# Time Series Clustering

(Special thanks to Eamonn J. Keogh and Michael J. Pazzani)

Report by

Kaushal Kishore

111601008

November 10, 2018

# Contents

# Problem Statement

The statement of this problem is quite straightforward. Given, some time series design a clustering algorithm so that the similar time series are grouped together.

# Dataset Description

The dataset is available on the following link: synthetic_control.data

The dataset contains 600 rows each with 60 columns, where each row contain a sequence of real valued numbers describing a time series.

The type of time series along with their row indexes are given below:

- [ 0 - 100) : Normal
- [ 100 - 200) : Cyclic
- [ 200 - 300) : Increasing Trend
- [ 300 - 400) : Decreasing Trend
- [ 400 - 500) : Upward Shift
- [ 500 - 600) : Downward Shift

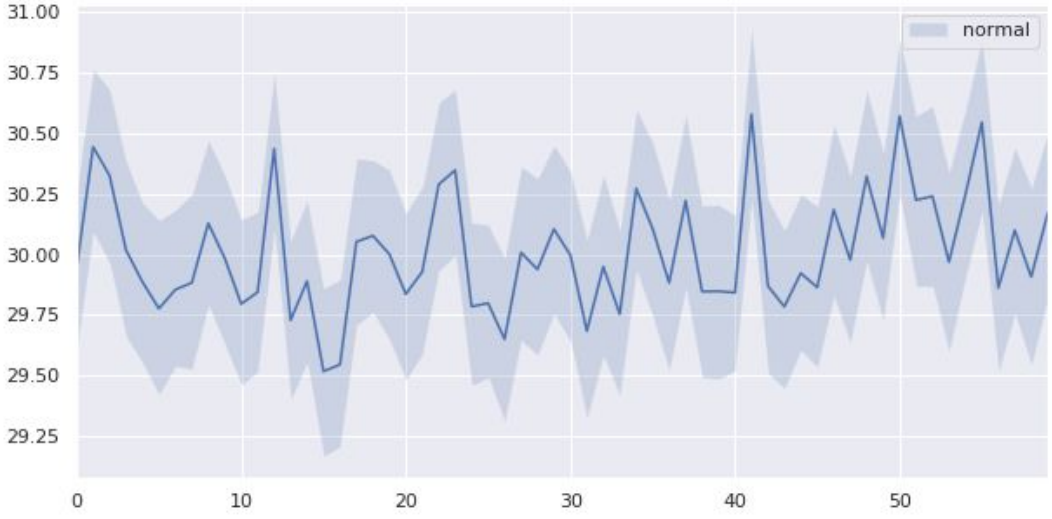
# Other Relevant Informations

Some other informations given on the website hosting this dataset:

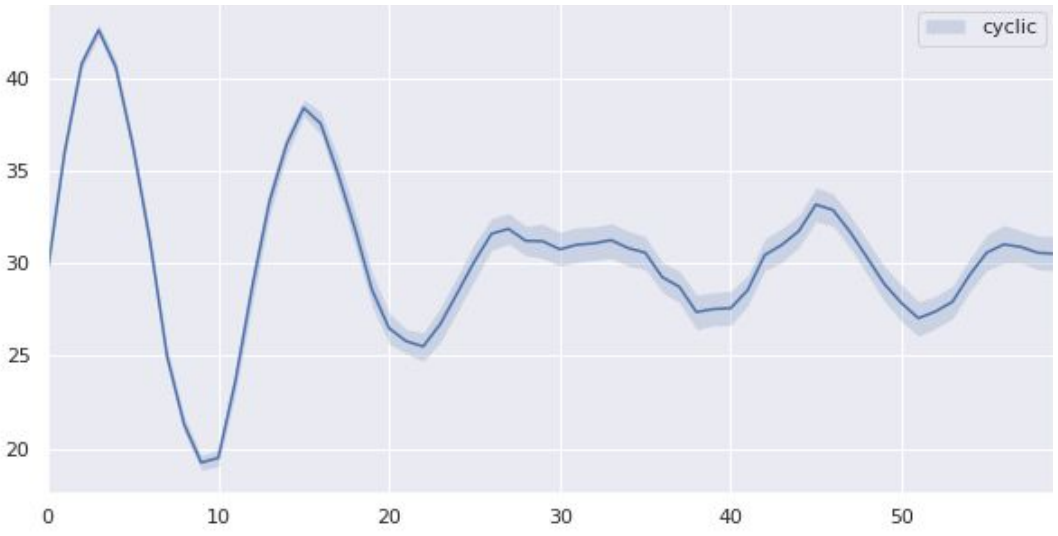
*"This is a good data set to test time series clustering (and classification) algorithms because Euclidean distance will not be able to achieve perfect accuracy. In particular the following pairs of classes will often be confused (Normal/Cyclic) (Decreasing trend/Downward shift) and (Increasing trend/ Upward shift)."*

# Dataset Visualization

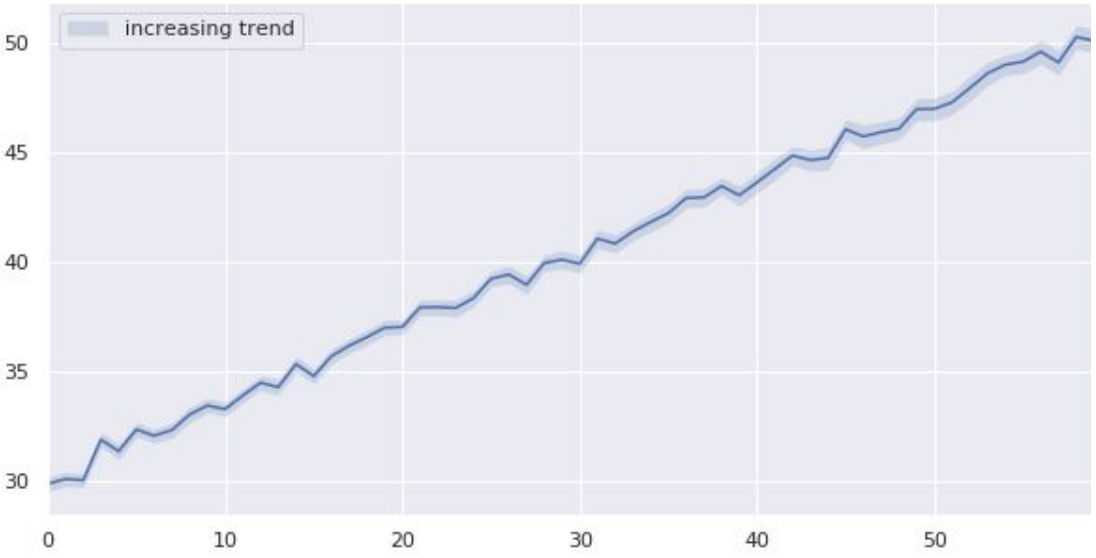
1) Normal:



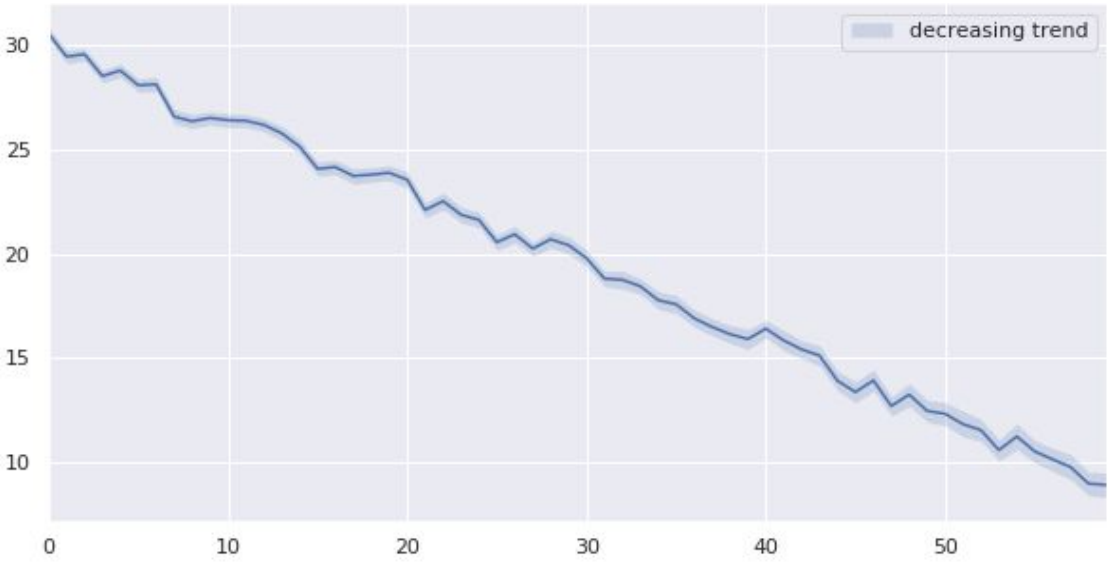

2) Cyclic:



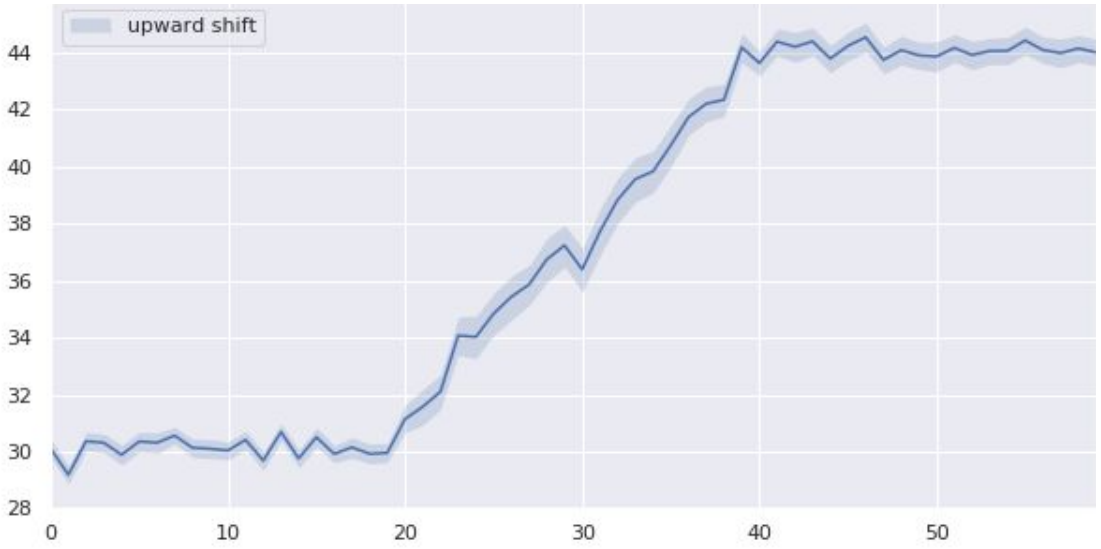

3) Increasing Trend:



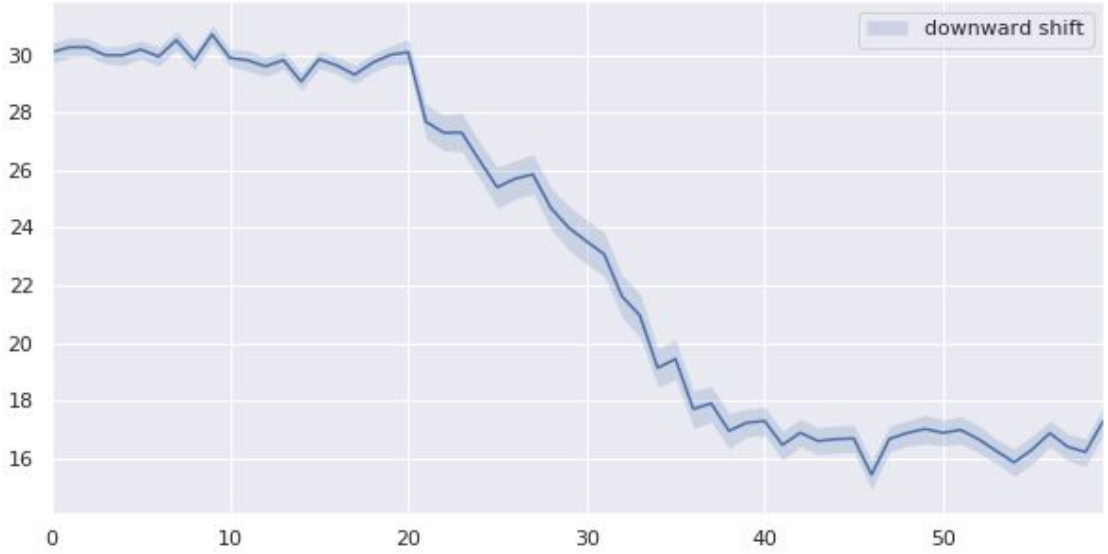

4) Decreasing Trend:


decreasing trend
30
25
20
15
10
0
10
20
30
40
50

5) Upward Shift:



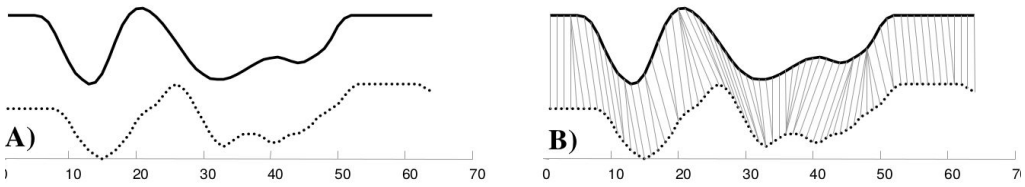

6) Downward Shift:



# Distance Measure: first step for clustering

No clustering is possible without a suitable choice of the similarity/dissimilarity measure. The first choice for a distance(dissimilarity) measure generally is euclidean but unfortunately for this task the euclidean measure is not suitable choice, let's try to understand why.:

Time series are a ubiquitous form of data occurring in virtually every scientific discipline. A common task with time series data is comparing one sequence with another. In some domains a very simple distance measure, such as Euclidean distance will suffice. However, it is often the case that two sequences have the approximately the same overall component shapes, but these shapes do not line up in X-axis.




Above figure(A) shows this with a simple example. In order to find the similarity between such sequences, or as a preprocessing step before averaging them, we must "warp" (fig B) the time axis of one (or both) sequences to achieve a better alignment. Dynamic time warping (DTW), is a technique for efficiently achieving this warping.

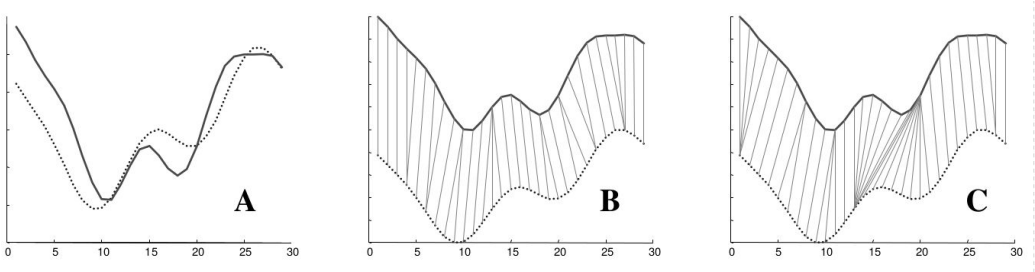
DTW can efficiently find an alignment between the two sequences that allows a more sophisticated distance measure to be calculated.

DTW can also be used as similarity measure between two temporal sequence.

Wikipedia states that: *"A nearest-neighbour classifier can achieve state-of-the-art performance when using dynamic time warping as a distance measure."*

Although DTW has been successfully used in many domains, it can produce pathological results. The crucial observation is that the algorithm may try to explain variability in the Y-axis by warping the X-axis. This can lead to unintuitive alignments where a single point on one time series maps onto a large subsection of another time series. We call examples of this undesirable behavior "s*ingularities".*

An additional problem with DTW is that the algorithm may fail to find obvious, natural alignments in two sequences simply because a feature (i.e peak, valley, inflection point, plateau etc.) in one sequence is slightly higher or lower than its corresponding feature in the other sequence.




The above figure illustrates the problem faced by DTW.

- Fig A: Two synthetic signals (with the same mean and variance)
- The natural "feature to feature" alignment.
- The alignment produced by dynamic time warping.

Note that DTW failed to align the two central peaks because they are slightly separated in the Y-axis.

Before we move on to address this problem first let's talk about how DTW works. Consider two time series Q and C of length n and m respectively, where:

$Q = q_1, q_2, \ldots\ldots\ldots q_i \ldots\ldots\ldots\ldots\ldots, q_n$

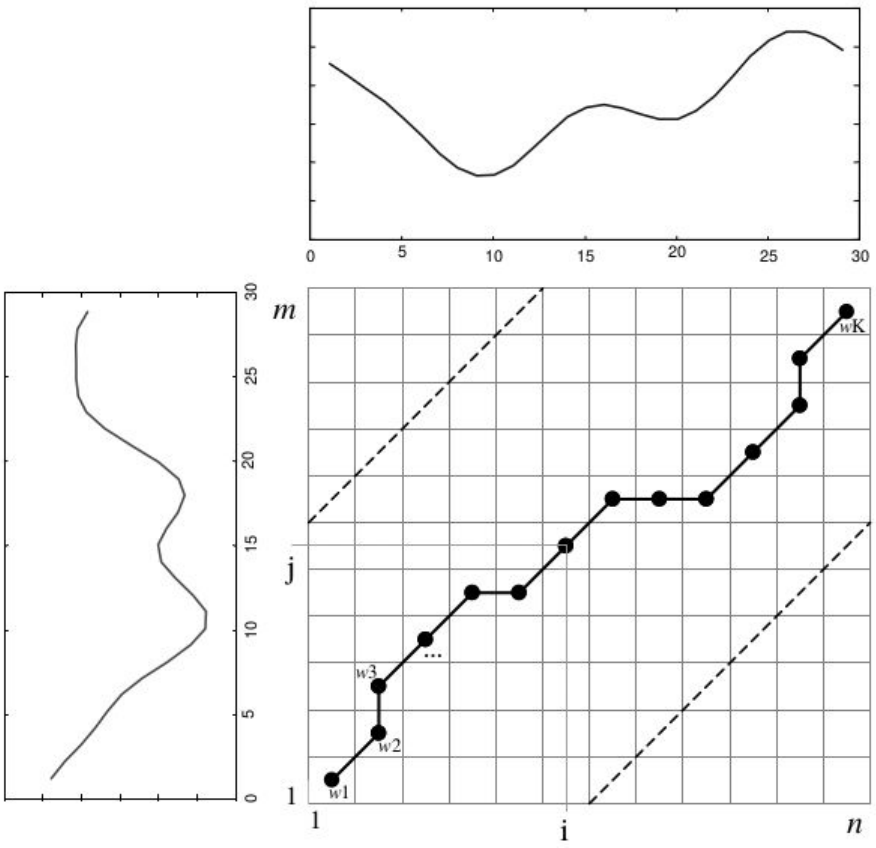
$C = c_1, c_2, \ldots\ldots\ldots c_i \ldots\ldots\ldots\ldots\ldots, c_m$

To align two sequences using DTW we construct an n-by-m matrix where the ($i^{th}$ , $j^{th}$ ) element of the matrix contains distance $\mathbf{d(q_i, c_j)}$ between the two points $q_i$ and $c_j$. Typically, the Euclidean distance is used, so $d(q_i, c_j) = (q_i - c_j)^2$. Each matrix element (i,j) corresponds to the alignment between the points $q_i$ and $c_j$. A warping path W, is a contiguous (in the sense stated below) set of matrix elements that defines a mapping between Q and C. The $k^{th}$ element of W is defined as $w_k = (i,j)_k$ so we have:

$$W = w_1 , w_2 , \ldots, w_k , \ldots, w_K$$

$$\text{where, } \max(m,n) \leq K < m+n-1$$

There are exponentially many warping paths that satisfy the above conditions, however we are interested only in the path which minimizes the warping cost.




This path can be found very efficiently using dynamic programming to evaluate the following recurrence which defines the cumulative distance $\gamma(i,j)$ as the distance d(i,j) found in the current cell and the minimum of the cumulative distances of the adjacent elements:

i.e.,

$$\gamma(i,j) = d(q_i, c_j) + \min\{ \gamma(i-1, j-1), \gamma(i-1, j), \gamma(i, j-1) \}$$

The above given equation is the most important line in this entire document. We are using the method of dynamic programming to efficiently calculate the cumulative distance measure.

The pseudo code for the DTW algorithm is given below:

```
int DTWDistance(q: array [1..n], c: array [1..m]) {
```

```
    DTW := array [0..n, 0..m]

    for i := 1 to n
        DTW[i, 0] := infinity
    for i := 1 to m
        DTW[0, i] := infinity
    DTW[0, 0] := 0

    for i := 1 to n
        for j := 1 to m
            cost := d(q[i], c[j])
            DTW[i, j] := cost + minimum( DTW[i-1, j  ],
                                         DTW[i  , j-1],
                                         DTW[i-1, j-1] )

    return DTW[n, m]
}
```

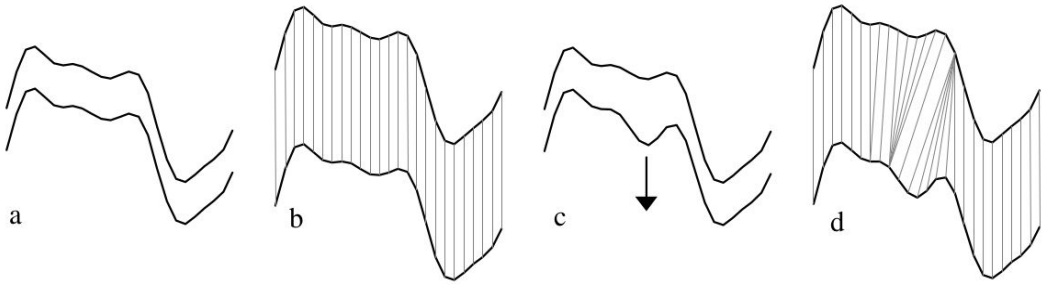
where **DTW[i, j]** is the distance between **q[1:i]** and **c[1:j]** with the best alignment.

The time complexity of DTW algorithm is O(NM), where N and M are the lengths of the two input sequences.

As already mentioned the vanilla DTW works quite well but everybody appreciate modifications which leads to better results. Following the same spirit let's present a modification which will allow us to achieve a better accuracy.

Introducing Derivative Dynamic Time Warping(DDTW)

If DTW attempts to align two sequences that are similar except for local accelerations and decelerations in the time axis, the algorithm is likely to be successful. The algorithm has problems when the two sequences also differ in the Y-axis. The two series may also have local differences in the Y-axis, for example a valley in one sequence may be deeper that the corresponding valley in the other time series. Below figure illustrates that.




Using DTW, two identical sequences (a) will clearly produce a one to one alignment (b). However, if we slightly change a local feature, in this case the depth of a valley (c), DTW attempts to explain the difference in terms of the time-axis and produces two singularities (d).

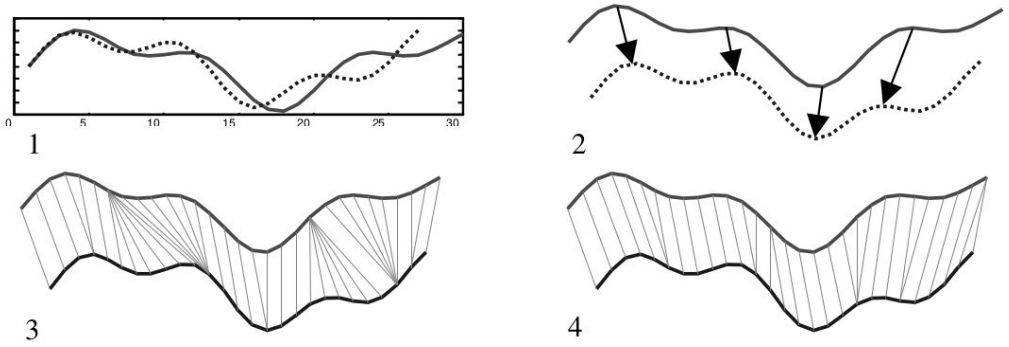
The weakness of DTW is in the features it considers. It only considers a data points Y-axis value. For example consider two data points $q_i$ and $c_j$ which have identical values, but $q_i$ is part of a rising trend and $c_j$ is part of a falling trend. DTW considers a mapping between these two points ideal, although intuitively we would prefer not to map a rising trend to a falling trend. To prevent this problem a modification of DTW that does not consider the Y-values of the data points, but rather considers the higher level feature of "shape" can be done. We obtain information about shape by considering the first derivative of the sequences, and thus the name **Derivative Dynamic Time Warping (DDTW)**.

Algorithm Details:

- Construct derivative sequences of real valued numbers from the given time sequence Q and C.
- To calculate the derivative sequence for Q use the following formula:
  - $D_q[i] = \frac{(q_i - q_{i-1}) + ((q_{i+1} - q_{i-1})/2)}{2}$ ; $1 < i < m$
  - $D_q[0] = D_q[1]$
  - $D_q[m] = D_q[m-1]$
- Similarly, calculate the derivative sequence $D_c$ for the time series C.

- Once the derivative sequences $D_q$ and $D_c$ are computed to pass these sequences to the DTW function which will return the distance between sequences $D_q$ and $D_c$ in their best alignment. This distance measure is the result of the DDTW algorithm on the time sequence Q and C.

Note that the derivative estimate given above is simply the average of the slope of the line through the point in question and its left neighbor, and the slope of the line through the left neighbor and the right neighbor. Empirically this estimate is more robust to outliers than any estimate considering only two data points. Note the estimate is not defined for the first and last elements of the sequence. Instead we use the estimates of the second and penultimate elements respectively.




The above image illustrates the comparison between the DTW and DDTW algorithm on given two time series. The fig. 2 shows the intuitive feature to feature warping alignment. Fig. 3 is the alignment produced by classic DTW and fig. 4 is the alignment produced by DDTW.

Conclusion:

- Don't use the plain euclidean distance to find the similarity/dissimilarity between two time series unless you want to screw up your project.
- DTW is a good similarity measure but with some slight modifications it can perform better.

- DDTW, seems to be a better distance measure for the current situation.

For the purpose of this report we will stick with the similarity measure DDTW as it includes all the features of the vanilla DTW with some extra flavours.

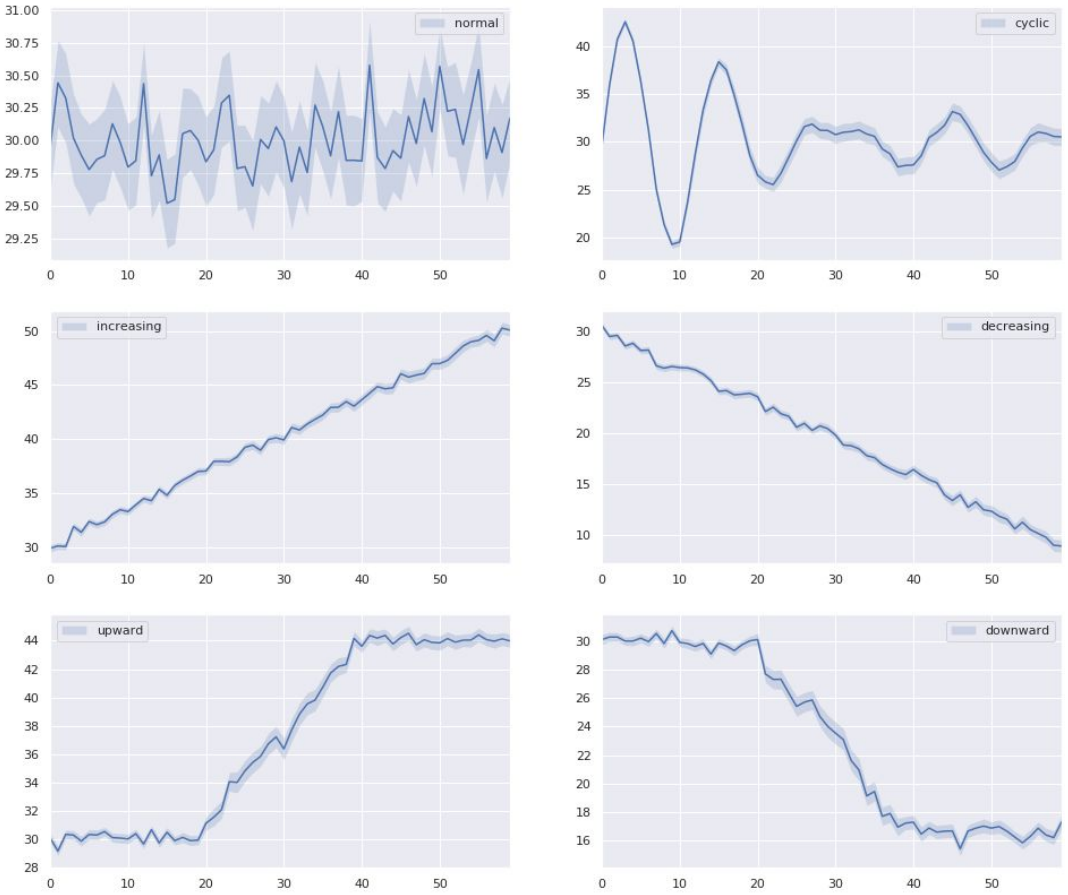
This ends the discussion on the choice of the method for computing the similarity/dissimilarity between the temporal sequences. From this point in this report, you may find some python code and the results of the implementation in a interactive way.

# Technology Stack

- Programming Language: Python 3.5
- Platform: JupyterLab
- Libraries used: Numpy, matplotlib, seaborn, scipy, sklearn and pandas

# Preprocessing

Visualizing the data once again:

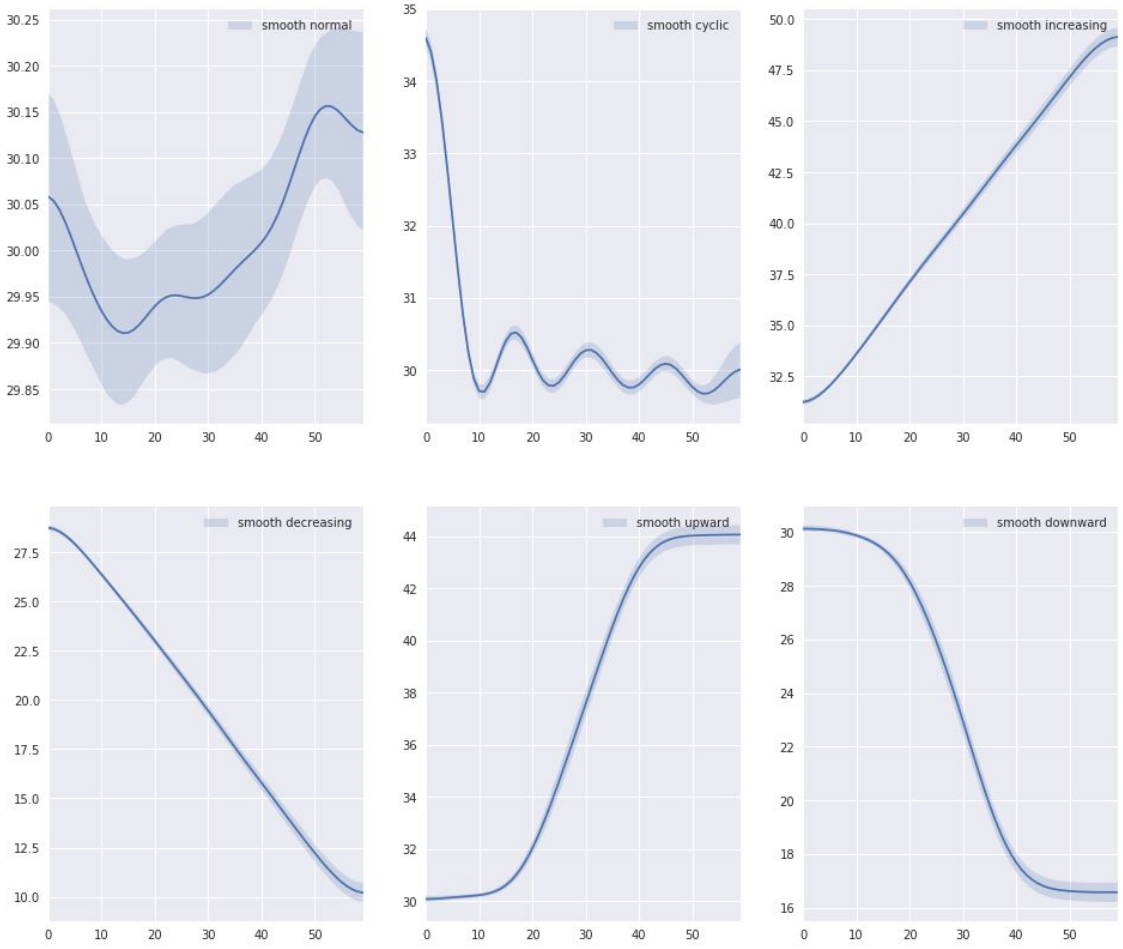



It's always a good practice to perform some preprocessing like noise cancellation, exponential smoothing, normalization, etc. before feeding into you model.

As we can see the above data too noisy to deal with. Hence the first step that one must take is to smooth them out. We can smooth them in python by using the following command in scipy.ndimage.filters:

```
time_series =  gaussian_filter1d(time_series , 5)
```

The results of the smoothing are given below:




Wow! the above smoothing is quite impressive. however the shape of the normal is quite surprising.

The next section discuss the methodology used for clustering the datasets.

# Method

The next step after preprocessing is to calculate the derivative sequence matrix. Each row of this matrix is the derivative sequence

of the corresponding data entry in the dataset matrix where each row corresponds to a time series.

Once we have computed this Derivative of Sequence matrix we can proceed to the process of calculating the distance matrix. Let's denote the distance matrix by the notation D, where an entry D[i,j] is the distance measure between the temporal sequence i and j computed by using the DDTW algorithm.

The python code for computing the derivative of sequence and distance matrix (which can be combined within a class declaration) is given below:

The class object is defined as follows:

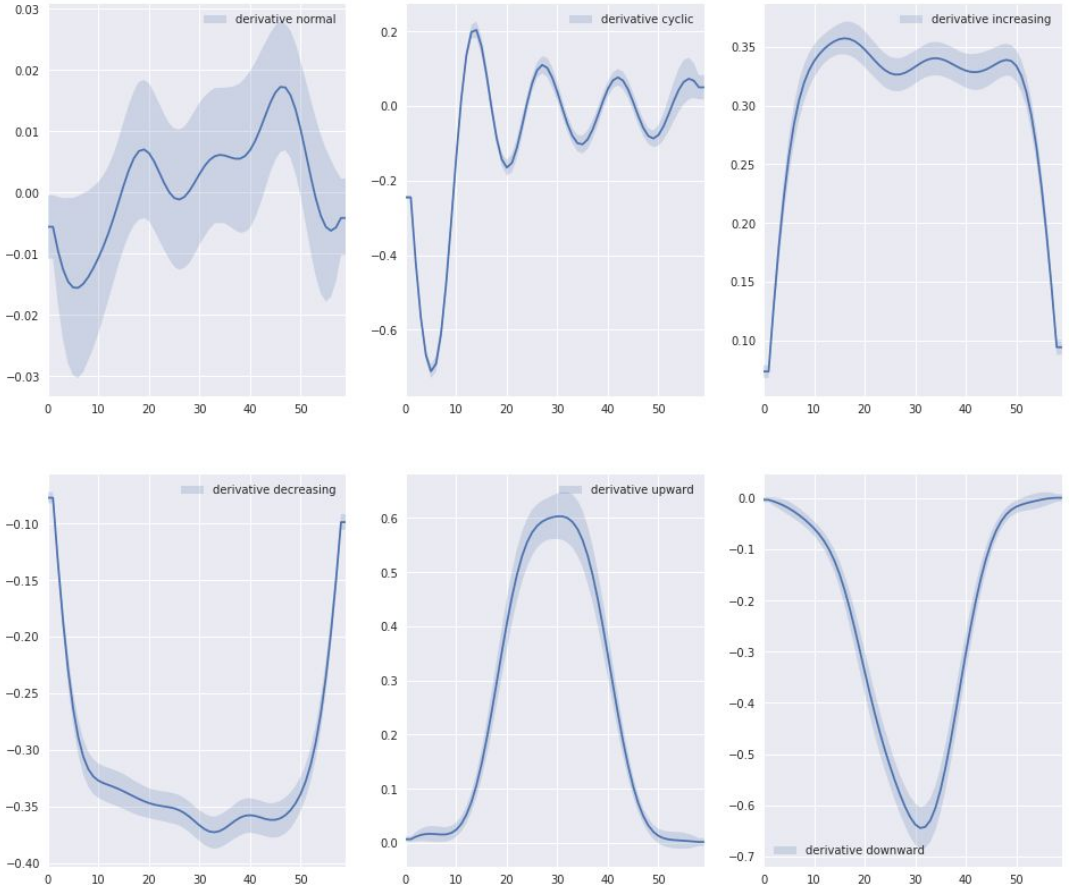
```
class DerivativeDTW(object):
    def __init__(self , time_series, filt=True, v=1):
        pass
```

```
def computeDerivativeMatrix(self):
    if (self.isDerivativeMatrixCreated == True):
        return

    # iterate through all the time series in the list
    for i in range(self.time_series.shape[0]):
        # for each time_series calculate the Derivative
        q = self.time_series[i , :]
        for j in range(1 , q.shape[0]-1):
            self.DerivativeMatrix[i , j] = ((q[j] - q[j-1]) + ((q[j+1] - q[j-1])/2))/2

        # set the boundary derivatives
        self.DerivativeMatrix[i , 0] = self.DerivativeMatrix[i , 1]
        self.DerivativeMatrix[i,-1] = self.DerivativeMatrix[i , -2]

    if self.filt == True:
        self.DerivativeMatrix = gaussian_filter1d(self.DerivativeMatrix , self.v)
    self.isDerivativeMatrixCreated = True
```

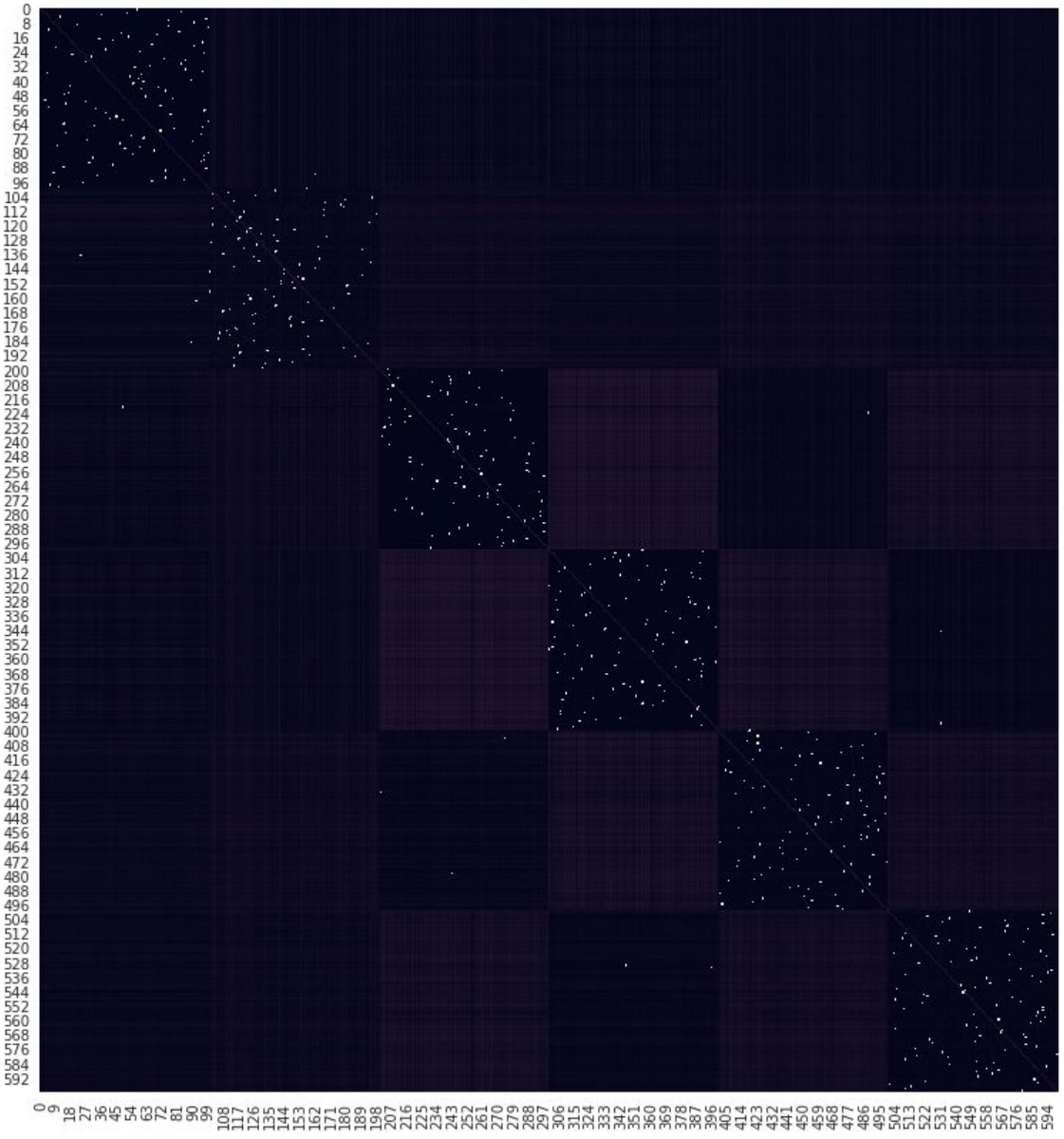
```
def computeDistanceMatrix(self):
    if self.isDistanceMatrixCreated == True:
        return

    if self.isDerivativeMatrixCreated == False:
        self.computeDerivativeMatrix()

    for i in range(self.time_series.shape[0]):
        for j in range(i):
            self.distanceMatrix[i,j] =
            self.distanceMatrix[j,i] =
                    self.computeDtw(q=self.DerivativeMatrix[i, :],
                                    c=self.DerivativeMatrix[j, :])
    self.dtwMatCreated = True
```

```
@staticmethod
def computeDtw(q , c):
    assert(q.shape == c.shape)
    m = q.shape[0]
    Y = np.zeros((m+1,m+1) , dtype=np.float)

    Y[: , 0] = np.inf
    Y[0 , :] = np.inf
    Y[0,0] = 0

    for i in range(m):
        for j in range(m):
            yi , yj = i+1 , j+1
            Y[yi , yj] = abs(q[i] - c[j]) +
                min(Y[yi-1 , yj-1] ,
                    Y[yi-1 , yj] ,
                    Y[yi , yj-1])
    return Y[m,m]
```

Let's visualize the derivative sequence:



Visualizing the distance matrix:



The white dots in the above distance matrix denotes the minimum value in each row. Hence for any row i, let the white dot be in column j, this means that the distance of i is minimum to j, for i not equal to j.

Now that we have implemented the Derivative Dynamic Time Warping algorithm to calculate the distance matrix. Let's use the

distance matrix to cluster the time series. Note that the distance is quite perfect, since the white dots are on the diagonal (why?).

Since we know the number of the clusters we will choose a algorithm which takes the number of the clusters as an argument in contrast to algorithms like DBSCAN where the concept of radius and minimum threshold arise which we have to adjust manually and is very hard to tune. Hence we are left with algorithms like K-Means, Agglomerative Clustering, and other variants of Hierarchical Clustering with the number of clusters as parameter.

Personally I found the Hierarchical clustering to have a better performance on this dataset, hence I am not going to discuss the other algorithms but you may try to explore the possibilities.

To perform hierarchical clustering using this distance matrix use the following python code:

```
ddtw = DerivativeDTW(time_series, filt=False)
distArray = ssd.squareform(ddtw.distanceMatrix)
Z = linkage(distArray, 'ward')
```

We can get the clustering prediction using the following command:

```
c = cut_tree(Z, n_clusters=6)
```

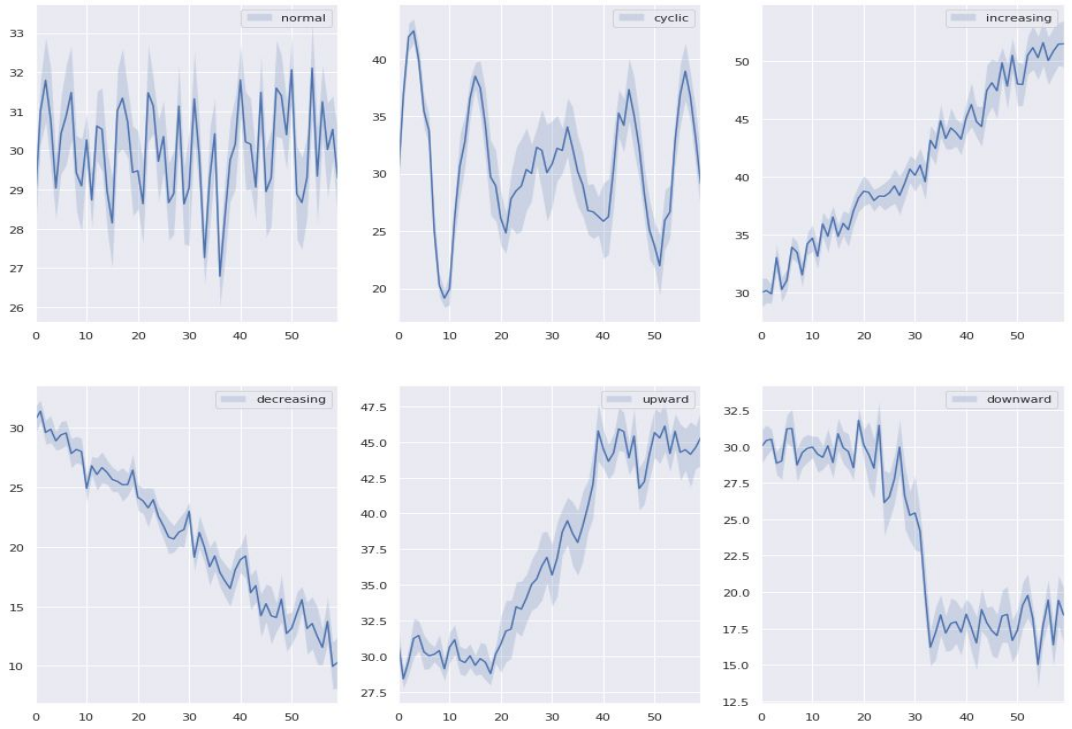
We can also draw the dendrogram by using the following command:

```
dn = dendrogram(Z ,leaf_font_size=10)
```

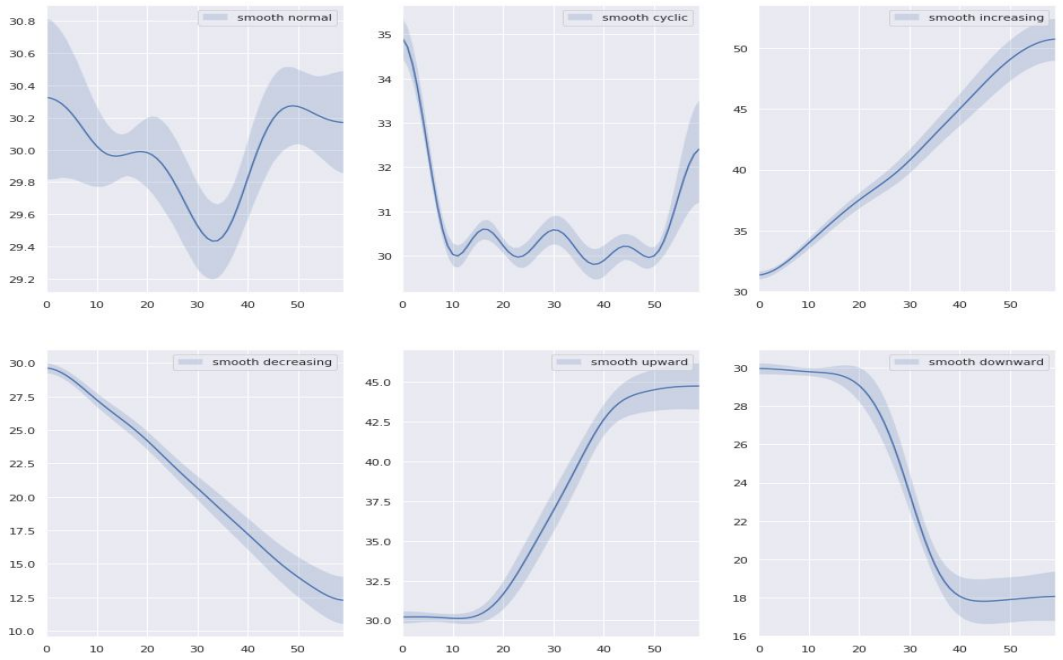
For the sake of brevity, I will take a subset of the whole dataset to check the model. For example let's take top 10 time series of each class and run the algorithm.

The results are shown below:

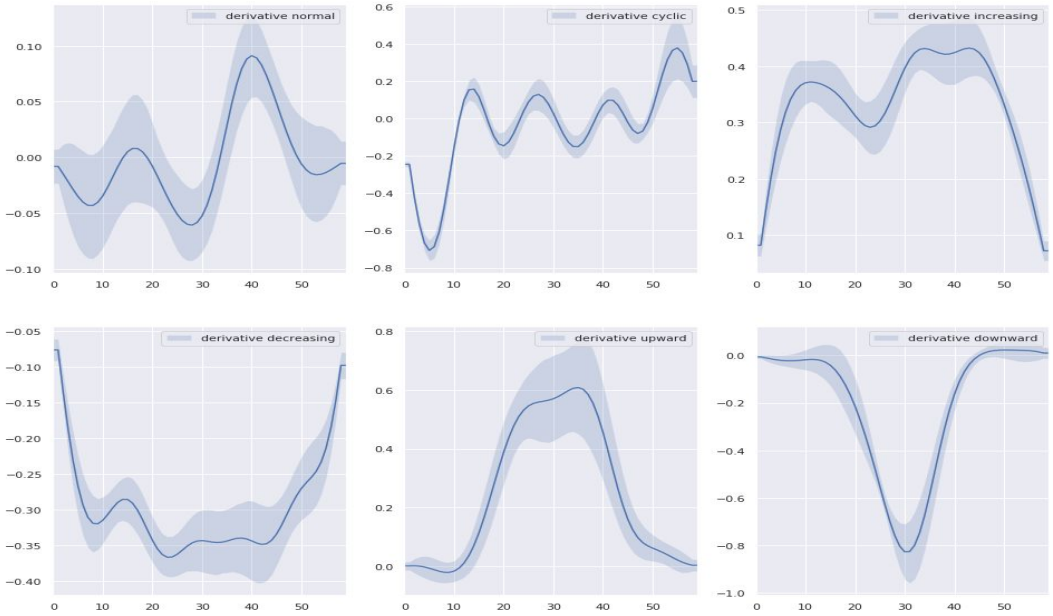
Data



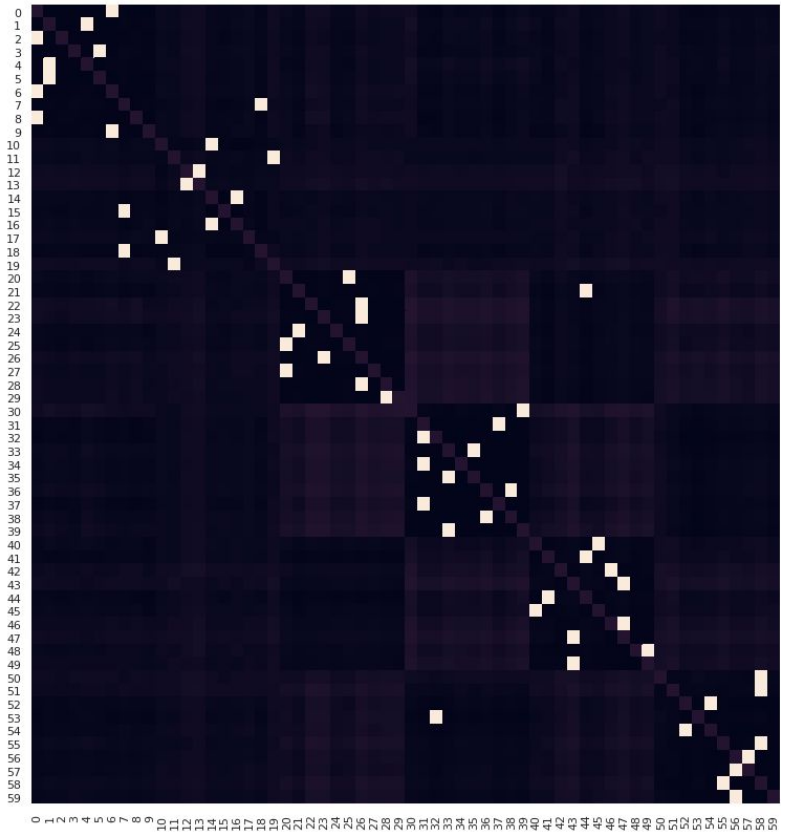

On smoothing:



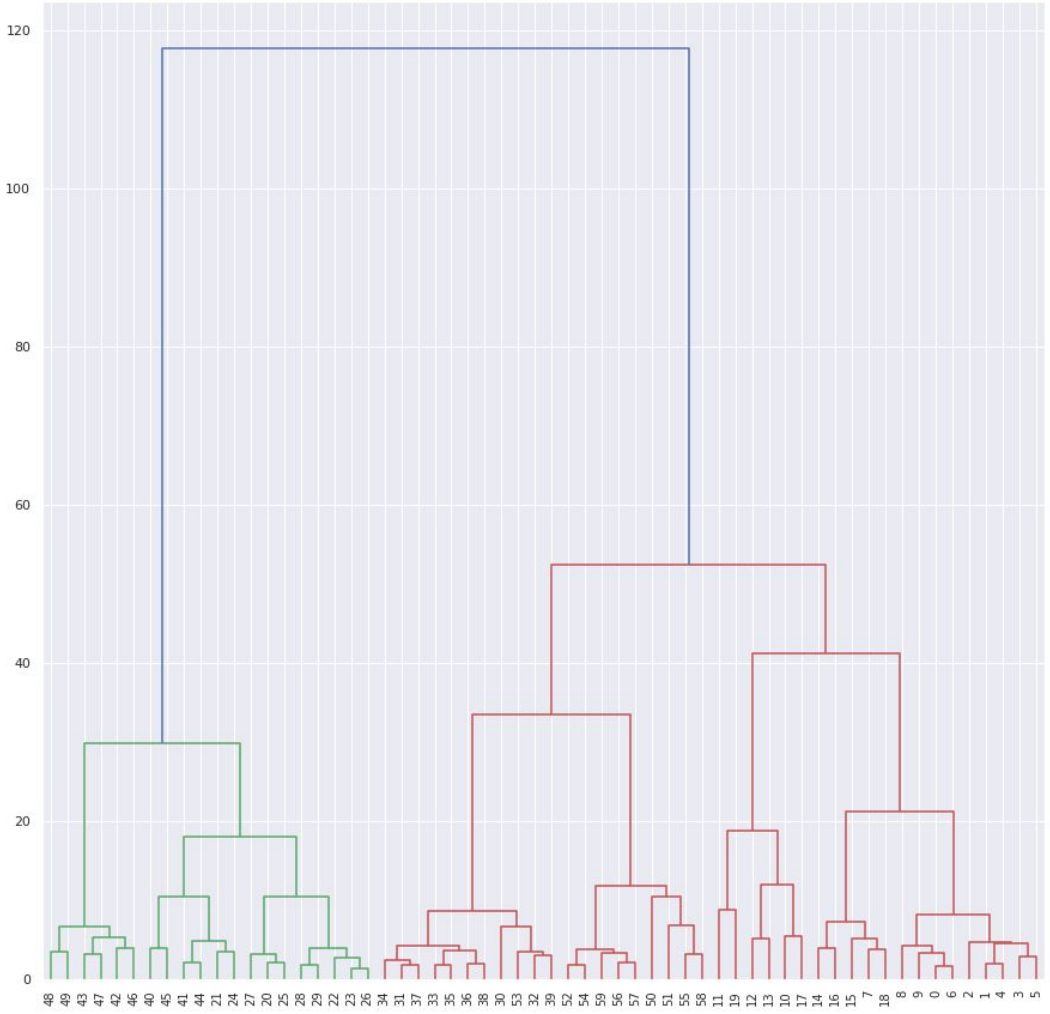
Derivative Sequences:




Distance Matrix:



Dendogram:




# Conclusion

Though there are some misclassifications but the accuracy is quite reasonable. I have tried on larger set size, the results were that the larger the number of the set the more the accuracy. However, after a certain value of the set size the accuracy started converging.

The time series are always a very difficult task to cluster many more methods have been formulated some of them use the advanced concepts of signal processing like Fourier analysis and frequency domain analysis. Here I limited myself to the scope of this course and my knowledge on time series.

---

# References

E. Keogh and M. J. Pazzani. Derivative Dynamic Time Warping. Technical Report.

---

Kaushal Kishore
CSE, B.Tech (Sem-5)
111601008