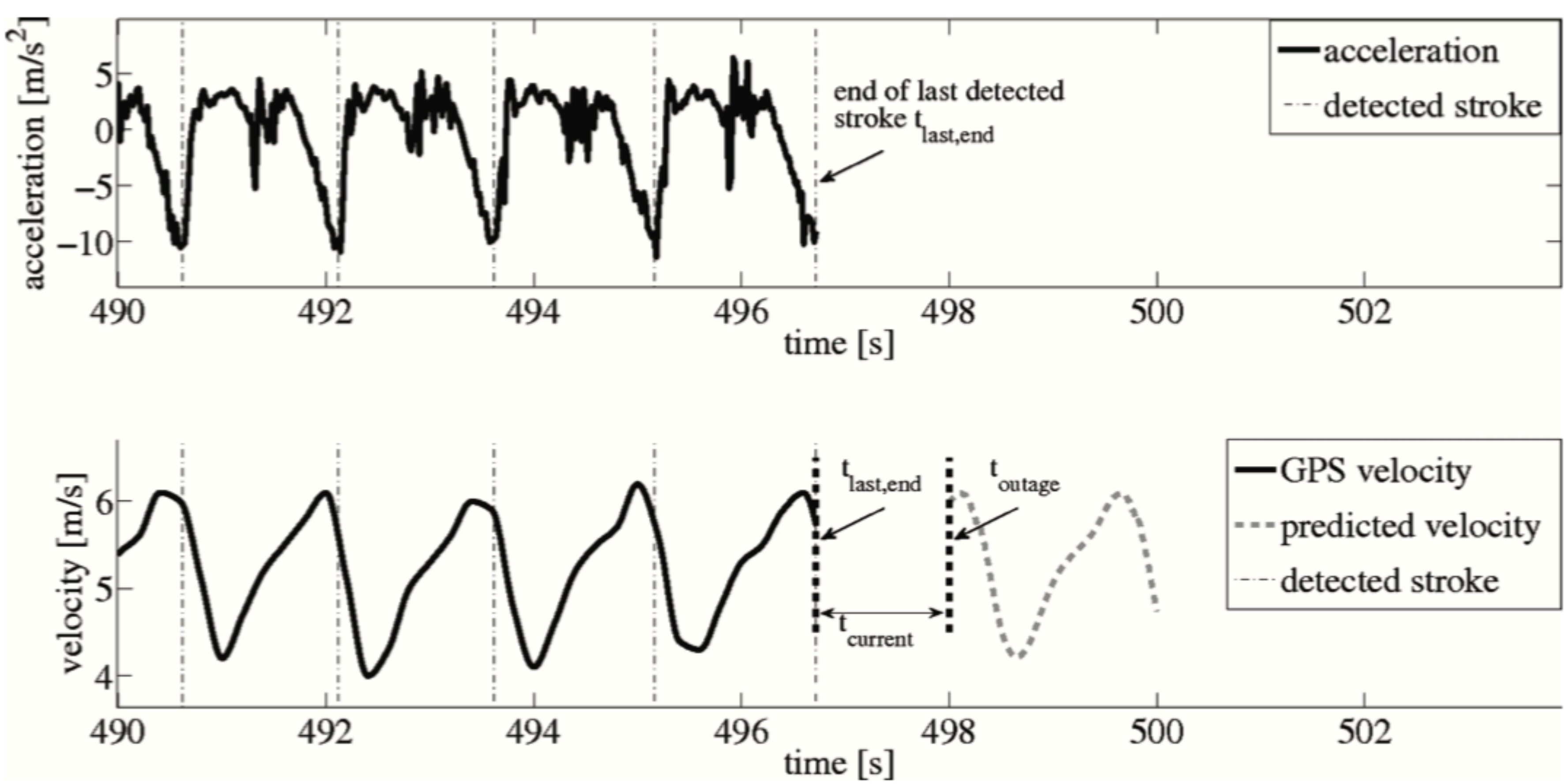
Source: "Speech Recognition - Intro and DTW", by Jan Černocky



Dynamic Time Warping

Stroke detection (rowing)

- Use DTW to detect stroke (used in rowing competitions)
- With strokes detected, predict boat's movement and position when sensor transmission lost

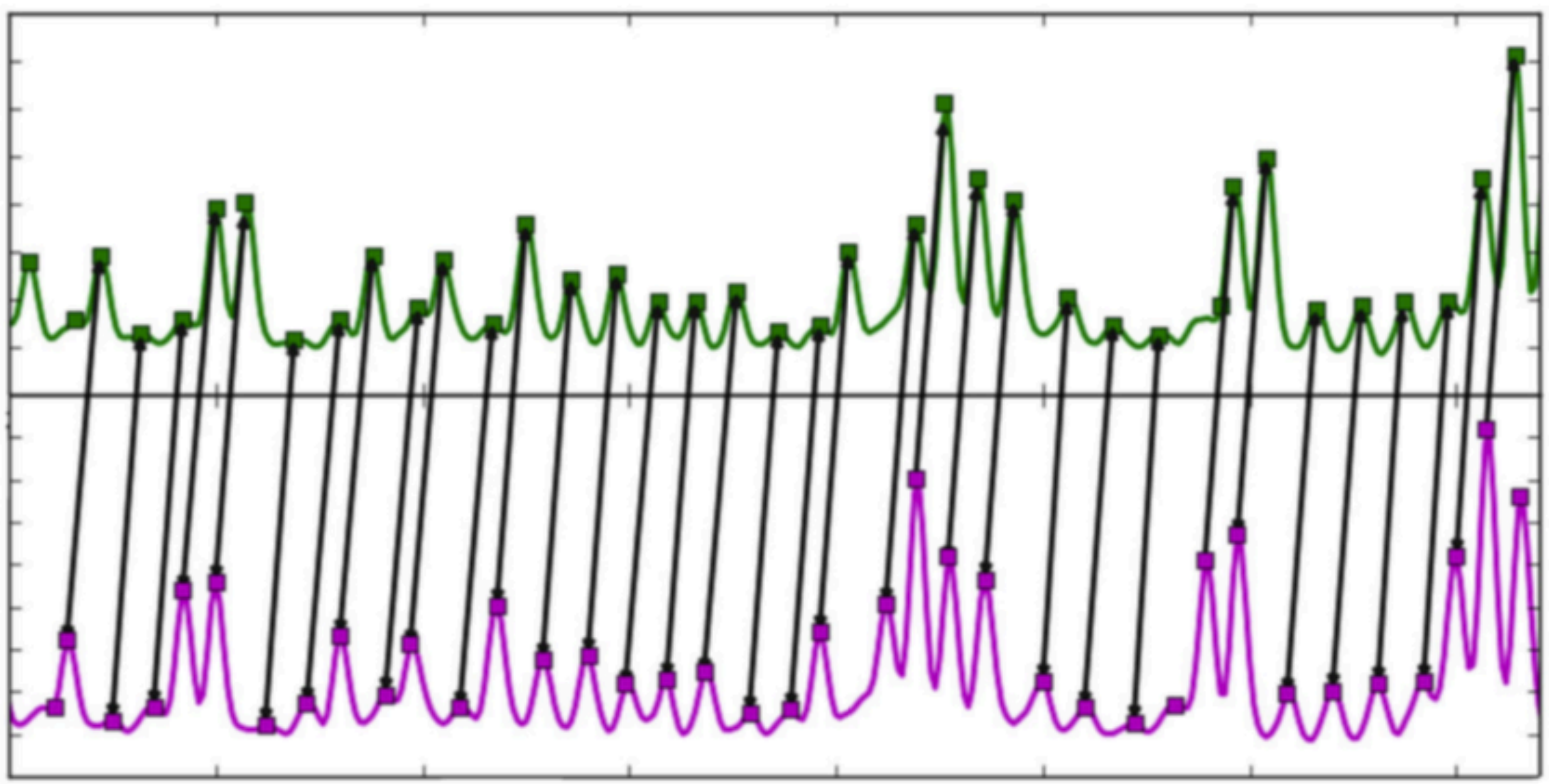


Source: "Movement prediction in rowing using a dynamic time warping base stroke detection", by Groh et al.

Dynamic Time Warping

Peak alignment in DNA sequencing

- Use DTW to align peaks in electropherogram (a plot generated by DNA sequencer)
- Accurate alignment gives better interpretation (e.g. better RNA secondary structure prediction)



with reagent

without reagent

Source: "A peak alignment algorithm with novel improvements in application to electropherogram analysis", by Karabiber

Dynamic Time Warping

Overview of algorithm

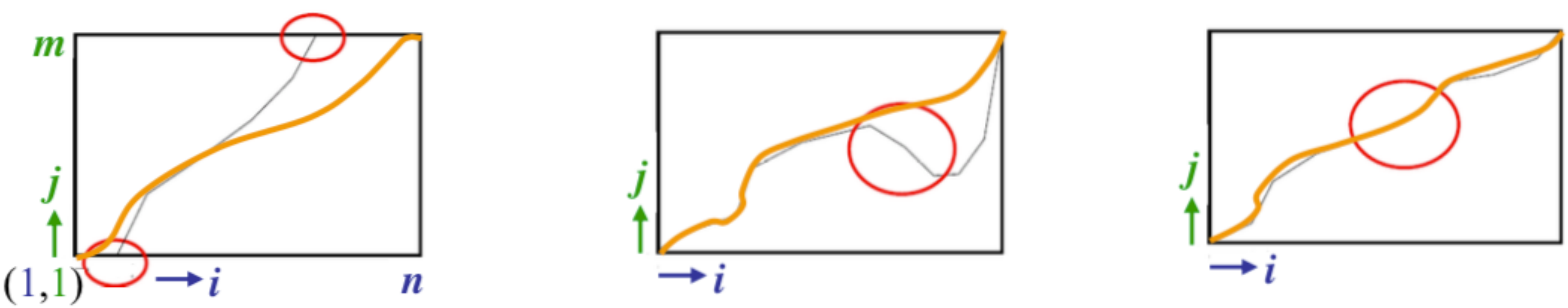
- Start by constructing *n* x *m* matrix *D*, in which

$$D_{i,j} = d_s(x_i, y_j)$$

- where

$$d_s(x_i, y_j) = (x_i - y_j)^2$$

- Create a warping path *w* that maps points between x and y, the path w must satisfy the following:
    - Boundary conditions
    - Monotonicity
    - Continuity



Source: "Dynamic time warping algorithm", by Elena Tsiporkova

Dynamic Time Warping

Overview of algorithm

- DTW algorithm consists of mainly 3 parts:
  1. Compute distance matrix
  2. Compute accumulated cost matrix
  3. Search the optimal path

- To start the code, import the necessary libraries, and setup a bit

```
> import numpy as np
> import matplotlib.pyplot as plt
> import pandas as pd

> plt.style.use('ggplot')
> plt.rcParams['ytick.right']     = True
> plt.rcParams['ytick.labelright']= True
> plt.rcParams['ytick.left']      = False
> plt.rcParams['ytick.labelleft'] = False
```

Dynamic Time Warping

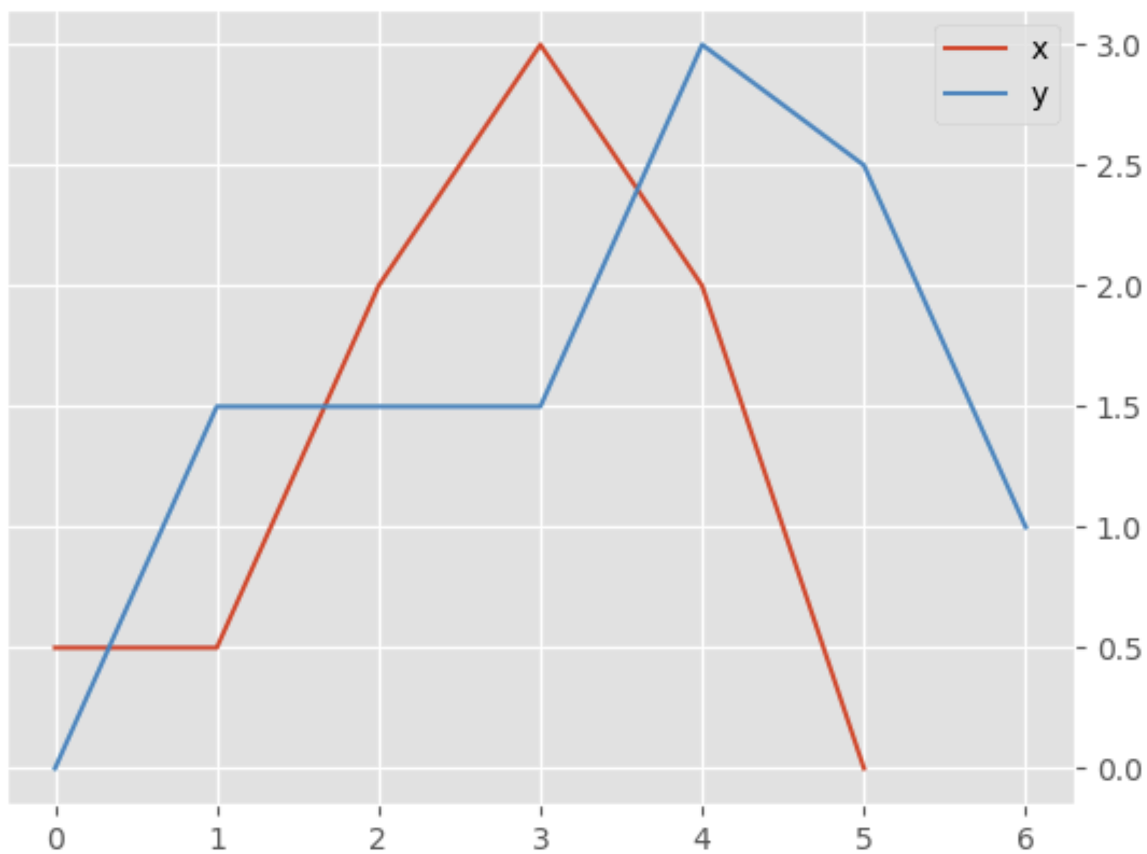
1. Compute distance matrix

- Define two simple short signals:

```
> x = np.array([0.5,0.5,2.0,3.0,2.0,0.0])
> y = np.array([0.0,1.5,1.5,1.5,3.0,2.5,1.0])
```

- Plot the two signals

```
> plt.figure()
> plt.plot(x,
           color="C0",
           label='x')
> plt.plot(y,
           color="C1",
           label='y')
> plt.legend()
```



Dynamic Time Warping

1. Compute distance matrix

- Compute the distance matrix is straightforward, since the matrix is defined as

$$D_{i,j} = d_s(x_i, y_j)$$

- and

$$d_s(x_i, y_j) = (x_i - y_j)^2$$

- The corresponding code

```
> dists = np.zeros((len(y),len(x)))

> for i in range(len(y)):
      for j in range(len(x)):
          dists[i,j]  = (y[i]-x[j])**2
```

dists

```
[[0.25, 0.25, 4.  , 9.  , 4.  , 0.  ],
 [1.  , 1.  , 0.25, 2.25, 0.25, 2.25],
 [1.  , 1.  , 0.25, 2.25, 0.25, 2.25],
 [1.  , 1.  , 0.25, 2.25, 0.25, 2.25],
 [6.25, 6.25, 1.  , 0.  , 1.  , 9.  ],
 [4.  , 4.  , 0.25, 0.25, 0.25, 6.25],
 [0.25, 0.25, 1.  , 4.  , 1.  , 1.  ]]
```

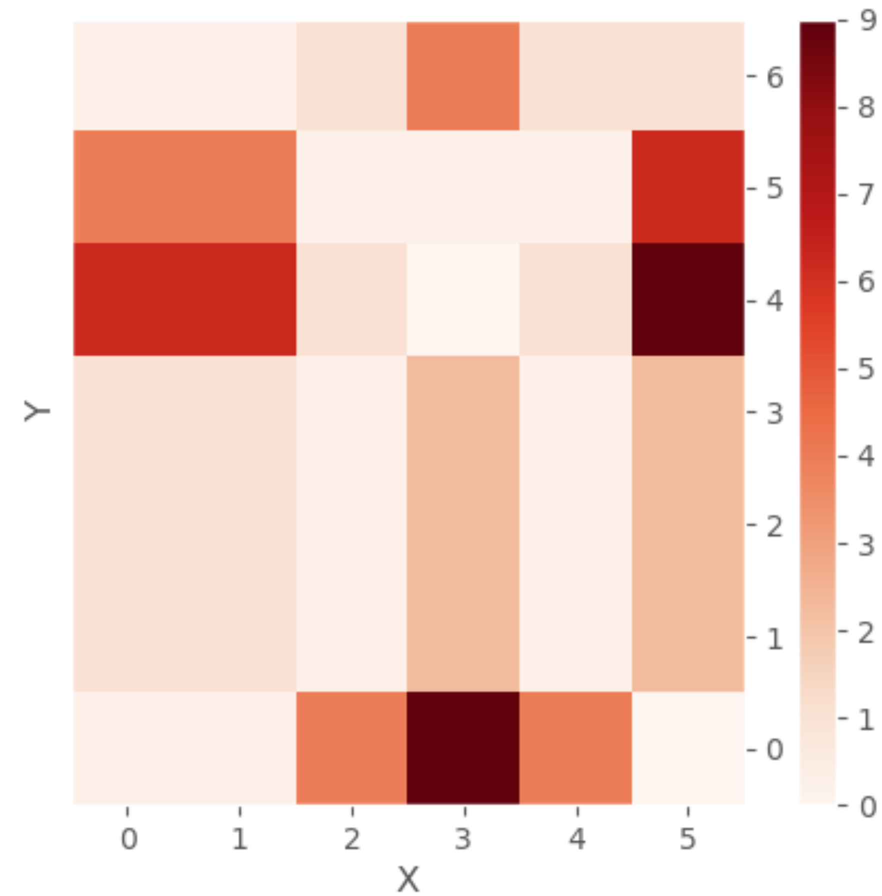
Dynamic Time Warping

1. Compute distance matrix

- Create a function to do a plot on the distance matrix

```
> def pltDistances(dists,xlab="X",ylab="Y",clrmap="viridis"):
      imgplt  = plt.figure()
      plt.imshow(dists,
                 interpolation='nearest',
                 cmap=clrmap)

      plt.gca().invert_yaxis()
      plt.xlabel(xlab)
      plt.ylabel(ylab)
      plt.grid()
      plt.colorbar()

      return imgplt
```

```
> pltDistances(dists,clrmap='Reds')
```

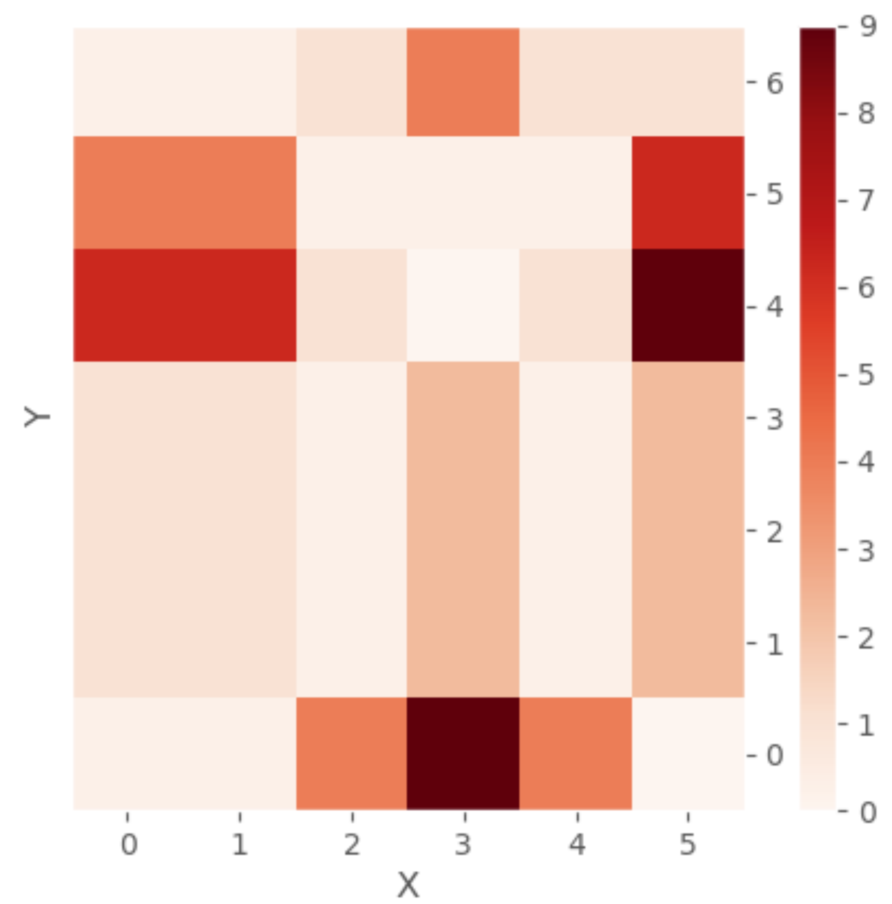


Dynamic Time Warping

1. Compute distance matrix

- Do take note, in the plot, the y axis is inverted
- Thus, first row of the matrix corresponds to the last row in the figure

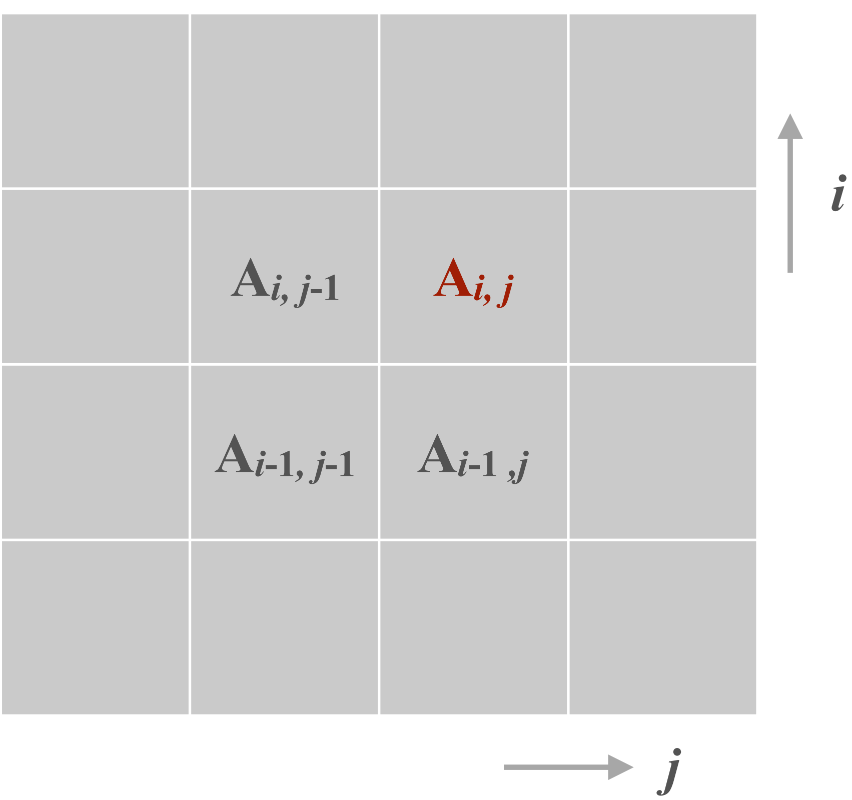
```
[[0.25, 0.25, 4.  , 9.  , 4.  , 0.  ],
 [1.  , 1.  , 0.25, 2.25, 0.25, 2.25],
 [1.  , 1.  , 0.25, 2.25, 0.25, 2.25],
 [1.  , 1.  , 0.25, 2.25, 0.25, 2.25],
 [6.25, 6.25, 1.  , 0.  , 1.  , 9.  ],
 [4.  , 4.  , 0.25, 0.25, 0.25, 6.25],
 [0.25, 0.25, 1.  , 4.  , 1.  , 1.  ]]
```



Dynamic Time Warping

2. Compute accumulated cost matrix

$\mathbf{A}_{i,j}$ equals to $\mathbf{D}_{i,j}$ plus either $\mathbf{A}_{i-1,j-1}$, $\mathbf{A}_{i,j-1}$ or $\mathbf{A}_{i-1,j}$, whichever has the lowest value



- The accumulated cost matrix is defined

$$A_{i,j} = D_{i,j} + \min\left(A_{i-1,j}, A_{i,j-1}, A_{i-1,j-1}\right)$$

- When i and j equals to 0

$$A_{0,0} = D_{0,0}$$

- When i equals to 0 (first row)

$$A_{0,j} = D_{0,j} + A_{0,j-1}$$

- When j equals to 0 (first column)

$$A_{i,0} = D_{i,0} + A_{i-1,0}$$

Dynamic Time Warping

2. Compute accumulated cost matrix

- Name the accumulated cost matrix as `acuCost`. It has the same shape as `dists`.

$$A_{0,0} = D_{0,0}$$

```
> acuCost      = np.zeros(dists.shape)
> acuCost[0,0]= dists[0,0]
```

$$A_{0,j} = D_{0,j} + A_{0,j-1}$$

```
> for j in range(1,dists.shape[1]):
      acuCost[0,j]    = dists[0,j]+acuCost[0,j-1]
```

$$A_{i,0} = D_{i,0} + A_{i-1,0}$$

```
> for i in range(1,dists.shape[0]):
      acuCost[i,0]    = dists[i,0]+acuCost[i-1,0]
```

$$A_{i,j} = D_{i,j} + \min(A_{i-1,j}, A_{i,j-1}, A_{i-1,j-1})$$

```
> for i in range(1,dists.shape[0]):
      for j in range(1,dists.shape[1]):
          acuCost[i,j]    = min(acuCost[i-1,j-1],
                                acuCost[i-1,j],
                                acuCost[i,j-1])+dists[i,j]
```

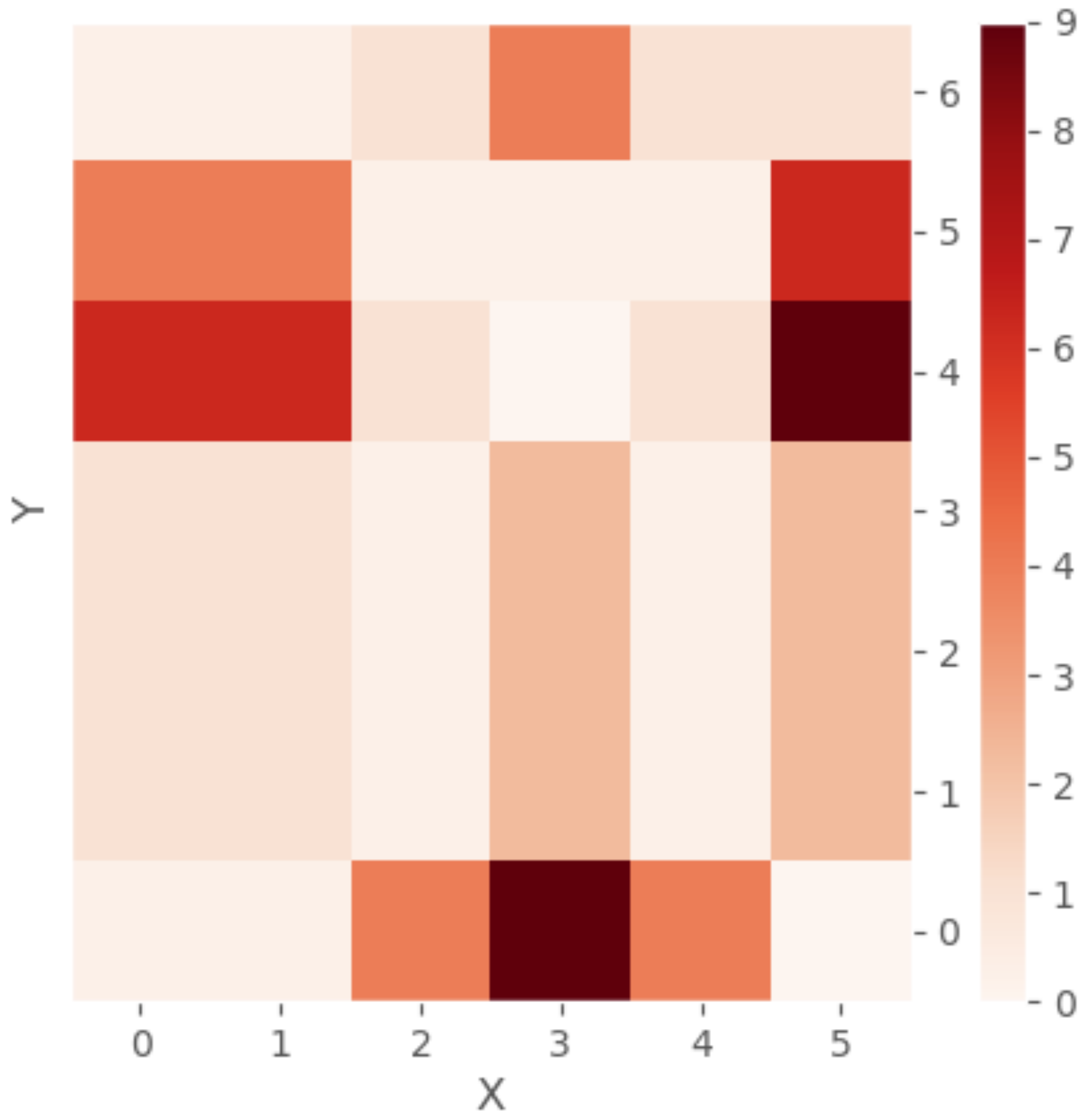
```
> pltDistances(acuCost,clrmap='Reds')
```

Dynamic Time Warping

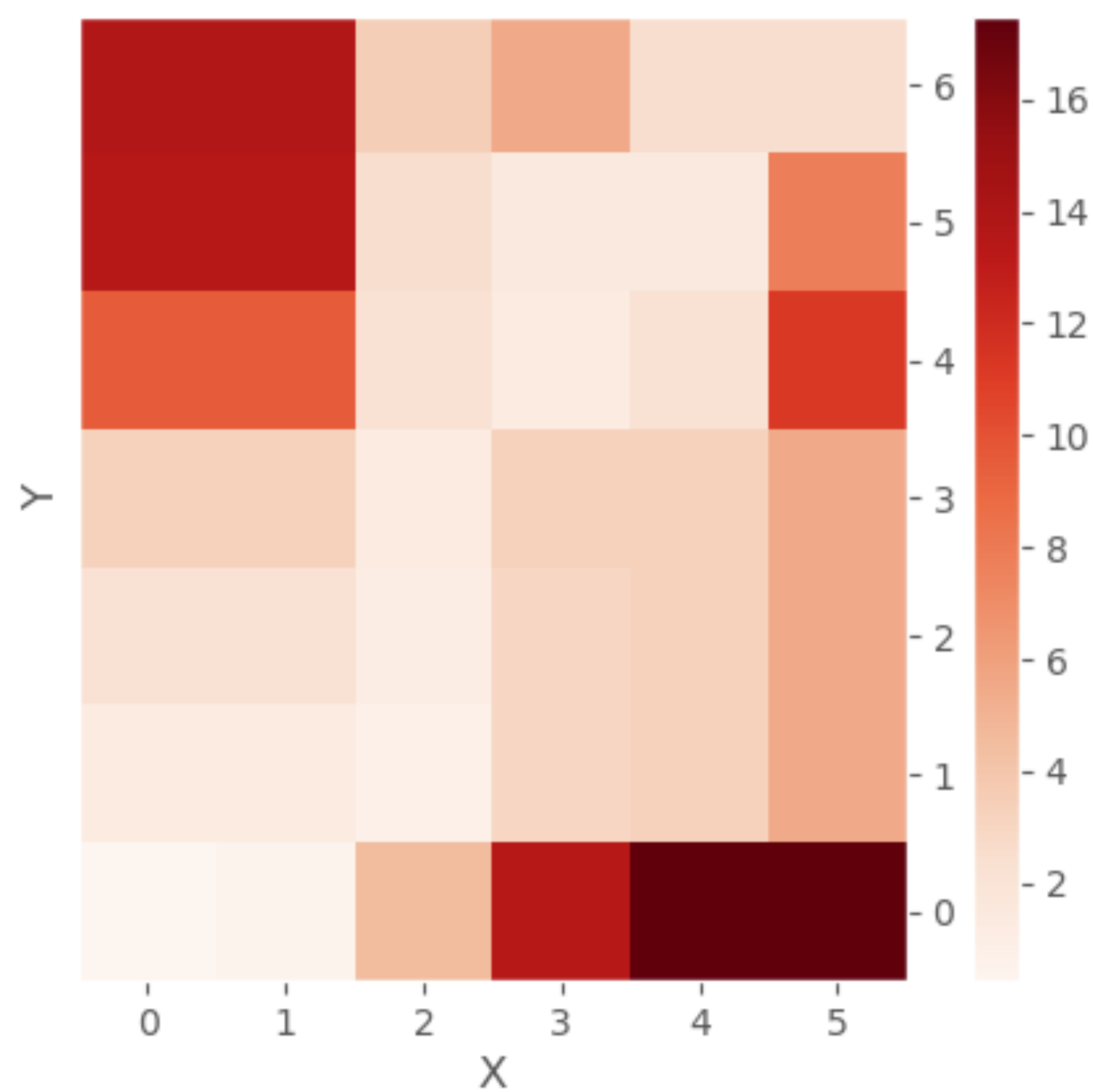
2. Compute accumulated cost matrix

- Can you see the warping / optimal path?

`dists`



`acuCost`

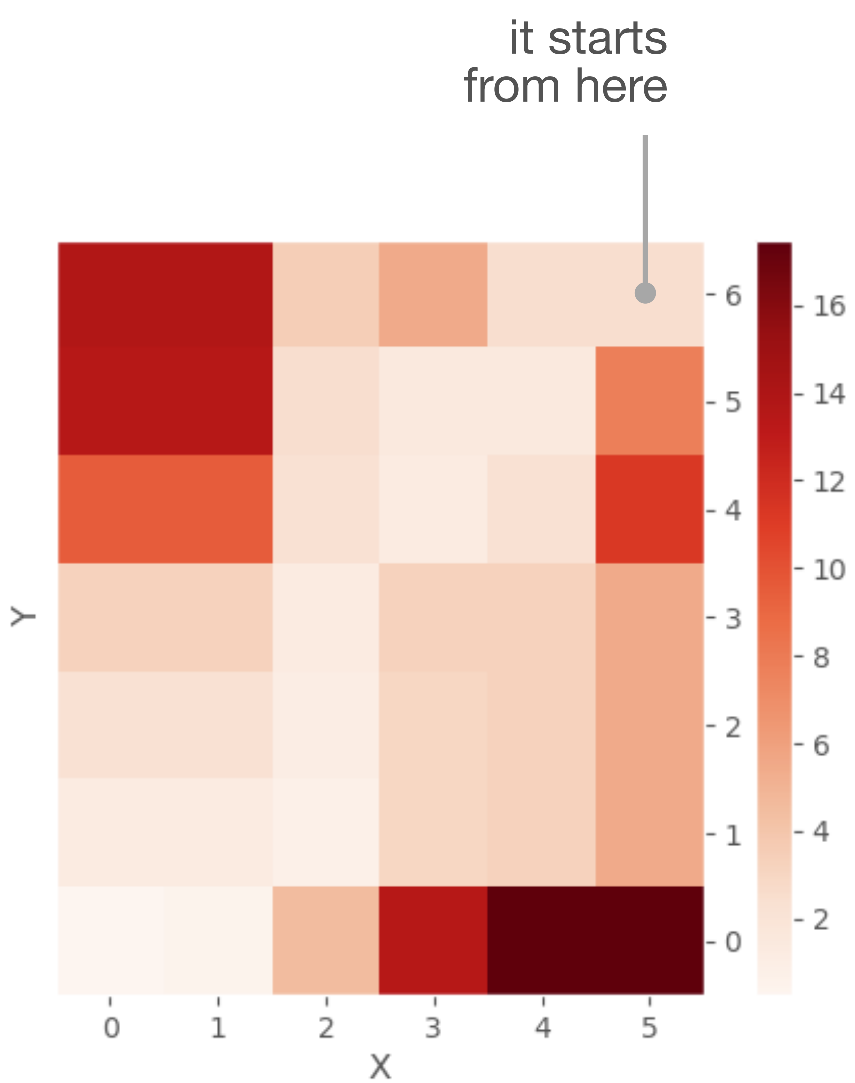


Dynamic Time Warping

3. Search the optimal path

- Name the warping path as `path`.

```
> i       = len(y)-1
> j       = len(x)-1
> path    = [[j,i]]
> while (i > 0) and (j > 0):
      if i==0:
          j   = j-1
      elif j==0:
          i   = i-1
      else:
          if acuCost[i-1,j] == min(acuCost[i-1,j-1],
                                   acuCost[i-1,j],
                                   acuCost[i,j-1]):
              i   = i-1
          elif acuCost[i,j-1] == min(acuCost[i-1,j-1],
                                     acuCost[i-1,j],
                                     acuCost[i,j-1]):
              j   = j-1
          else:
              i   = i-1
              j   = j-1
      path.append([j,i])
> path.append([0,0])
```

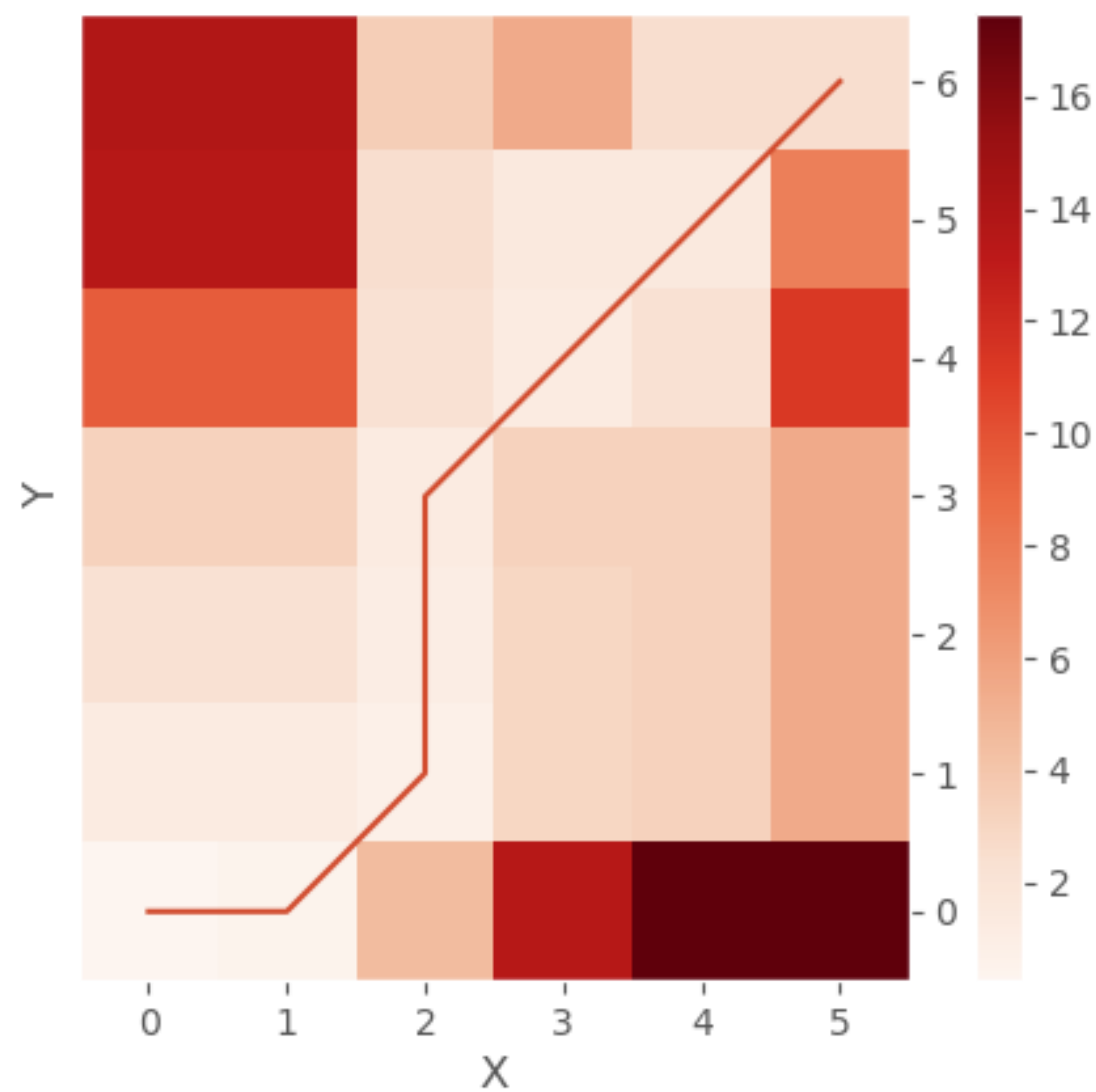


Dynamic Time Warping

3. Search the optimal path

- Create a function that plots the path

```
> def pltCostAndPath(acuCost,path,xlab="X",ylab="Y",clrmap="viridis"):
    px      = [pt[0] for pt in path]
    py      = [pt[1] for pt in path]

    imgplt  = pltDistances(acuCost,
                           xlab=xlab,
                           ylab=ylab,
                           clrmap=clrmap)
    plt.plot(px,py)

    return imgplt

> pltCostAndPath(acuCost,path,clrmap='Reds')
```



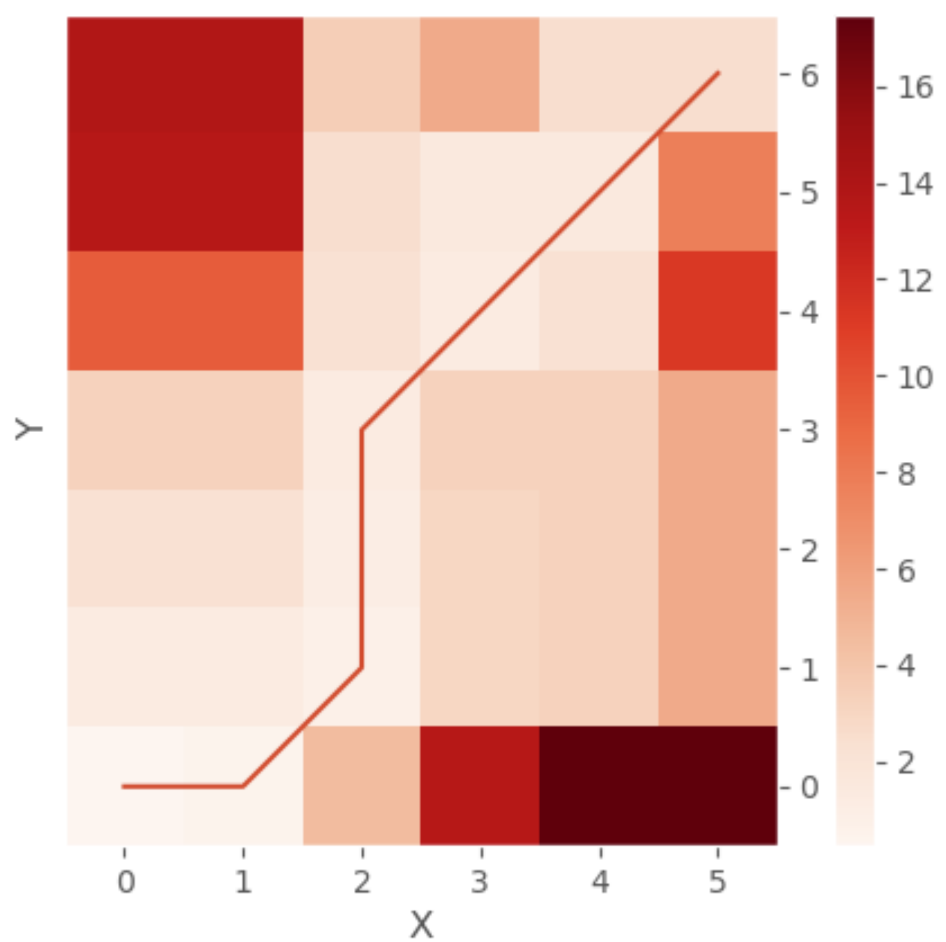
Dynamic Time Warping

3. Search the optimal path

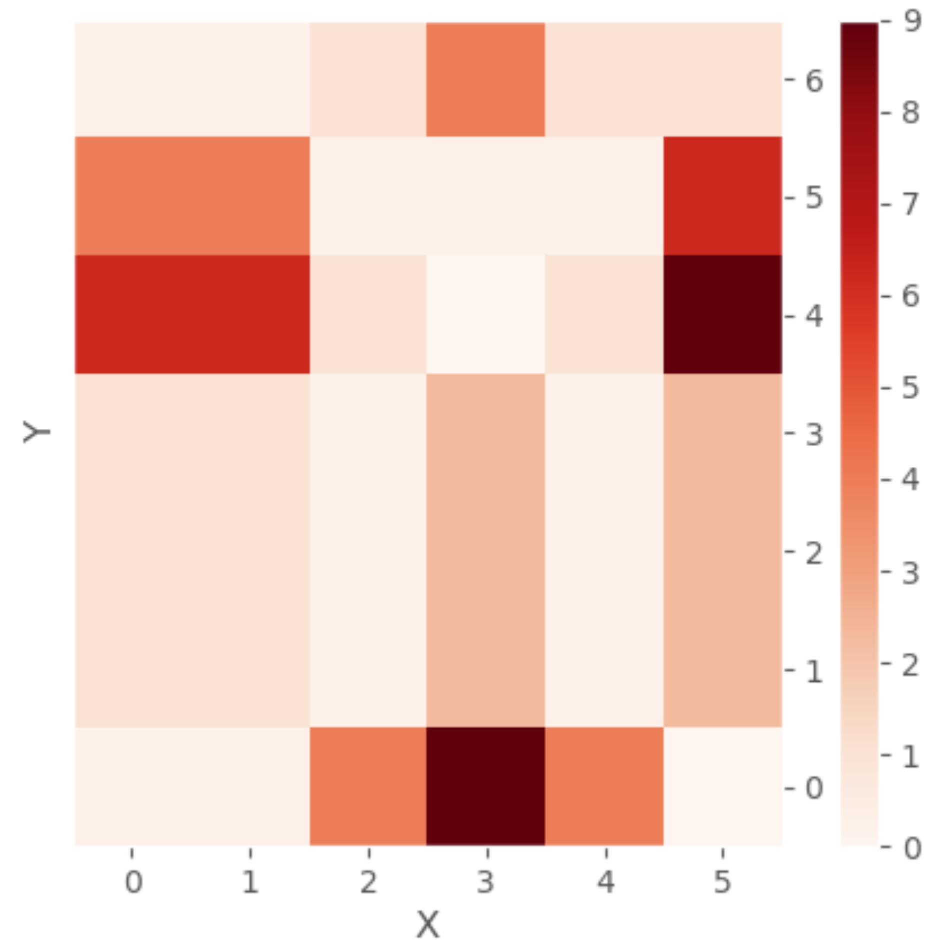
- Calculate the cost, which can be considered as a measure for similarity / distance

```
> cost        = 0
> for [j,i] in path:
      cost    = cost+dists[i,j]
> cost
: 2.5
```

acuCost



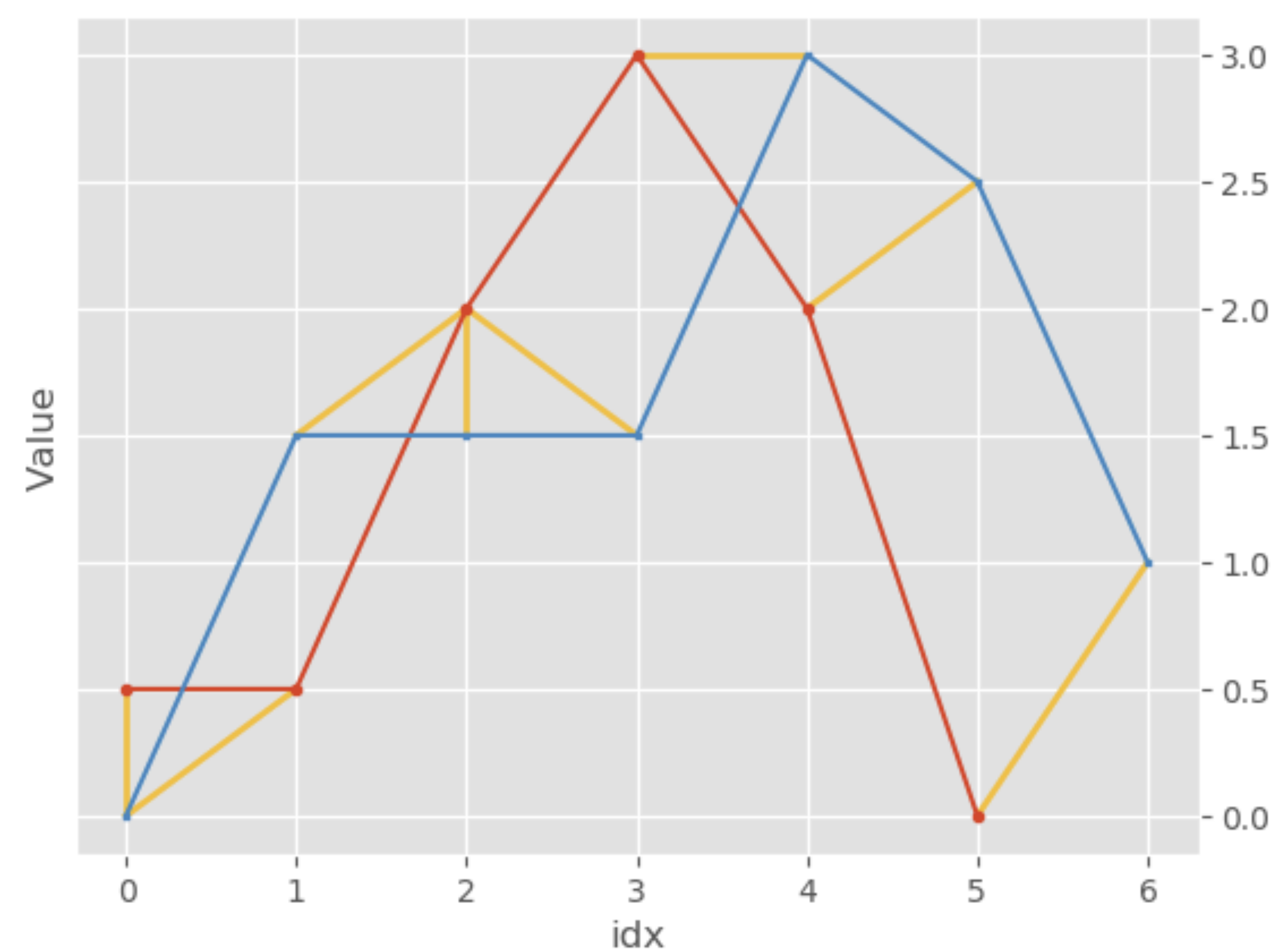
dists



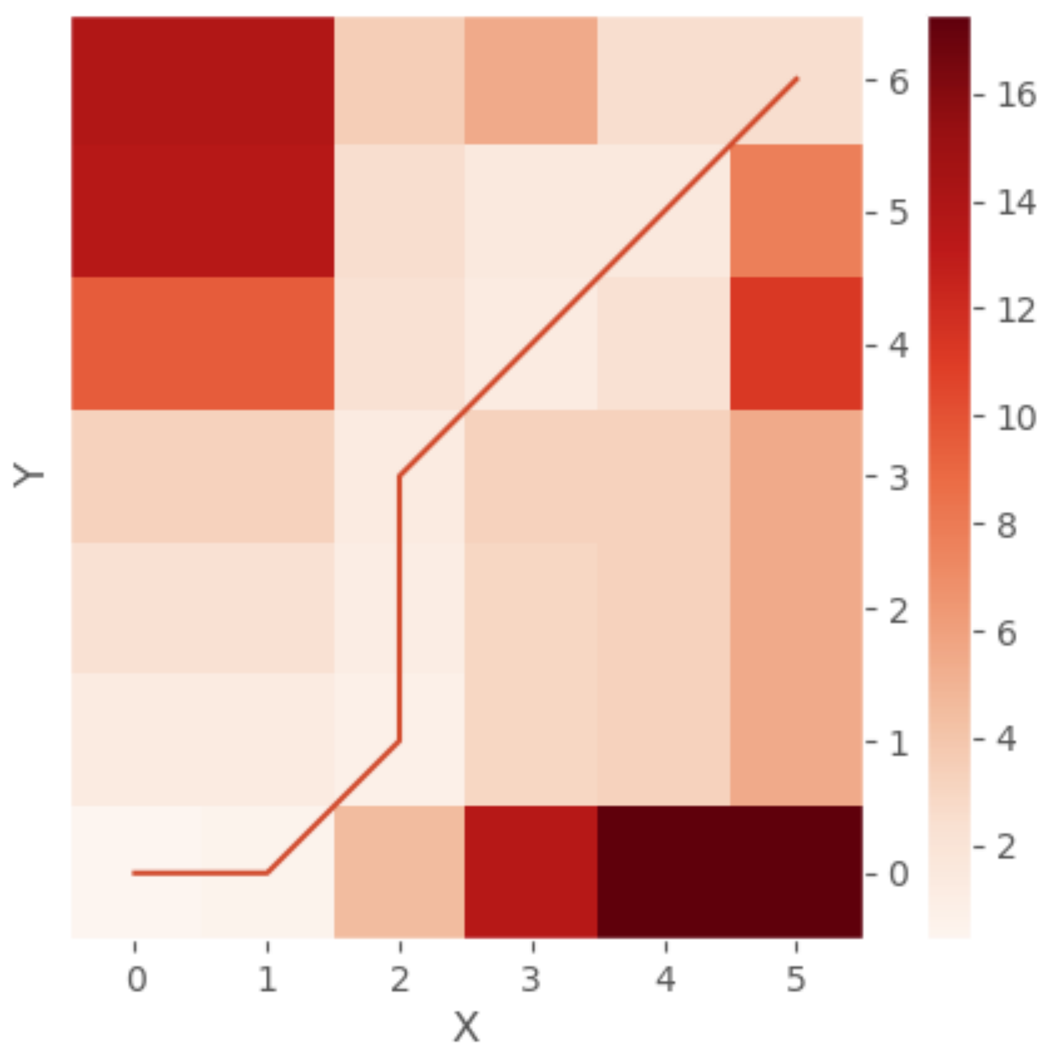
Dynamic Time Warping
3. Search the optimal path

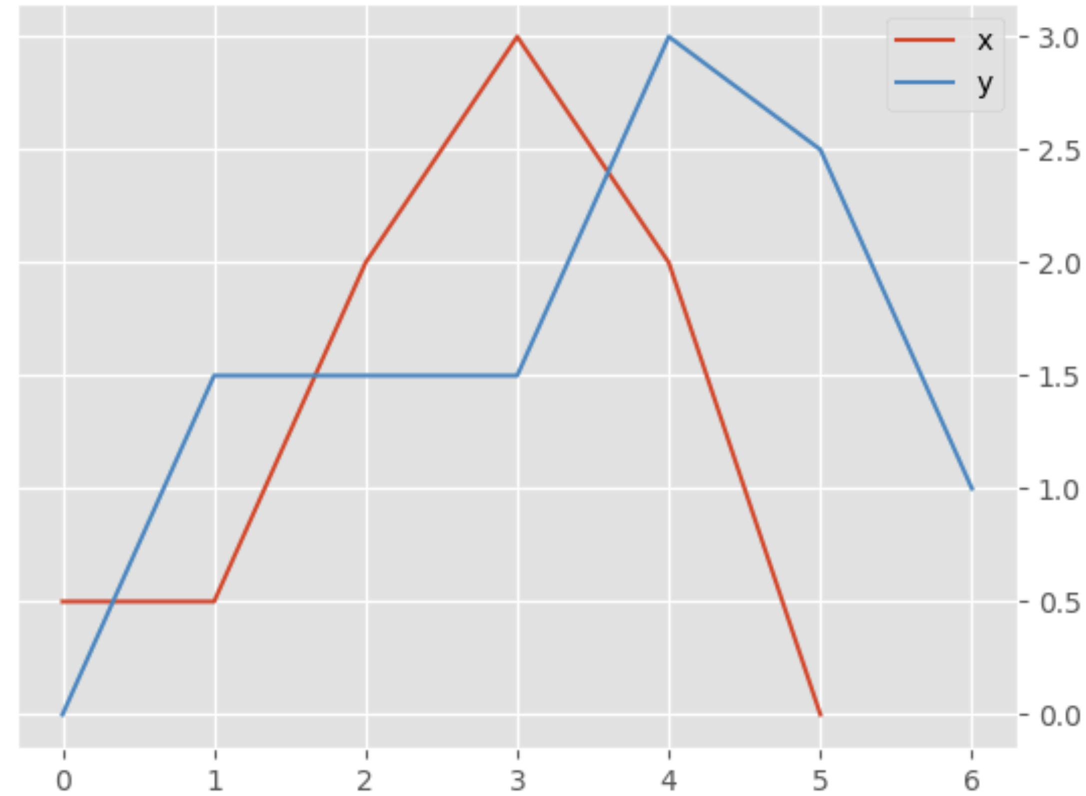
- Plot the mapping of points between two signals

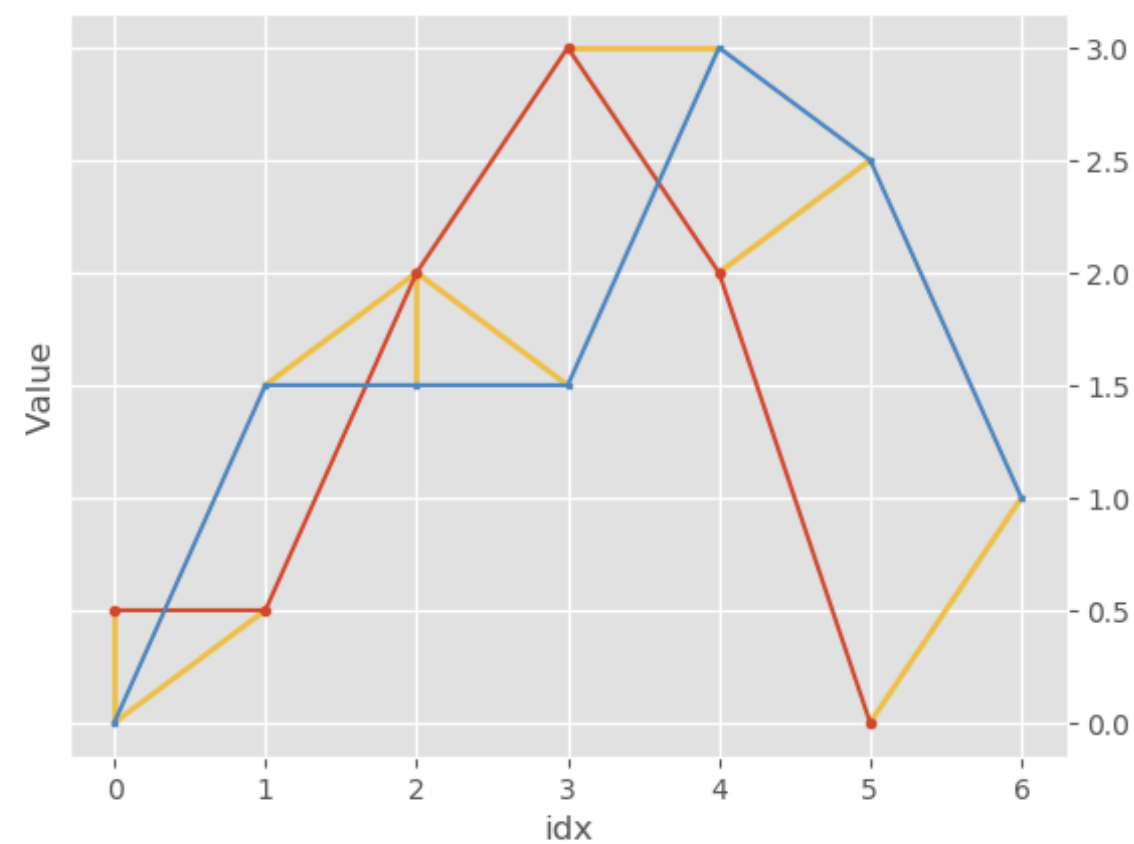
```
> def pltWarp(s1,s2,path,xlab="idx",ylab="Value"):
    imgplt      = plt.figure()

    for [idx1,idx2] in path:
        plt.plot([idx1,idx2],[s1[idx1],s2[idx2]],
                 color="C4",
                 linewidth=2)
    plt.plot(s1,
             'o-',
             color="C0",
             markersize=3)
    plt.plot(s2,
             's-',
             color="C1",
             markersize=2)
    plt.xlabel(xlab)
    plt.ylabel(ylab)

    return imgplt

> pltWarp(x,y,path)
```



Dynamic Time Warping







Dynamic Time Warping

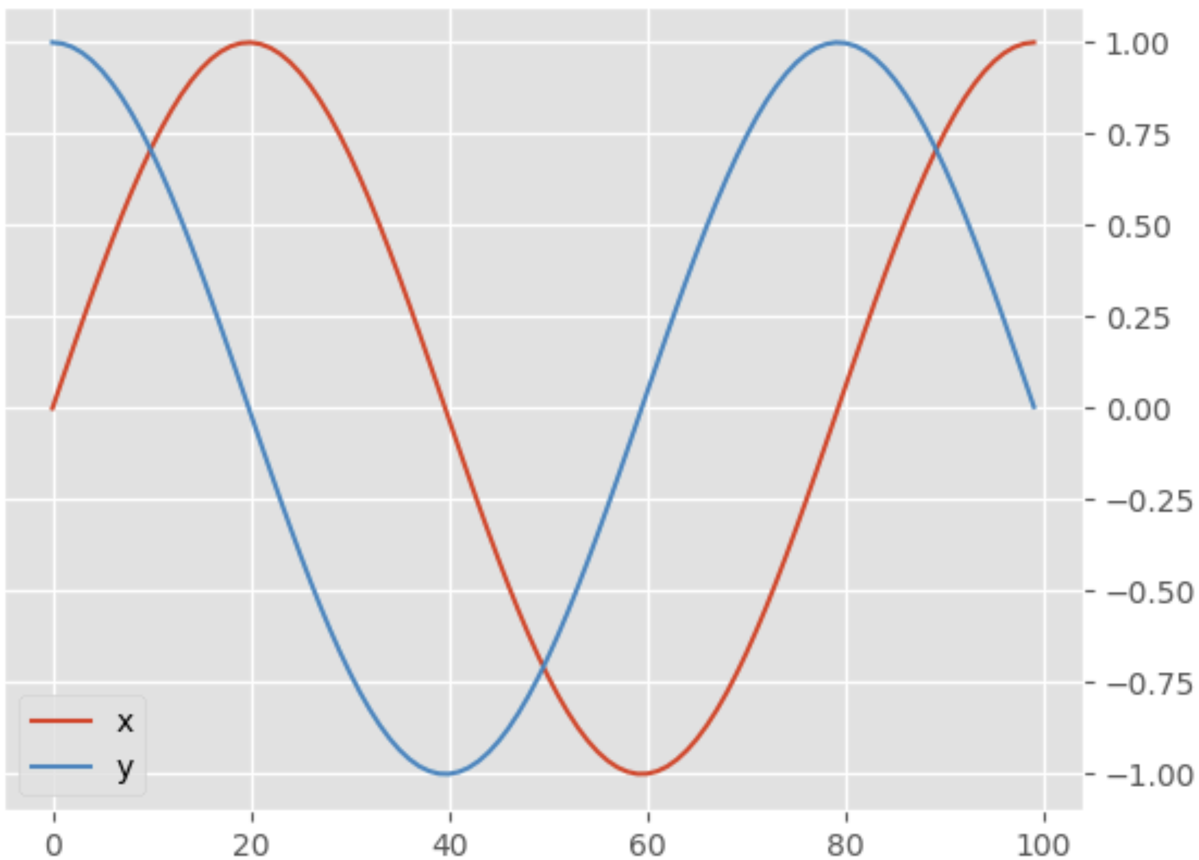
Another example

- Define two signals as:

```
> x = np.sin(np.linspace(0,7.85,100))
> y = np.cos(np.linspace(0,7.85,100))
```

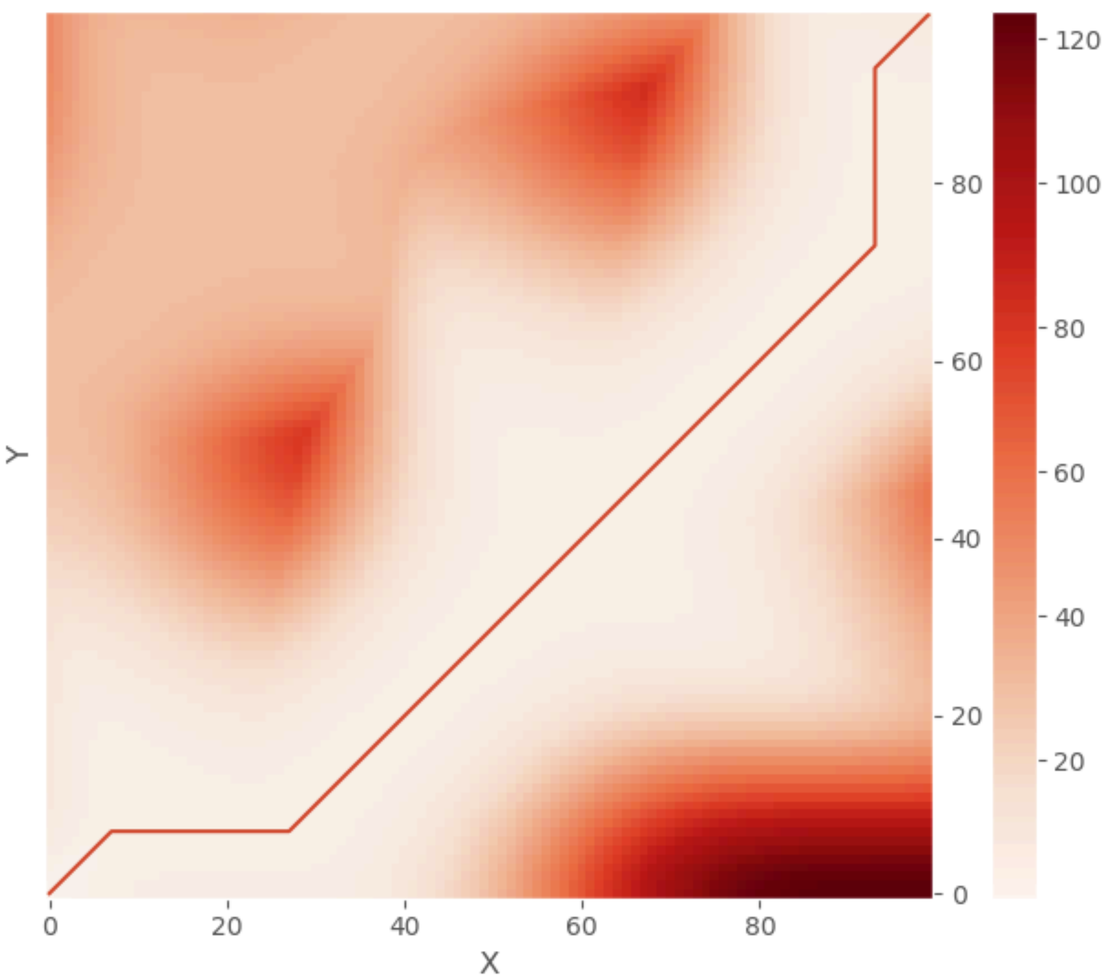
- Plot the two signals

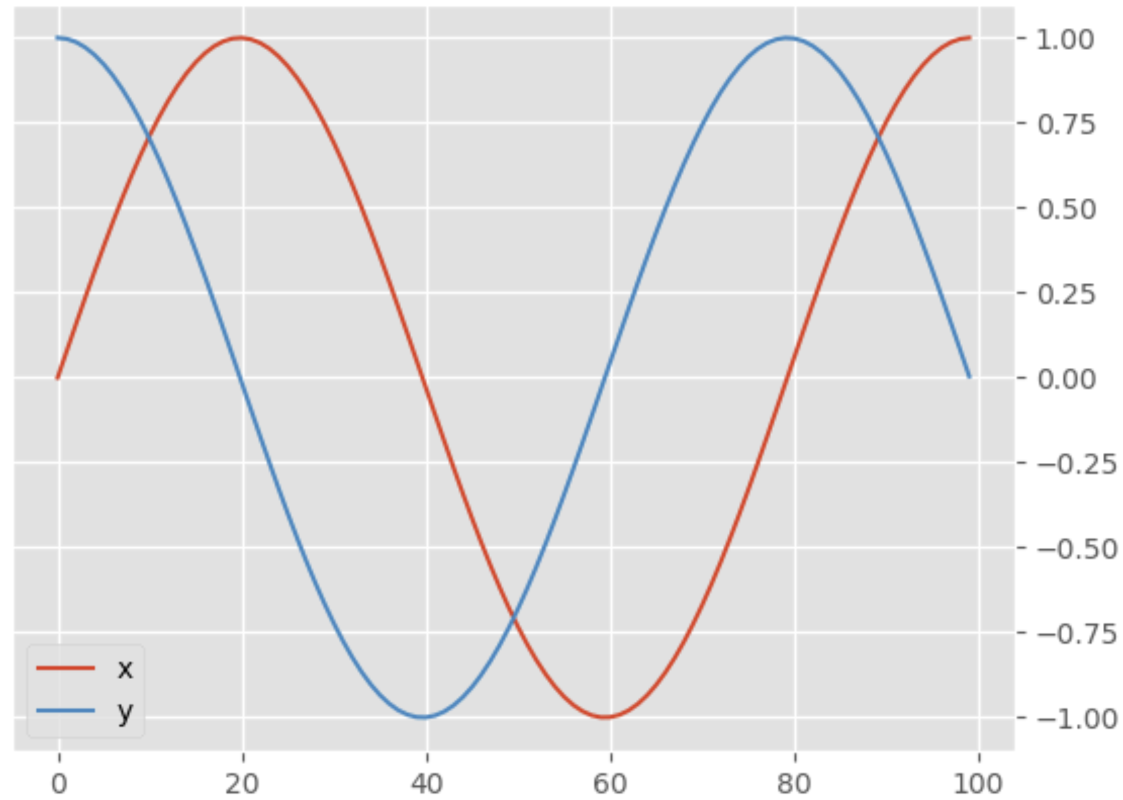
```
> plt.figure()
> plt.plot(x,
           color="C0",
           label='x')
> plt.plot(y,
           color="C1",
           label='y')
> plt.legend()
```

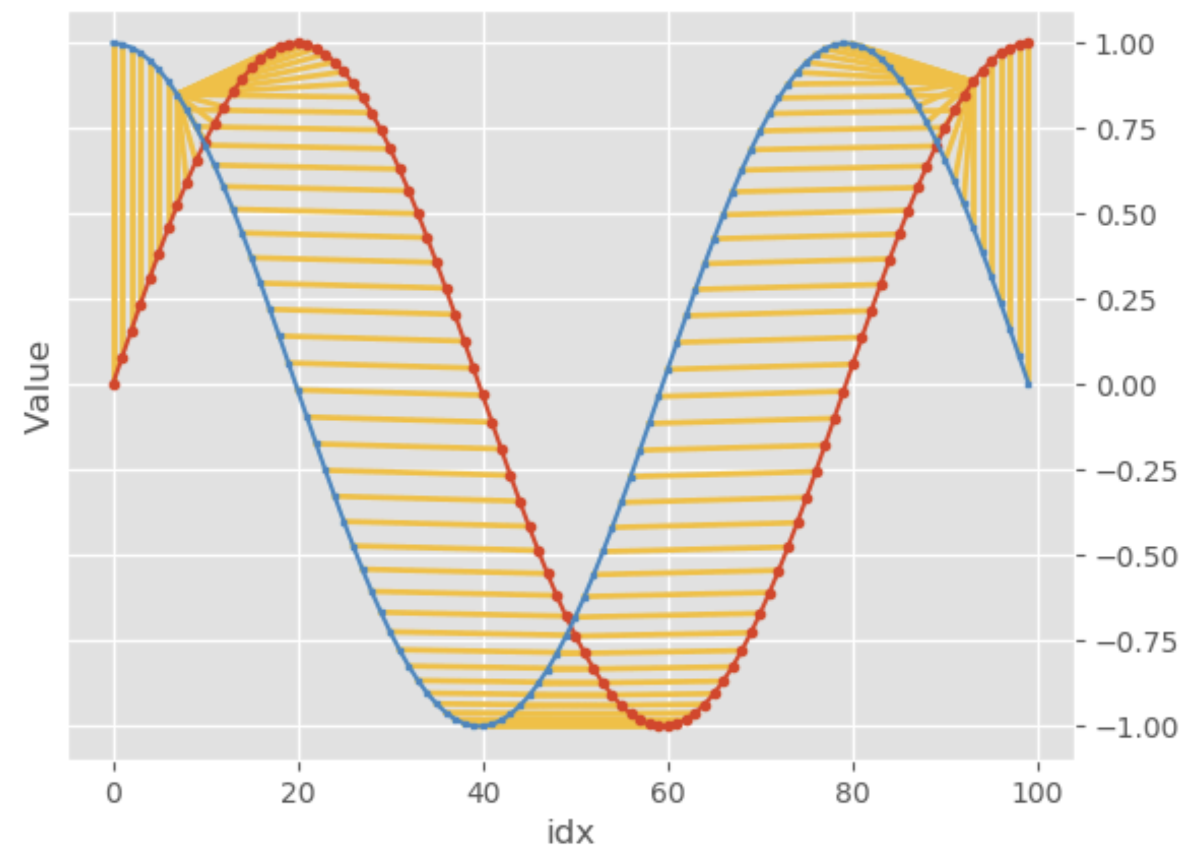


Dynamic Time Warping

Another example



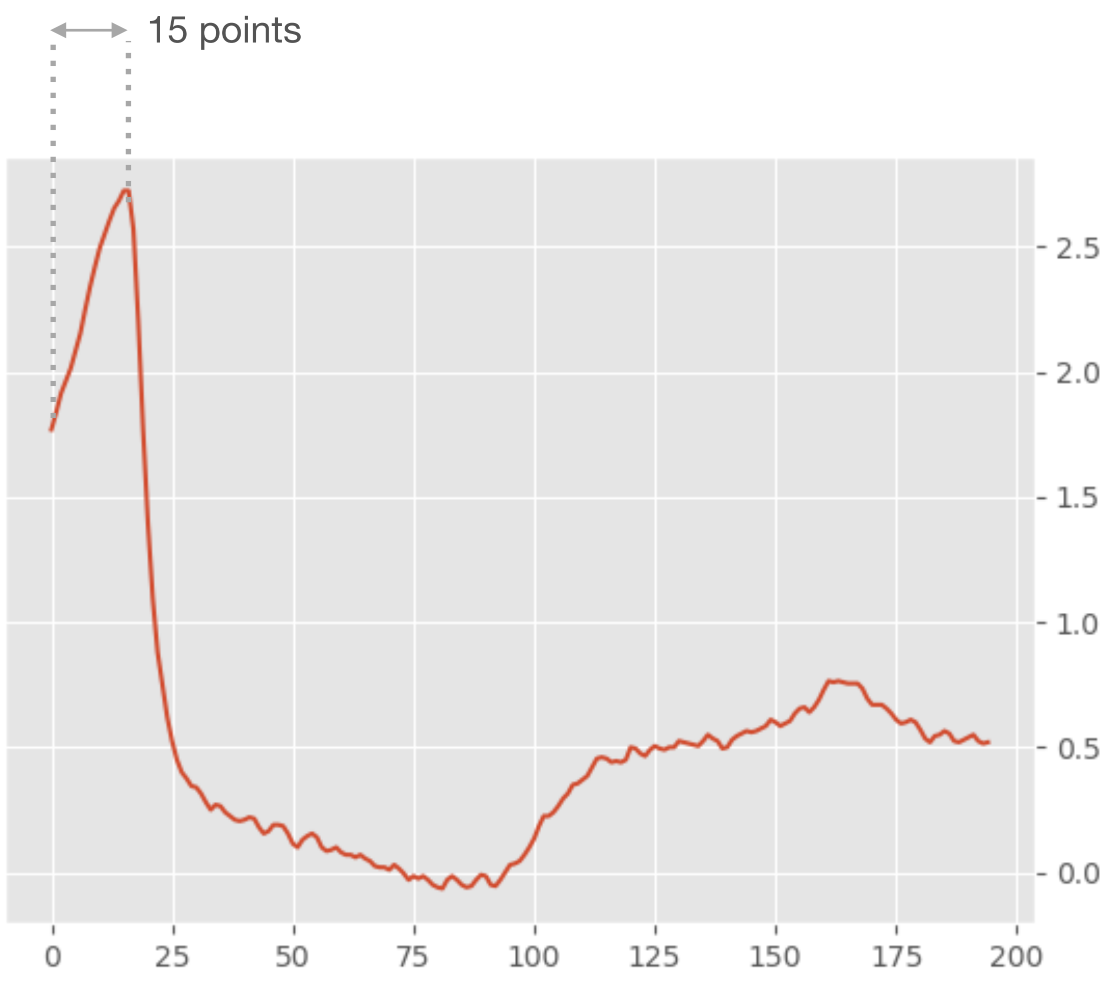


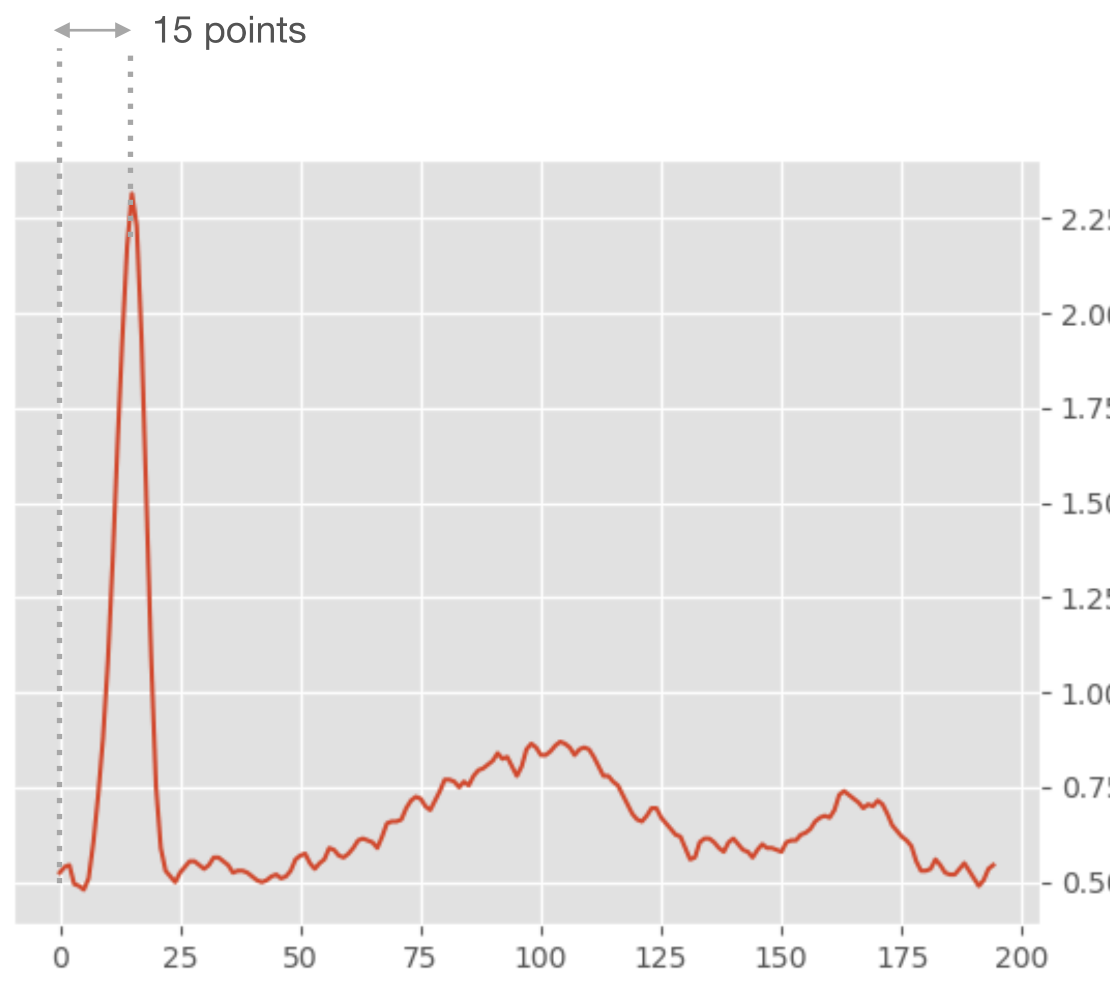


Back to the problem

How to start?

- Before we compute similarity, must segment individual heartbeat signal
- With signal segmented, perform DTW





Workshop

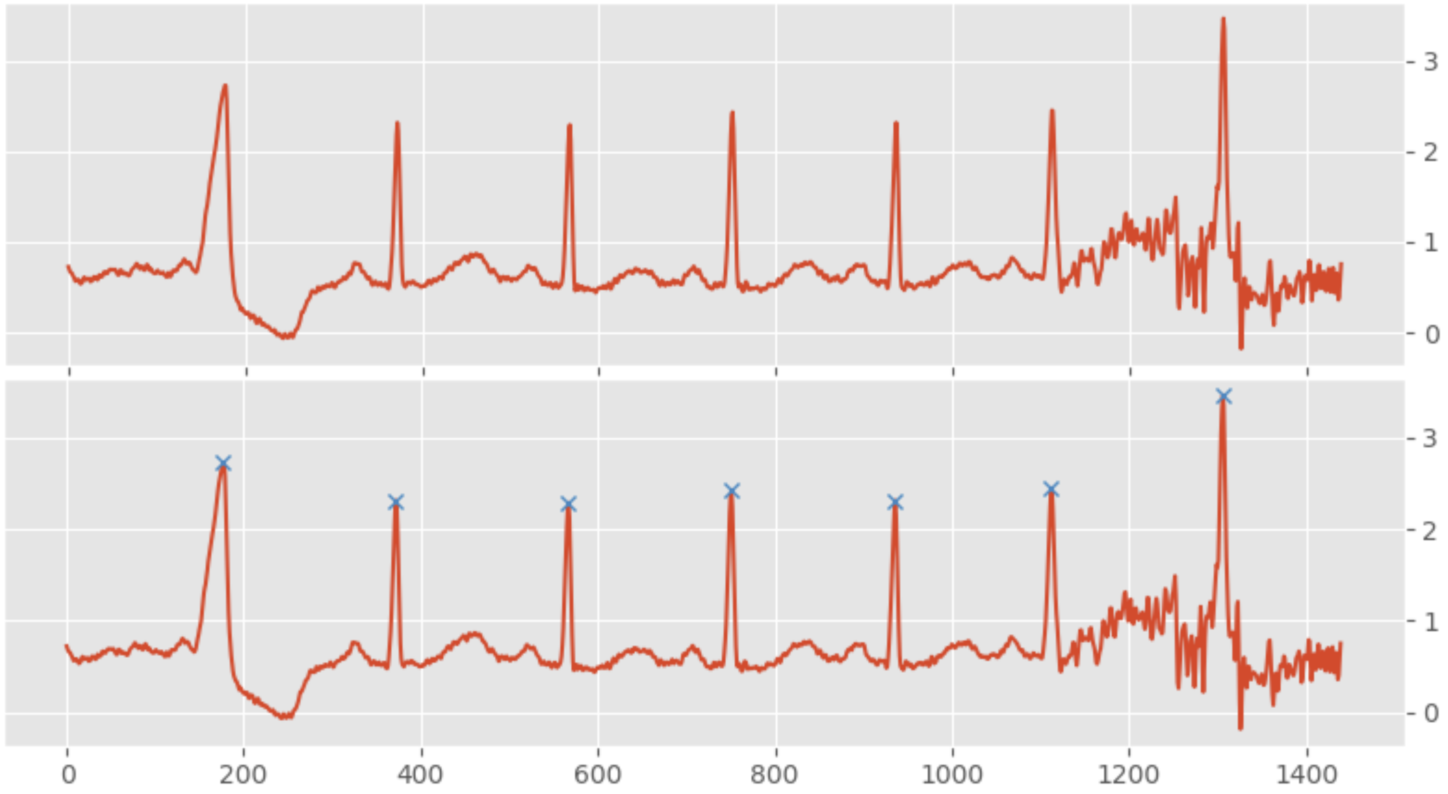
To start

- Load the data

```
> l2D      = pd.read_csv('ecg2D.csv',
                          header=None)
> ECGs     = l2D[1].values
```

- Create a function with the below signature. The output is a list consists of all the ECG segments in a ECG signal

```
def extractECG(ecg,pks,offset=15):
```



Workshop

Compute the accumulated costs, plot the optimal paths and warp

- Make the comparisons between segment
  - 1 and 2
  - 2 and 3
  - 2 and 6