Mining Time Series Data 1

Acknowledgements

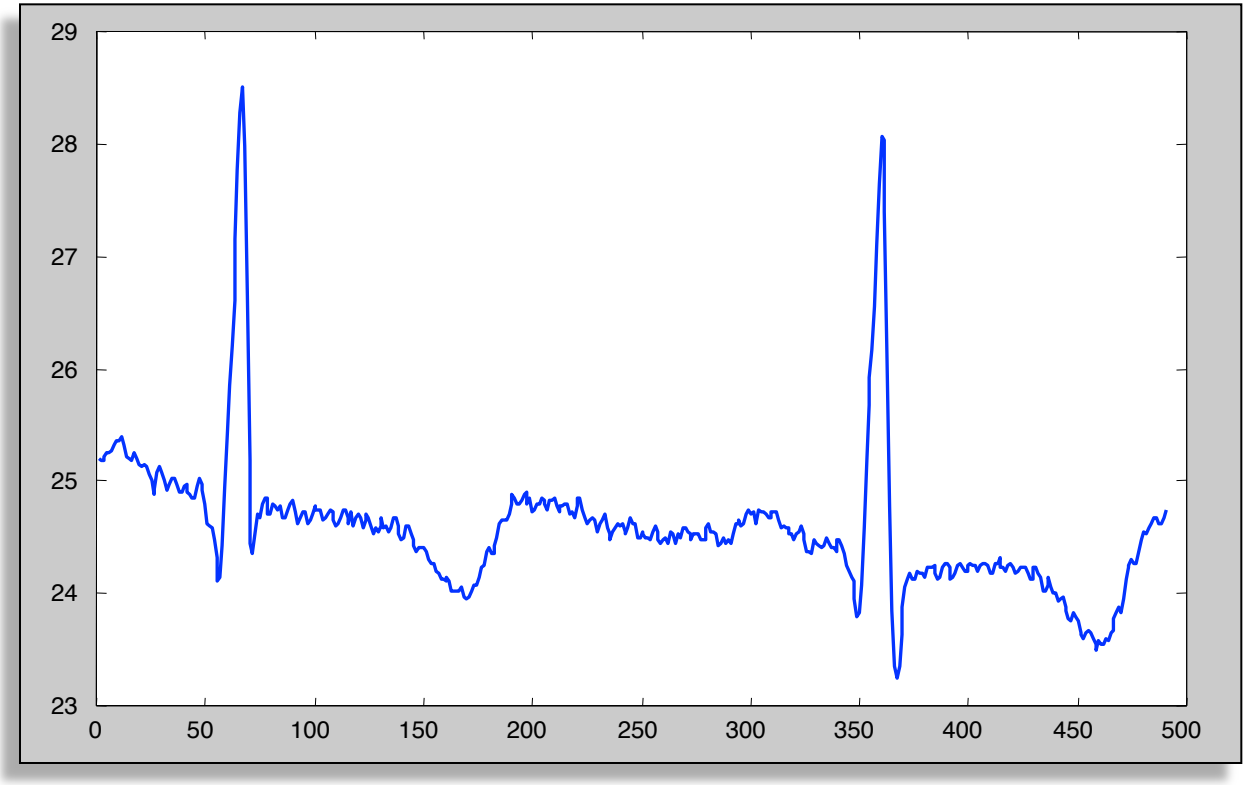
Some slides are provided by

- Eamonn Keogh (various time series data mining tutorials)
- Michail Vlachos (ICDM 2006 tutorial)

25.1750
25.2250
25.2500
25.2500
25.2750
25.3250
25.3500
25.3500
25.4000
25.4000
25.3250
25.2250
25.2000
25.1750
••
••
24.6250
24.6750
24.6750
24.6250
24.6250
24.6250
24.6750
24.7500

What are Time Series?

A time series is a collection of observations made sequentially in time.

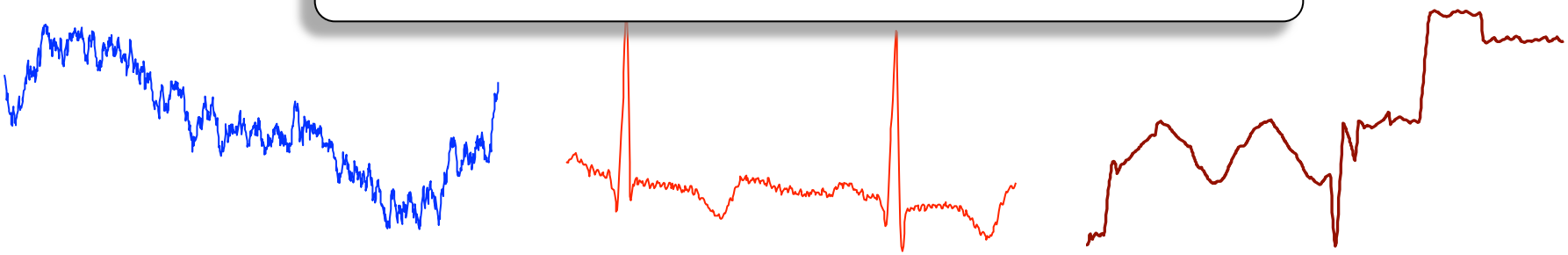


Time Series are Ubiquitous! I

People measure things…

- *Their blood pressure*
- *George Bush's popularity rating*
- *The annual rainfall in Seattle*
- *The value of their Google stock*

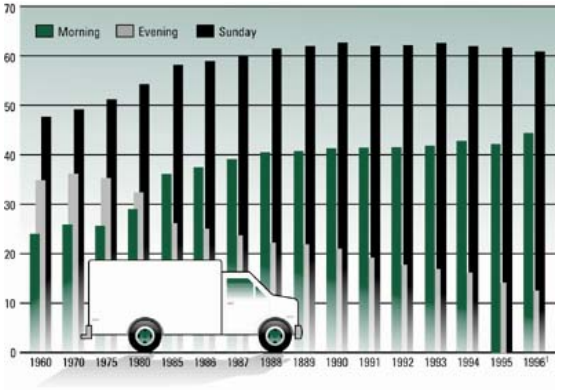
…and things change over time…



Thus time series occur in virtually every medical, scientific and businesses domain

Time Series are Ubiquitous! II

A random sample of 4,000 graphics from 15 of the world's newspapers published from 1974 to 1989 found that more than 75% of all graphics were time series (Tufte, 1992).



Shapes

Recognize type of leaf based on its shape

Ulmus carpinifolia

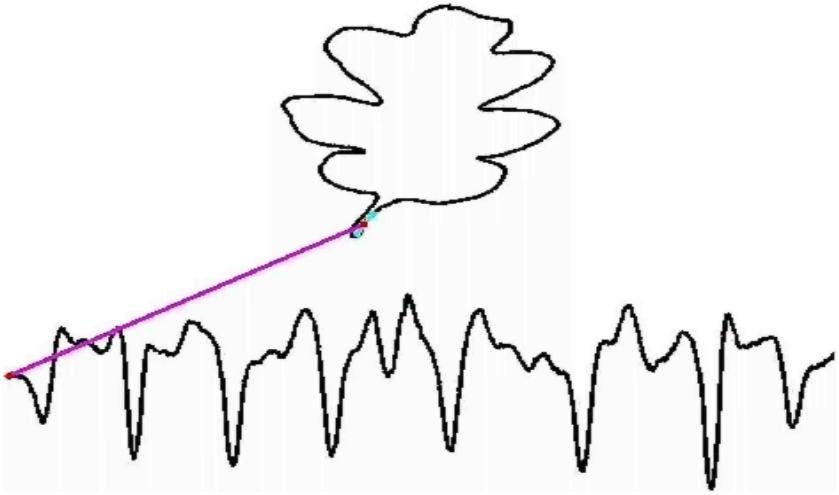
Acer platanoides

Salix fragilis

Tilia

Quercus robur

Convert perimeter into a sequence of values



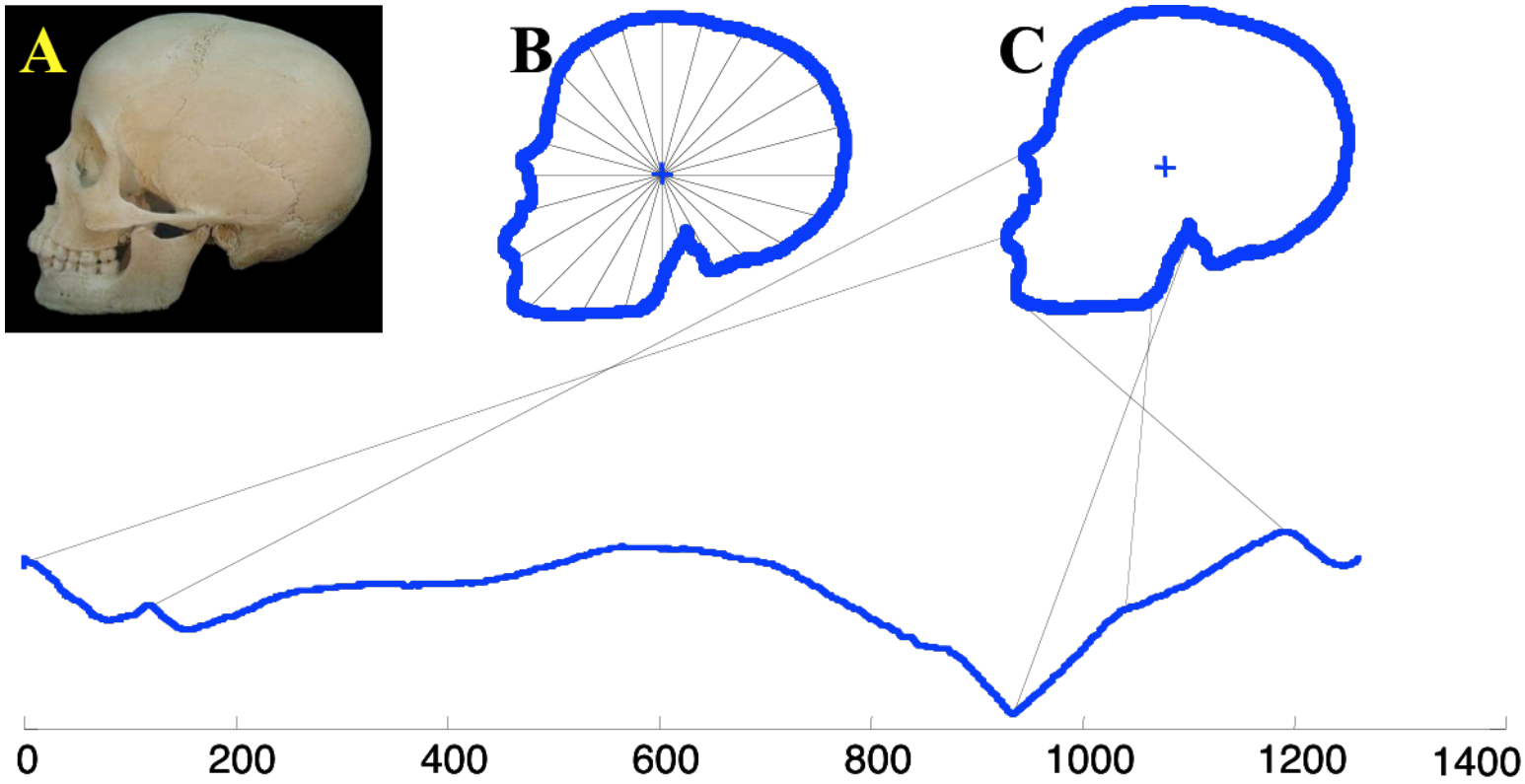
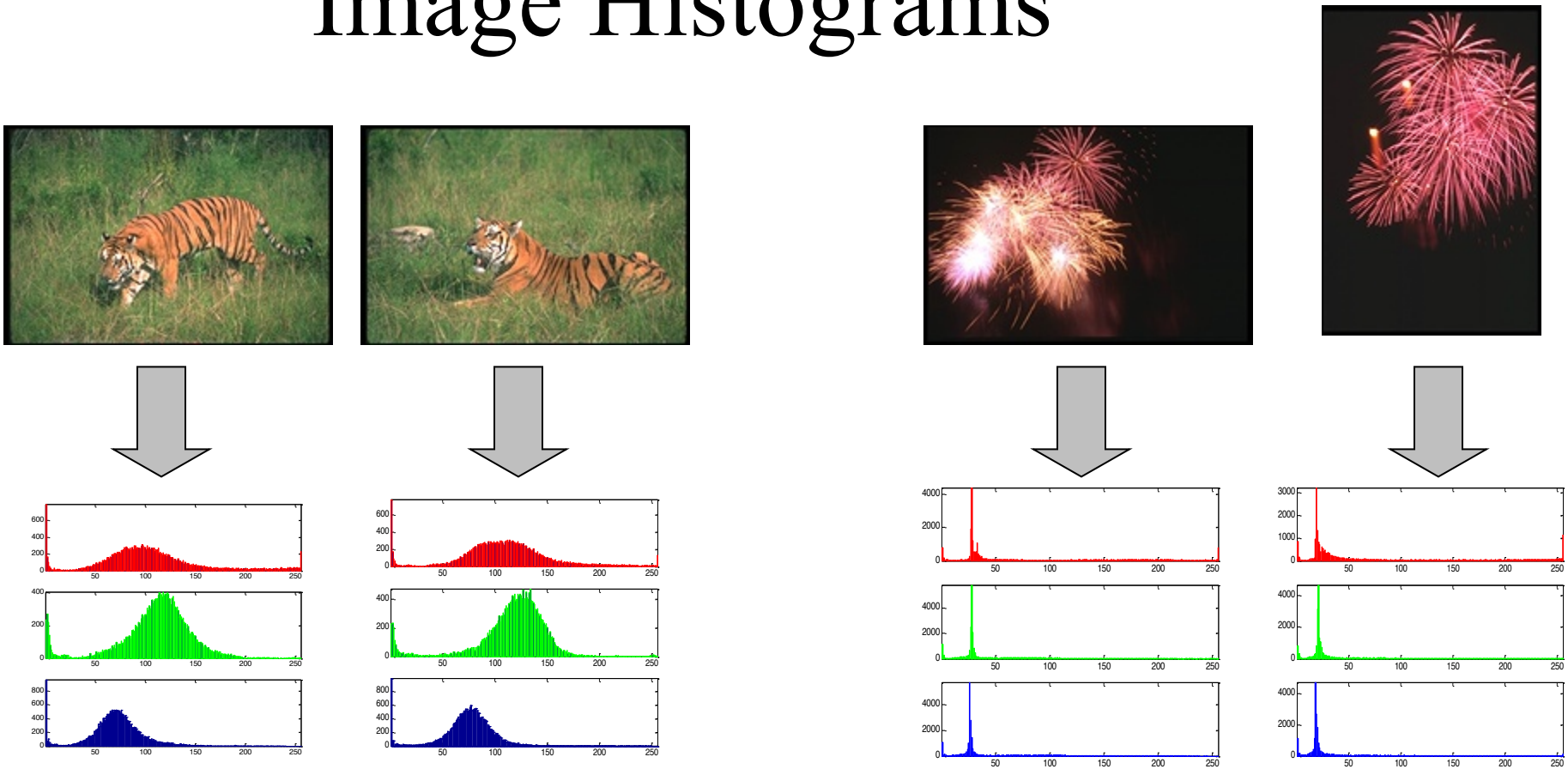

A
B
C
0
200
400
600
800
1000
1200
1400

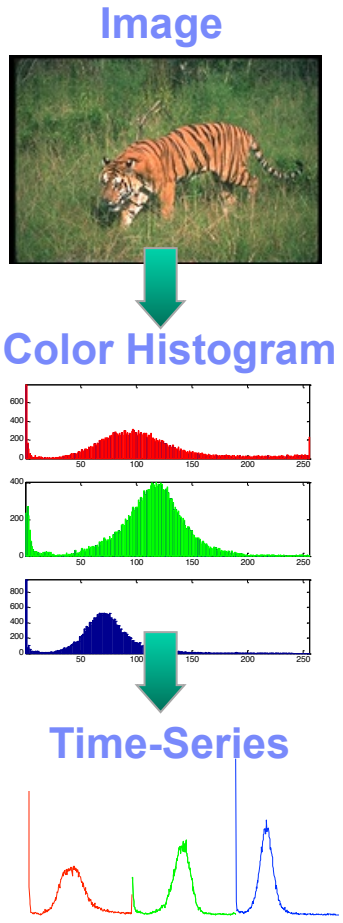
Image Histograms



Convert to Time-Series

Applications (Image Matching)

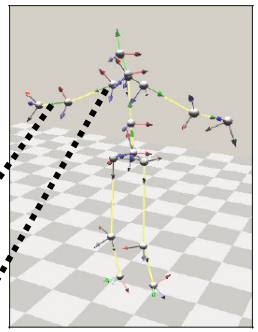
Many types of data can be converted to time-series

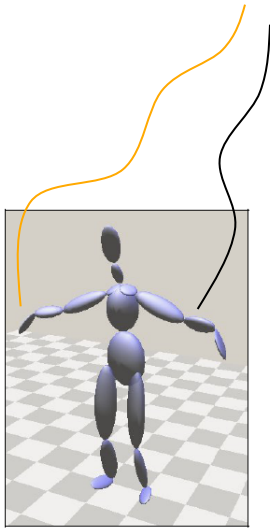


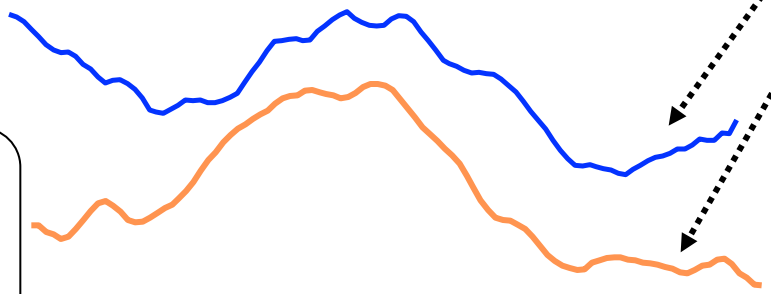
Applications (Motion Capture)

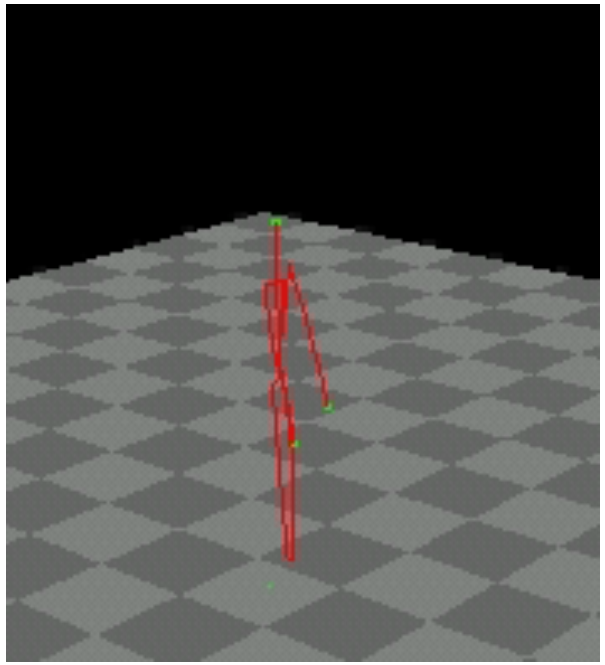
***Motion-Capture (MOCAP) Data* (Movies, Games)**

- Track position of several joints over time
- 3*17 joints = **51 parameters per frame**







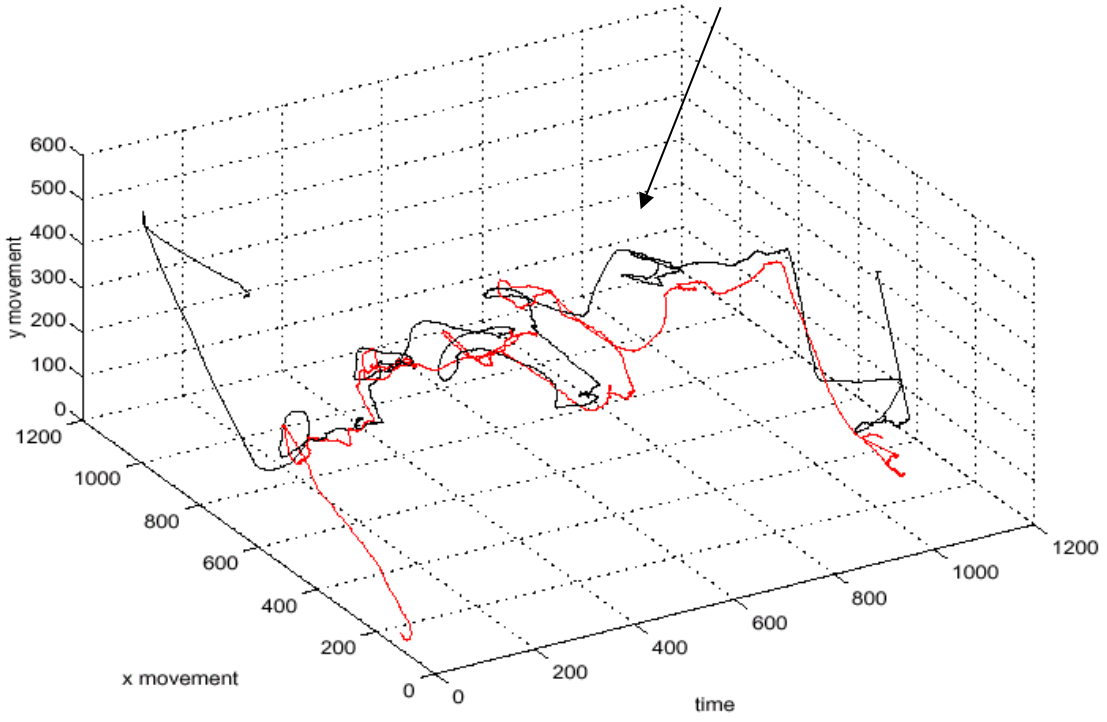


Applications (Video)

Video-tracking / Surveillance

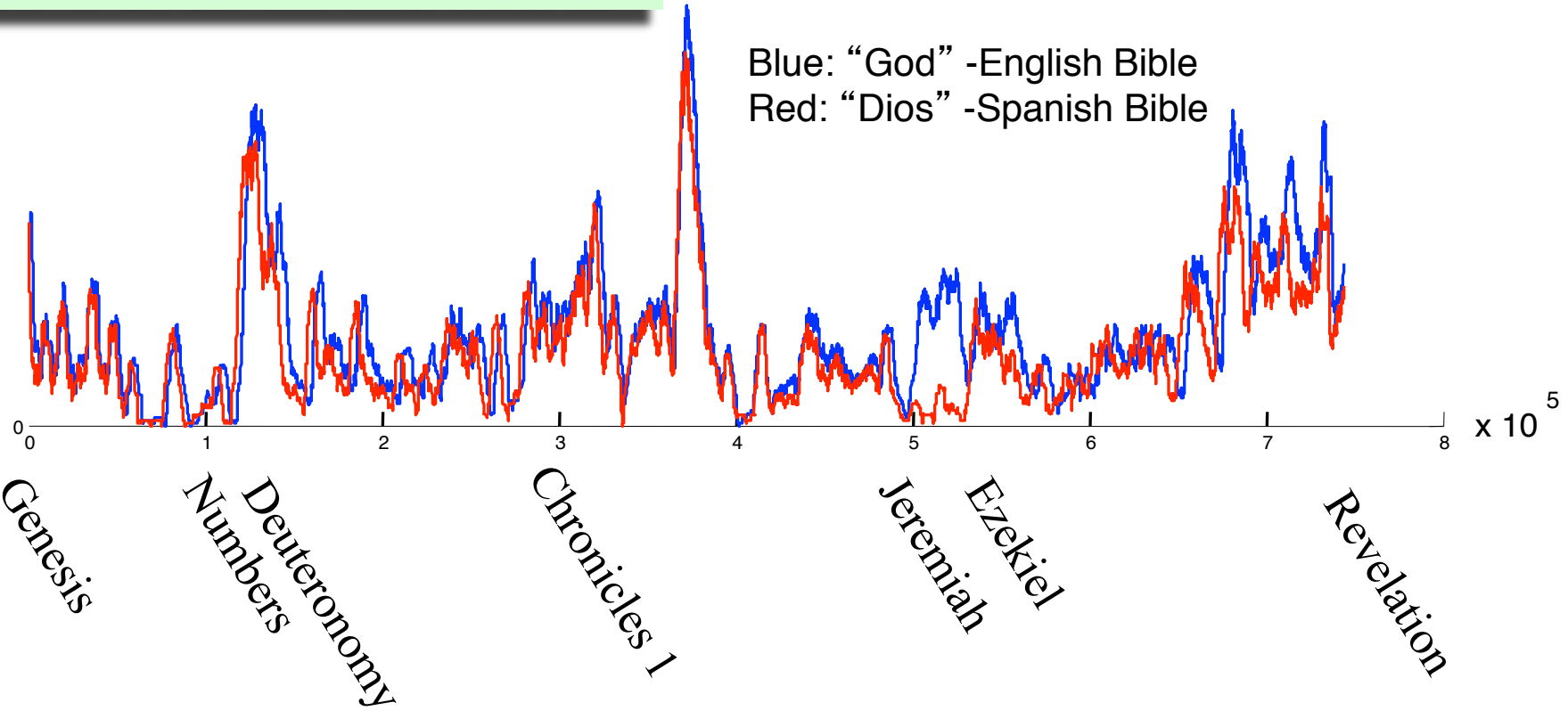
- Visual tracking of body features (2D time-series)
  - Sign Language recognition (3D time-series)

Video Tracking of body feature over time (Athens1, Athens2)



Text data, may best be thought of as time series…

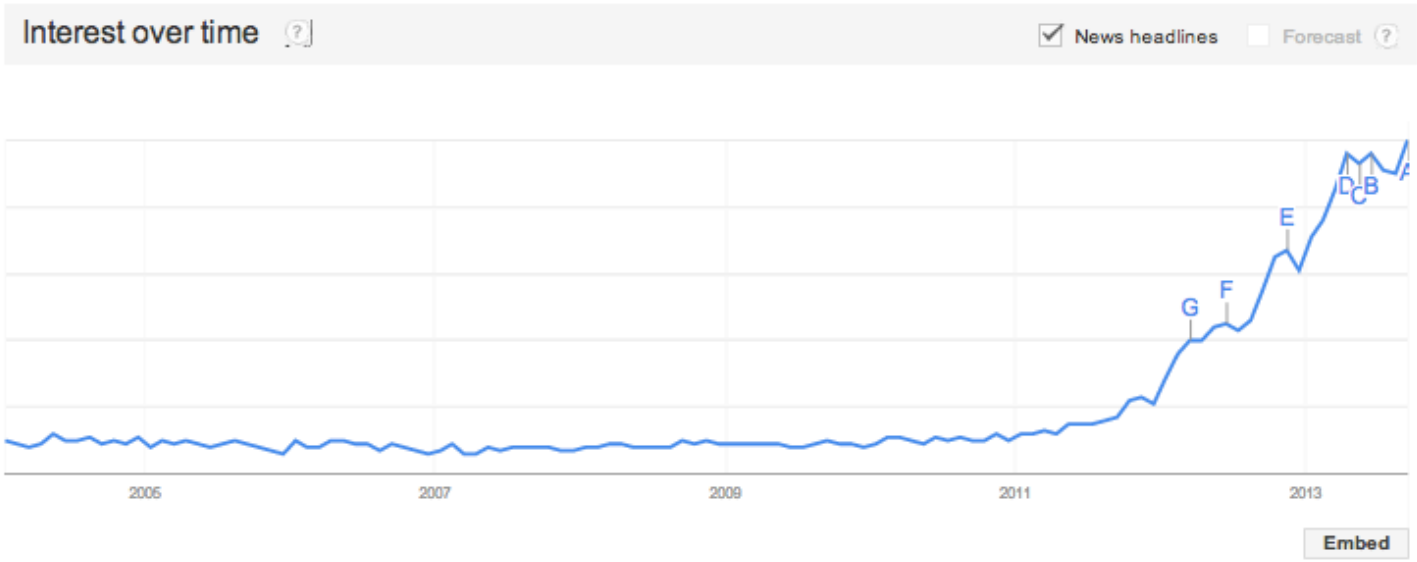
The local frequency of words in the Bible



Gray: “El Senor” -Spanish Bible

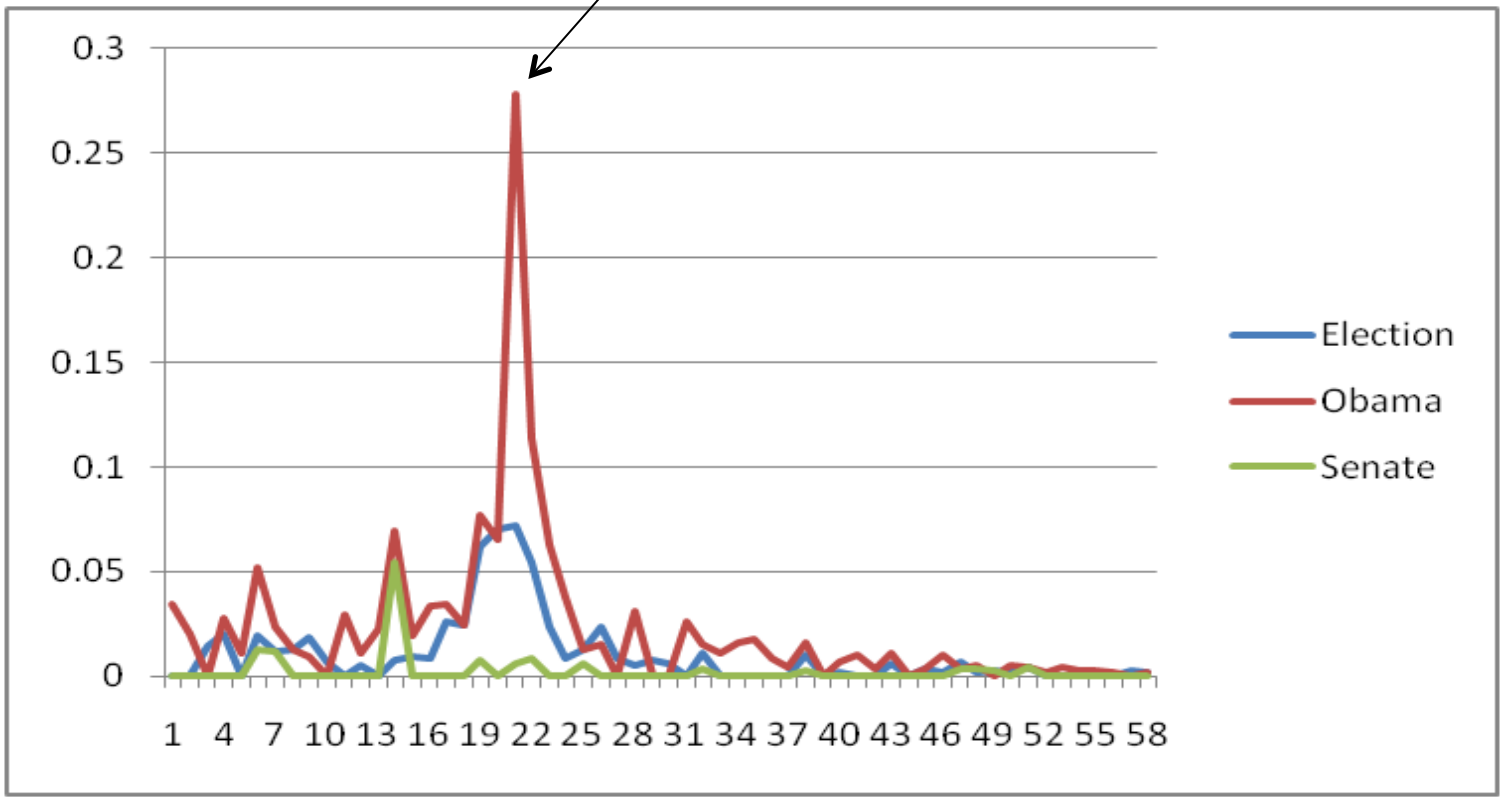
Text Data As Time Series II

"big data"

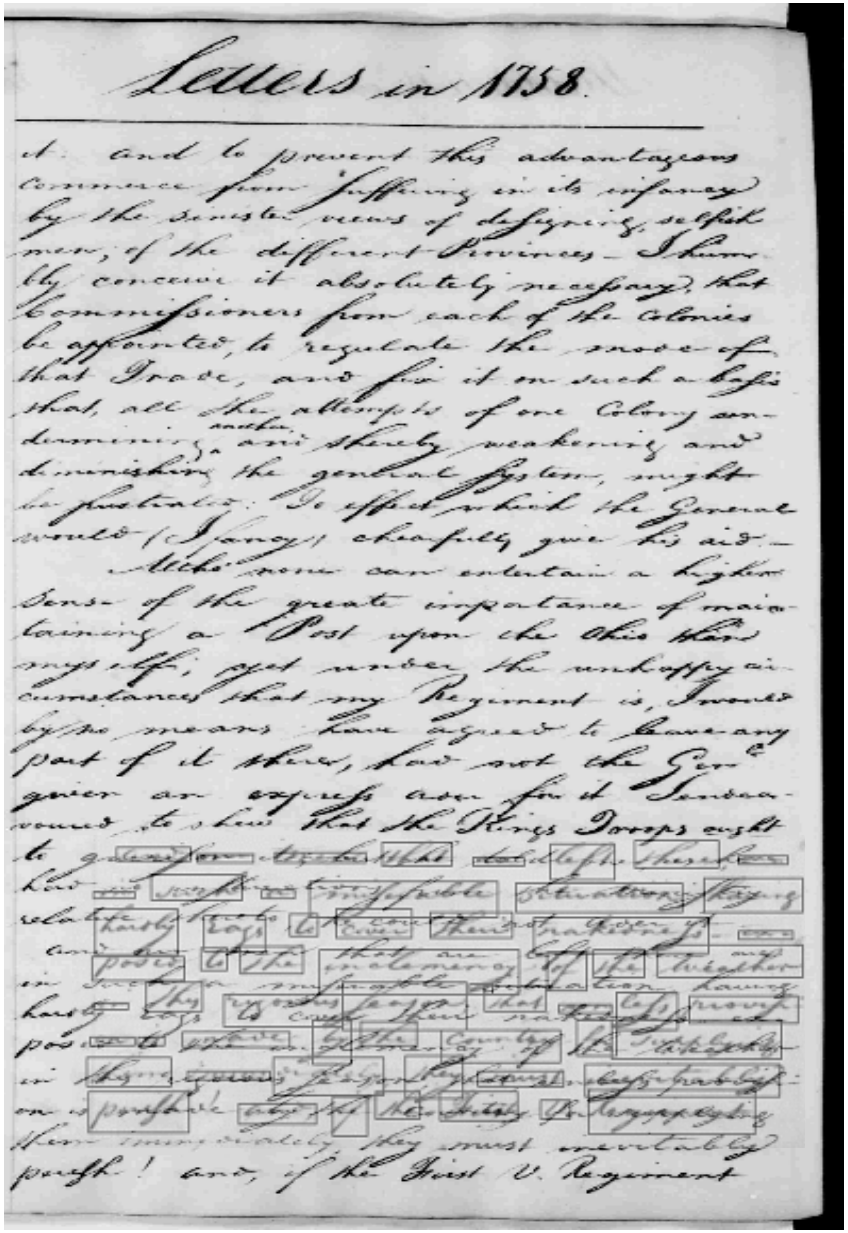


Text Data As Time Series III

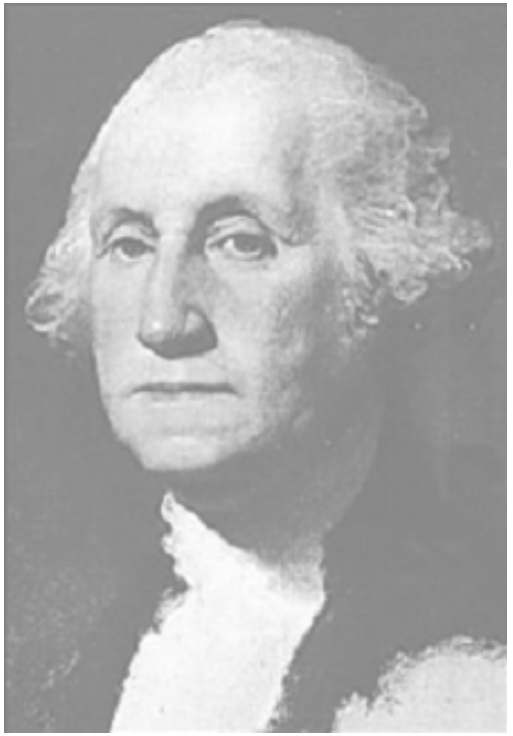
The day after election (11/05/08)



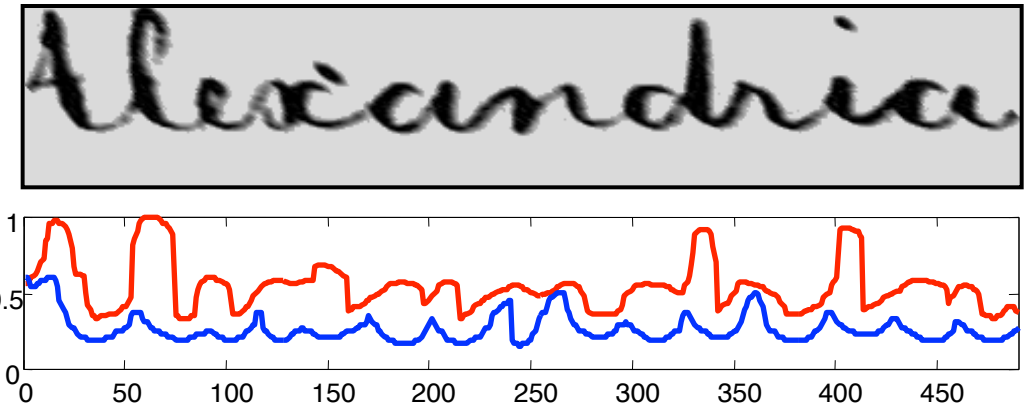
Handwriting data, may best be thought of as time series…



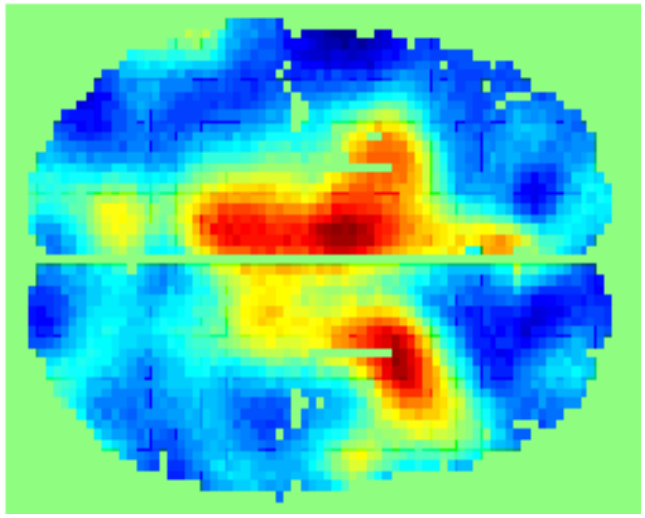
George Washington Manuscript

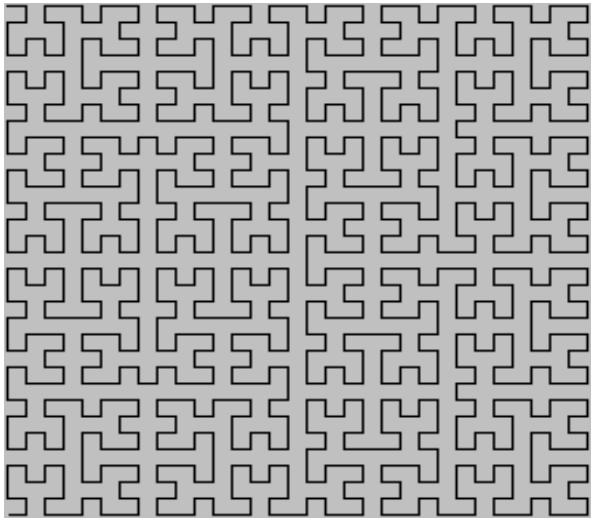


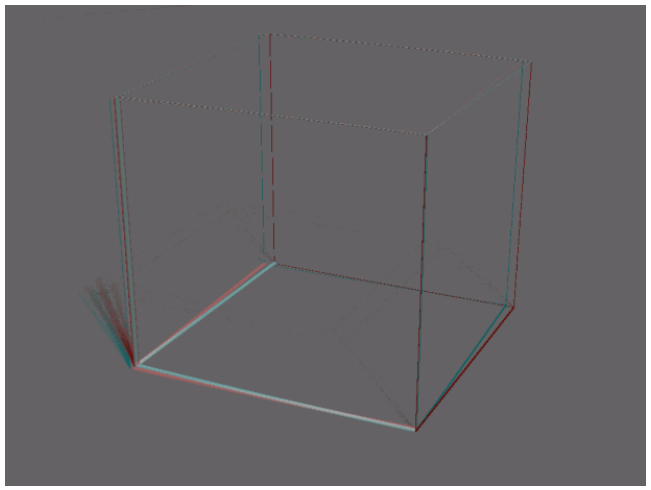
George Washington
1732-1799

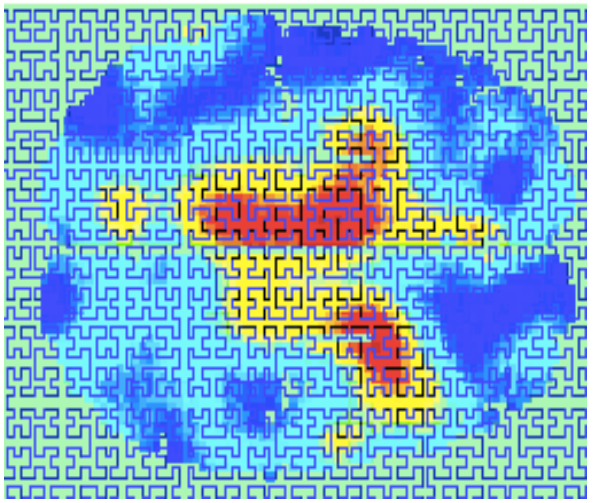


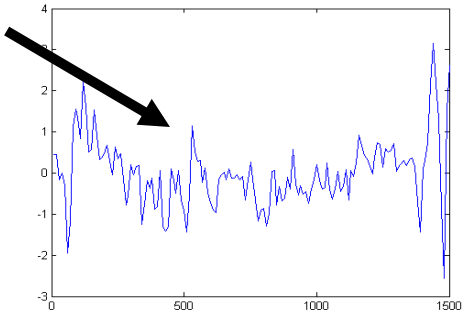
Brain scans (3D voxels), may best be thought of as time series..











Works with
3D glasses!

Wang, Kontos, Li and Megalooikonomou ICASSP 2004

Why is Working With Time Series so Difficult? Part I

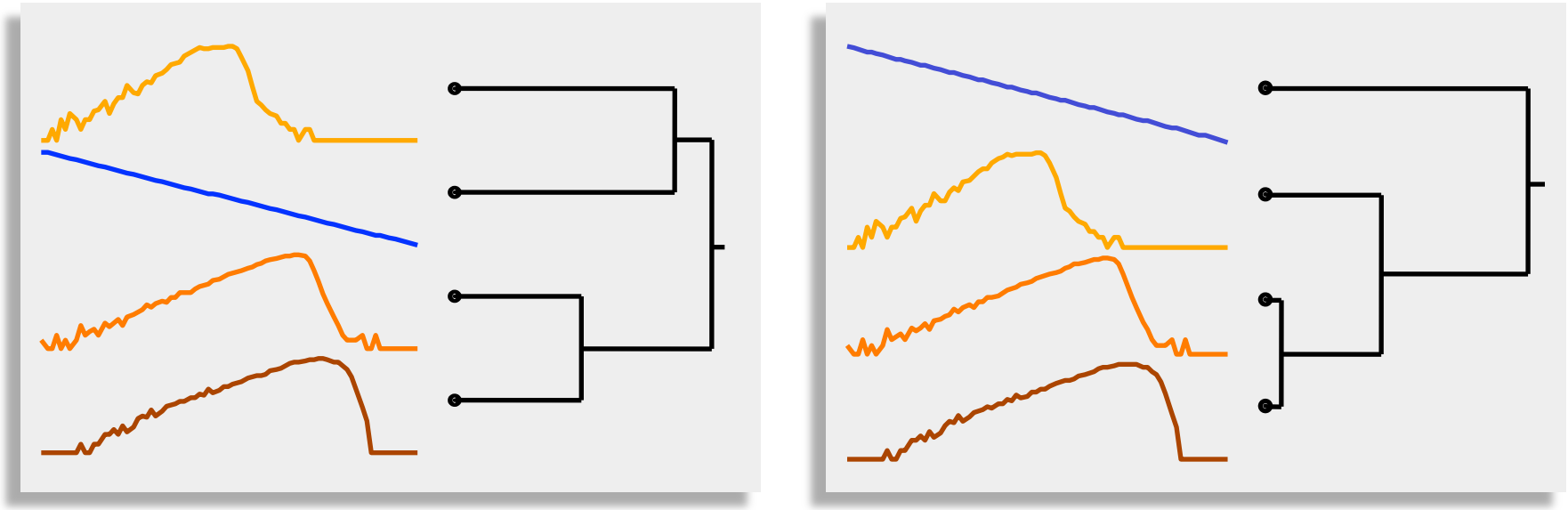
Answer: How do we work with very large databases?

- 1 Hour of ECG data: 1 GB
- Twitter: over 400 million tweets per day (in 2013, compared to 340M in 2012, and 200M in 2011)
- Astronomy Databases:
  - LSST (Large Synoptic Survey Telescope) project: 20 PB science data & 100 PB image archive
- Satellite data
- Sensor data
- Biological data

Since most of the data lives on disk (or tape), we need a representation of the data we can efficiently manipulate.

Why is Working With Time Series so Difficult? Part II

Answer: We are dealing with subjectivity



The definition of similarity depends on the user, the domain and the task at hand. We need to be able to handle this subjectivity.

Why is working with time series so difficult? Part III

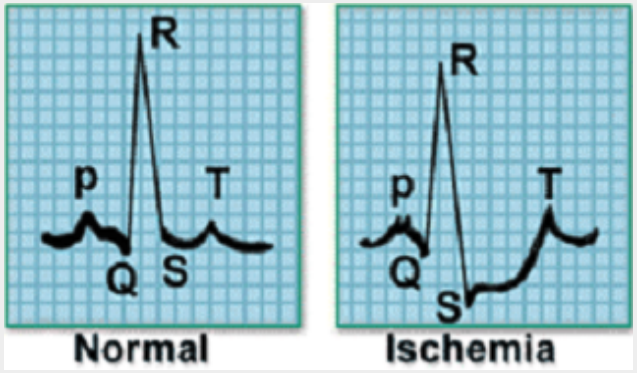
Answer: Miscellaneous data handling problems.

- Differing data formats.
- Differing sampling rates.
- Noise, missing values, etc.

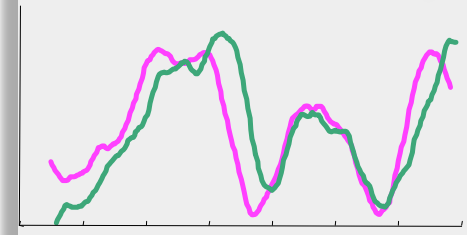
What do we want to do with the time series data?

Clustering

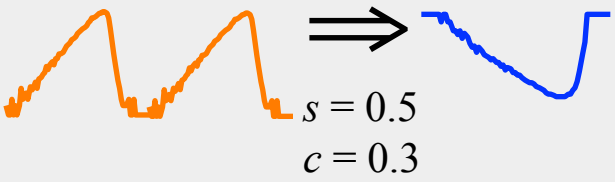
Classification



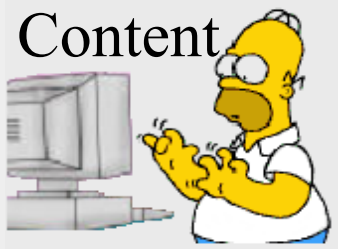
Motif Discovery

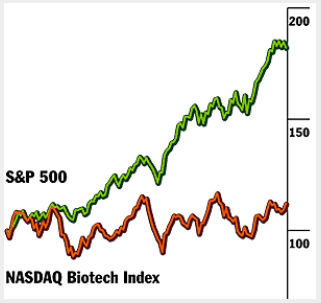


Rule Discovery



Query by Content

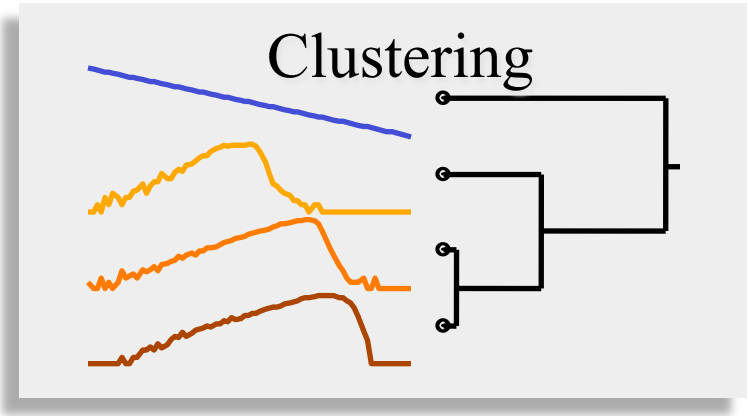


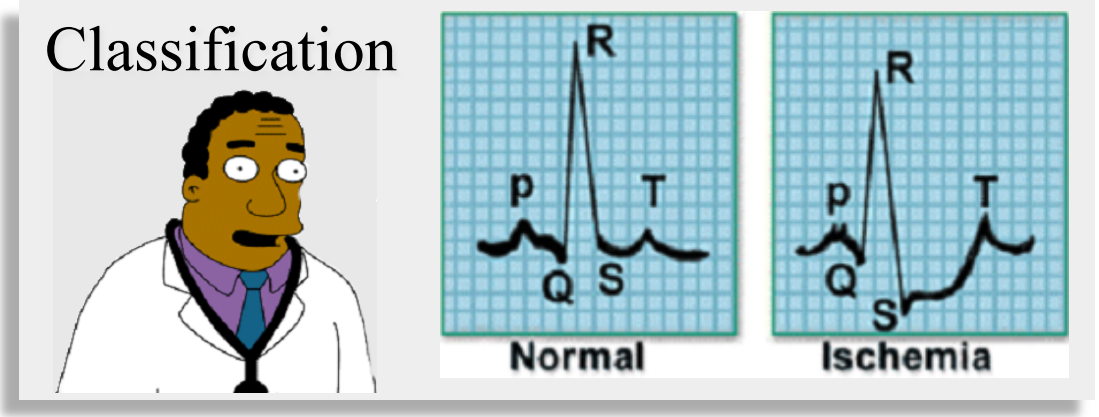


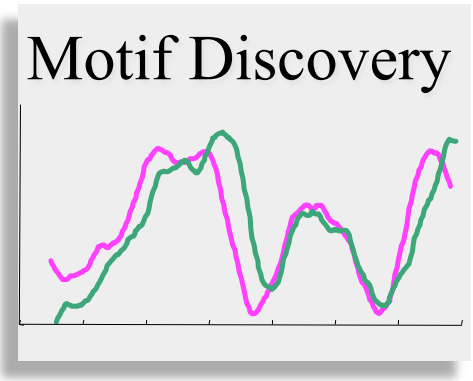
Visualization

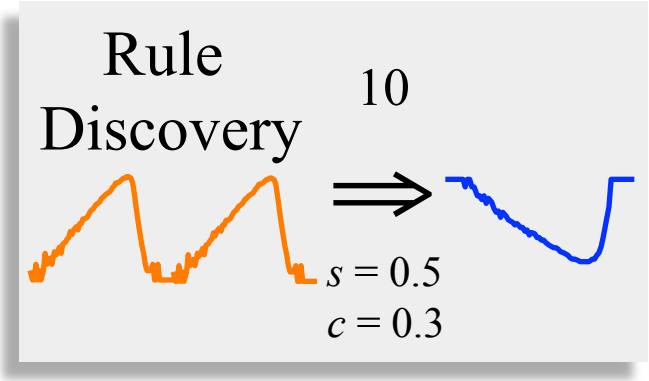
Novelty Detection

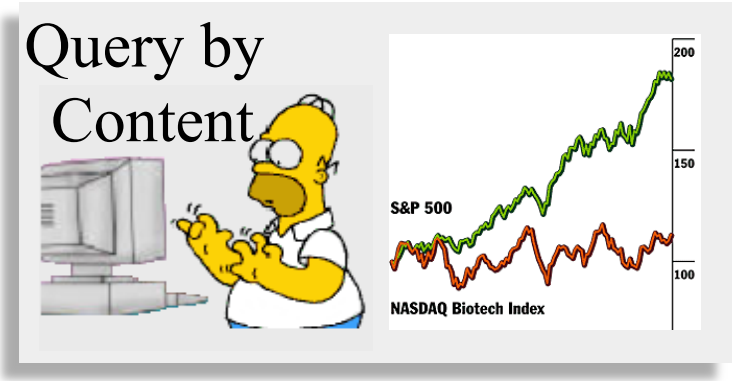
All these problems require similarity matching

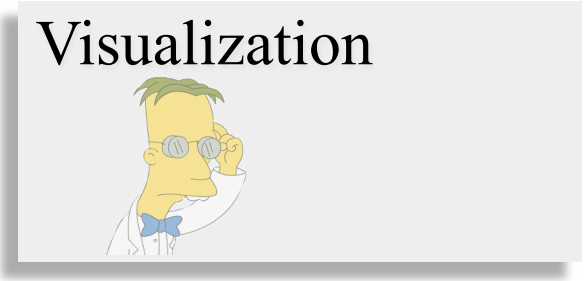


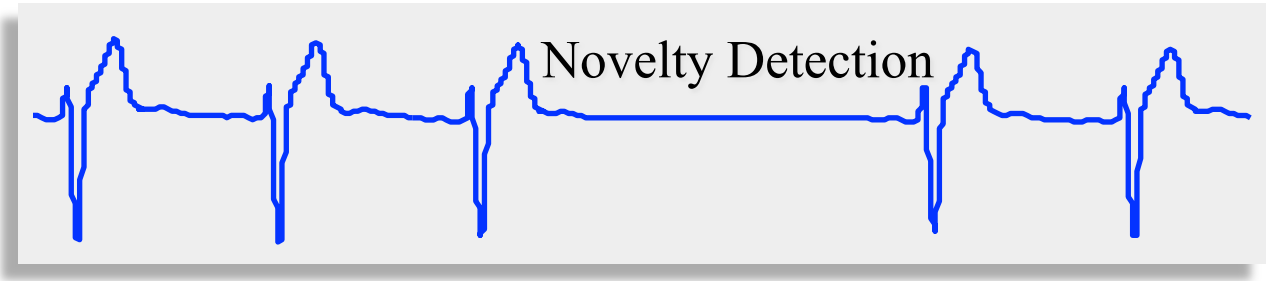




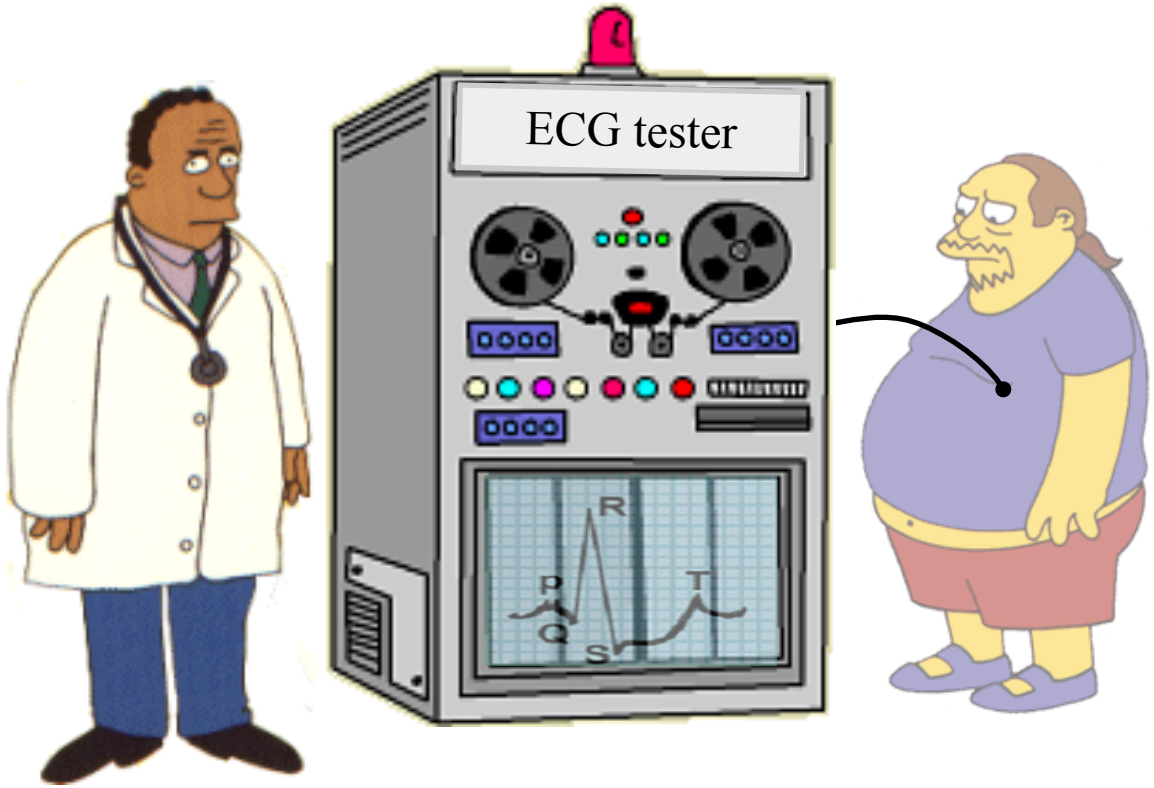








A simple motivation



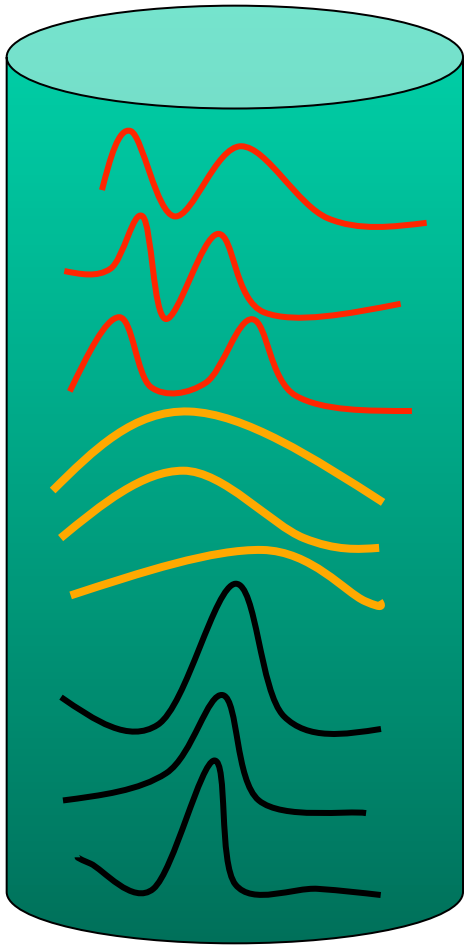
You go to the doctor because of chest pains. Your ECG looks strange…

You doctor wants to search a database to find **similar** ECGs, in the hope that they will offer clues about your condition...

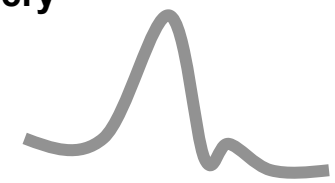
Two questions:

- **How do we define similar?**
- How do we search quickly?

Basic Time-Series Matching Problem



query



Linear Scan:

Objective: ***Compare the query with all sequences in DB and return the k most similar sequences to the query.***

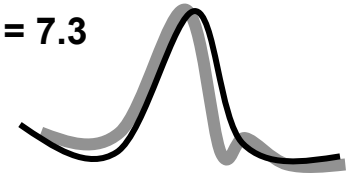
Database with time-series:

- *Medical sequences*
- *Images, etc*

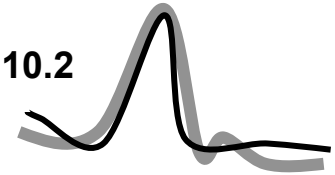
Sequence Length:100-1000pts
DB Size: 1 TByte

Distance

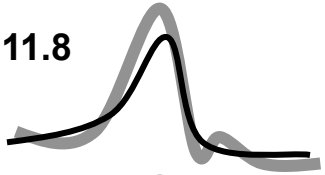
D = 7.3



D = 10.2



D = 11.8

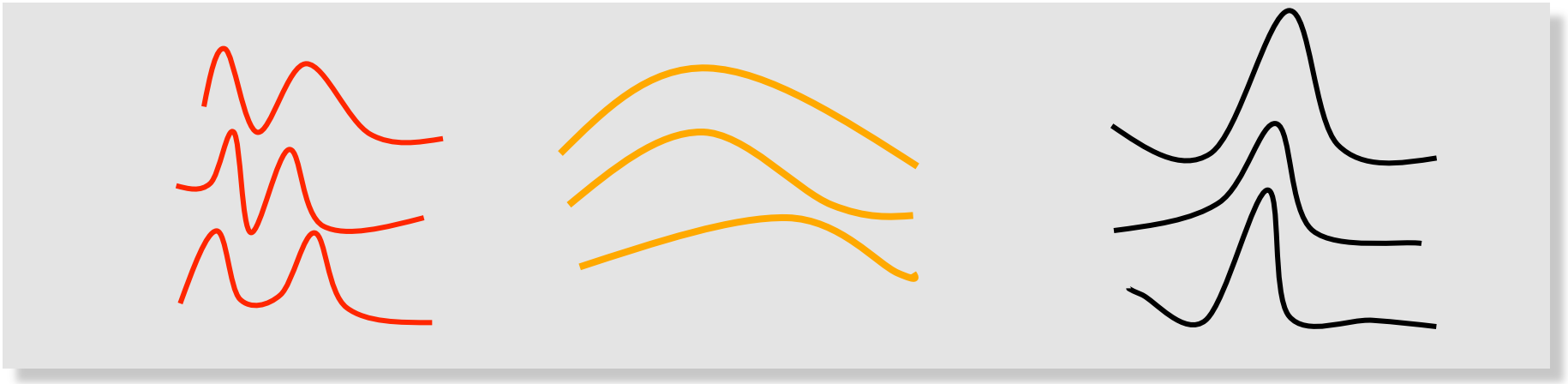


D = 17

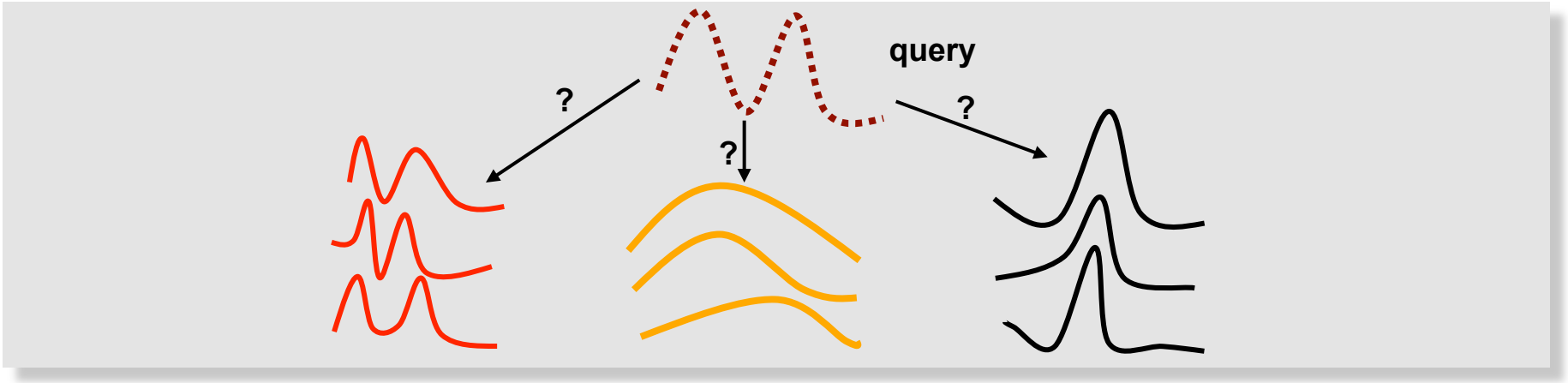
D = 22

What other problems can we solve?

Clustering: "Place time-series into 'similar' groups"



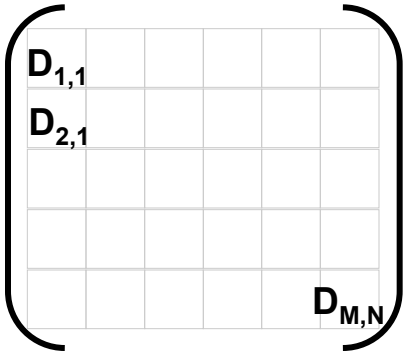
Classification: "To which group is a time-series most 'similar' to?"



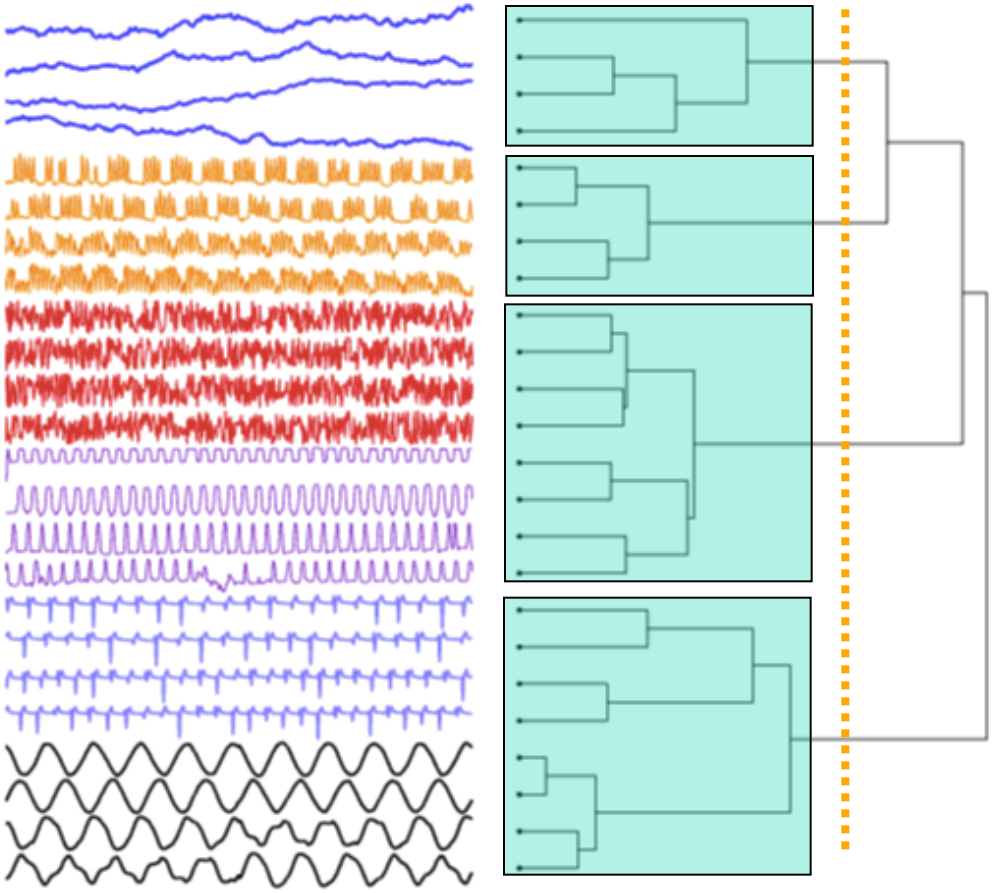
Hierarchical Clustering

- *Very generic & powerful tool*
- *Provides visual data grouping*

Pairwise distances



1. *Merge objects with smallest distance*
2. *Re-evaluate distances*
3. *Repeat process*



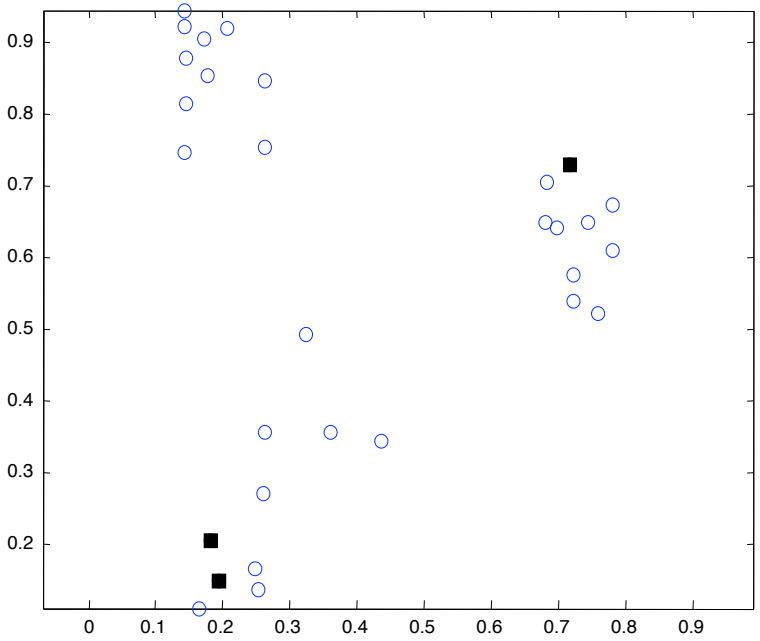
```
Z = linkage(D);
H = dendrogram(Z);
```

Partitional Clustering

- ***Faster than hierarchical clustering***
- ***Typically provides suboptimal solutions (local minima)***
- ***Not good performance for high dimensions***

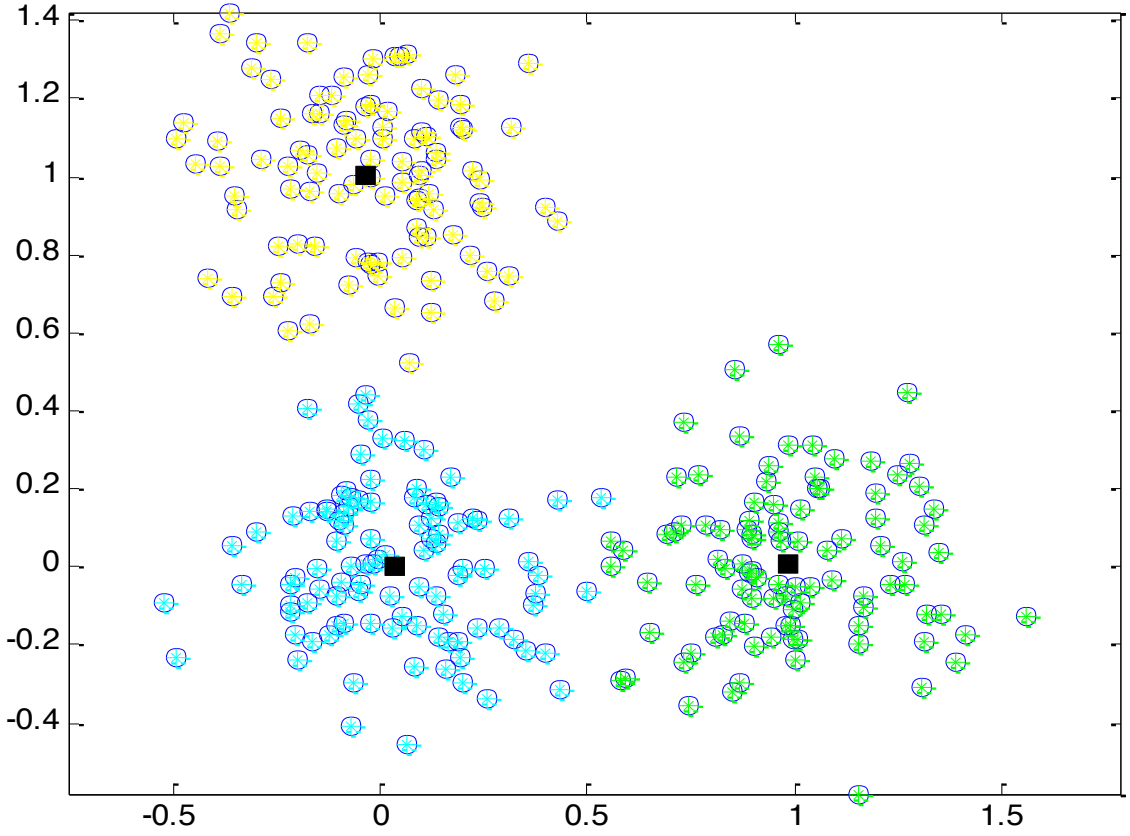
K-Means Algorithm:

1. ***Initialize k clusters (k specified by user) randomly.***
2. ***Repeat until convergence***
   1. *Assign each object to the nearest cluster center.*
   2. *Re-estimate cluster centers.*



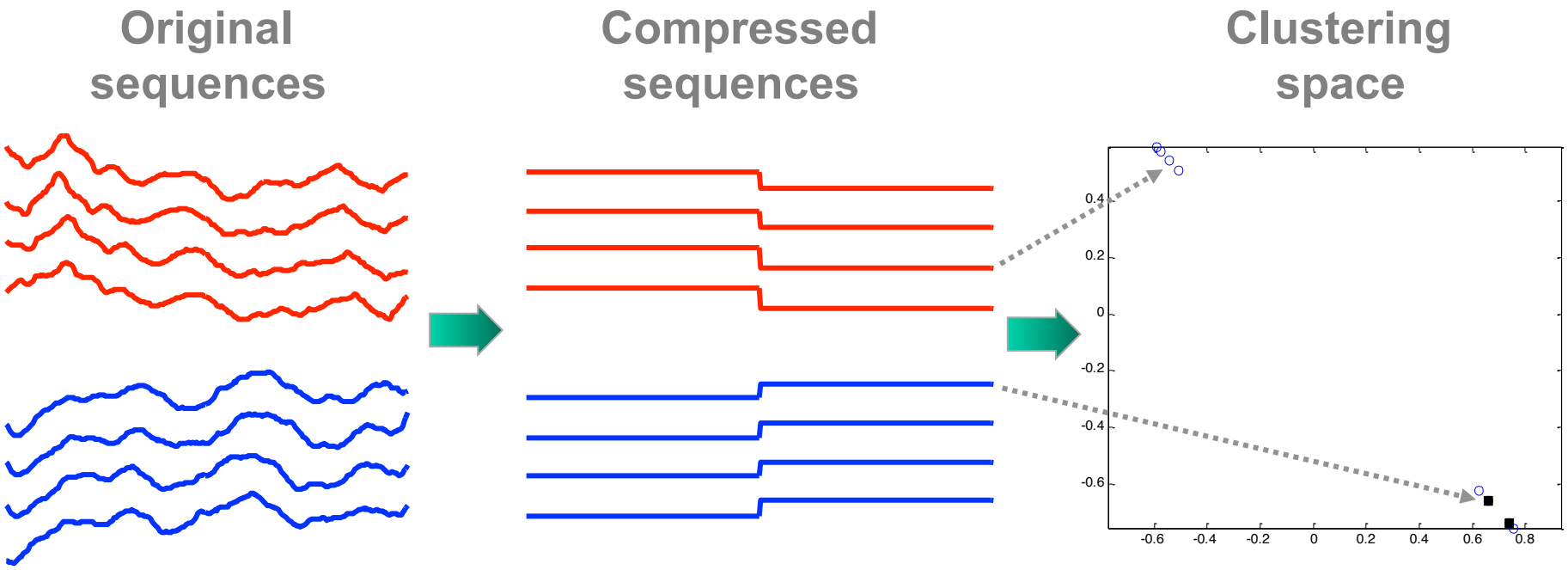
See: kmeans

K-Means Demo



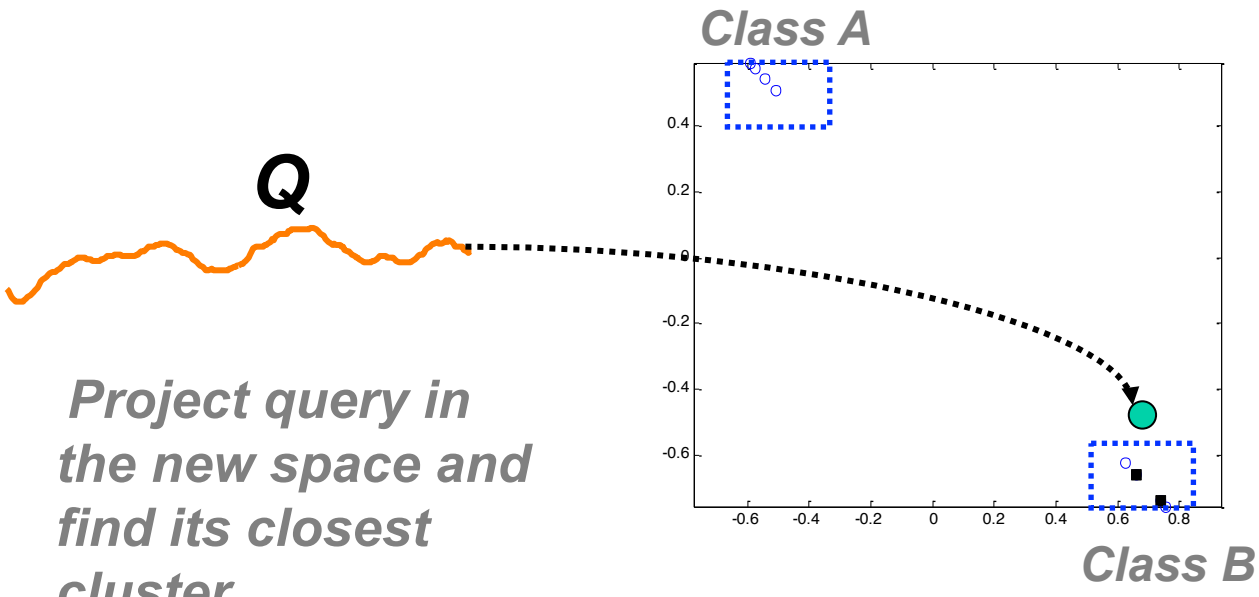
K-Means Clustering for Time-Series

- ***So how is kMeans applied for Time-Series that are high-dimensional?***
- ***Perform kMeans on a compressed dimensionality***



Classification

Typically classification can be made easier if we have clustered the objects

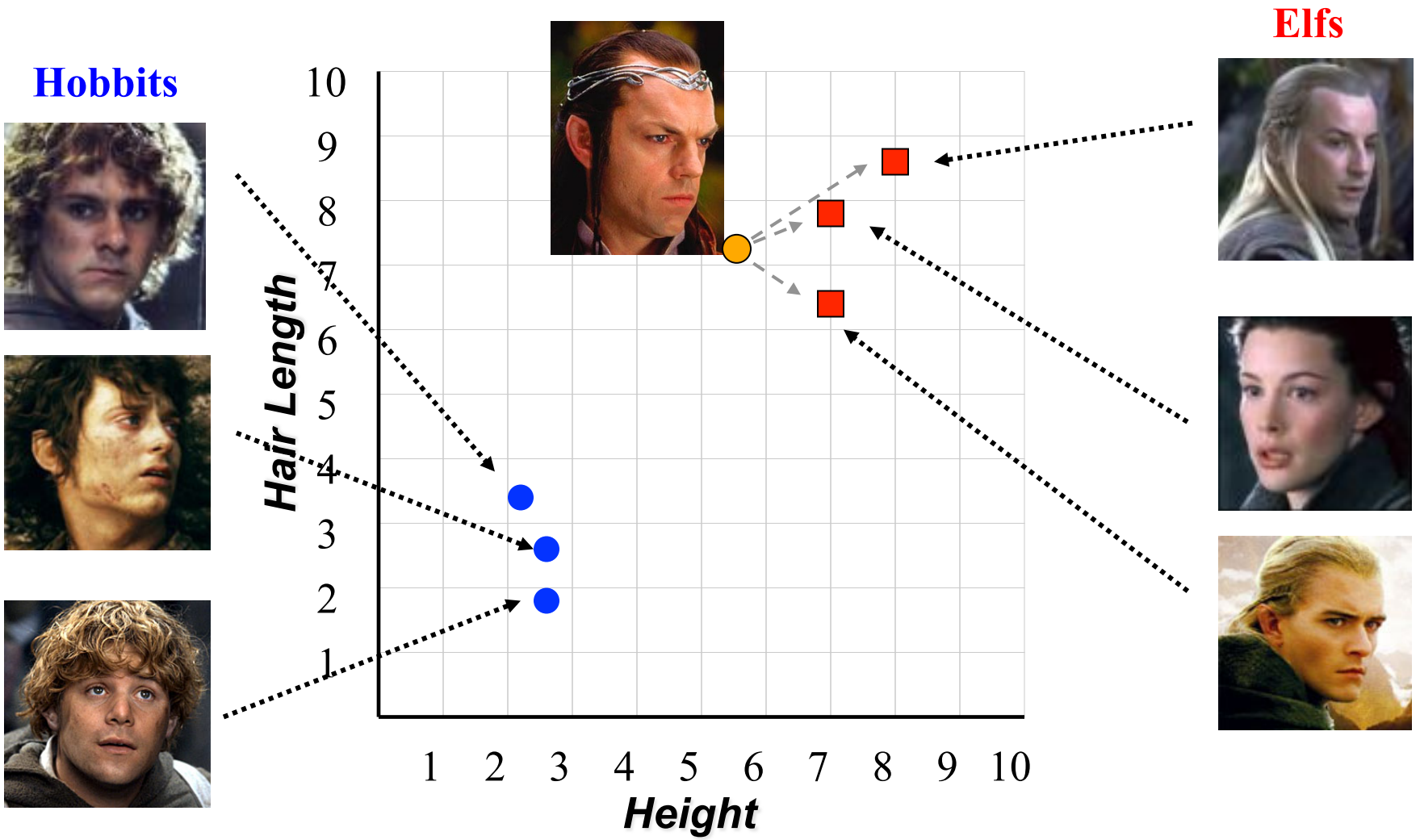


Project query in the new space and find its closest cluster

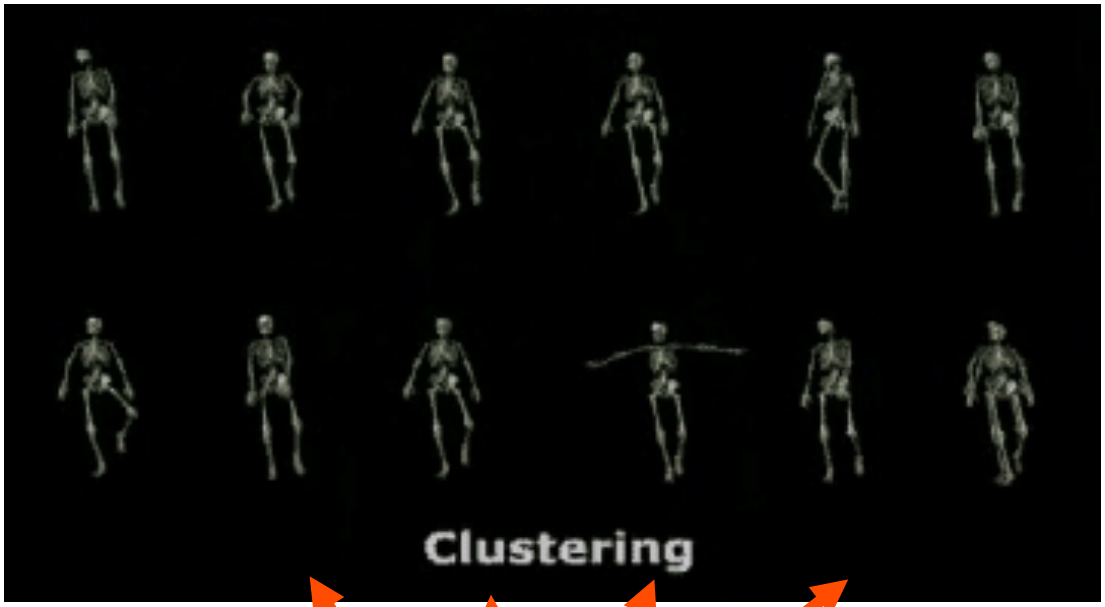
So, query Q is more similar to class B

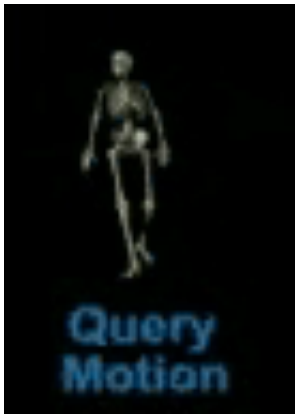
Nearest Neighbor Classification

We need not perform clustering before classification. We can classify an object based on the class majority of its nearest neighbors/matches.



Example





What do we need?

1. Define Similarity

2. Search fast

- Dimensionality Reduction (compress data)

What is Similarity?

The quality or state of being similar; likeness; resemblance; as, a similarity of features. **Webster's Dictionary**

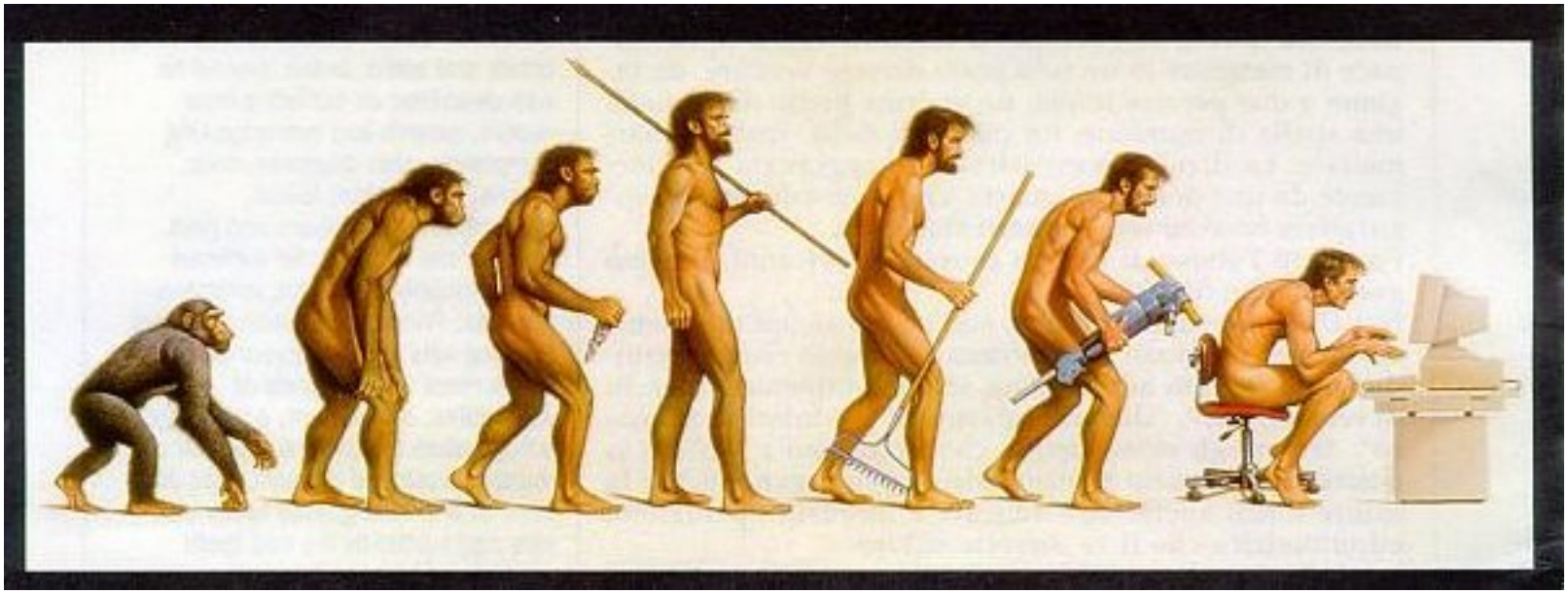
Similarity is hard to define, but…
"*We know it when we see it*"

The real meaning of similarity is a philosophical question.

We will take a more pragmatic approach.

Notion of Similarity I

- ***Solution to any time-series problem, boils down to a proper definition of *similarity****



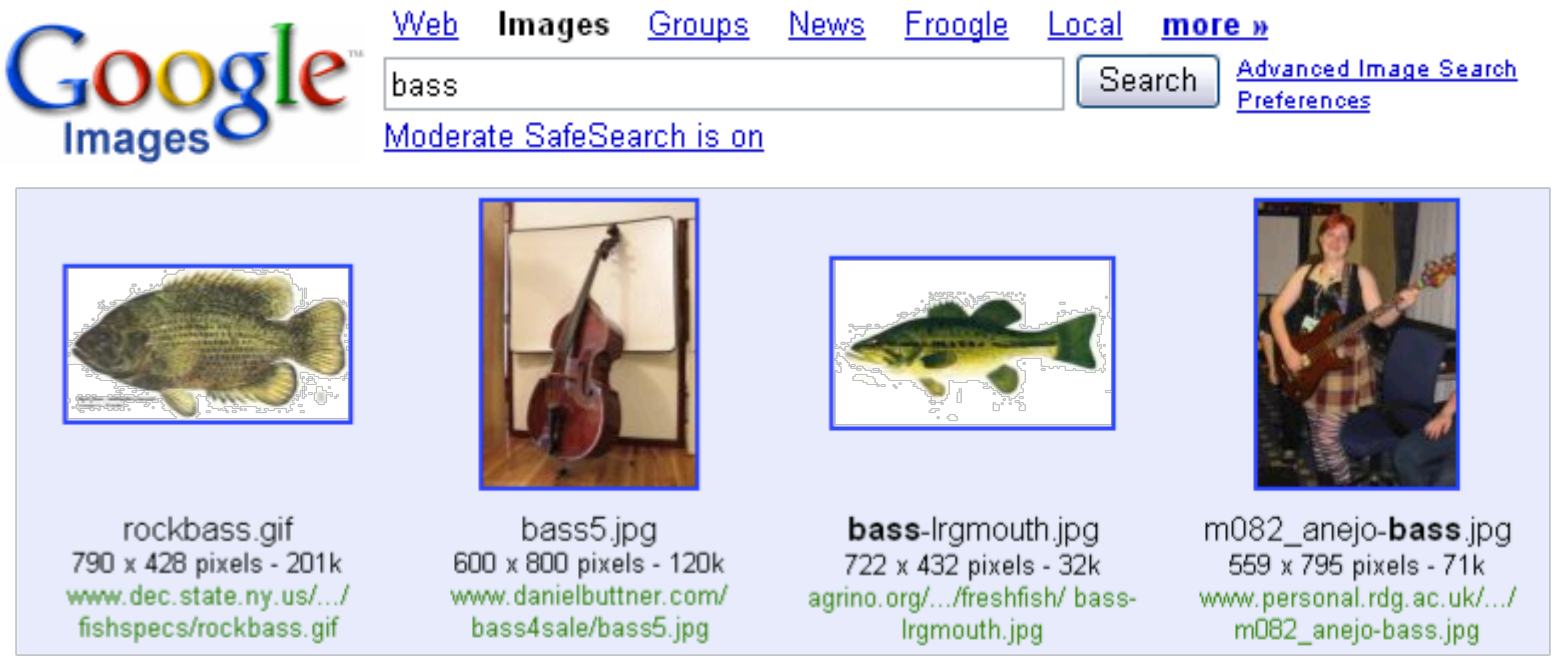
Similarity is always subjective.

(i.e. it depends on the application)

Notion of Similarity II

Similarity depends on the features we consider

(i.e. how we will describe or compress the sequences)

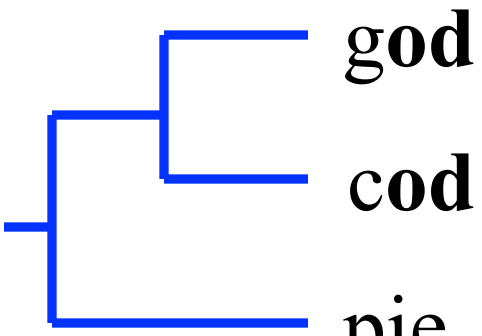


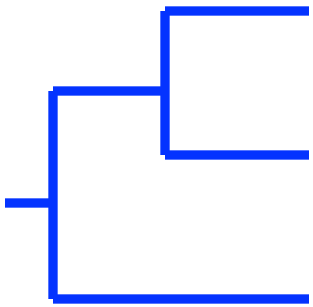
Two Kinds of Similarity

text

Similarity at the level of *individual* characters

Similarity at the *structural* level





SLY I'll pheeze you, in faith. **Hostess** A pair of stocks, you ro

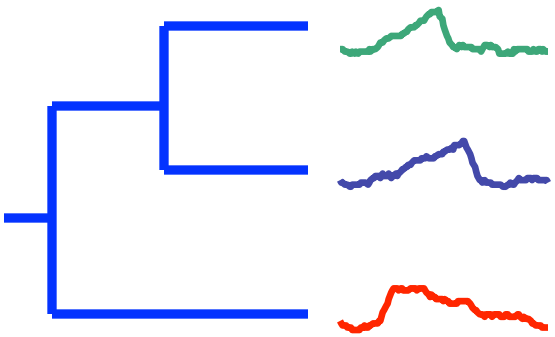
VALENTINE Cease to persuade, my loving Proteus:Home-k

In the beginning God created the heavens and the earth. The e

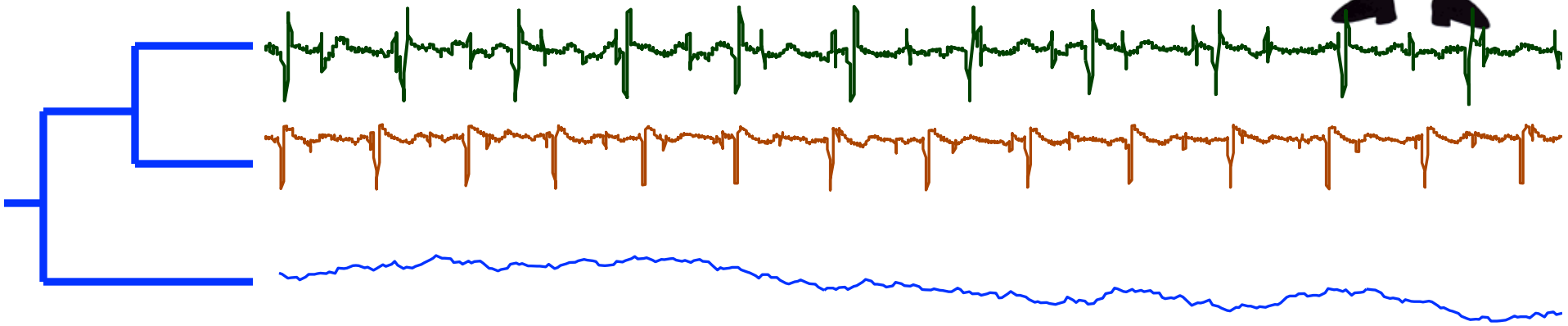
Two Kinds of Similarity

time series

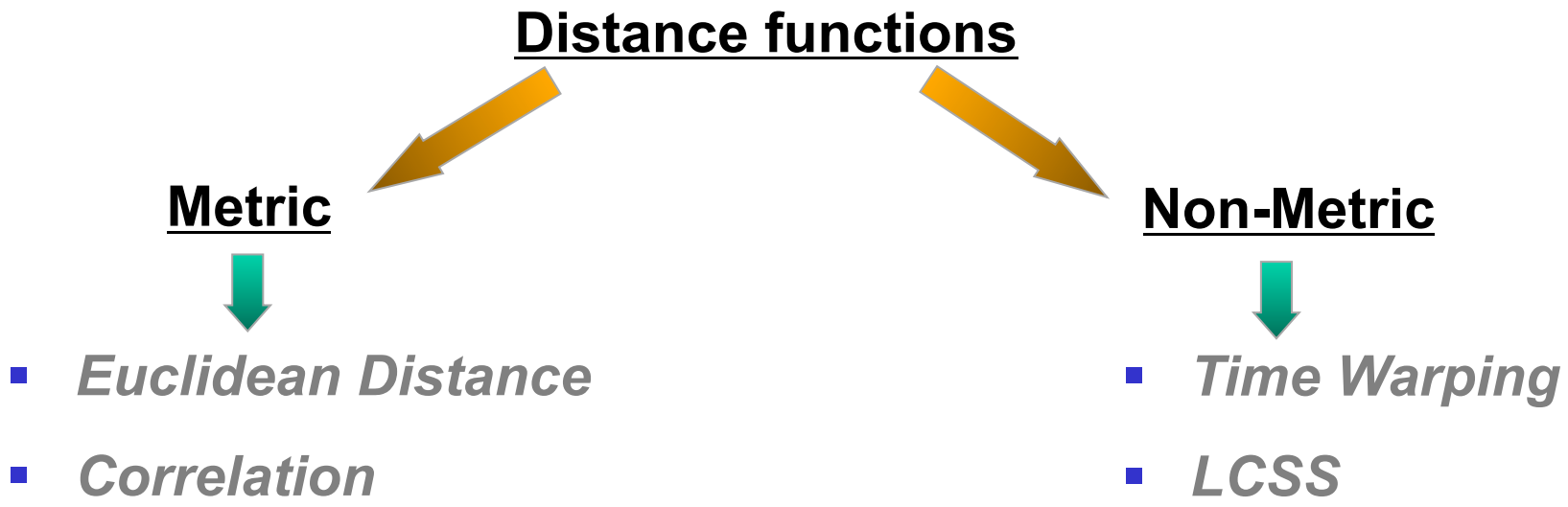
Similarity at the level of *shape*



Similarity at the *structural* level



Metric and Non-metric Distance Functions



Properties

Positivity and Constancy:

$d(x,y) \geq 0$ and $d(x,y)=0$, if $x=y$

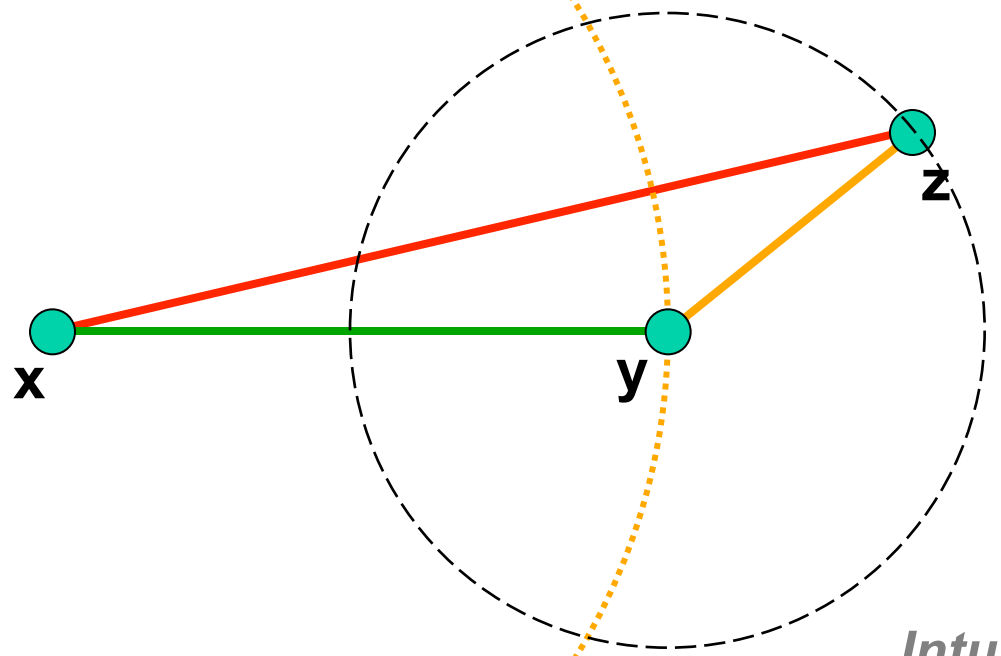
Symmetry: $d(x,y) = d(y,x)$

Triangle Inequality: $d(x,z) \leq d(x,y) + d(y,z)$

If any of these is not obeyed then the distance is a non-metric

Triangle Inequality

Triangle Inequality: **d(x,z) ≤ d(x,y) + d(y,z)**

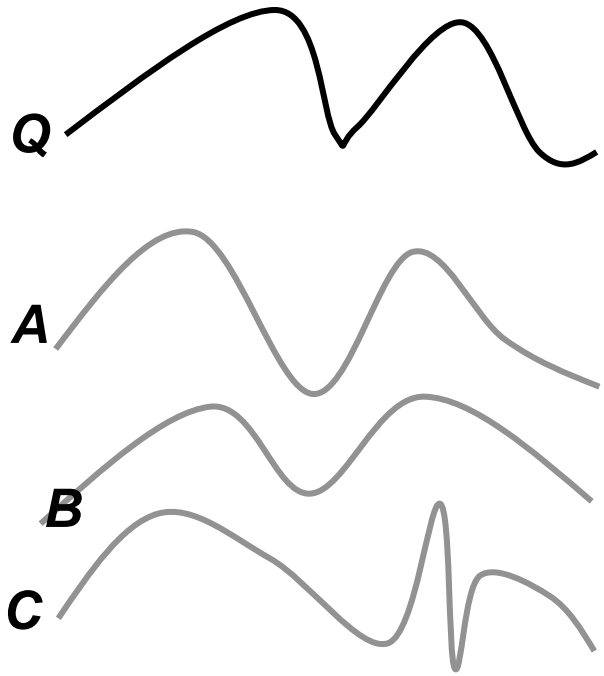


Metric distance functions can exploit the triangle inequality to speed-up search

Intuitively, if:
- x is similar to y and,
- y is similar to z, then,
- x is similar to z too.

Triangle Inequality (Importance)

Triangle Inequality: **d(x,z) ≤ d(x,y) + d(y,z)**



Assume: *d(Q,bestMatch) = 20*

and *d(Q,B) =150*

Then, since d(A,B)=20

$$d(Q,A) \geq d(Q,B) - d(B,A)$$

$$d(Q,A) \geq 150 - 20 = 130$$

So we don't have to retrieve A from disk

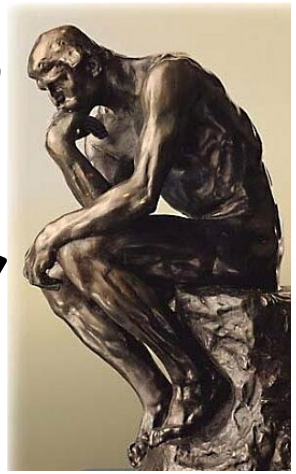
	A	B	C
A	0	20	110
B	20	0	90
C	110	90	0

Non-Metric Distance Functions

Man similar to bat??

Bat similar to batman

Batman similar to man



- ***Matching flexibility***
- ***Robustness to outliers***
- ***Stretching in time/space***
- ***Support for different sizes/lengths***

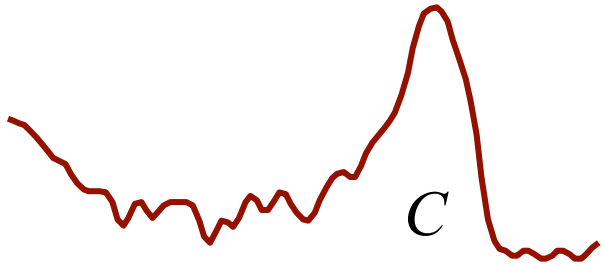
- ***Speeding-up search can be difficult***

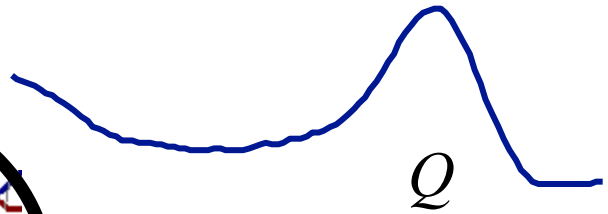
Euclidean Distance Metric

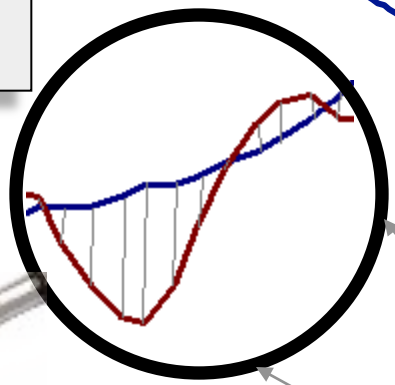
Given two time series:

$$Q = q_1 \ldots q_n$$
$$C = c_1 \ldots c_n$$

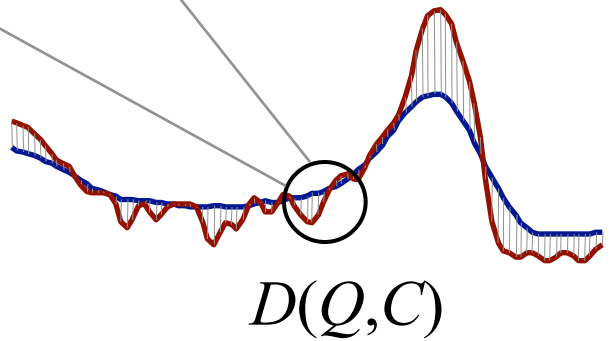
$$D(Q,C) \equiv \sqrt{\sum_{i=1}^{n} (q_i - c_i)^2}$$







About 80% of published work in data mining uses Euclidean distance




```
L2 = sqrt(sum((a-b).^2)); % return Euclidean distance
```

Optimizing the Euclidean Distance Calculation

$$D(Q,C) \equiv \sqrt{\sum_{i=1}^{n}(q_i - c_i)^2}$$

$$D_{squared}(Q,C) \equiv \sum_{i=1}^{n}(q_i - c_i)^2$$

Euclidean distance and Squared Euclidean distance are equivalent in the sense that they return the same rankings, clusterings and classifications

Instead of using the **Euclidean distance**

we can use the

Squared Euclidean distance

This optimization helps with CPU time, but most problems are I/O bound.

Euclidean Distance (Vectorization)

A Matlab trick

Question: *If I want to compare many sequences to each other do I have to use a for-loop?*

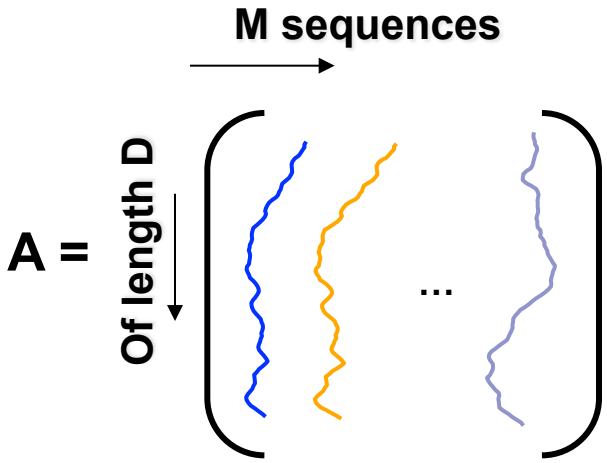
Answer: *No, one can use the following equation to perform matrix computations only…*

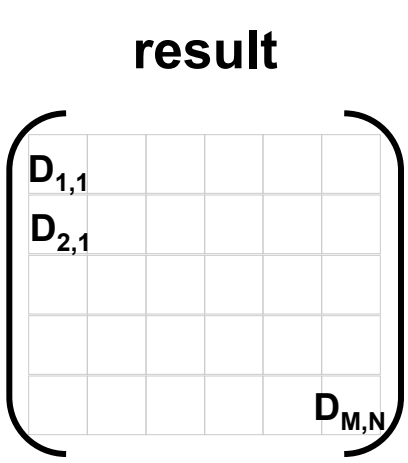
$$||A-B|| = \text{sqrt}\left(||A||^2 + ||B||^2 - 2*A.B \right)$$

A: DxM matrix

B: DxN matrix

Result is MxN matrix






```
aa=sum(a.*a); bb=sum(b.*b); ab=a'*b;
d = sqrt(repmat(aa',[1 size(bb,2)]) + repmat(bb,[size(aa,2) 1]) - 2*ab);
```

Preprocessing the data before distance calculations

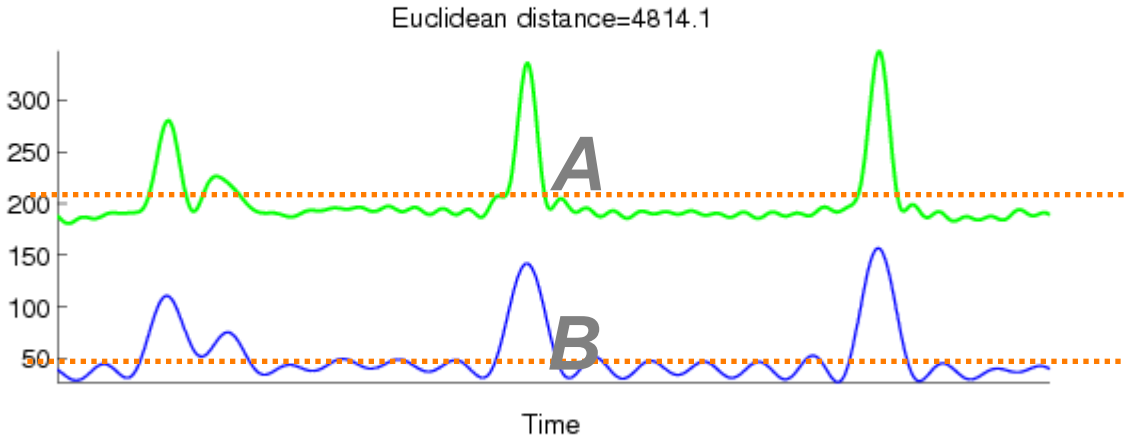
If we naively try to measure the distance between two "raw" time series, we may get very unintuitive results

This is because Euclidean distance is very sensitive to some "distortions" in the data. For most problems these distortions are not meaningful, and thus we can and should remove them

In the next few slides we will discuss the 4 most common distortions, and how to remove them

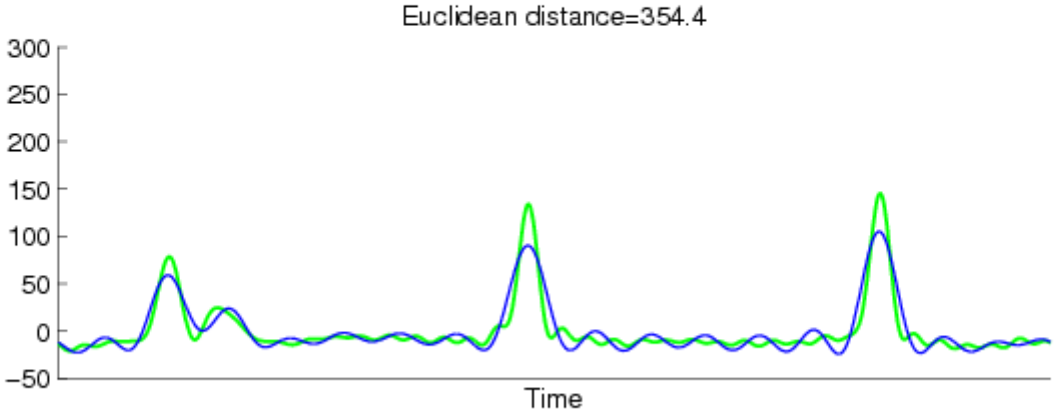
- Offset Translation
- Amplitude Scaling
- Linear Trend
- Noise

Transformation I: Offset Translation



average value of A

average value of B

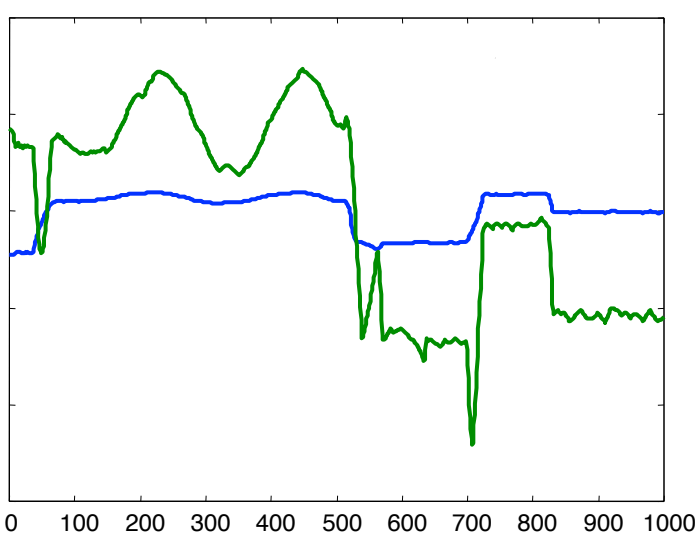


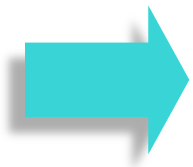
$A = A - mean(A)$

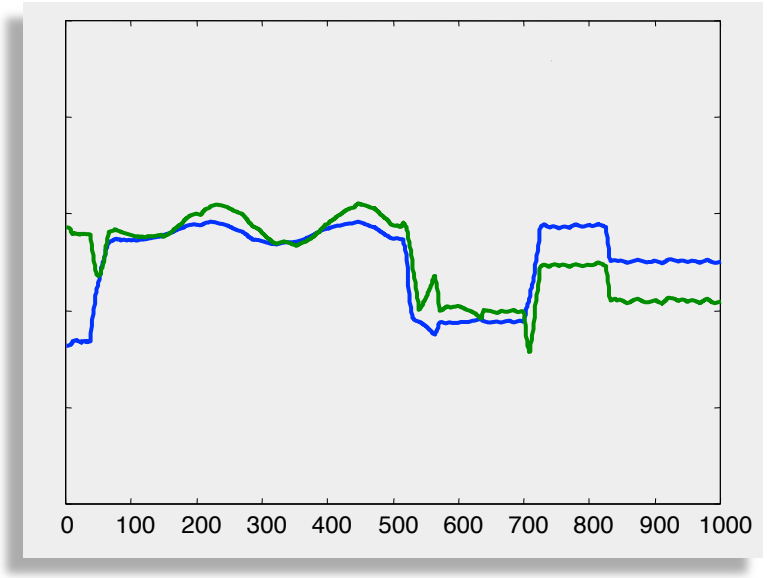
$B = B - mean(B)$

$D(A, B)$

Transformation II: Amplitude Scaling





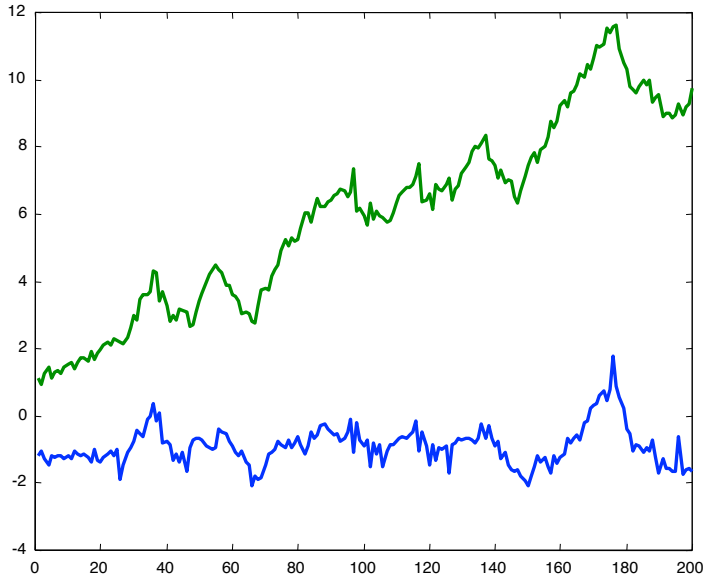


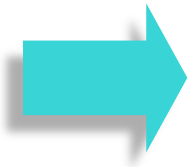
$Q = (Q - \text{mean}(Q)) / \text{std}(Q)$

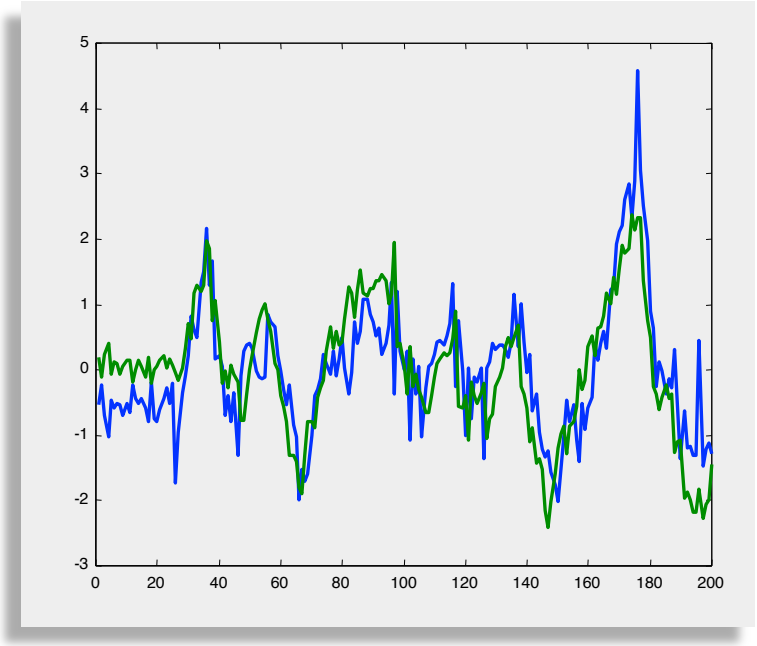
$C = (C - \text{mean}(C)) / \text{std}(C)$

$D(Q,C)$

Transformation III: Linear Trend







The intuition behind removing linear trend is…

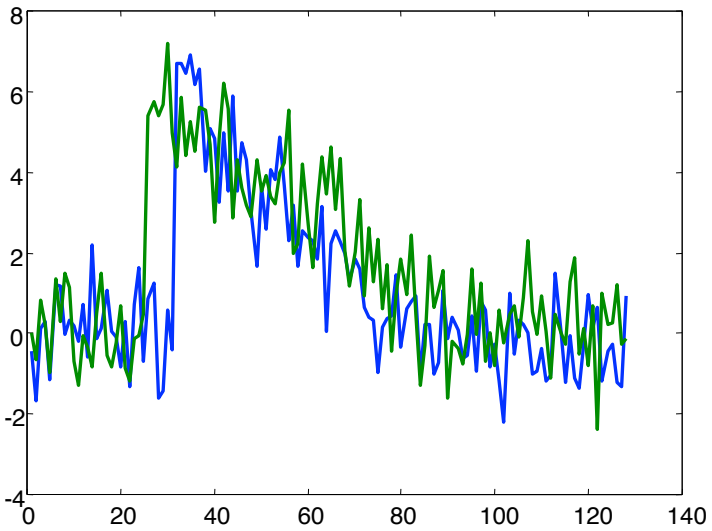
Fit the best fitting straight line to the time series, then subtract that line from the time series.

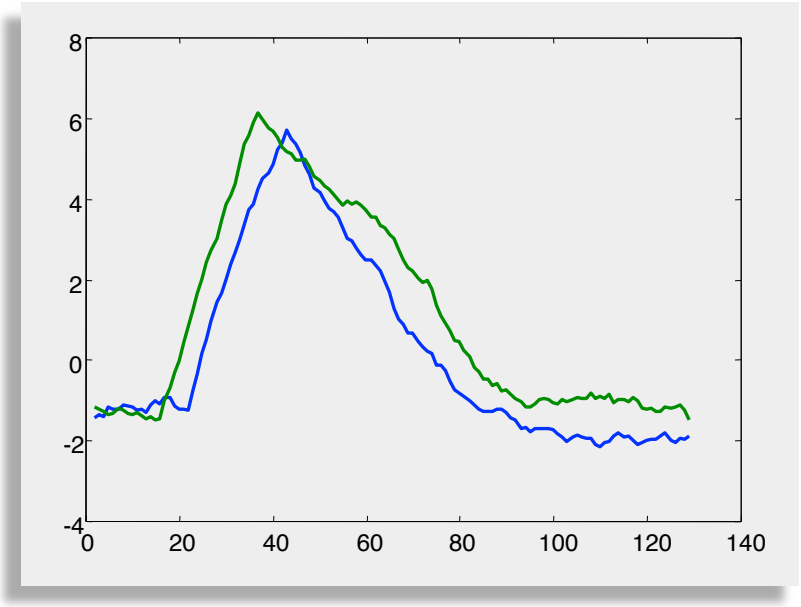
Removed **linear trend**

Removed offset translation

Removed amplitude scaling

Transformation IIII: Noise





The intuition behind removing noise is...

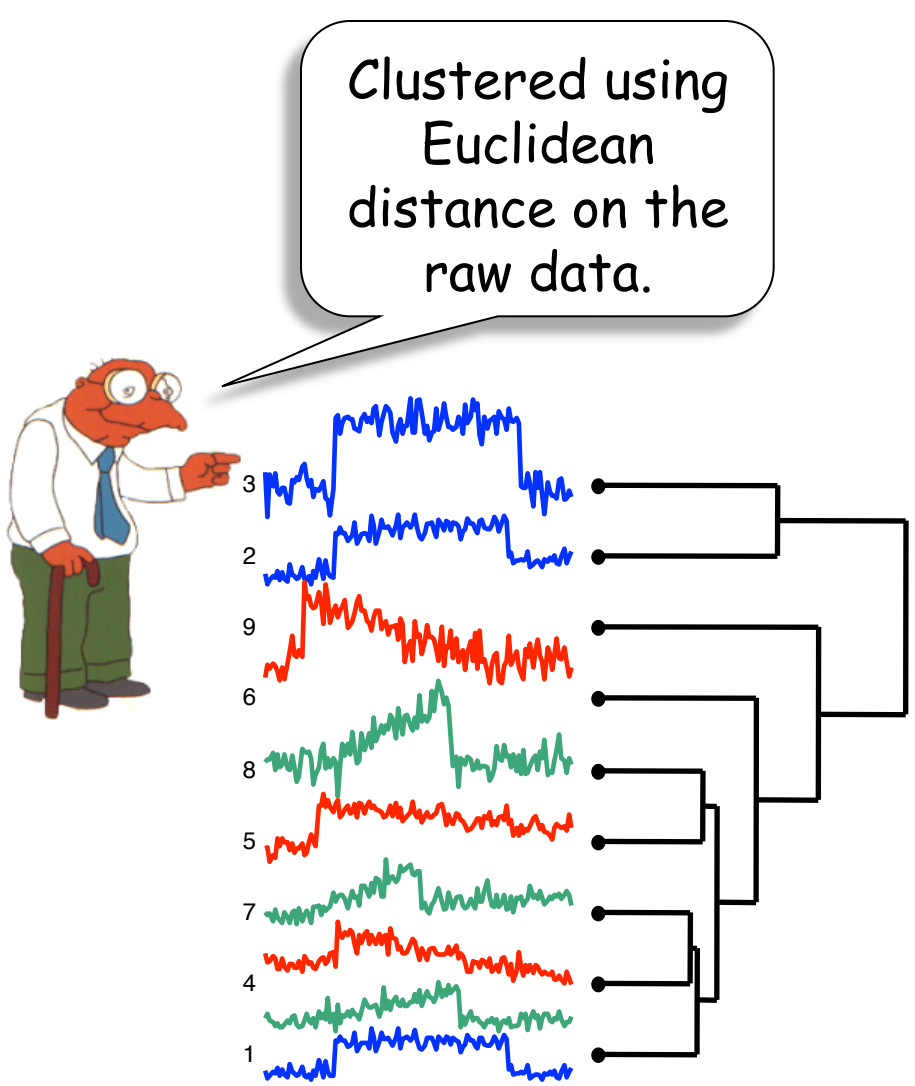
Average each datapoints value with its neighbors.

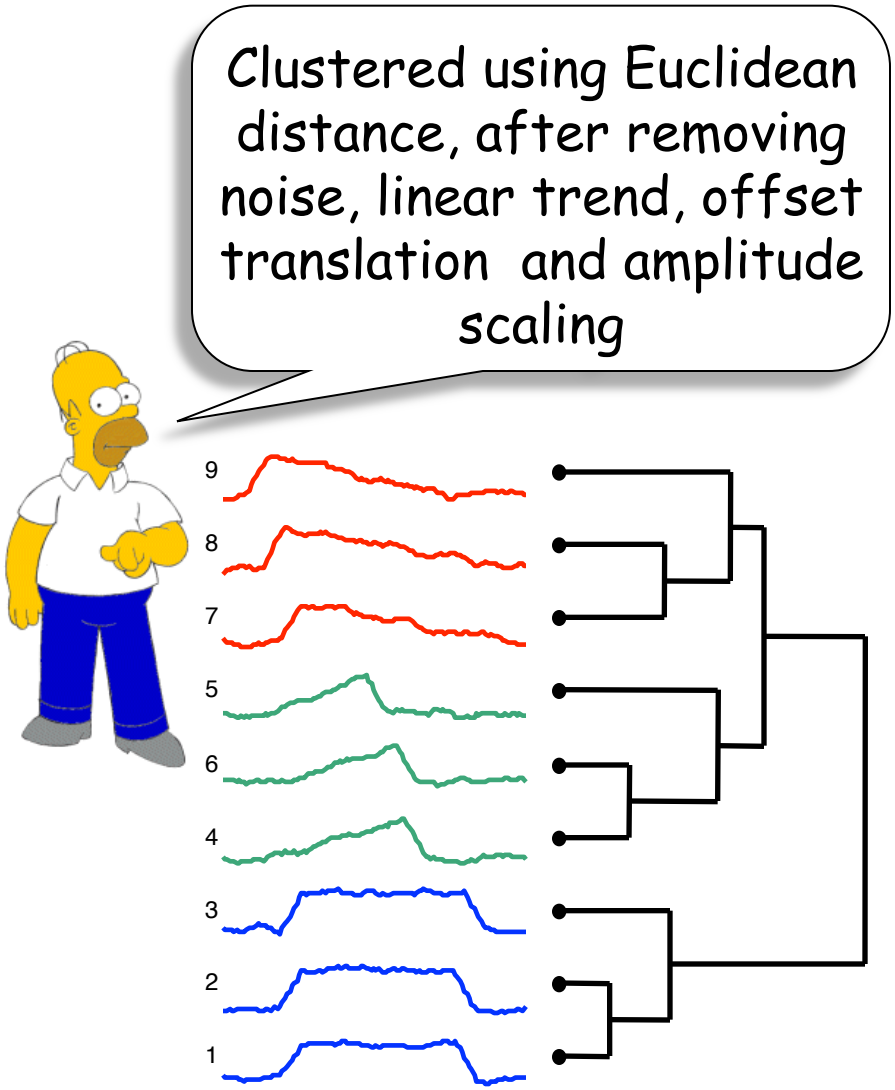
$Q = \text{smooth}(Q)$

$C = \text{smooth}(C)$

$D(Q,C)$

A Quick Experiment to Demonstrate the Utility of Preprocessing the Data





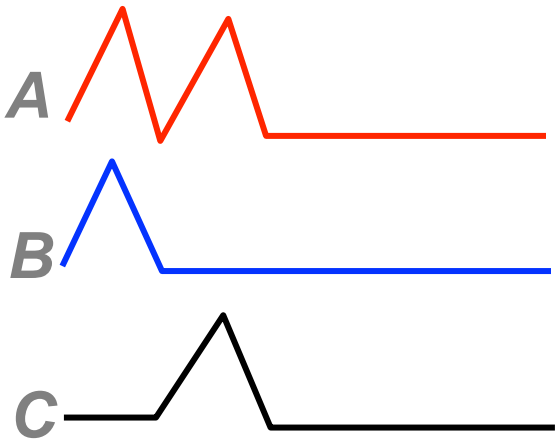
Summary of Preprocessing

The "raw" time series may have distortions which we should remove before clustering, classification etc

Of course, sometimes the distortions are the most interesting thing about the data, the above is only a general rule

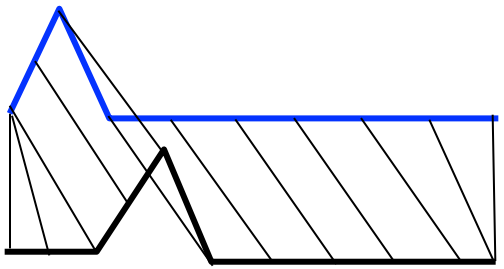
We should keep in mind these problems as we consider the high level representations of time series which we will encounter later (DFT, Wavelets etc). Since these representations often allow us to handle distortions in elegant ways

Euclidean distance cannot compensate for small distortions in time axis.

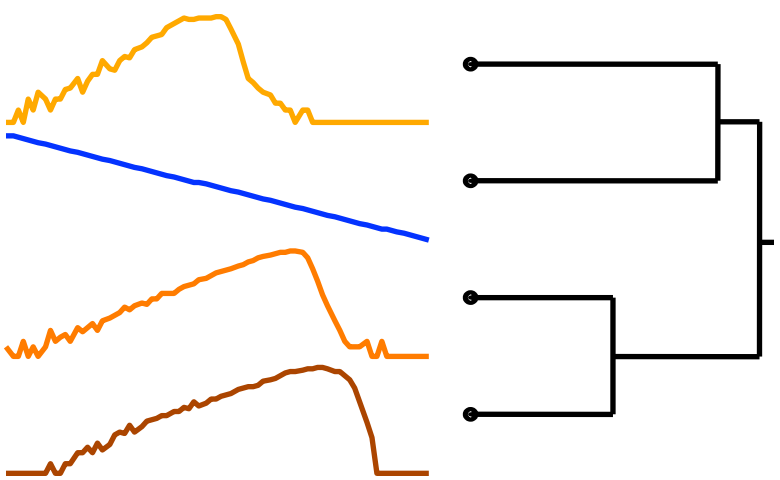


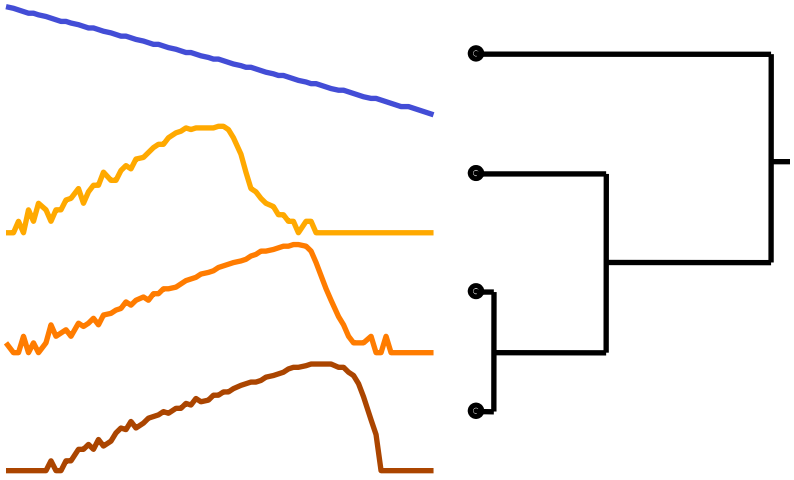
According to Euclidean distance
B is more similar to A than to C

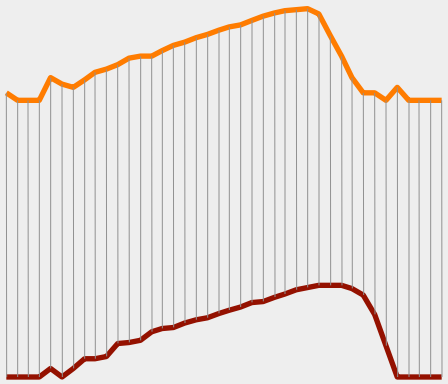
Solution: Allow for compression & decompression in time



Dynamic Time Warping

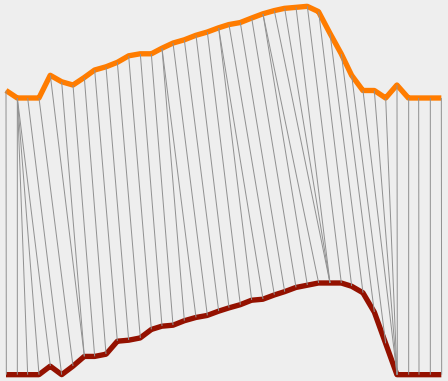






Fixed Time Axis

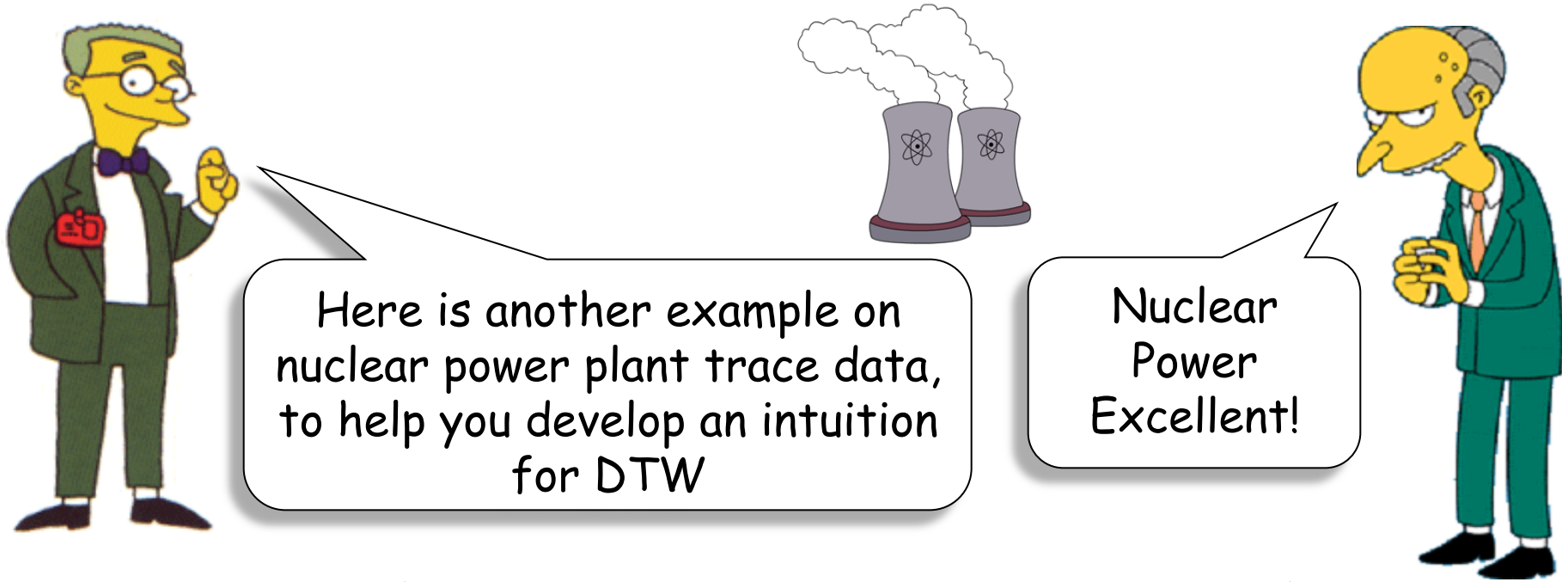
Sequences are aligned "one to one".

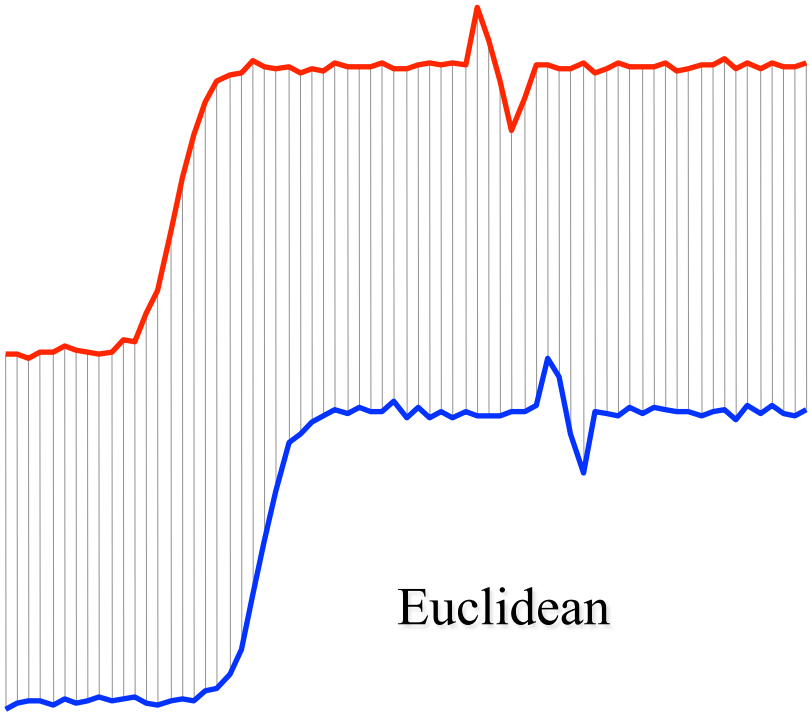


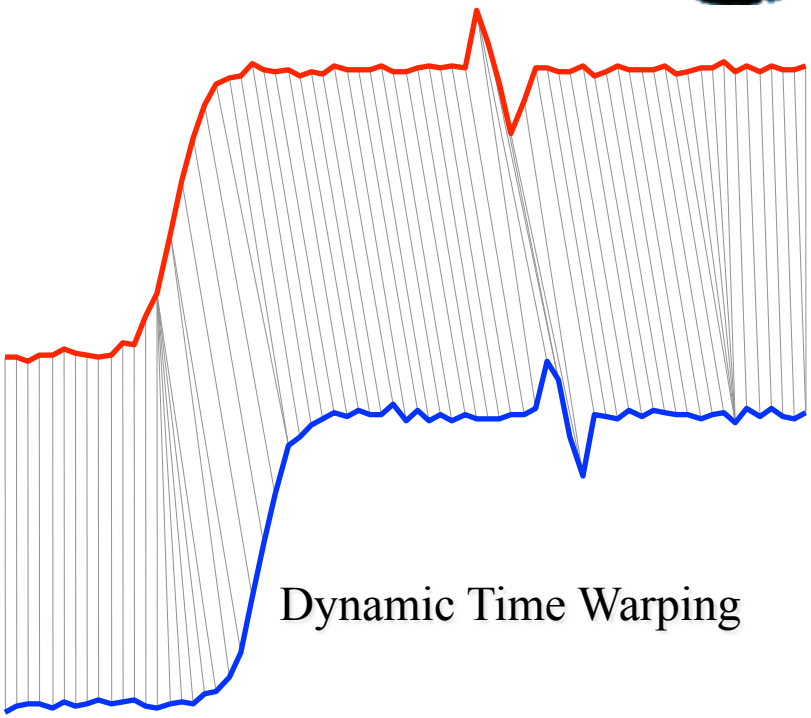
"Warped" Time Axis

Nonlinear alignments are possible.

Note: We will first see the utility of DTW, then see how it is calculated.


Here is another example on nuclear power plant trace data, to help you develop an intuition for DTW
Nuclear Power Excellent!

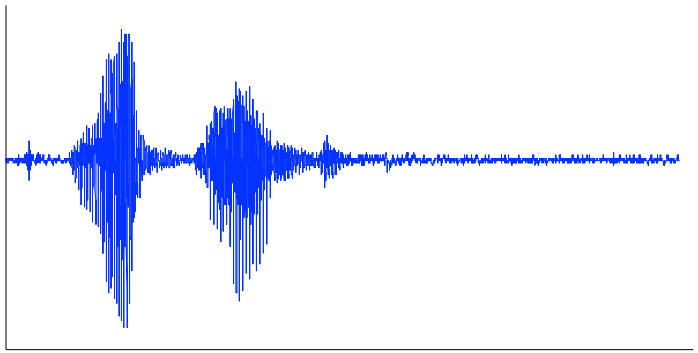

Euclidean


Dynamic Time Warping

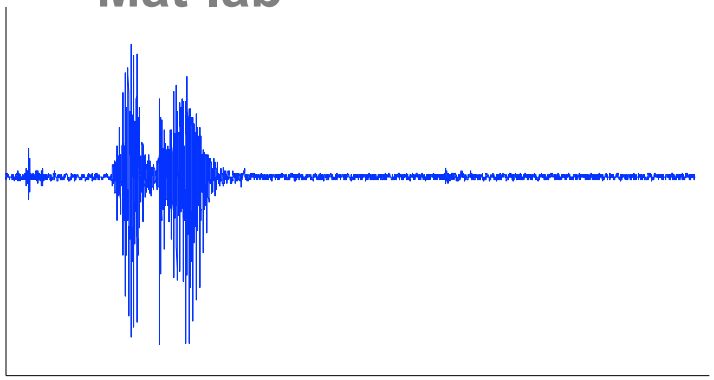
Dynamic Time-Warping

First used in speech recognition for recognizing words spoken at different speeds

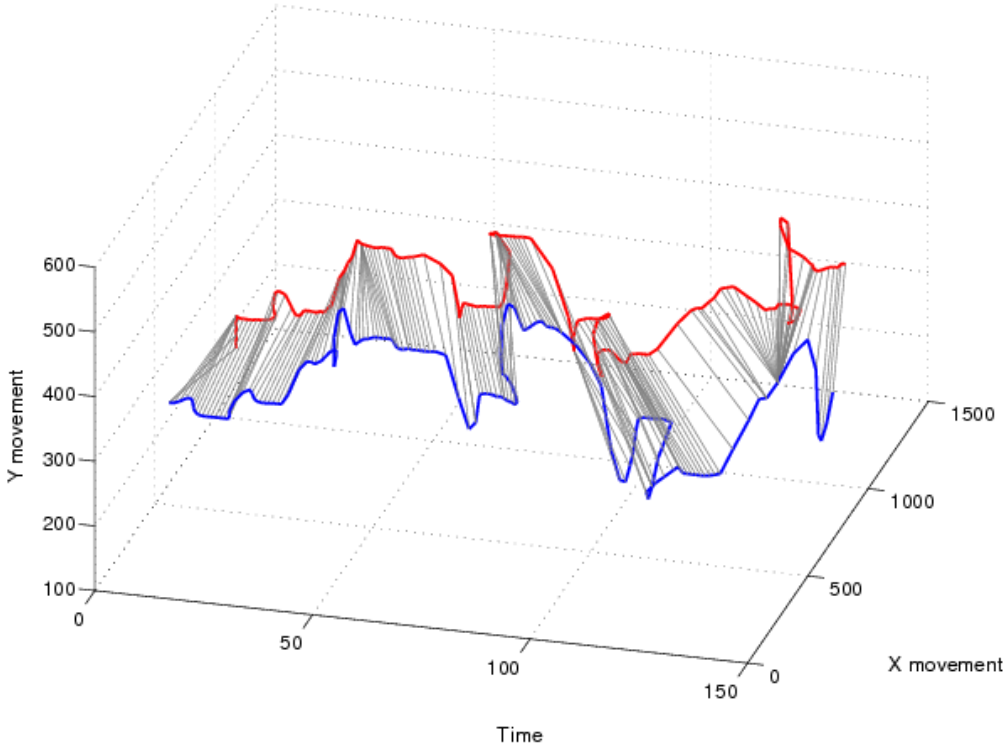
---Maat--llaabb----------



----Mat-lab----------



Same idea can work equally well for generic time-series data



Dynamic Time-Warping (how does it work?)

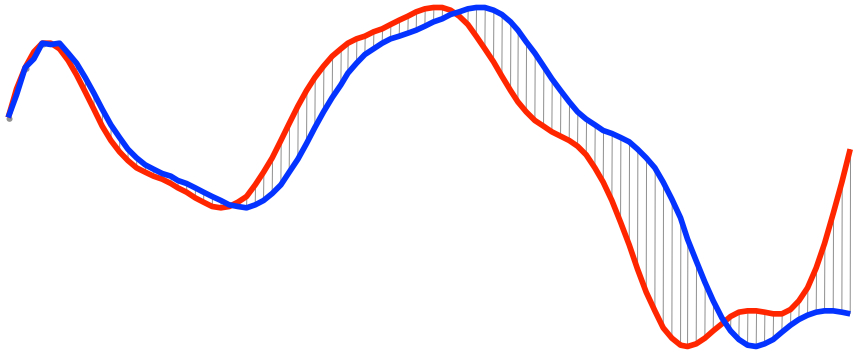
The intuition is that we copy an element multiple times so as to achieve a better matching

Euclidean distance

T1 = [1, 1, 2, 2]

| | | | d = 1

T2 = [1, 2, 2, 2]



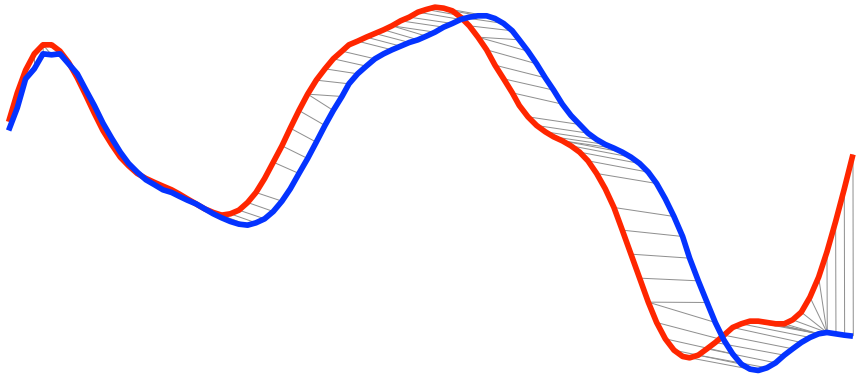
One-to-one linear alignment

Warping distance

T1 = [1, 1, 2, 2]

\ \ | d = 0

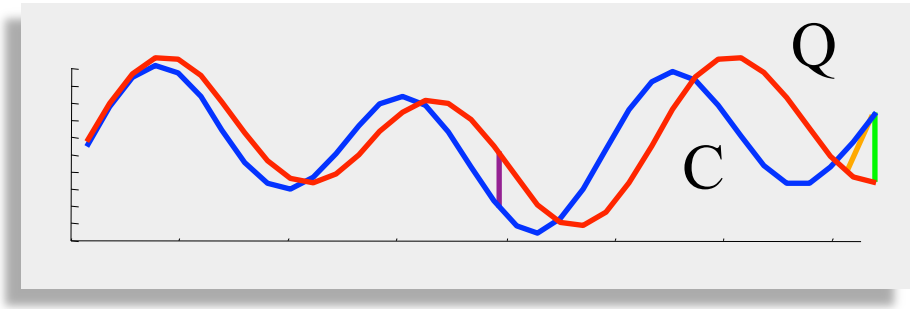
T2 = [1, 2, 2, 2]

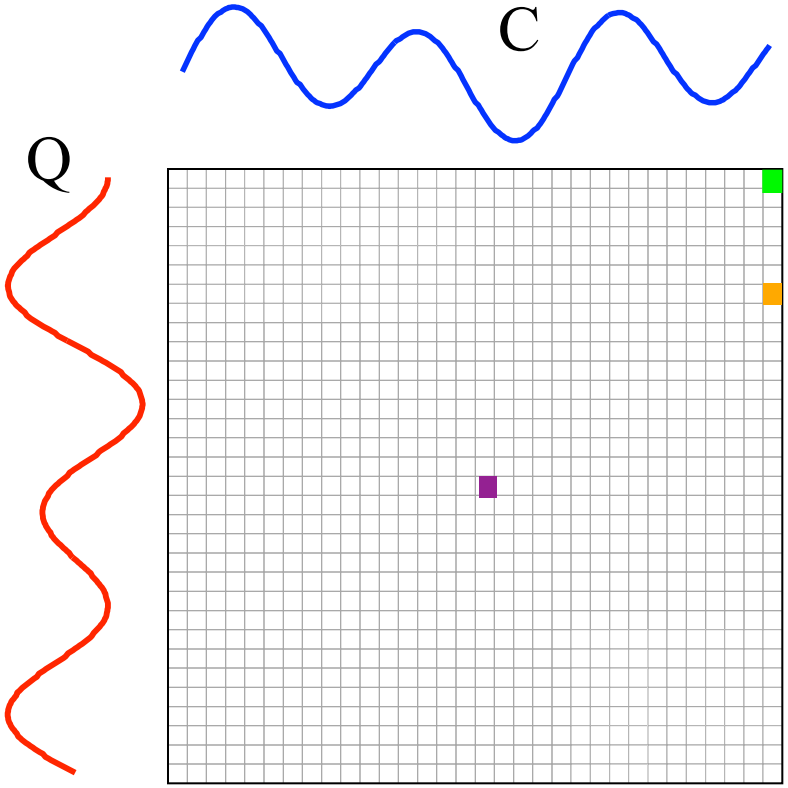


One-to-many non-linear alignment

How is DTW Calculated? I

We create a matrix the size of |Q| by |C|, then fill it in with the distance between every pair of point in our two time series.

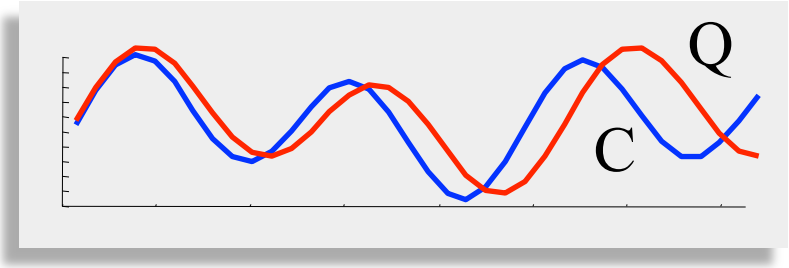


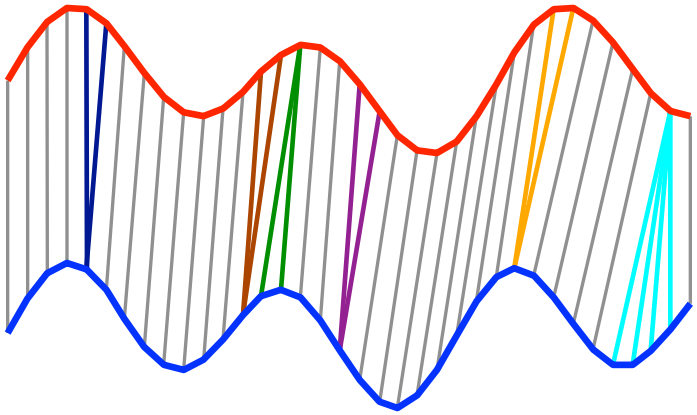


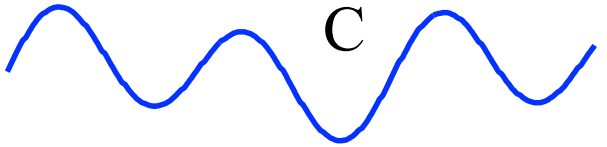
How is DTW Calculated? II

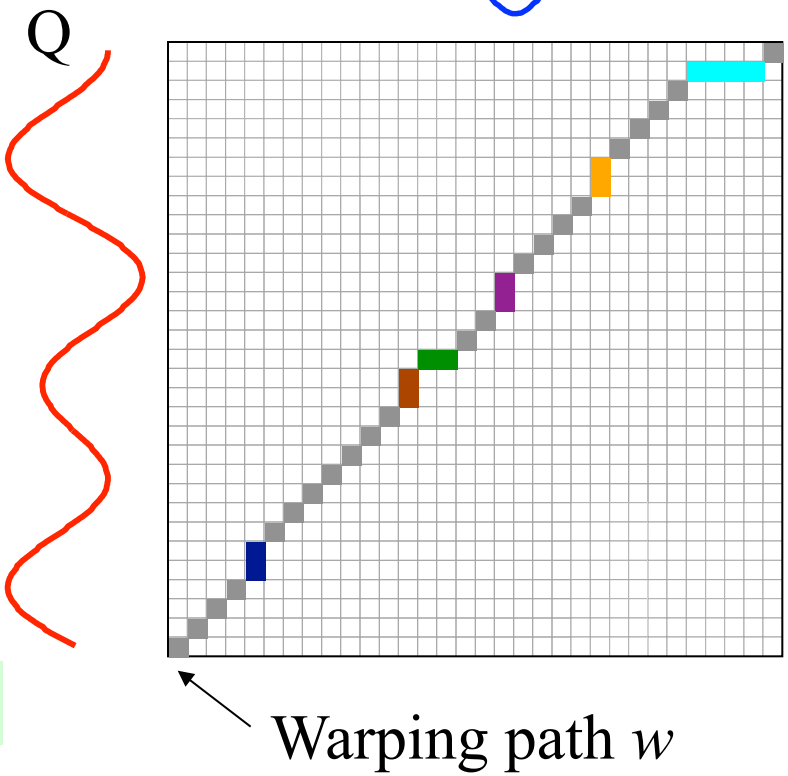
Every possible warping between two time series, is a path though the matrix. We want the best one…

$$DTW(Q,C) = \min\left\{ \sqrt{\sum_{k=1}^{K} w_k} \Big/ K \right\}$$









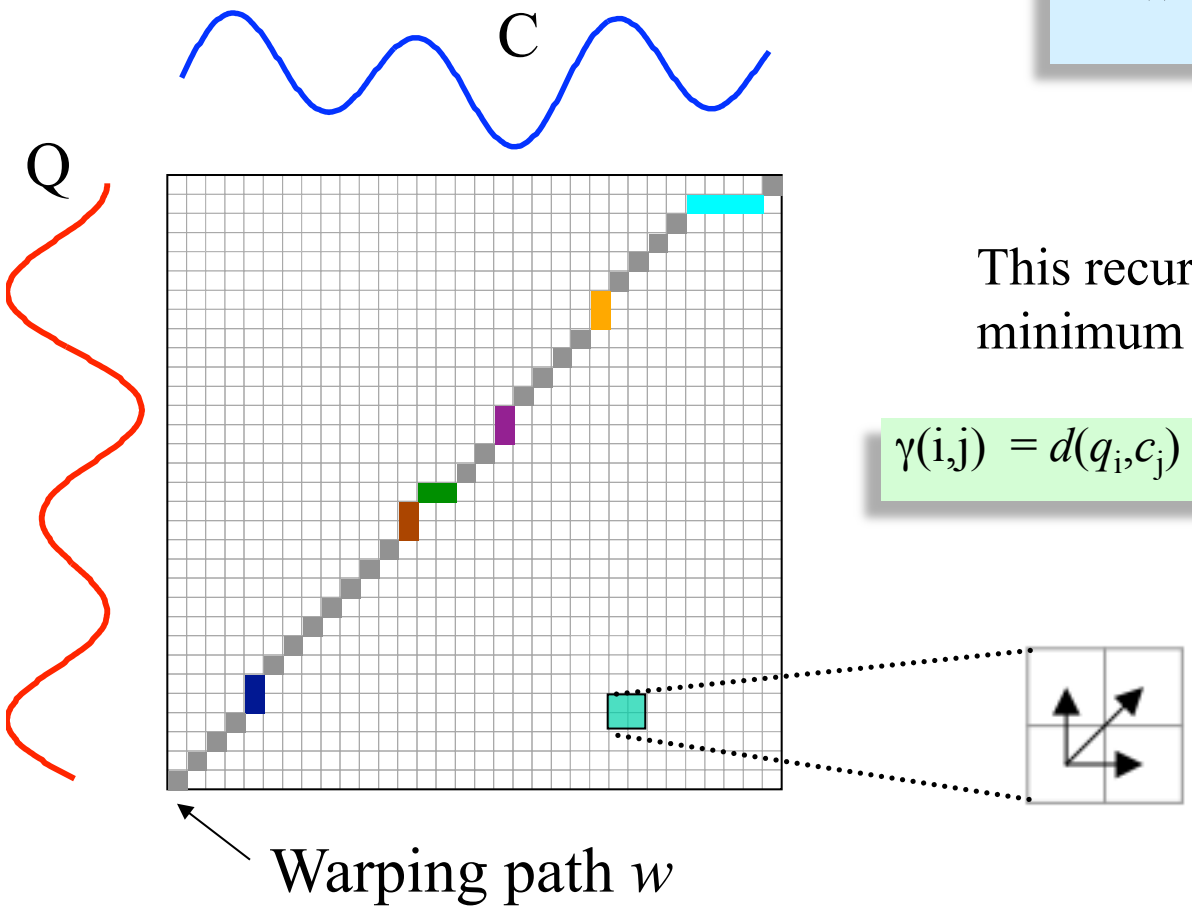
This recursive function gives us the minimum cost path

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

How is DTW Calculated? II

Every possible warping between two time series, is a path though the matrix. We want the best one…

$$DTW(Q,C) = \min\left\{ \sqrt{\sum_{k=1}^{K} w_k} \Big/ K \right\}$$

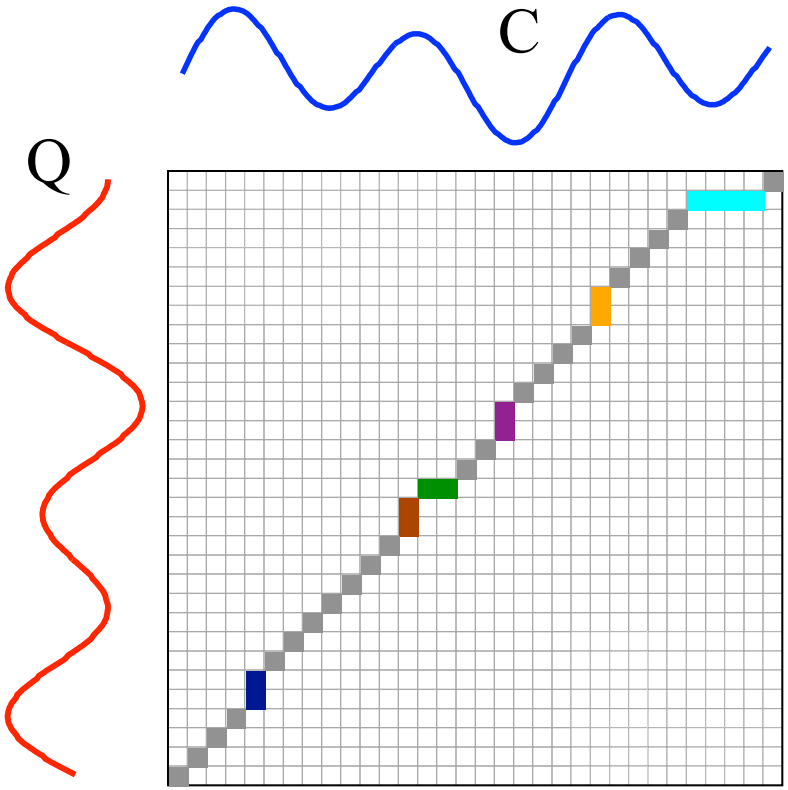


This recursive function gives us the minimum cost path

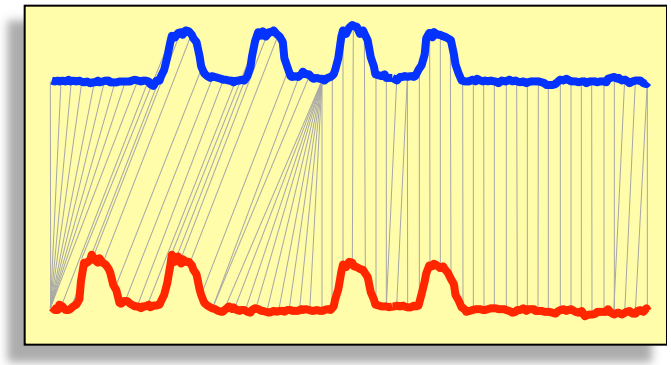
$$\gamma(i,j) = d(q_i,c_j) + \min\{ \gamma(i-1,j-1), \gamma(i-1,j), \gamma(i,j-1) \}$$

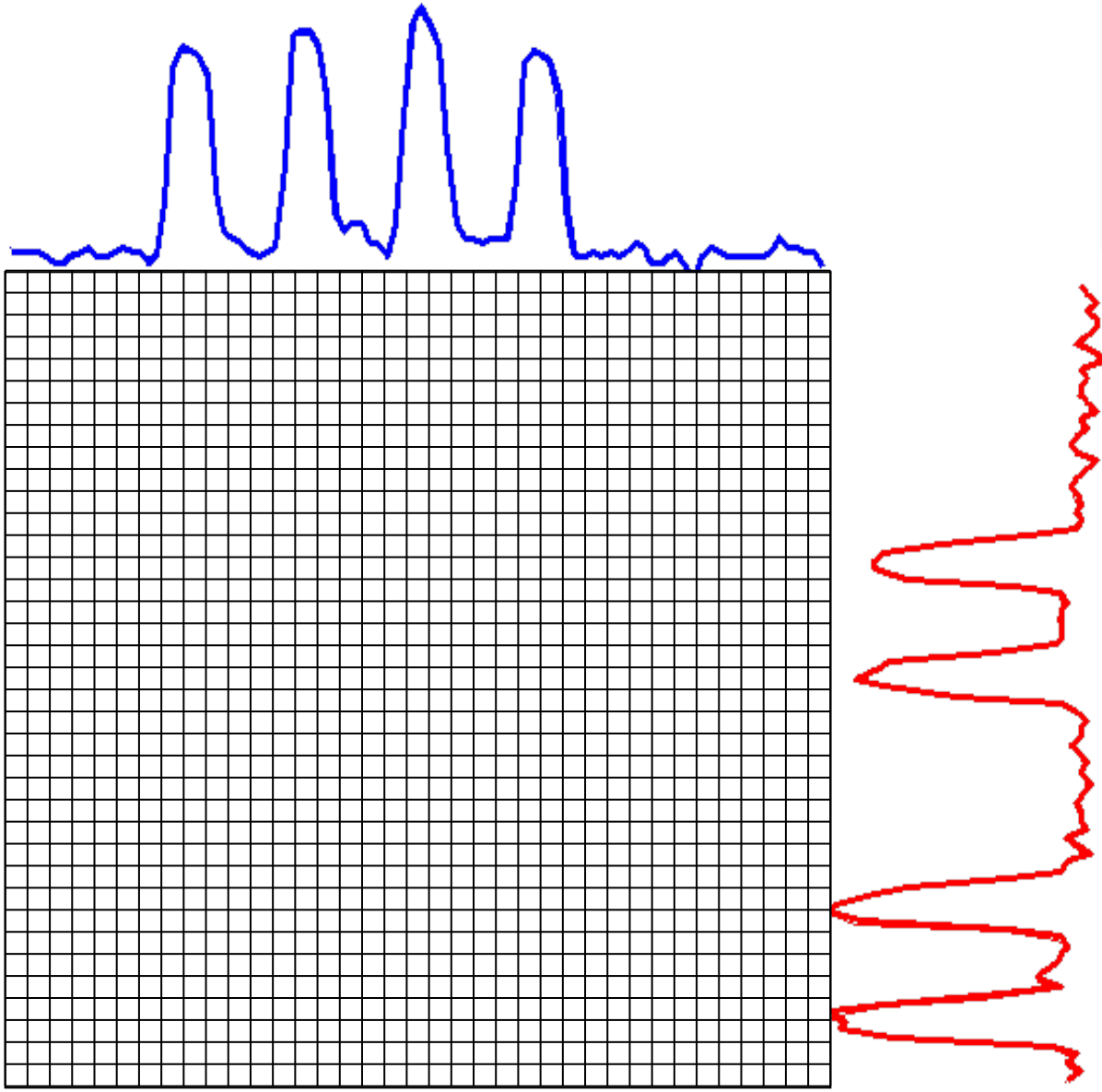
Three Constraints: Given a path *W*…

- **Boundary Constraint**: *W* must start and finish in the first and last points of the sequences
- **Continuity**: at any given point in *W*, we can only travel to neighboring points
- **Monotonicity**: points in *W* must be monotonically ordered



Let us visualize the cumulative matrix on a real world problem I

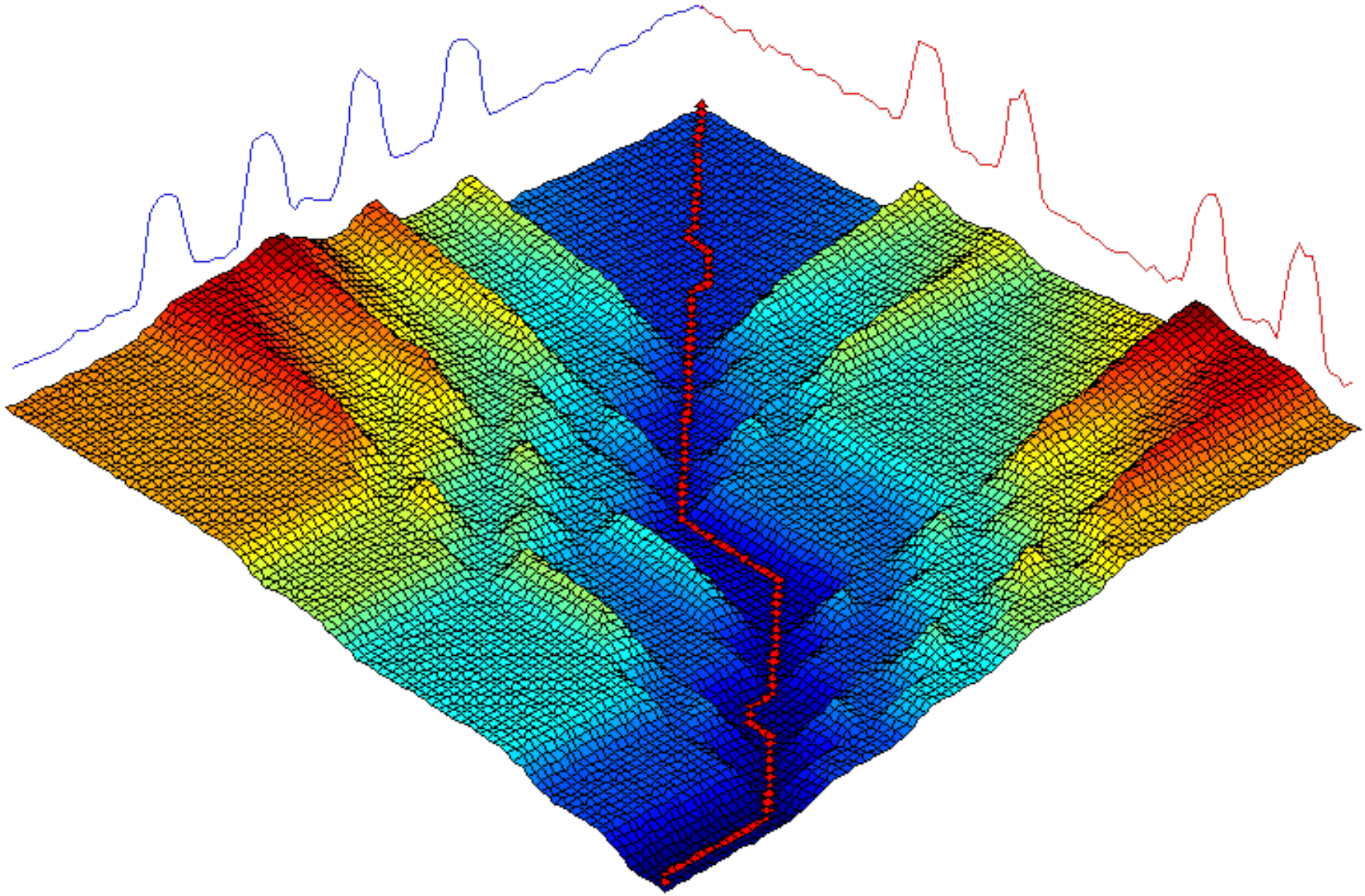




This example shows 2 one-week periods from the power demand time series.

Note that although they both describe 4-day work weeks, the blue sequence had Monday as a holiday, and the red sequence had Wednesday as a holiday.

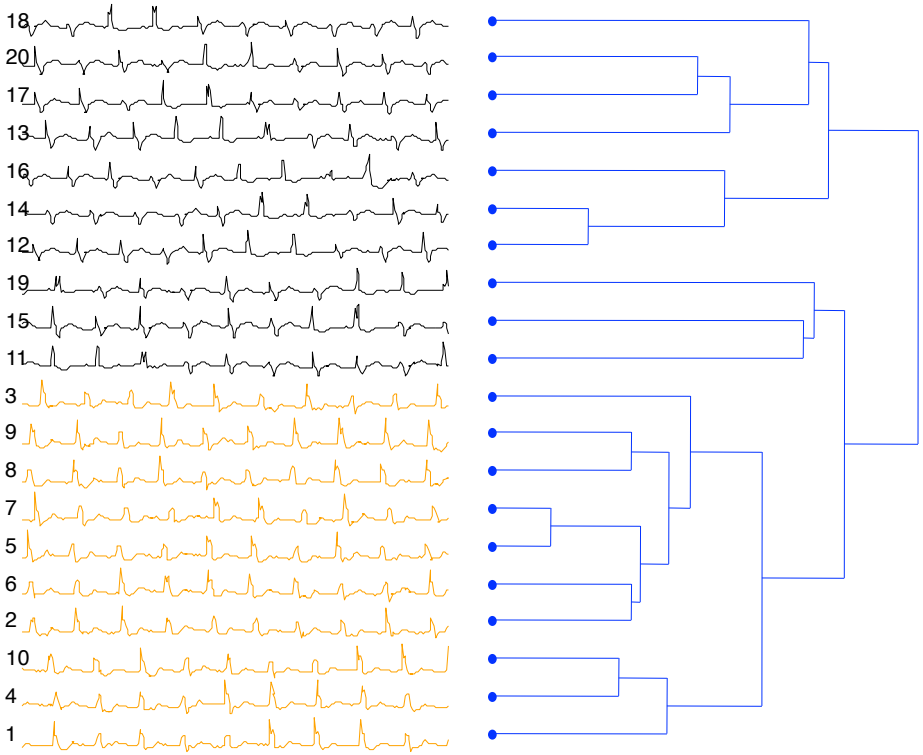
Let us visualize the cumulative matrix on a real world problem II



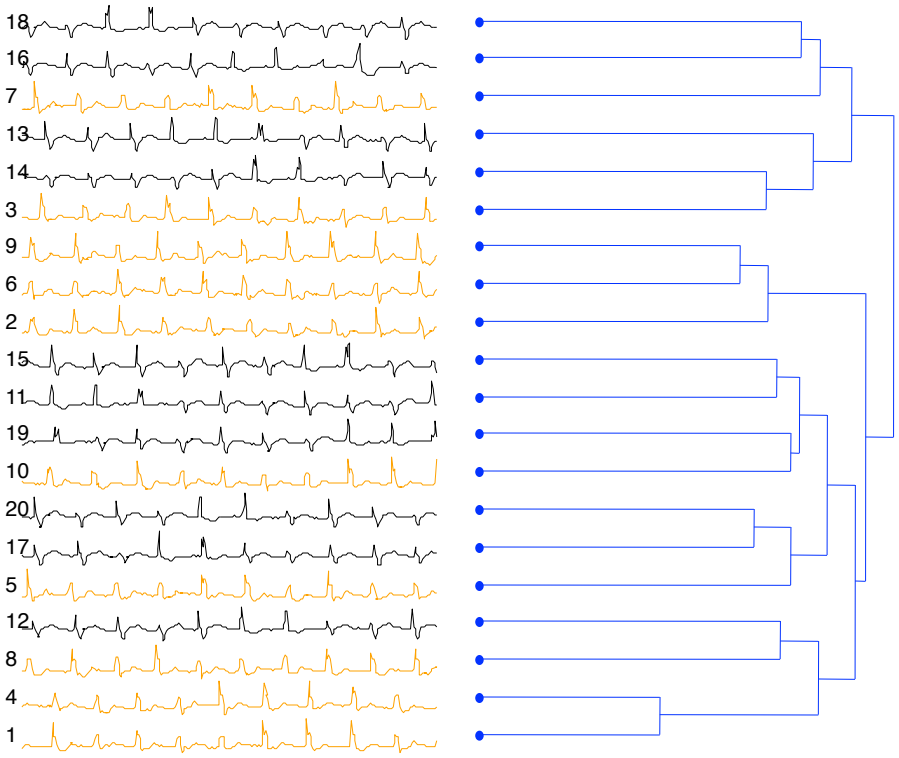
Dynamic Time-Warping (Examples)

So does it work better than Euclidean? Well yes! After all it is more costly..

Dynamic Time Warping

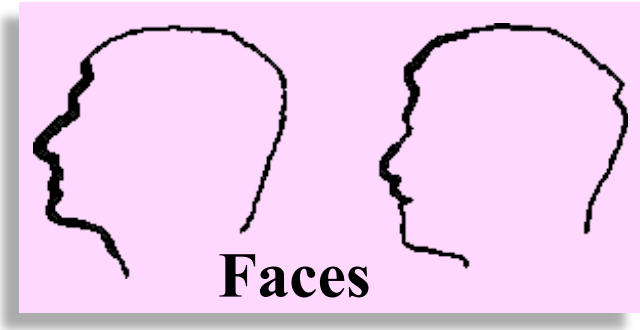


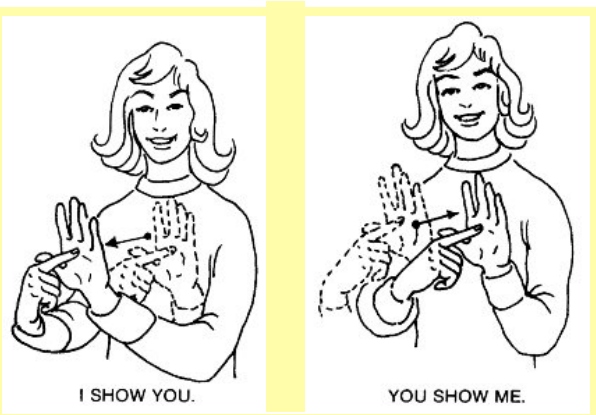
Euclidean Distance

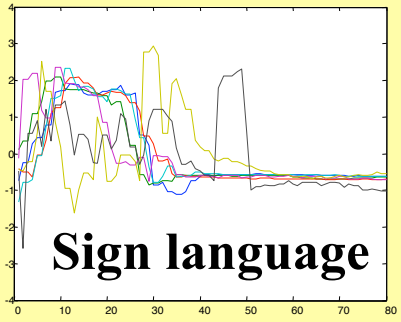


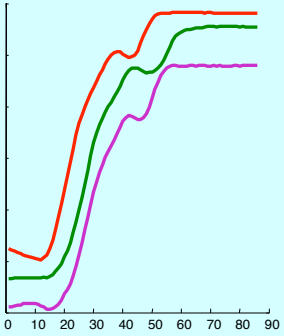
MIT arrhythmia database

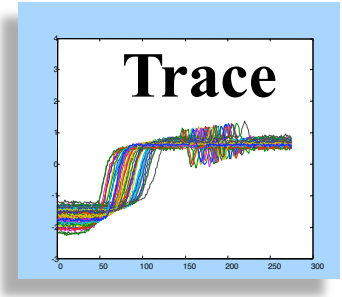
Let us compare Euclidean Distance and DTW on some problems

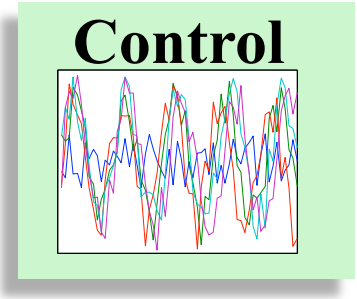


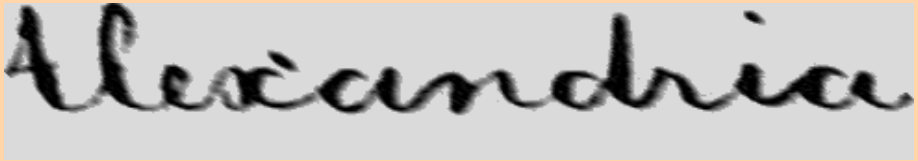


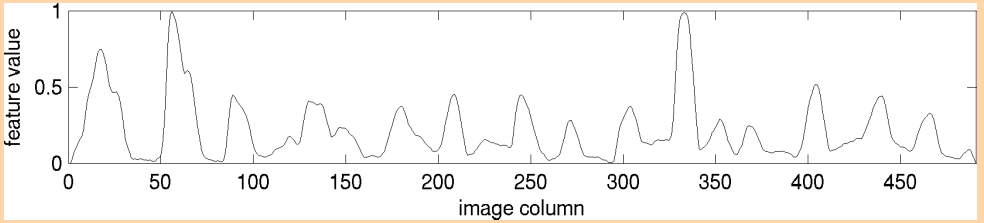


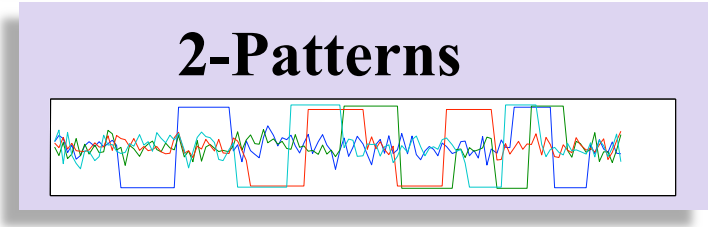










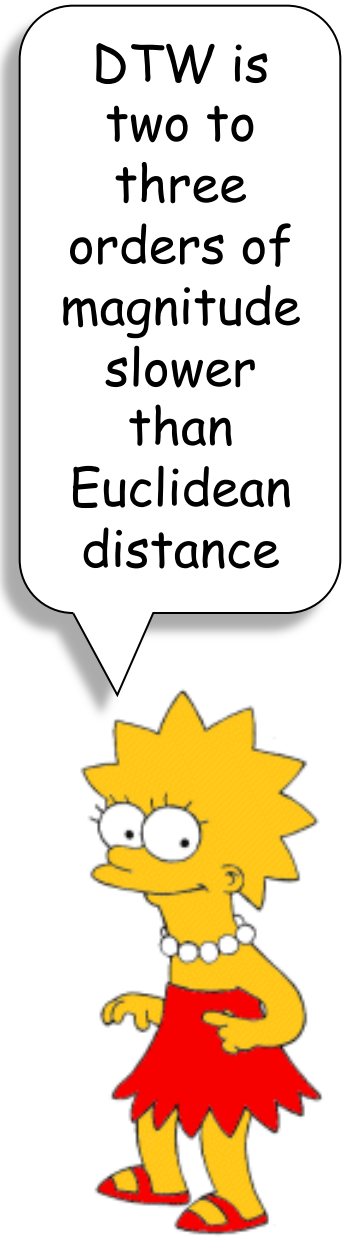


Results: Error Rate

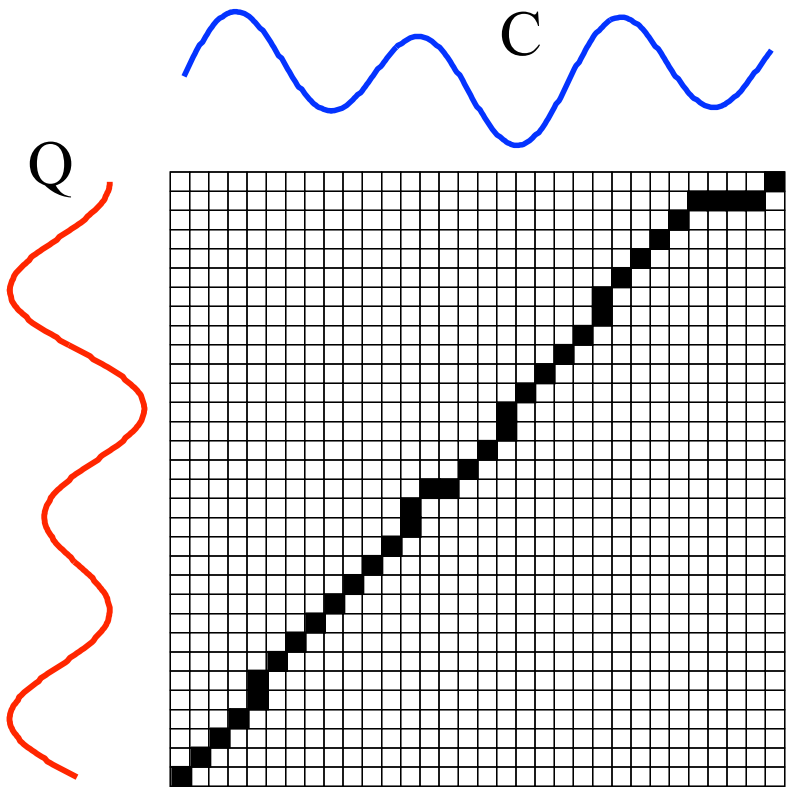
Dataset	Euclidean	DTW
Word Spotting	4.78	1.10
Sign language	28.70	25.93
GUN	5.50	1.00
Nuclear Trace	11.00	0.00
Leaves#	33.26	4.07
(4) Faces	6.25	2.68
Control Chart*	7.5	0.33
2-Patterns	1.04	0.00

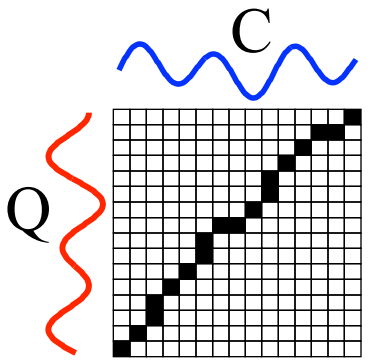
Results: Time (**msec**)

Dataset	Euclidean	DTW	
Word Spotting	40	8,600	215
Sign language	10	1,110	110
GUN	60	11,820	197
Nuclear Trace	210	144,470	687
Leaves	150	51,830	345
(4) Faces	50	45,080	901
Control Chart	110	21,900	199
2-Patterns	16,890	545,123	32



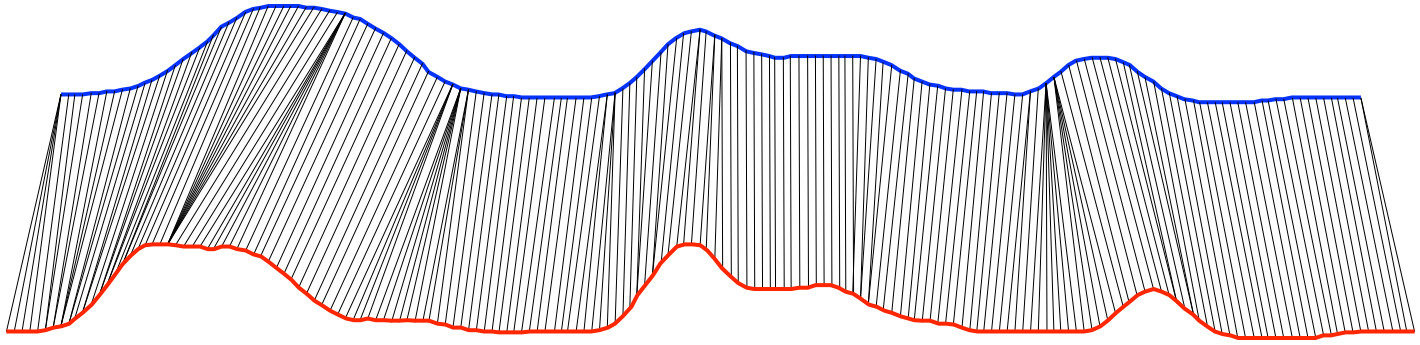
Fast Approximations to Dynamic Time Warp Distance I



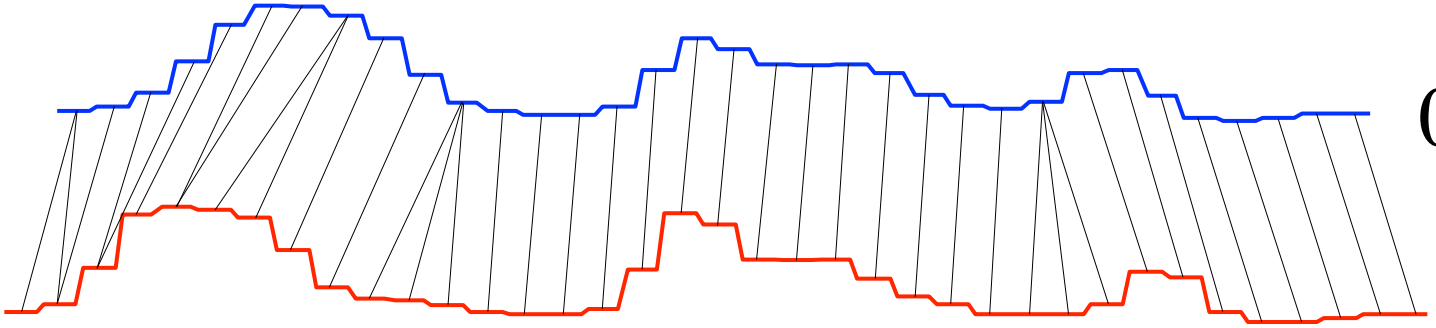


Simple Idea: Approximate the time series with some compressed or downsampled representation, and do DTW on the new representation. How well does this work...

Fast Approximations to Dynamic Time Warp Distance II



1.03 sec



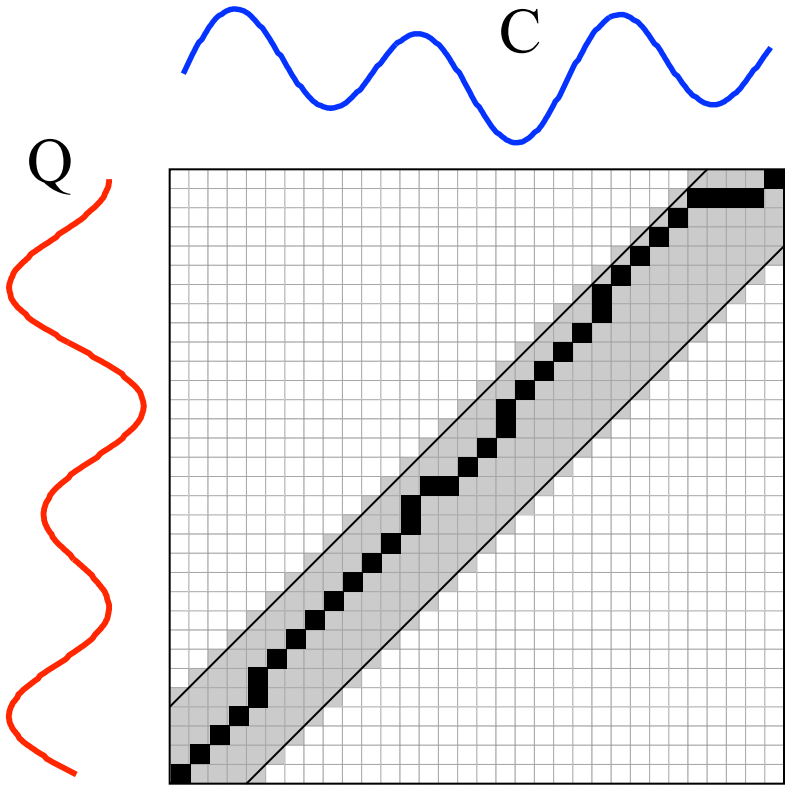
0.07 sec

… there is strong visual evidence to suggests it works well

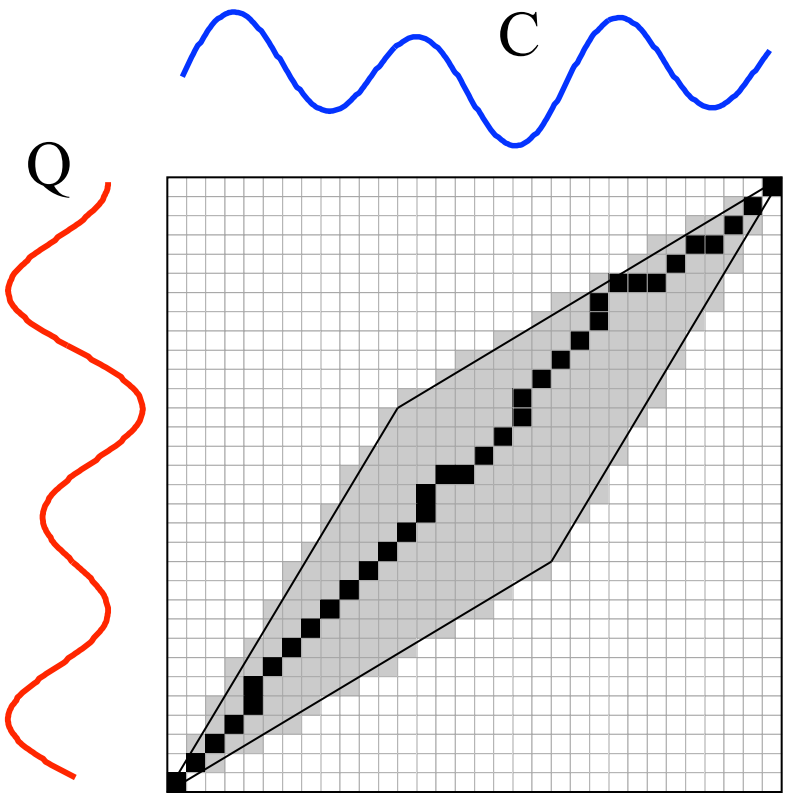
There is good experimental evidence for the utility of the approach on clustering, classification, etc

Global Constraints

- Slightly speed up the calculations
- Prevent pathological warpings



Sakoe-Chiba Band

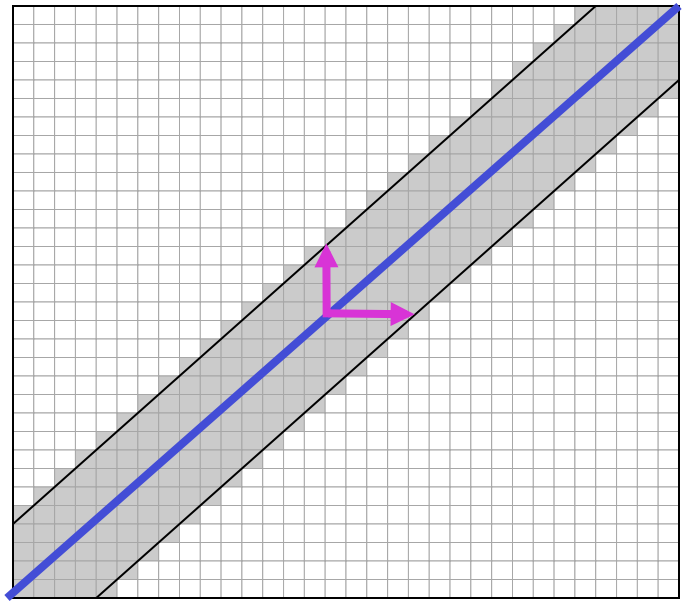


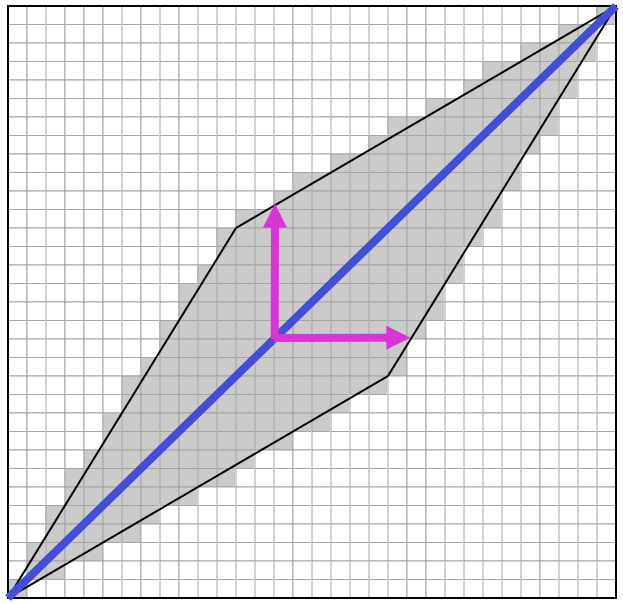
Itakura Parallelogram

A global constraint constrains the indices of the warping path $w_k = (i,j)_k$ such that $j-r \leq i \leq j+r$

Where r is a term defining allowed range of warping for a given point in a sequence.

We can learn the best r using a training set

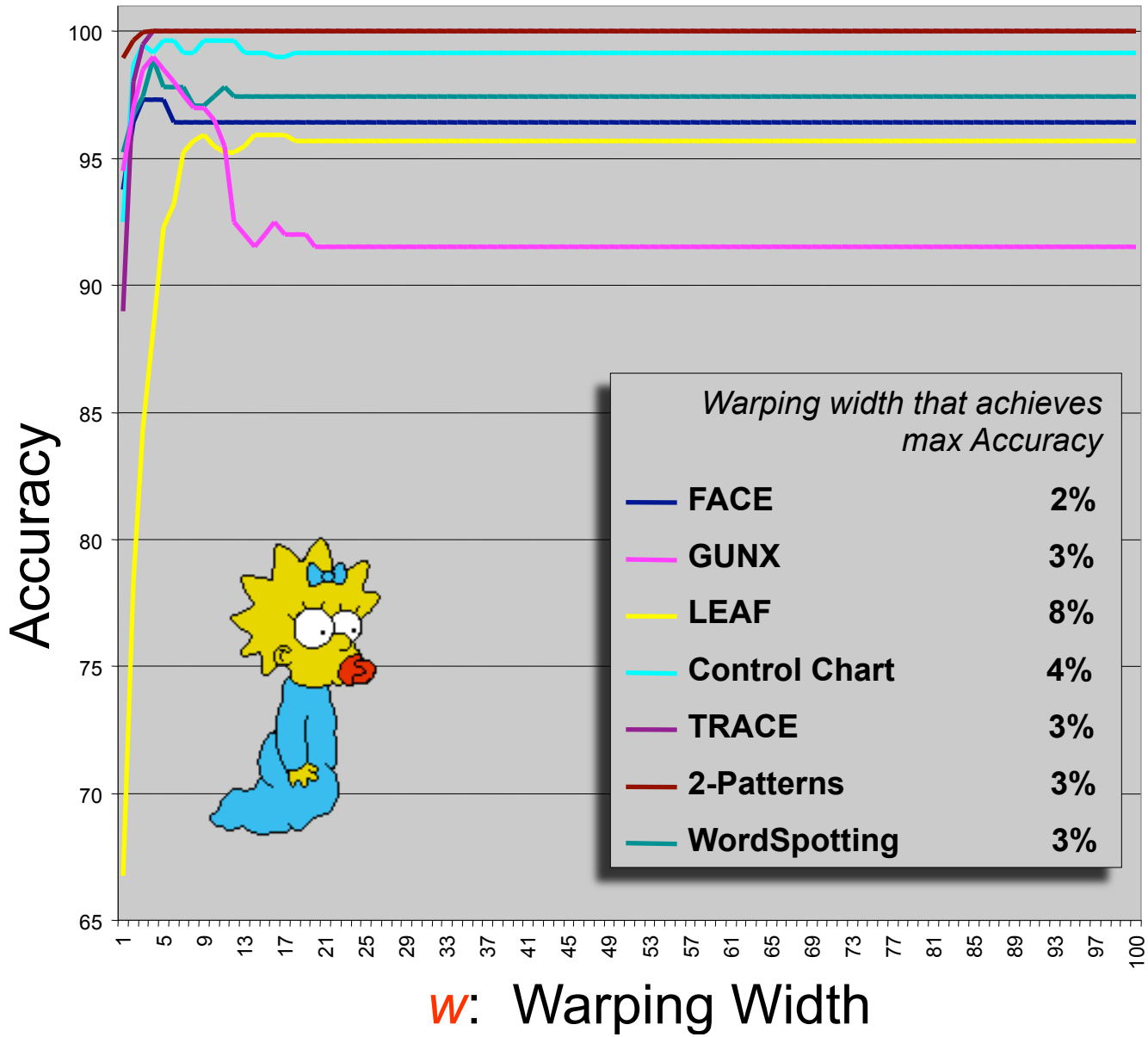


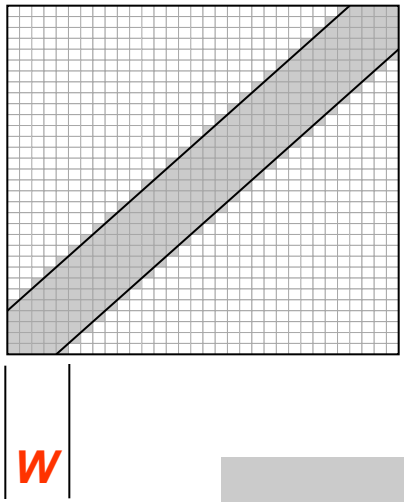


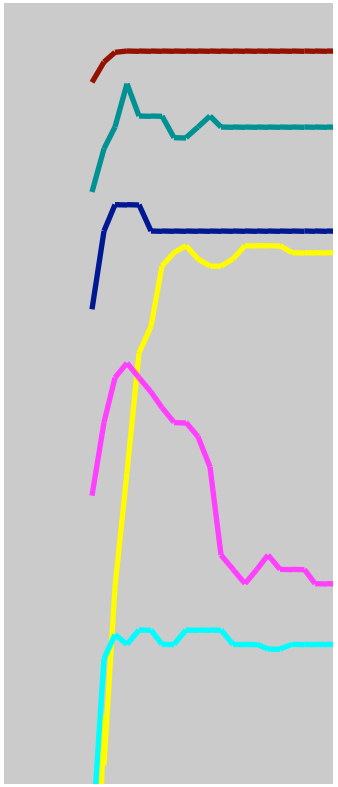
Sakoe-Chiba Band

Itakura Parallelogram

Accuracy vs. Width of Warping Window







In general, it's hard to speed up a single DTW calculation

However, if we have to make many DTW calculations (which is almost always the case), we can potentiality speed up the whole process by *lowerbounding*.

Keep in mind that the *lowerbounding* trick works for any situation where you have an expensive calculation that can be *lowerbounded* (string edit distance, graph edit distance etc)

I will explain how *lowerbounding* works in a generic fashion in the next two slides, then show concretely how *lowerbounding* makes dealing with massive time series under DTW possible…

Lower Bounding I

Assume that we have two functions:

- DTW(A,B)
- lower_bound_distance(A,B)

The true DTW function is very slow…

The *lower bound* function is very fast…

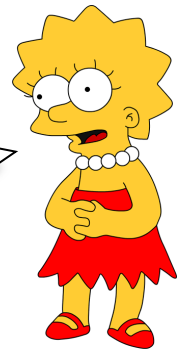
By definition, for all A, B, we have

$$\text{lower\_bound\_distance}(A,B) \leq \text{DTW}(A,B)$$

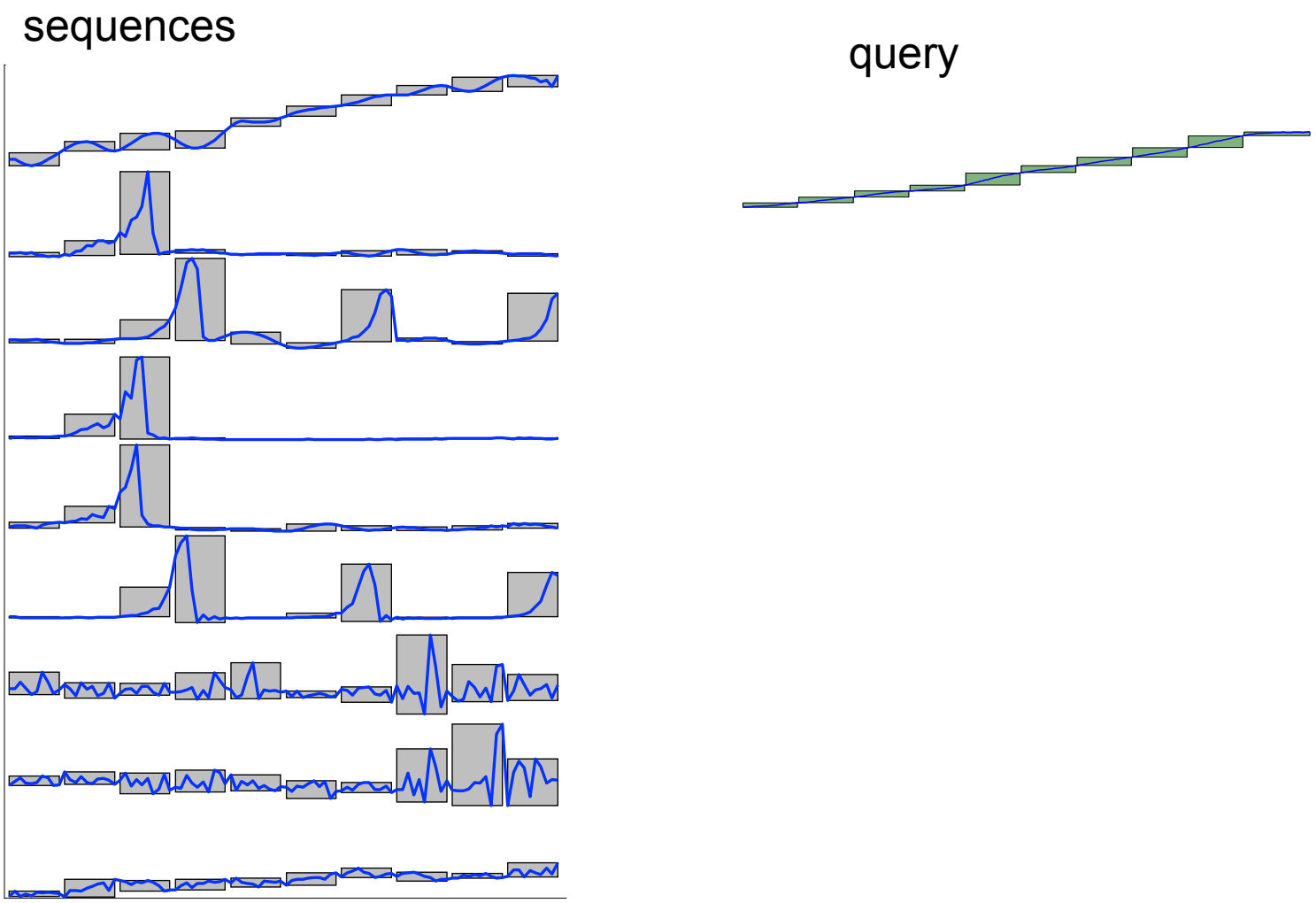
Lower Bounding II

We can speed up similarity search under DTW by using a lower bounding function

```
Algorithm Lower_Bounding_Sequential_Scan(Q)
1.   best_so_far = infinity;
2.   for all sequences in database
3.     LB_dist = lower_bound_distance(C_i, Q);
4.     if LB_dist < best_so_far
5.       true_dist = DTW(C_i, Q);
6.       if true_dist < best_so_far
7.         best_so_far = true_dist;
8.         index_of_best_match = i;
9.       endif
10.    endif
11.  endfor
```

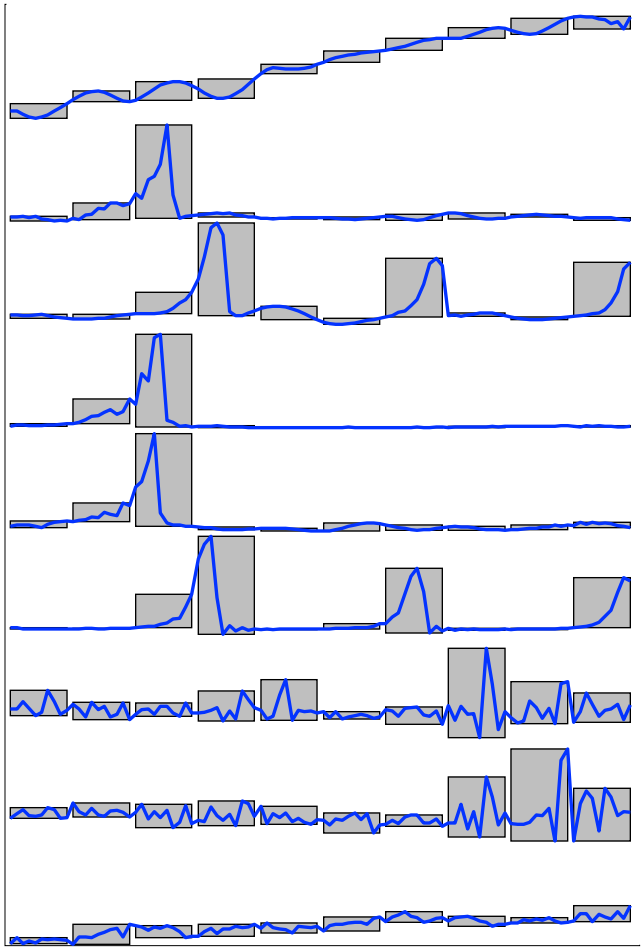


Lower Bounding Example



Lower Bounding Example

sequences



Lower Bounds

4.6399

37.9032

19.5174

72.1846

67.1436

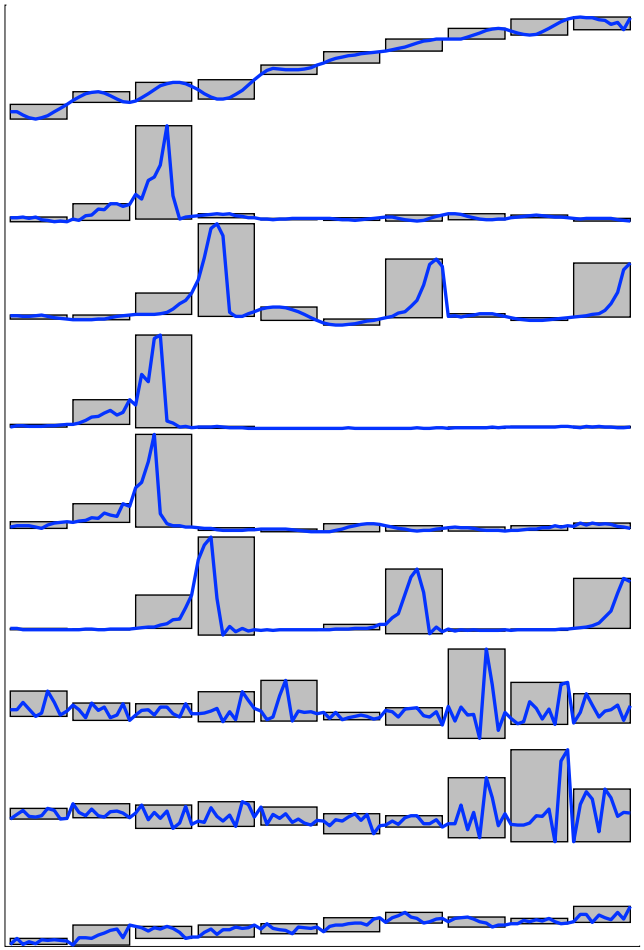
78.0920

70.9273

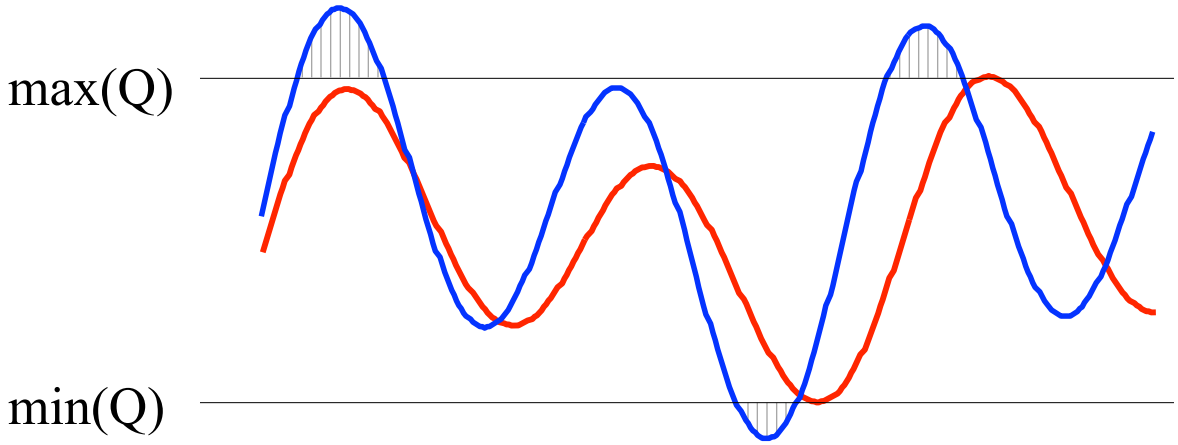
63.7253

1.4121

Lower Bounding Example



Lower Bound of Yi

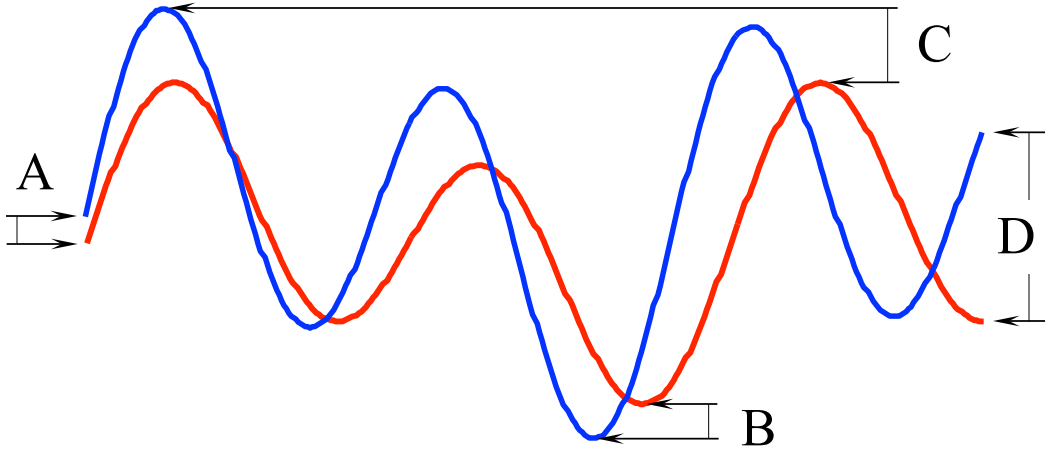


LB_Yi

Yi, B, Jagadish, H & Faloutsos, C. *Efficient retrieval of similar time sequences under time warping*. ICDE 98, pp 23-27.

The sum of the squared length of gray lines represent the minimum corresponding points contribution to the overall DTW distance, and thus can be returned as the lower bounding measure

Lower Bound of Kim

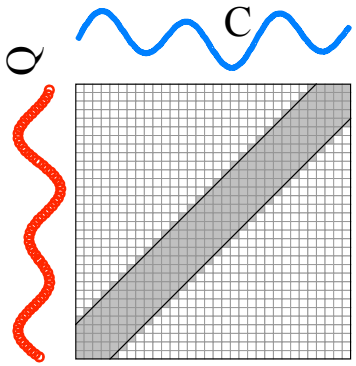


LB_Kim

Kim, S, Park, S, & Chu, W. *An index-based approach for similarity search supporting time warping in large sequence databases*. ICDE 01, pp 607-614

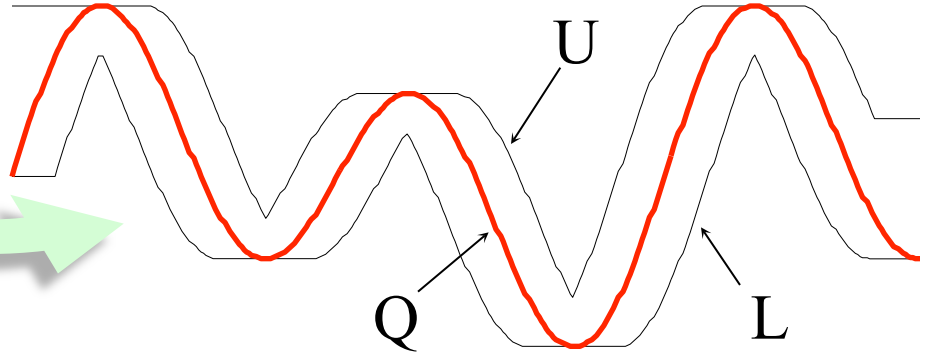
The (maximum) squared difference between the two sequence's first (A), last (D), minimum (B) and maximum points (C) is returned as the lower bound

Lower Bound of Keogh



Sakoe-Chiba Band

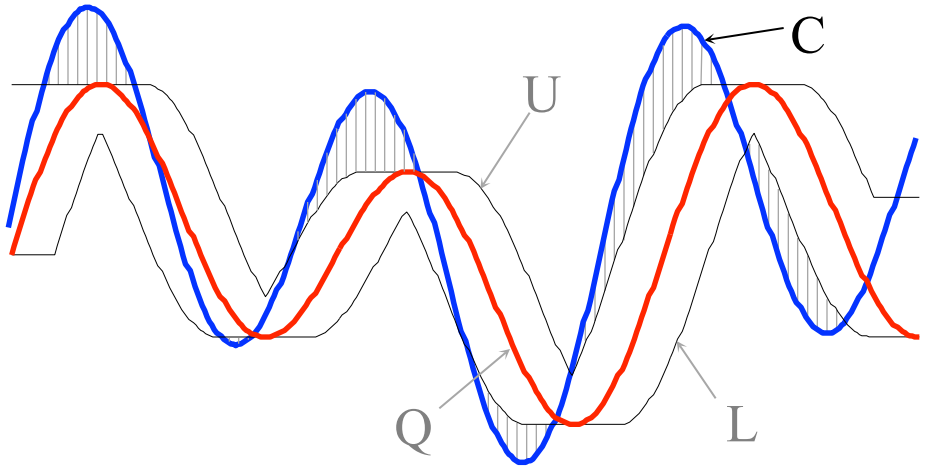
$U_i = \max(q_{i-r} : q_{i+r})$
$L_i = \min(q_{i-r} : q_{i+r})$



LB_Keogh

Envelope-Based Lower Bound

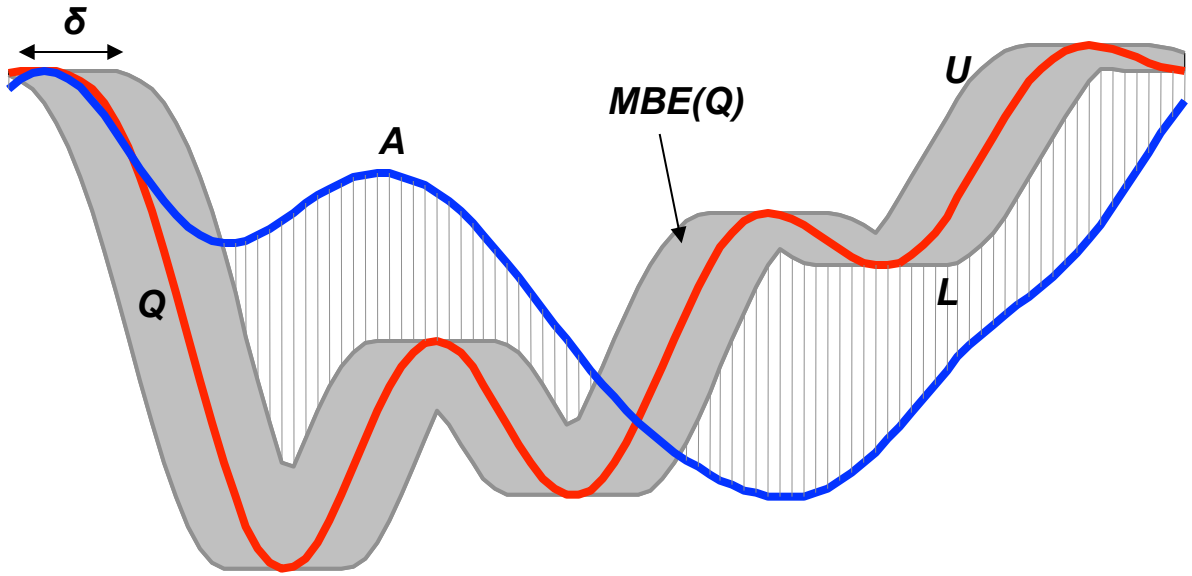
$$LB\_Keogh(Q,C) = \sum_{i=1}^{n} \begin{cases} (C_i - U_i)^2 & if\ C_i > U_i \\ (C_i - L_i)^2 & if\ C_i < L_i \\ 0 & otherwise \end{cases}$$



Lower Bounding the Dynamic Time Warping

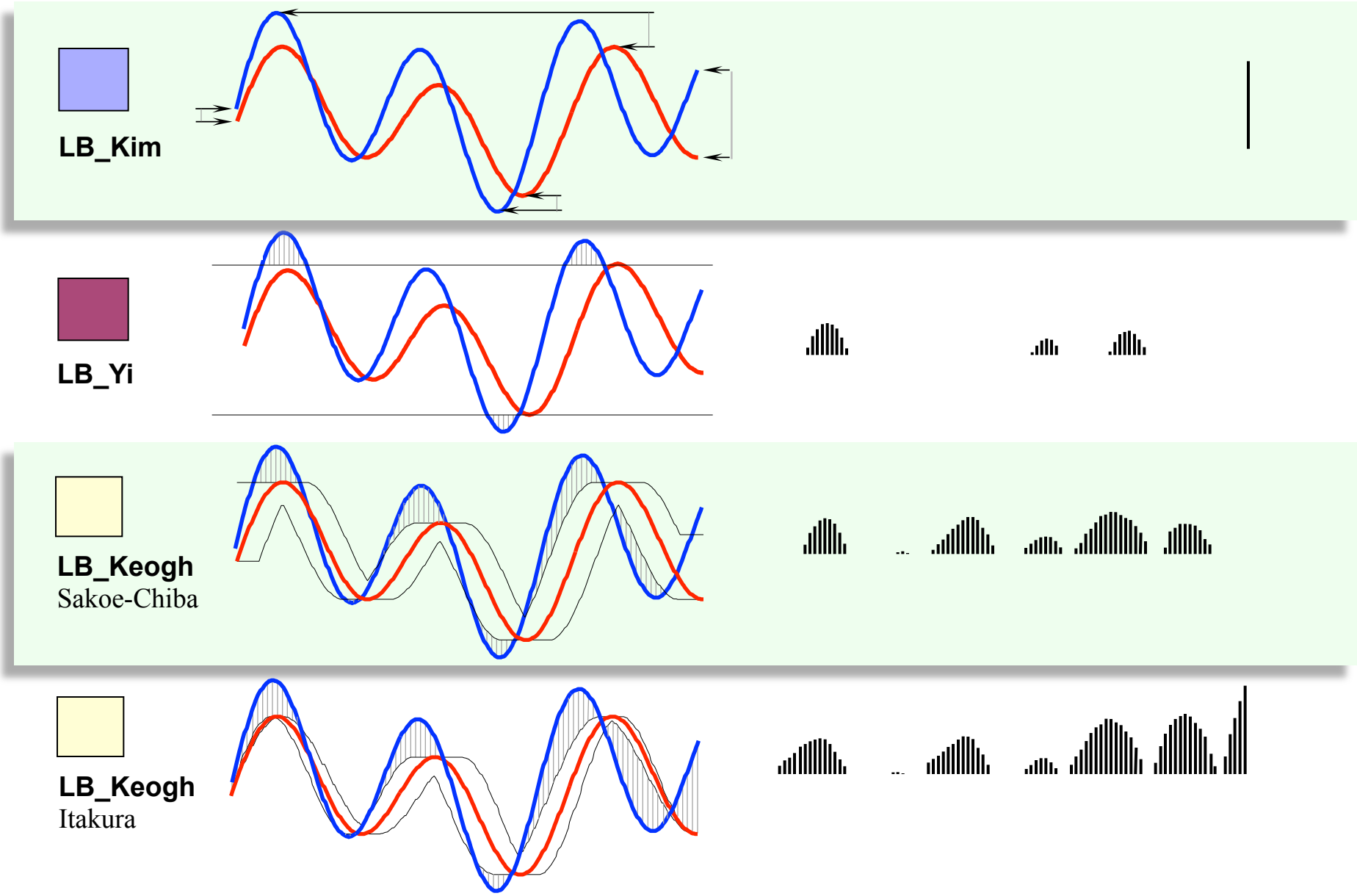
Minimum Bounding Envelope for bounding the DTW

- *Create Minimum Bounding Envelope (MBE) of query Q*
- *Calculate distance between MBE of Q and any sequence A*
- *One can show that:* ***$D(MBE(Q)_\delta, A) < DTW(Q,A)$***



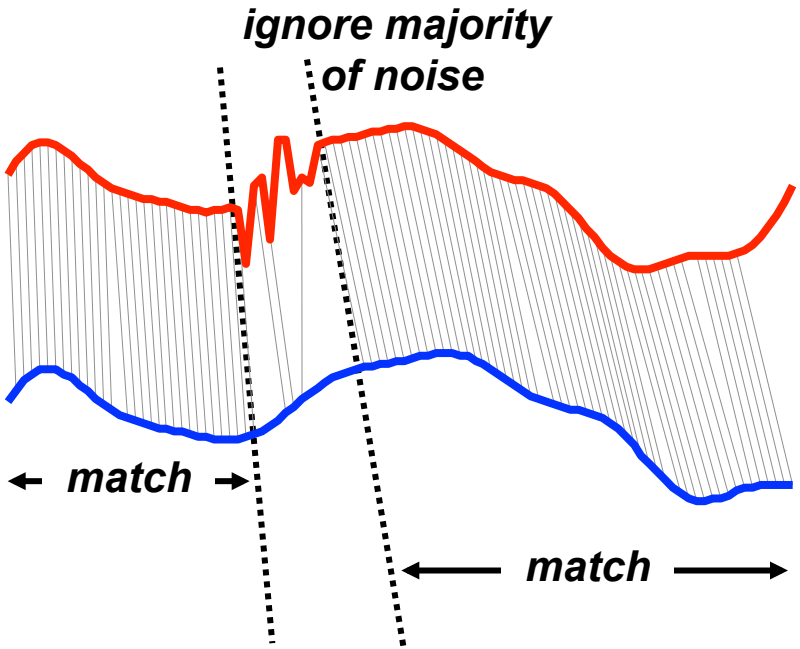

```
LB = sqrt(sum([[A > U].* [A-U]; [A < L].* [L-A]].^2));
```

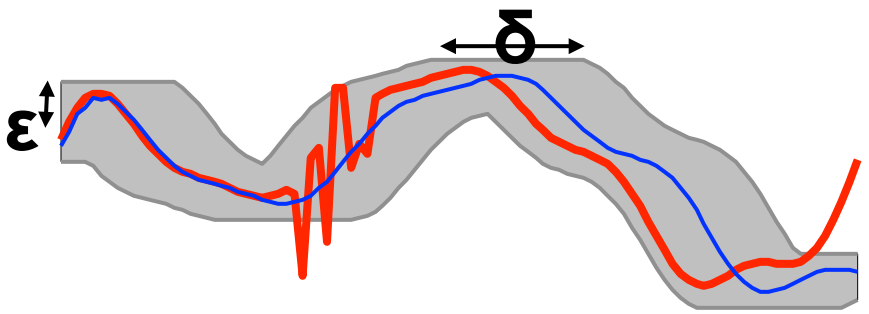
The tightness of the lower bound for each technique is proportional to the length of gray lines used in the illustrations



Longest Common Subsequence (LCSS)

With Time Warping extreme values (outliers) can destroy the distance estimates. The LCSS model can offer more resilience to noise and impose spatial constraints too.



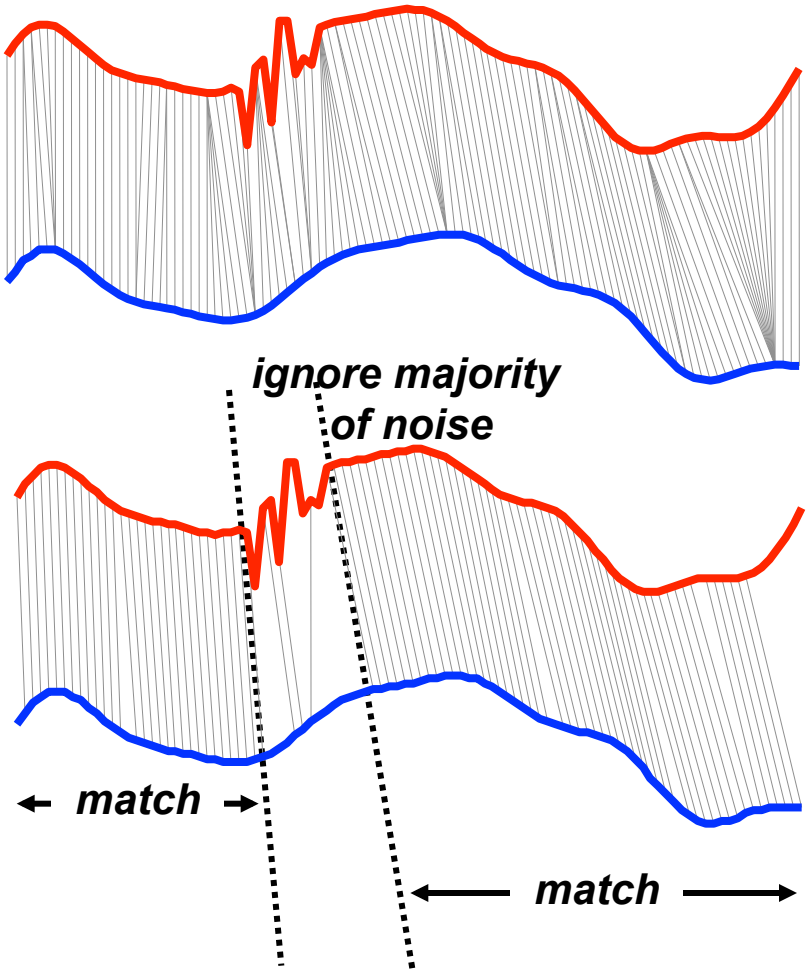


Matching within δ time and ε in space

Everything that is outside the bounding envelope can never be matched

Longest Common Subsequence (LCSS)

LCSS is more resilient to noise than DTW.



Disadvantages of DTW:

A. All points are matched

B. Outliers can distort distance

C. One-to-many mapping

Advantages of LCSS:

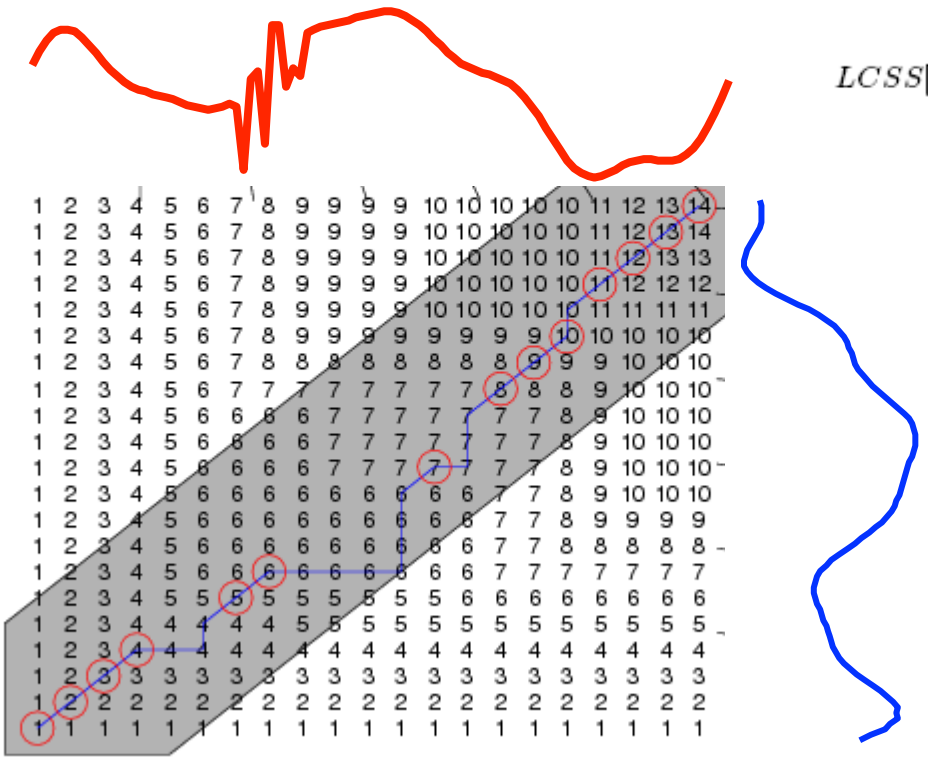
A. Outlying values not matched

B. Distance/Similarity distorted less

C. Constraints in time & space

LCSS (Implementation)

Similar dynamic programming solution as DTW, but now we measure similarity not distance.



$$LCSS[i,j] = \begin{cases} 0 & \text{if} \quad i = 0 \\ 0 & \text{if} \quad j = 0 \\ 1 + LCSS[i-1, j-1] & \text{if} \quad |a_{i,k} - b_{j,k}| < \epsilon, \\ max(LCSS[i-1,j], LCSS[i,j-1]) & \text{otherwise} \end{cases}$$

Can also be expressed as distance

$$D_{LCSS}(A,B) = 1 - \frac{LCSS_{\delta,\epsilon}(A,B)}{\min(n,m) \; or \max(n,m)}$$

Distance Measure Comparison

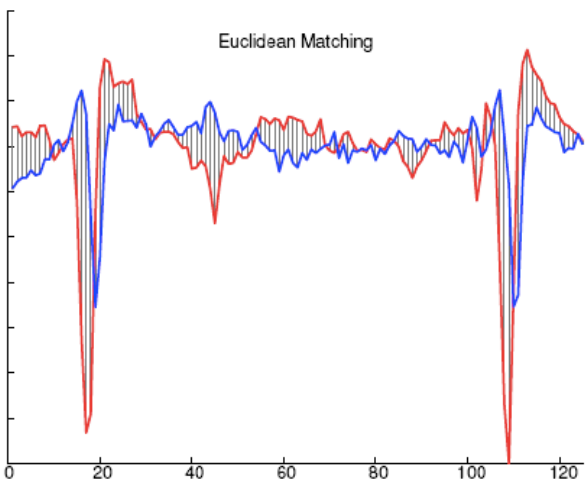
Table 5. Error Rates for each Similarity Measure by Dataset

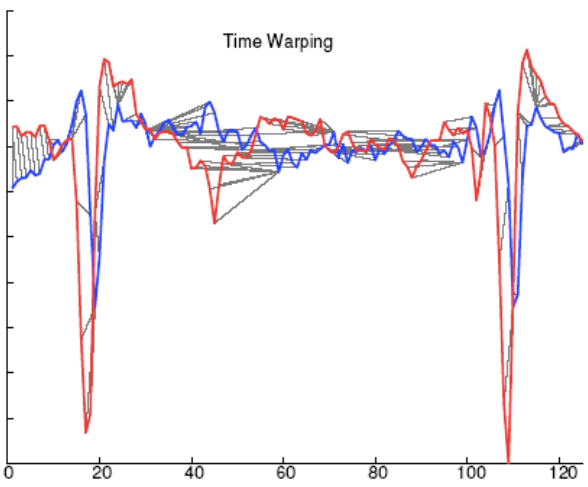
	AUSLAN	Japanese Vowels	Wafer	ECG	Star Light Curve
Euclidean	.1722	.3838	.08333	.1778	.1767
DTW (full)	.1556	**.1063**	.0909	.1889	.1647
DTW (window)	**.1444**	**.1063**	.0656	.1722	**.1566**
EDR	.4167	.1417	.3131	.2000	.1606
ERP	.2778	.2970	.0556	.1944	.3936
LCSS	.4185	.1226	.1363	**.1278**	.1687
LCSS Relaxed	.2259	.1172	.1091	**.1278**	.1687
TWED	.1741	**.1063**	**.0318**	**.1278**	.1606

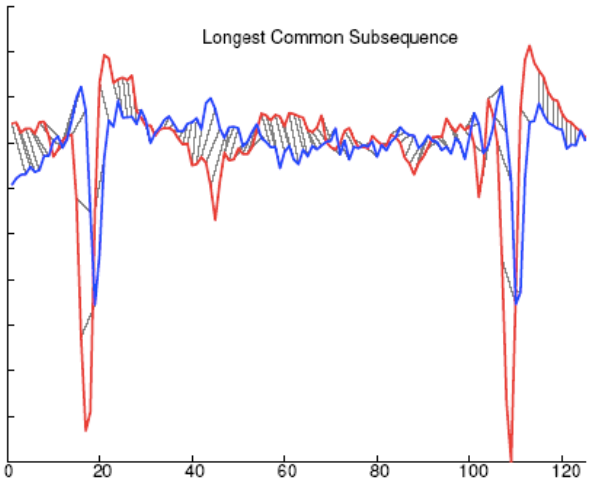
LCSS is the least sensitive to noise because it includes a threshold to define a “match.”

Distance Measure Comparison (Overview)

Method	*Complexity*	*Elastic Matching*	*One-to-one Matching*	*Noise Robustness*
Euclidean	*O(n)*	×	✓	×
DTW	*O(n*δ)*	✓	×	×
LCSS	*O(n*δ)*	✓	✓	✓

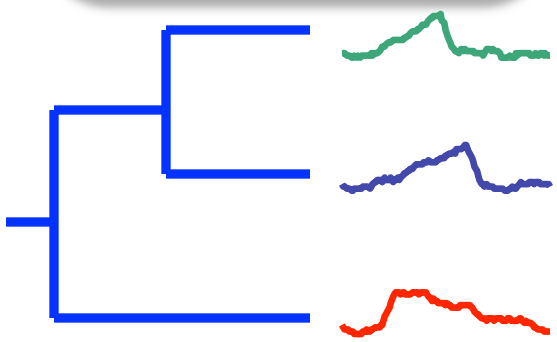




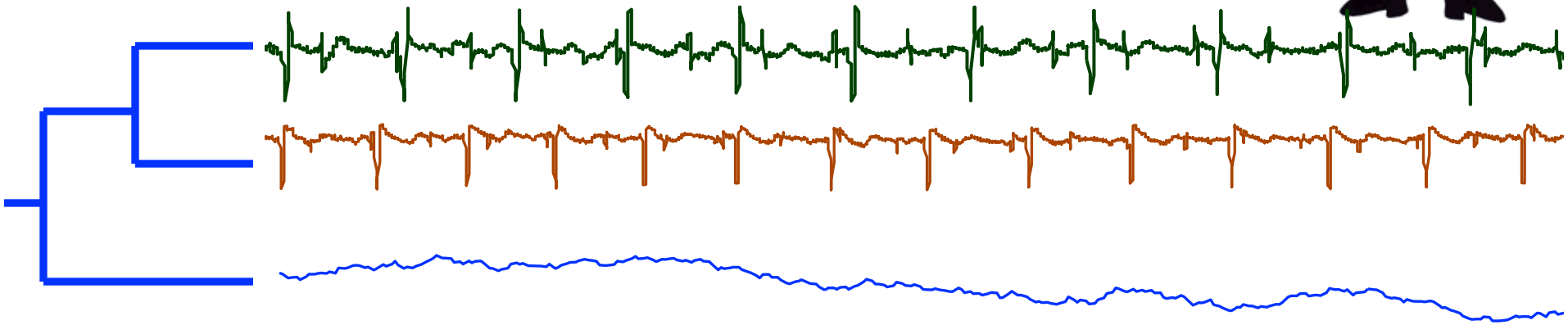


Two Kinds of Similarity

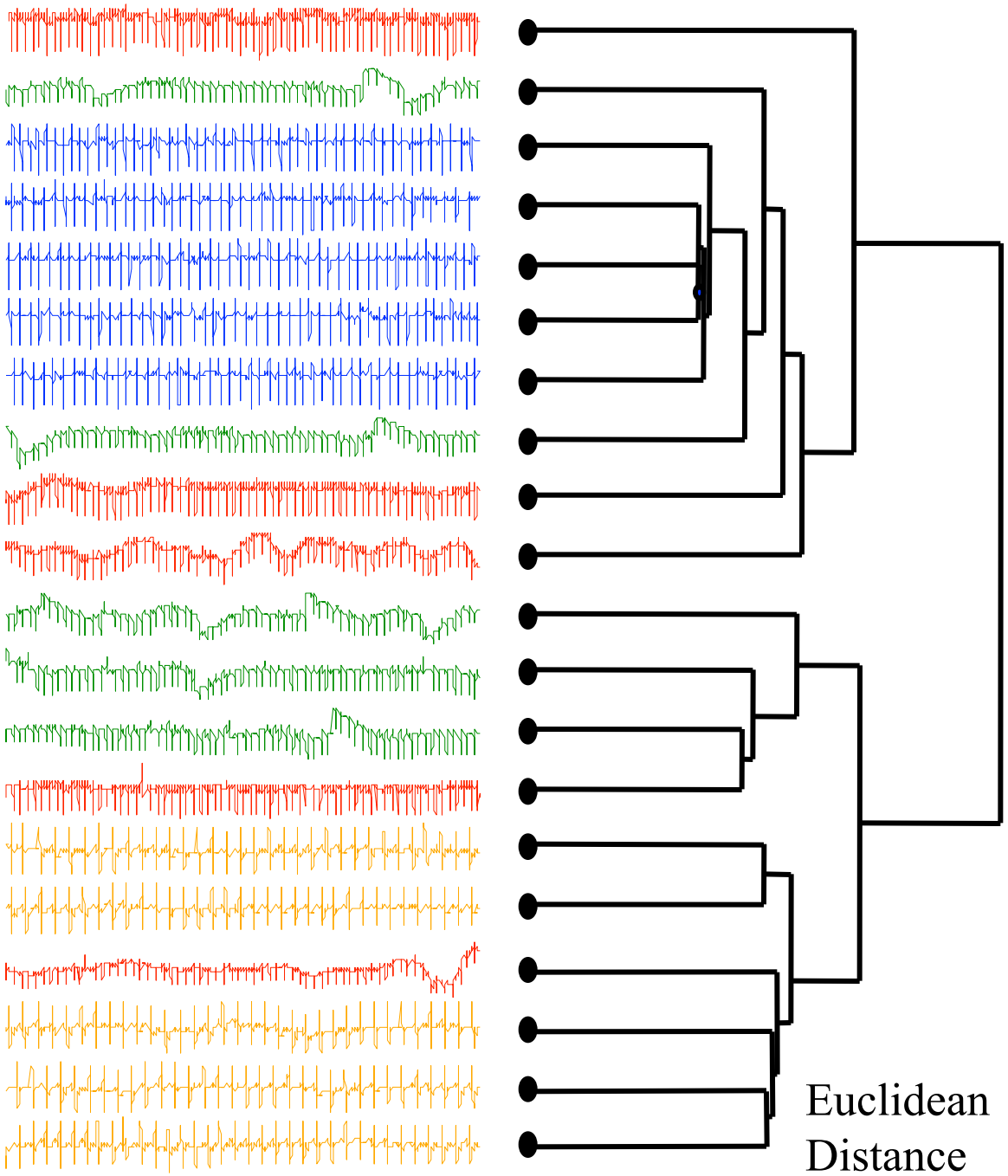
We are done with *shape* similarity



Let us consider similarity at the *structural* level for the next 10 minutes



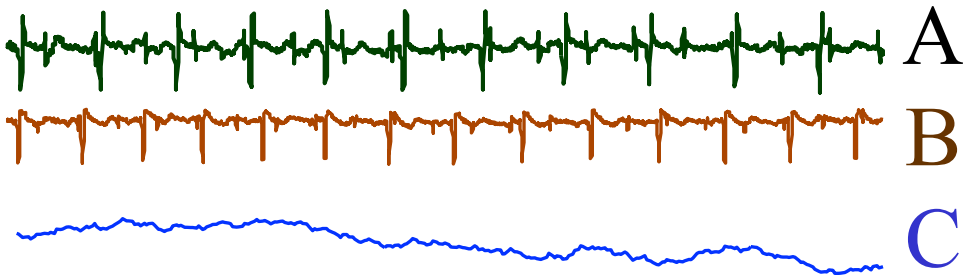
For long time series, *shape* based similarity will give very poor results. We need to measure similarly based on high level *structure*



Structure or Model Based Similarity

The basic idea is to extract *global* features from the time series, create a feature vector, and use these feature vectors to measure similarity and/ or classify

But which
- **features**?
- **distance measure/ learning algorithm**?



Feature \ Time Series	A	B	C
Max Value	11	12	19
Autocorrelation	0.2	0.3	0.5
Zero Crossings	98	82	13
ARIMA	0.3	0.4	0.1
…	…	…	…

Feature-based Classification of Time-series Data

Nanopoulos, Alcock, and Manolopoulos

- features?
- distance measure/ learning algorithm?

Learning Algorithm

multi-layer perceptron neural network

Features
mean
variance
skewness
kurtosis
mean (1st derivative)
variance (1st derivative)
skewness (1st derivative)
kurtosis (1st derivative)

Learning to Recognize Time Series: Combining ARMA Models with Memory-Based Learning

Deng, Moore and Nechyba

- features?
- distance measure/ learning algorithm?

Distance Measure

Euclidean distance (between coefficients)

- Used to detect drunk drivers!
- Independently rediscovered and generalized by Kalpakis et. al. and expanded by Xiong and Yeung

Features

The parameters of the Box Jenkins model.

More concretely, the coefficients of the ARMA model.

Deformable Markov Model Templates for Time Series Pattern Matching

Ge and Smyth

- features?
- distance measure/ learning algorithm?

Features

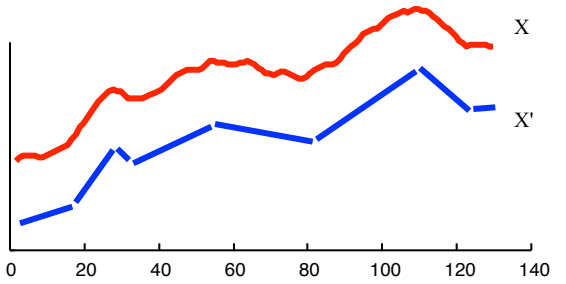
The parameters of a Markov Model

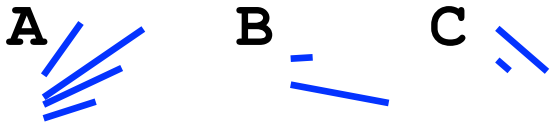
The time series is first converted to a piecewise linear model

Variations independently developed by Li and Biswas, Ge and Smyth, Lin, Orgun and Williams etc

There tends to be lots of parameters to tune...

	A	B	C
A	0.1	0.4	0.5
B	0.4	0.2	0.2
C	0.5	0.2	0.3





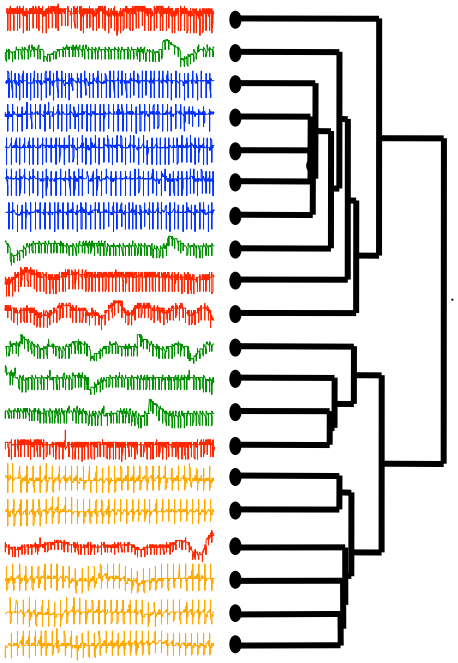
Compression Based Dissimilarity

(**In general**) Li, Chen, Li, Ma, and Vitányi: (**For time series**) Keogh, Lonardi and Ratanamahatana

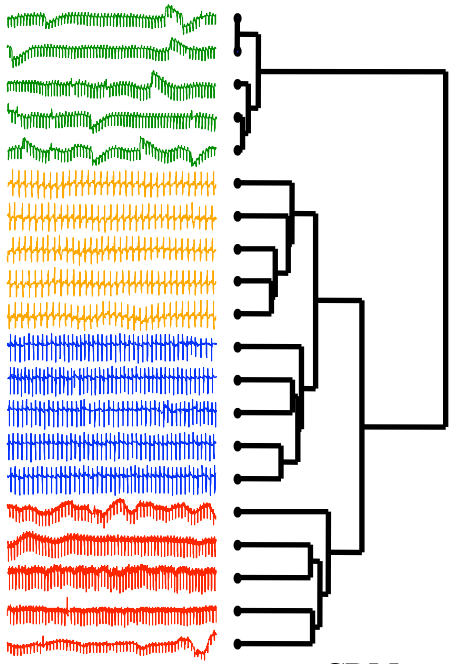
- features?
- distance measure/ learning algorithm?

Distance Measure

Co-Compressibility $$CDM(x,y) = \frac{C(xy)}{C(x)+C(y)}$$



Euclidean



CDM

Features

Whatever structure the compression algorithm finds...

The time series is first converted to the SAX symbolic representation*

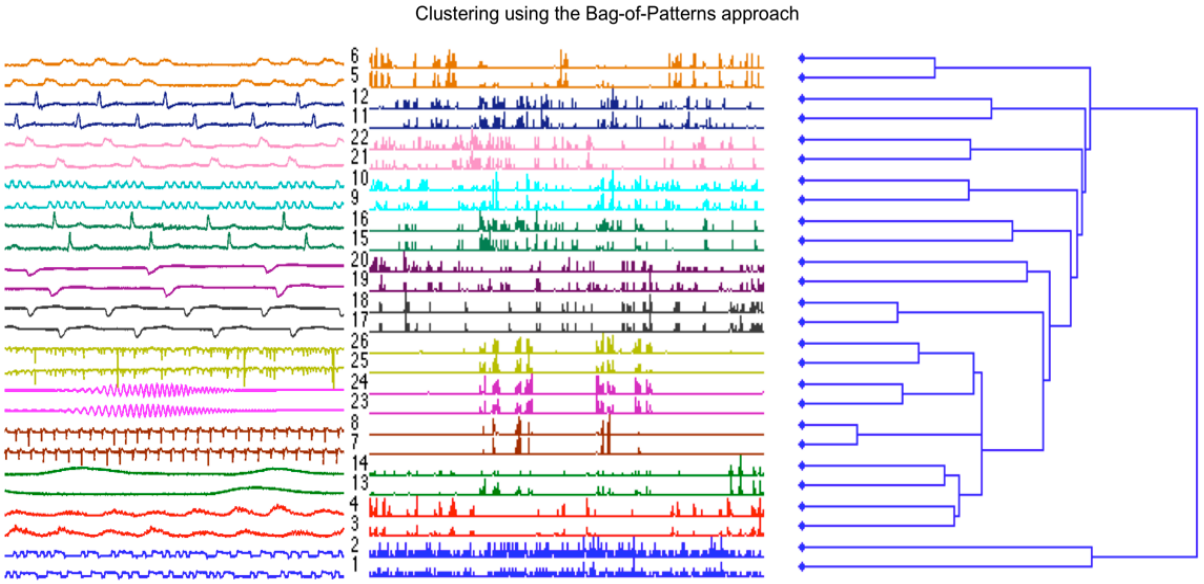
Pattern-Histogram Based Similarity

Lin and Li

- features?
- distance measure/ learning algorithm?

Distance Measure

Euclidean Distance on Bag of Patterns



Features

Represent time series as "bag of patterns" (much like the "bag of words" for text data)

The time series is first converted to the SAX symbolic representation*

Summary of Time Series Similarity

- If you have *short* time series, use DTW after searching over the warping window size (and shape), or LCSS
- Then use envelope based lower bounds to speed things up.

- If you have *long* time series, and you know nothing about your data, try compression based dissimilarity or bag-of-patterns similarity.
- If you do know something about your data, try to leverage of this knowledge to extract features.