Indexing Time Series

Based on original slides by Prof. Dimitrios Gunopulos and Prof. Christos Faloutsos with some slides from tutorials by Prof. Eamonn Keogh and Dr. Michalis Vlachos. Excellent tutorials (and not only) about Time Series can be found there:
http://www.cs.ucr.edu/~eamonn/tutorials.html
A nice tutorial on Matlab and Time series is also there:
http://www.cs.ucr.edu/~mvlachos/ICDM06/

Time Series Databases

- A time series is a sequence of real numbers, representing the measurements of a real variable at equal time intervals
  - Stock prices
  - Volume of sales over time
  - Daily temperature readings
  - ECG data

- A time series database is a large collection of time series

Time Series Data

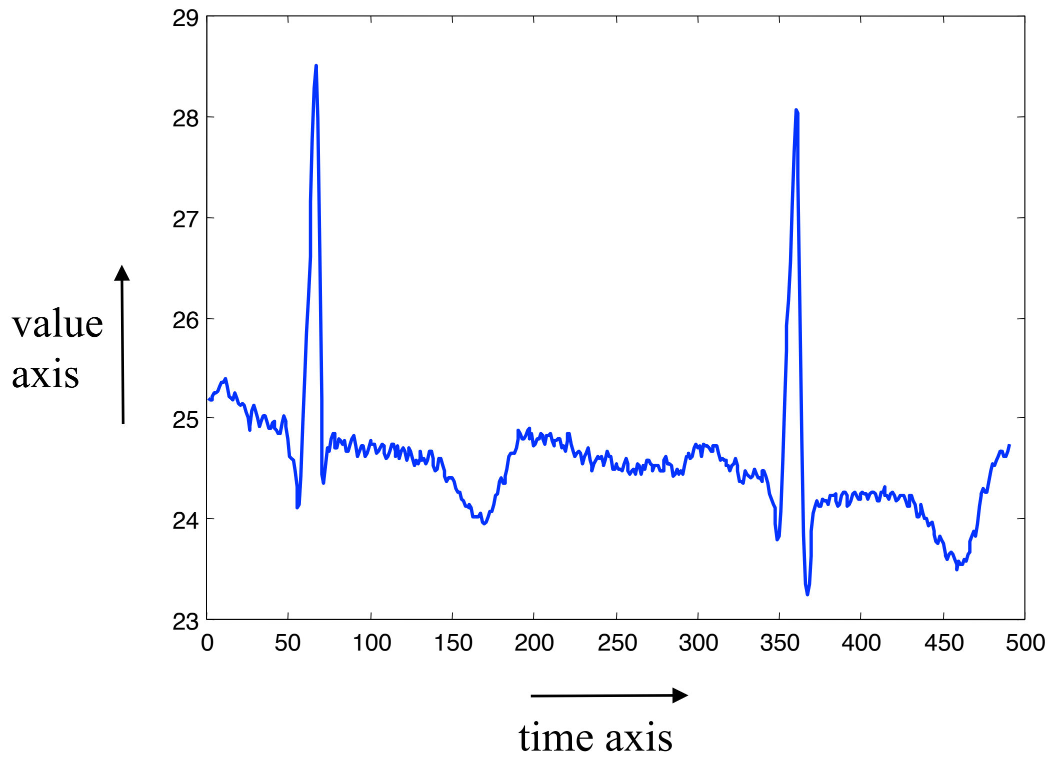
A time series is a collection of observations made sequentially in time.

25.1750
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



Time Series Problems
(from a database perspective)

- The Similarity Problem

  $X = x_1, x_2, \ldots, x_n$ and $Y = y_1, y_2, \ldots, y_n$

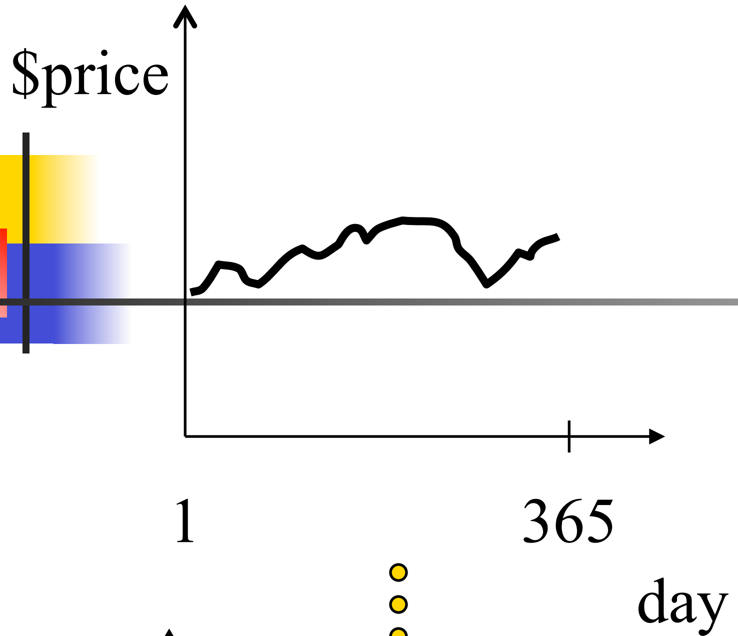
- Define and compute Sim(X, Y)
  - E.g. do stocks X and Y have similar movements?
- Retrieve efficiently similar time series (Indexing for Similarity Queries)

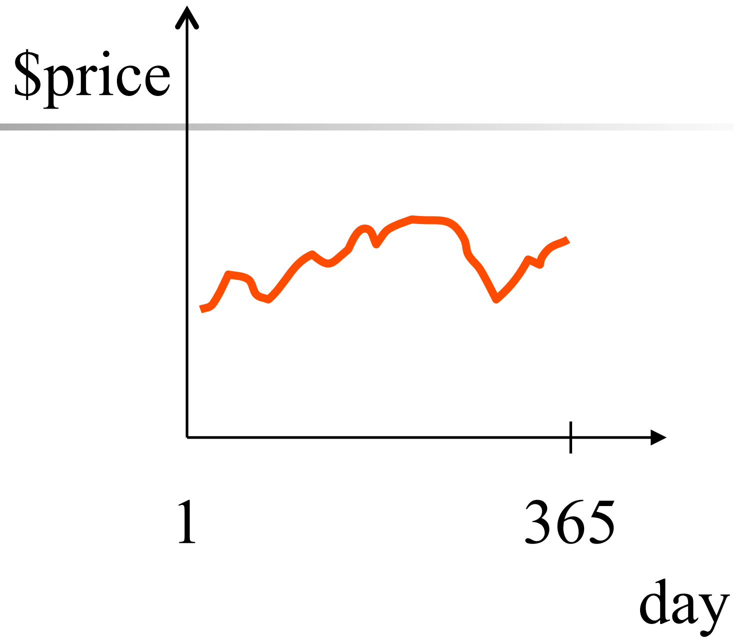
Types of queries

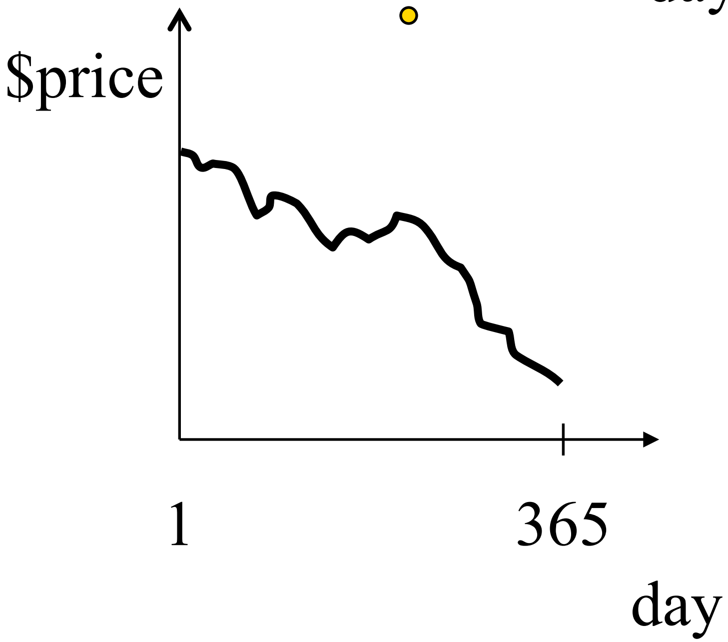
- whole match vs sub-pattern match
- range query vs nearest neighbors
- all-pairs query

Examples

- Find companies with similar stock prices over a time interval
- Find products with similar sell cycles
- Cluster users with similar credit card utilization
- Find similar subsequences in DNA sequences
- Find scenes in video streams







distance function: by expert

(eg, Euclidean distance)

Problems

- Define the similarity (or distance) function
- Find an efficient algorithm to retrieve similar time series from a database
  - (Faster than sequential scan)

The Similarity function depends on the Application

Metric Distances

- What properties should a similarity distance have to allow (easy) indexing?
- $D(A,B) = D(B,A)$ *Symmetry*
- $D(A,A) = 0$ *Constancy of Self-Similarity*
- $D(A,B) >= 0$ *Positivity*
- $D(A,B) \leq D(A,C) + D(B,C)$ *Triangular Inequality*

- Some times the distance function that best fits an application is not a metric… then indexing becomes interesting….

Euclidean Similarity Measure

- View each sequence as a point in n-dimensional Euclidean space (n = length of each sequence)
- Define (dis-)similarity between sequences X and Y as

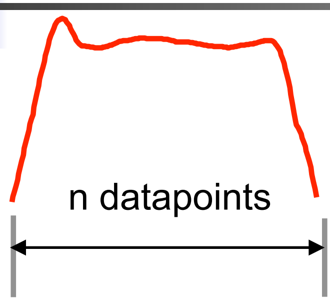
$$\boxed{L_p = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}}$$

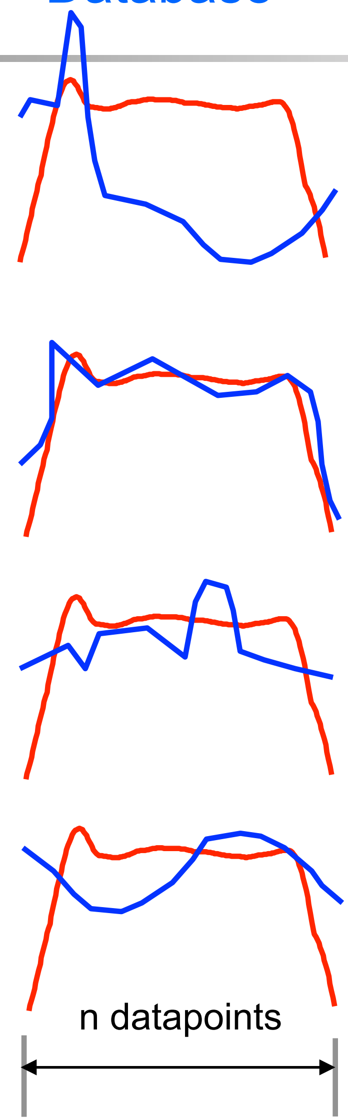
p=1 Manhattan distance

p=2 Euclidean distance

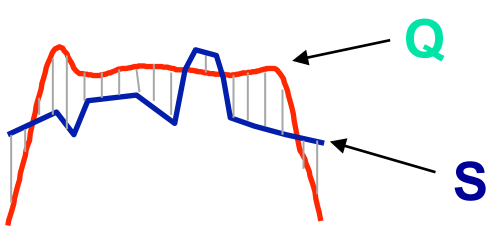
Euclidean model

Query Q



Database	Distance	Rank
		
	0.98	4
	0.07	1
	0.21	2
	0.43	3

Euclidean Distance between two time series $Q = \{q_1, q_2, \ldots, q_n\}$ and $S = \{s_1, s_2, \ldots, s_n\}$



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

Advantages

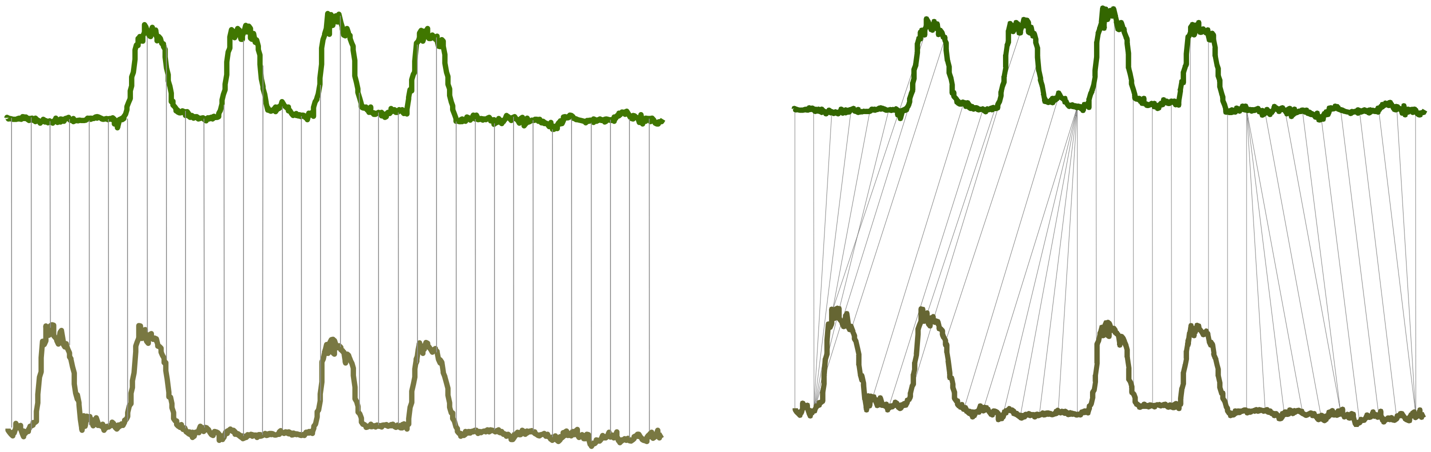
- Easy to compute: O(n)
- Allows scalable solutions to other problems, such as
  - indexing
  - clustering
  - etc...

Dynamic Time Warping
[Berndt, Clifford, 1994]

- Allows acceleration-deceleration of signals along the time dimension

- Basic idea
  - Consider $X = x_1, x_2, \ldots, x_n$, and $Y = y_1, y_2, \ldots, y_n$
  - We are allowed to extend each sequence by repeating elements
  - Euclidean distance now calculated between the extended sequences X' and Y'
  - Matrix M, where $m_{ij} = d(x_i, y_j)$

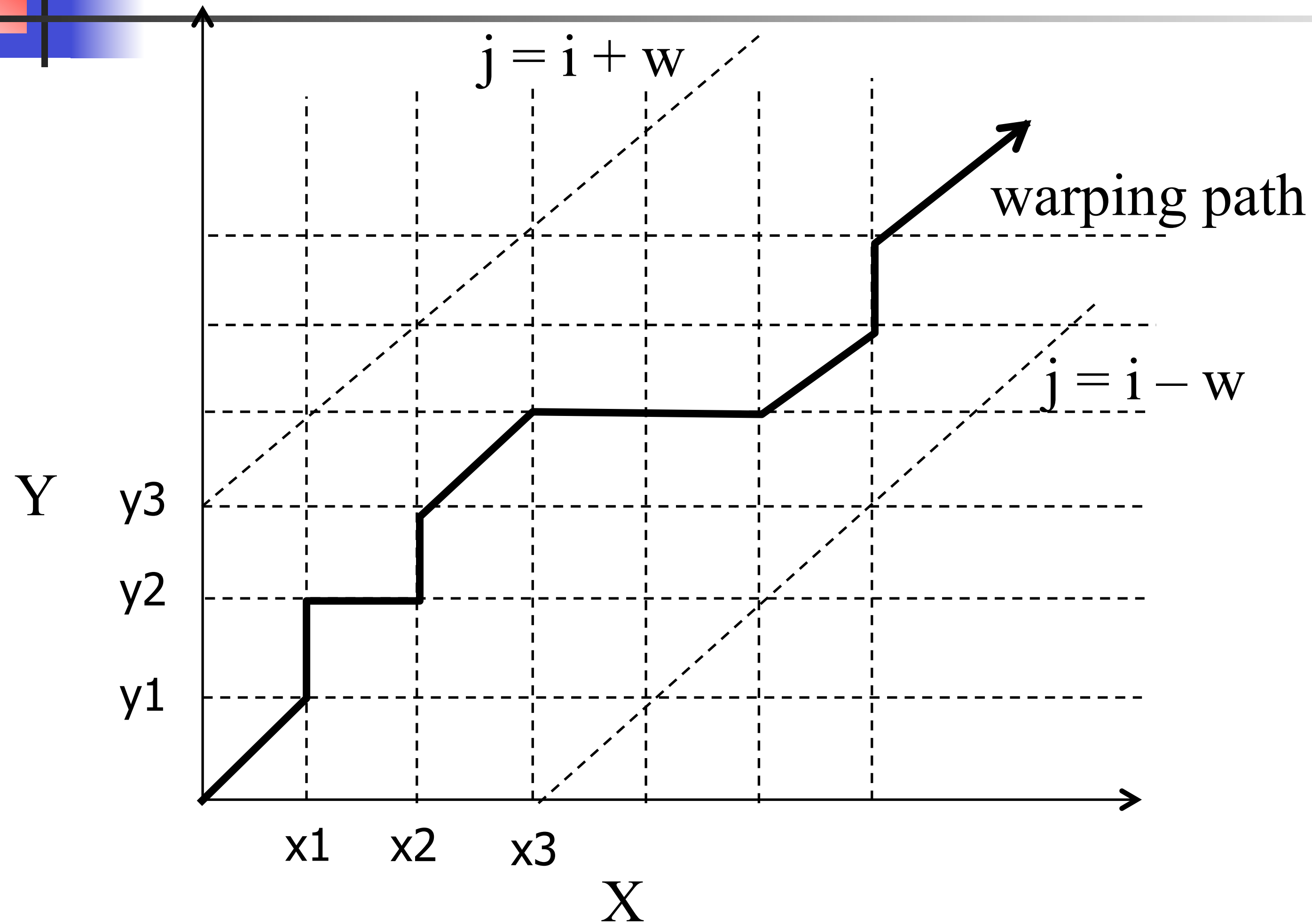
Example



Euclidean distance vs DTW

Dynamic Time Warping

[Berndt, Clifford, 1994]



Restrictions on Warping Paths

- Monotonicity
  - Path should not go down or to the left
- Continuity
  - No elements may be skipped in a sequence
- Warping Window

$$| i - j | <= w$$

Example

	s1	s2	s3	s4	s5	s6	s7	s8	s9
q1	3.76	8.07	1.64	1.08	2.86	0.00	0.06	1.88	1.25
q2	2.02	5.38	0.58	2.43	4.88	0.31	0.59	3.57	2.69
q3	6.35	11.70	3.46	0.21	1.23	0.29	0.11	0.62	0.29
q4	16.8	25.10	11.90	1.28	0.23	4.54	3.69	0.64	1.10
q5	3.20	7.24	1.28	1.42	3.39	0.04	0.16	2.31	1.61
q6	3.39	7.51	1.39	1.30	3.20	0.02	0.12	2.16	1.49
q7	4.75	9.49	2.31	0.64	2.10	0.04	0.00	1.28	0.77
q8	0.96	3.53	0.10	4.00	7.02	1.00	1.46	5.43	4.33
q9	0.02	1.08	0.27	8.07	12.18	3.39	4.20	10.05	8.53

Matrix of the pair-wise distances for element si with qj

Example

	$s1$	$s2$	$s3$	$s4$	$s5$	$s6$	$s7$	$s8$	$s9$
$q1$	**3.76**	11.83	13.47	14.55	17.41	17.41	17.47	19.35	20.60
$q2$	5.78	**9.14**	**9.72**	12.15	17.03	17.34	17.93	21.04	22.04
$q3$	12.13	17.48	12.60	**9.93**	11.16	11.45	11.56	12.18	12.47
$q4$	29.02	37.23	24.50	11.21	**10.16**	14.70	15.14	12.20	13.28
$q5$	32.22	36.26	25.78	12.63	13.55	**10.20**	10.36	12.67	13.81
$q6$	35.61	39.73	27.17	13.93	15.83	**10.22**	10.32	12.48	13.97
$q7$	40.36	45.10	29.48	14.57	16.03	10.26	**10.22**	11.50	12.27
$q8$	41.32	43.89	29.58	18.57	21.59	11.26	11.68	**15.65**	15.83
$q9$	41.34	42.40	29.85	26.64	30.75	14.65	15.46	21.73	**24.18**

Matrix computed with Dynamic Programming based on the:
dist(i,j) = dist(si, yj) + min {dist(i-1,j-1), dist(i, j-1), dist(i-1,j))

Formulation

- Let D(i, j) refer to the dynamic time warping distance between the subsequences

  $x_1, x_2, \ldots, x_i$

  $y_1, y_2, \ldots, y_j$

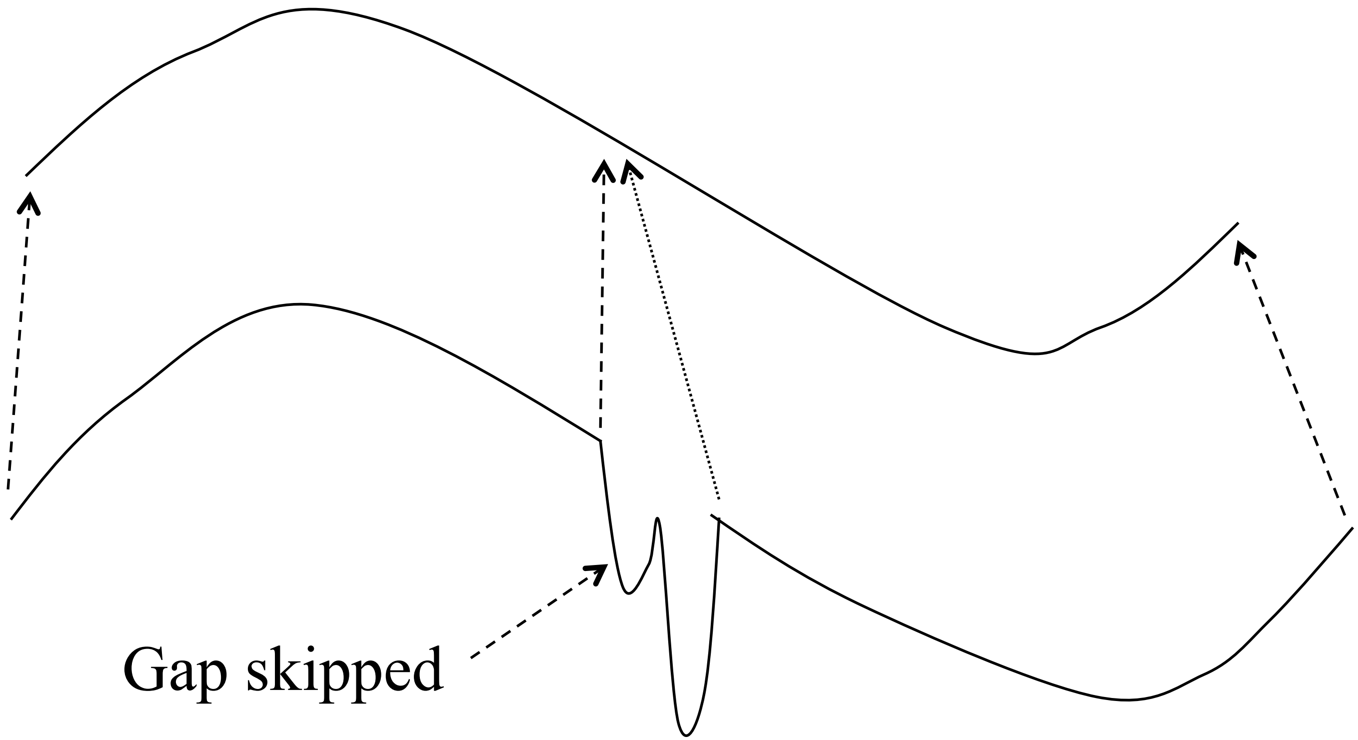
$$D(i, j) = | x_i - y_j | + \min\{ D(i-1, j), D(i-1, j-1), D(i, j-1) \}$$

Solution by Dynamic Programming

- Basic implementation = $O(n^2)$ where n is the length of the sequences
  - will have to solve the problem for each (i, j) pair
- If warping window is specified, then O(nw)
  - Only solve for the (i, j) pairs where $| i – j | <= w$

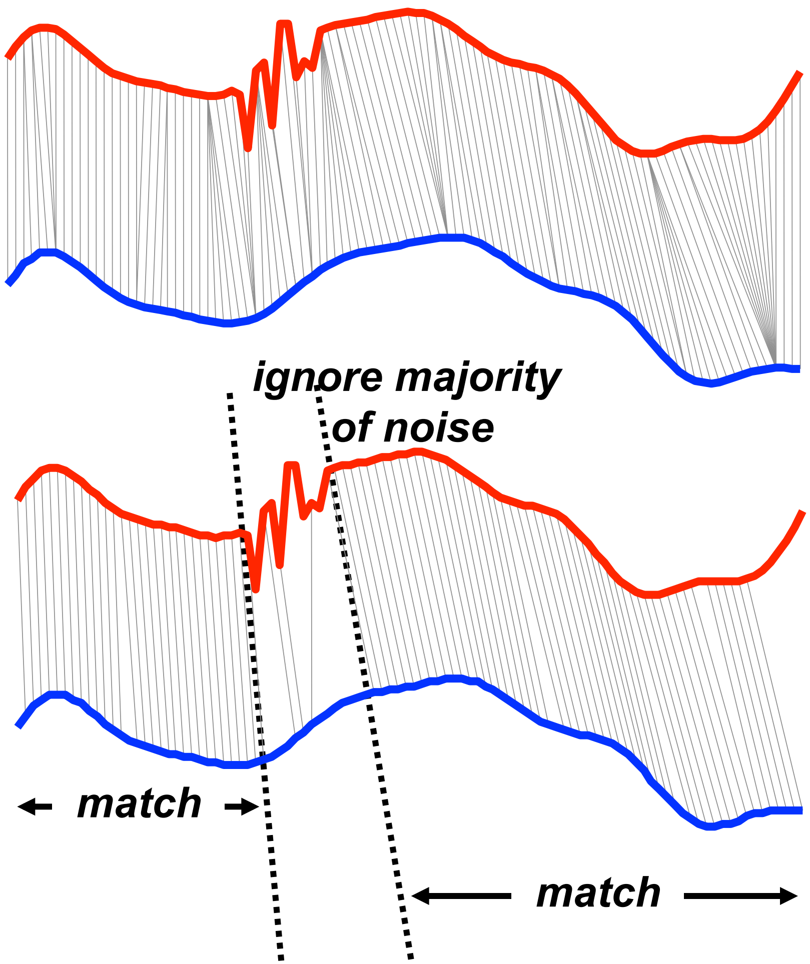
Longest Common Subsequence Measures

(Allowing for Gaps in Sequences)



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

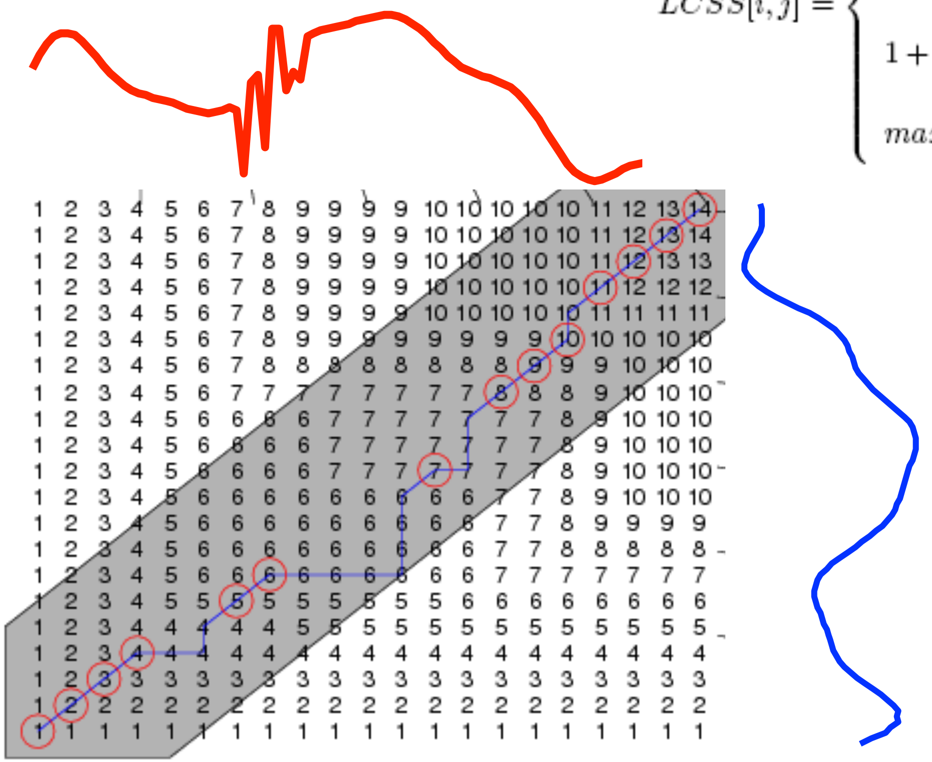
B. Distance/Similarity distorted less

C. Constraints in time & space

Longest Common Subsequence

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

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



Can also be expressed as distance

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

Similarity Retrieval

- Range Query
  - Find all time series S where $D(Q,S) \leq \varepsilon$
- Nearest Neighbor query
  - Find all the k most similar time series to Q
- A method to answer the above queries: Linear scan … very slow

- A better approach GEMINI

GEMINI

Solution: Quick-and-dirty' filter:

- extract *m* features (numbers, eg., avg., etc.)
- map into a point in *m*-d feature space
- organize points with off-the-shelf spatial access method (‘SAM’)
- retrieve the answer using a NN query
- discard false alarms

GEMINI Range Queries

Build an index for the database in a feature space using an R-tree

Algorithm RangeQuery(Q, ε)

1. Project the query Q into a point q in the feature space
2. Find all candidate objects in the index within ε
3. Retrieve from disk the actual sequences
4. Compute the actual distances and discard false alarms

GEMINI NN Query

Algorithm K_NNQuery(Q, K)

1. Project the query Q in the same feature space
2. Find the candidate K nearest neighbors in the index
3. Retrieve from disk the actual sequences pointed to by the candidates
4. Compute the actual distances and record the maximum
5. Issue a RangeQuery(Q, εmax)
6. Compute the actual distances, return best K

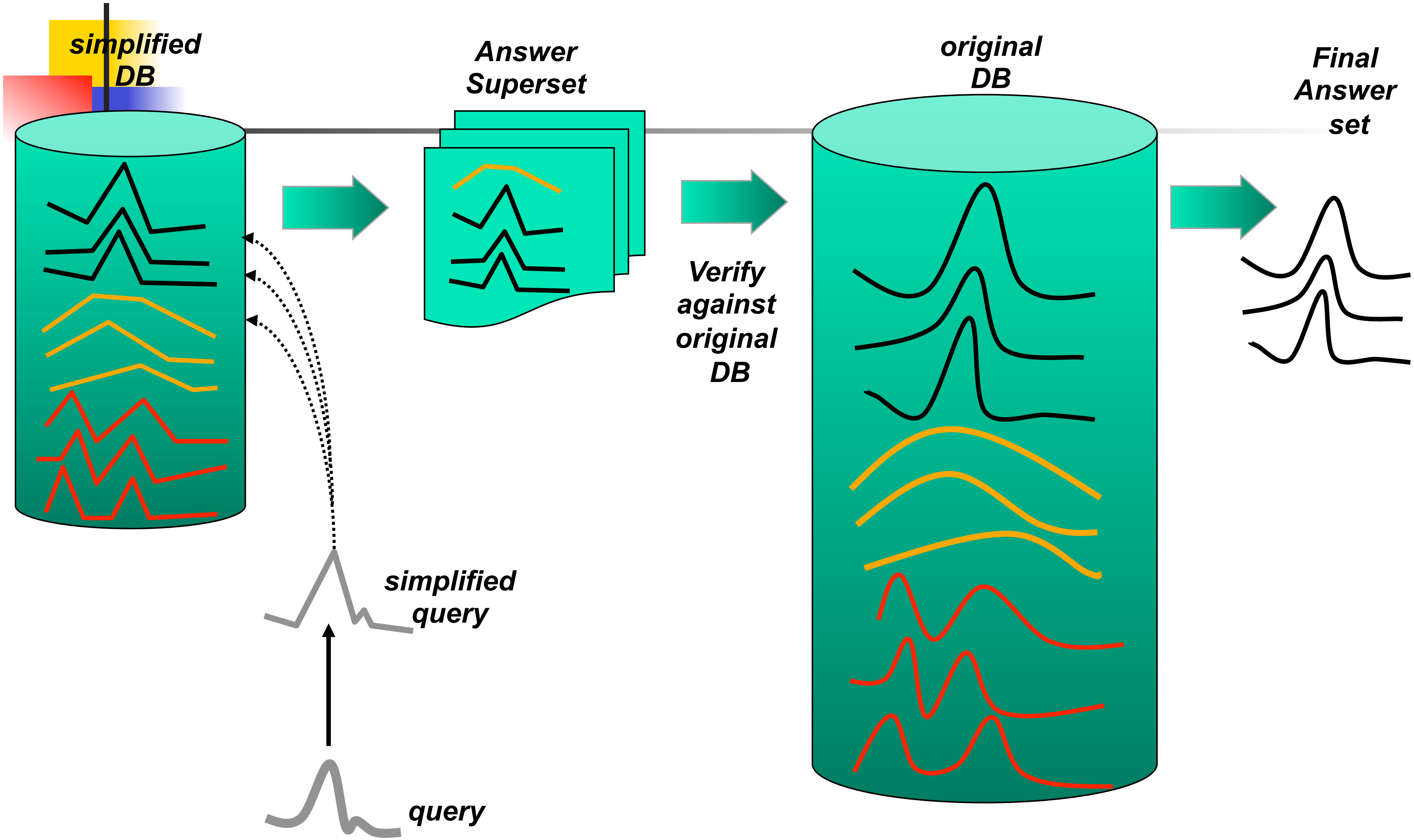
GEMINI

- GEMINI works when:

  $D_{feature}(F(x), F(y)) <= D(x, y)$

- *Note that, the closer the feature distance to the actual one, the better.*

Generic Search using Lower Bounding



Problem

- How to extract the features? How to define the feature space?

- Fourier transform
- Wavelets transform
- Averages of segments (Histograms or APCA)
- Chebyshev polynomials
- .... your favorite curve approximation...

Fourier transform

- DFT (Discrete Fourier Transform)
- Transform the data from the time domain to the frequency domain
- highlights the periodicities
- SO?

DFT

A: several real sequences are periodic

Q: Such as?

A:

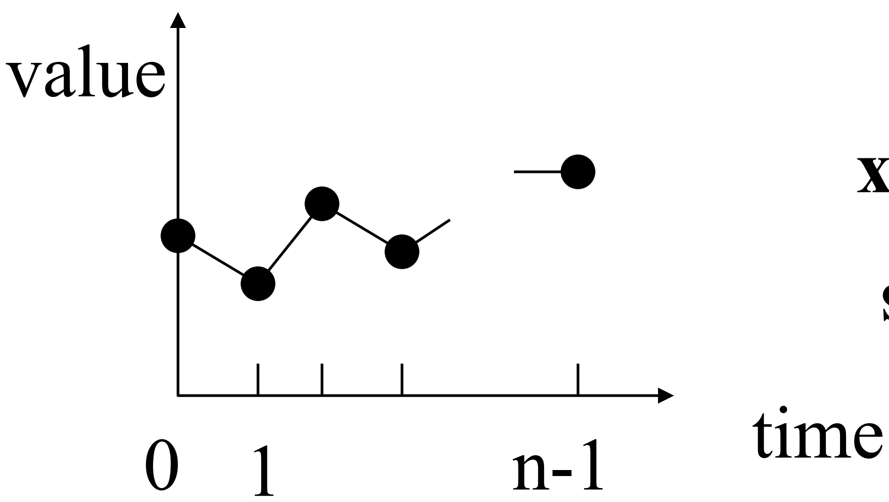
- sales patterns follow seasons;
- economy follows 50-year cycle (or 10?)
- temperature follows daily and yearly cycles

Many real signals follow (multiple) cycles

How does it work?

Decomposes signal to a sum of sine and cosine waves.

Q:How to assess 'similarity' of **x** with a (discrete) wave?



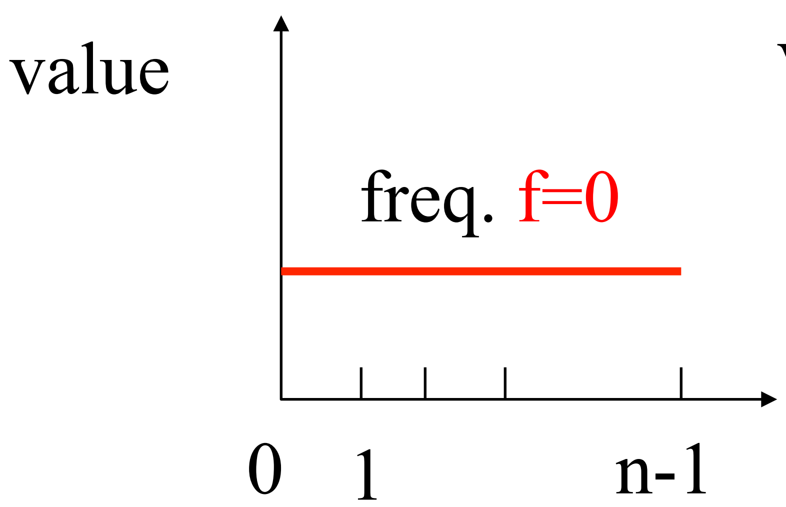
$$\mathbf{x} = \{x_0, x_1, \ldots x_{n-1}\}$$

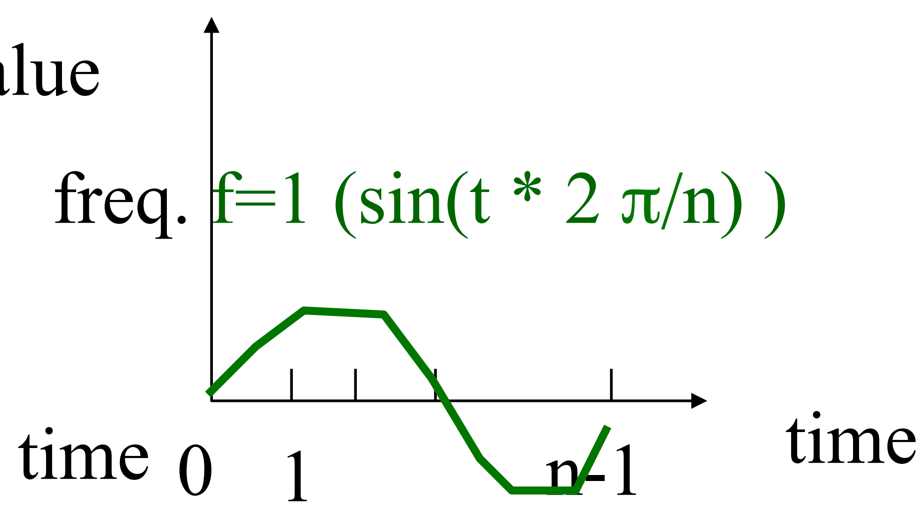
$$\mathbf{s} = \{s_0, s_1, \ldots s_{n-1}\}$$

How does it work?

A: consider the waves with frequency 0, 1, ...; use the inner-product (~cosine similarity)

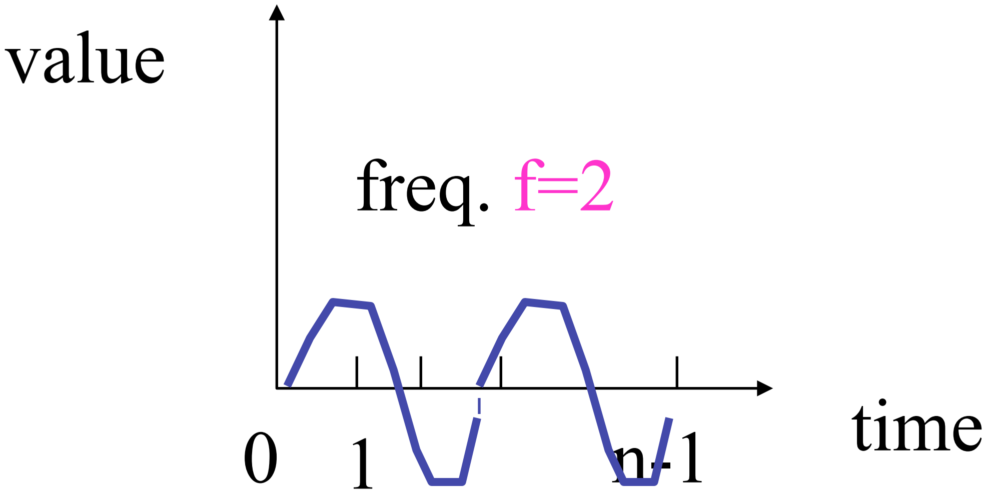
Freq=1/period





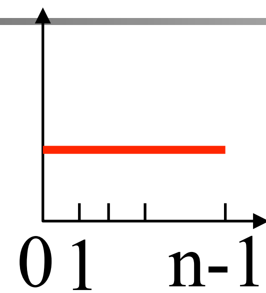
How does it work?

A: consider the waves with frequency 0, 1, ...; use the inner-product (~cosine similarity)

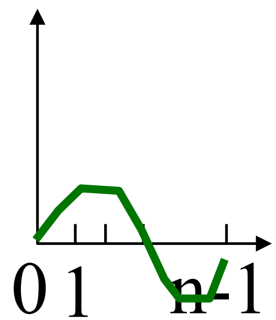


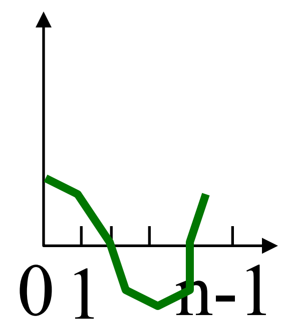
How does it work?

<u>**'basis' functions**</u>



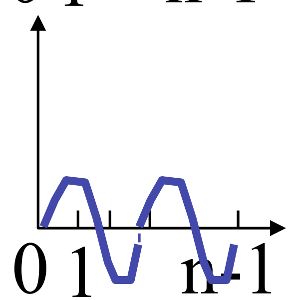
sine, freq =1

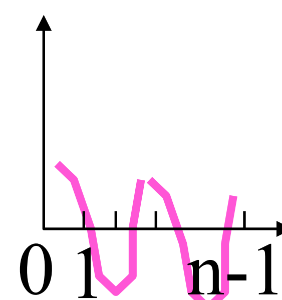




cosine, f=1

sine, freq = 2





cosine, f=2

How does it work?

- Basis functions are actually n-dim vectors, **orthogonal** to each other
- ‘similarity’ of **x** with each of them: inner product
- DFT: ~ all the similarities of **x** with the basis functions

How does it work?

Since $e^{jf} = cos(f) + j\ sin(f)$ (j=$sqrt$(-1)),

we finally have:

DFT: definition

- Discrete Fourier Transform (n-point):

$$X_f = 1/\sqrt{n} \sum_{t=0}^{n-1} x_t * \exp(-j 2\pi\, tf / n)$$

$$(j = \sqrt{-1}\,)$$

inverse DFT

$$x_t = 1/\sqrt{n} \sum_{t=0}^{n-1} X_f * \exp(+j 2\pi\, tf / n)$$

DFT: properties

Observation - SYMMETRY property:

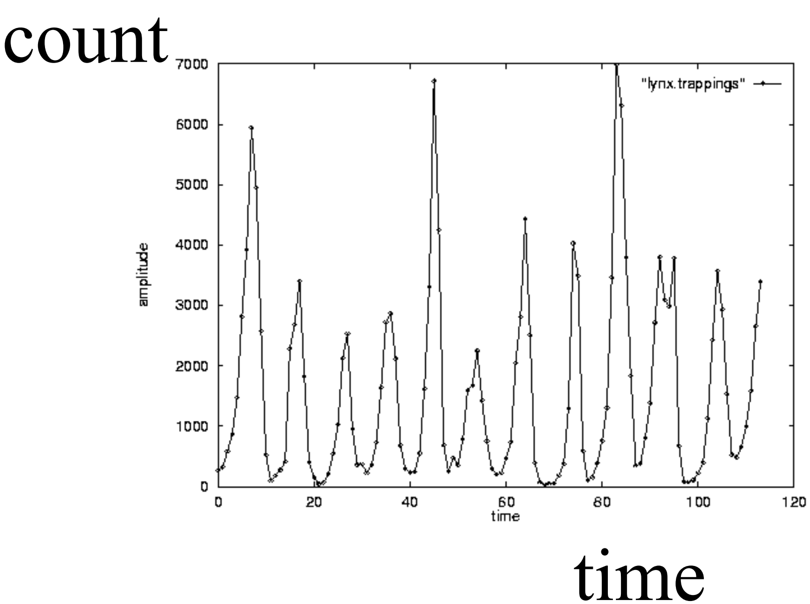
$$X_f = (X_{n-f})^*$$

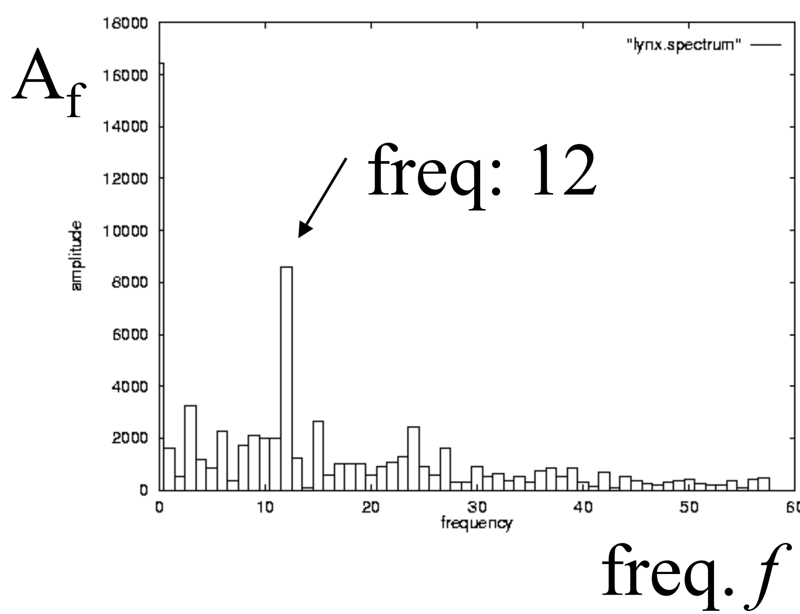
(“*”: complex conjugate*: (a + b j)* = a - b j)*

Thus we use only the first half numbers

DFT: Amplitude spectrum

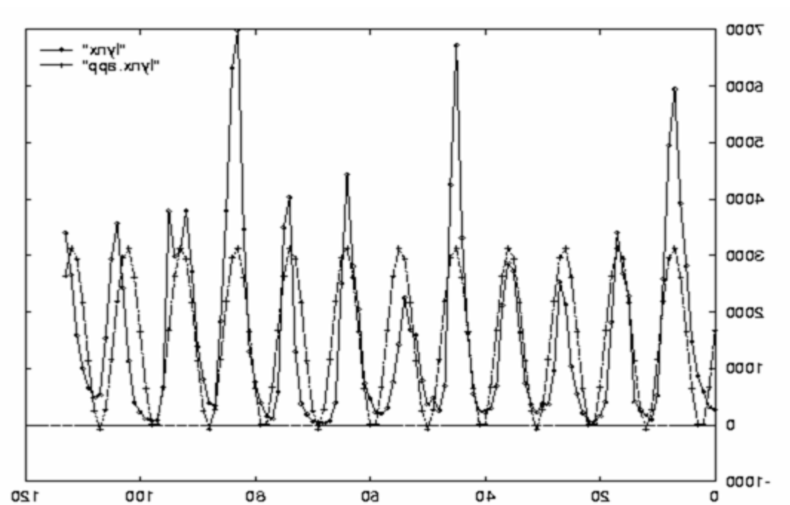
- Amplitude $$A_f^{\,2} = \mathrm{Re}^2(X_f) + \mathrm{Im}^2(X_f)$$
- Intuition: strength of frequency 'f'

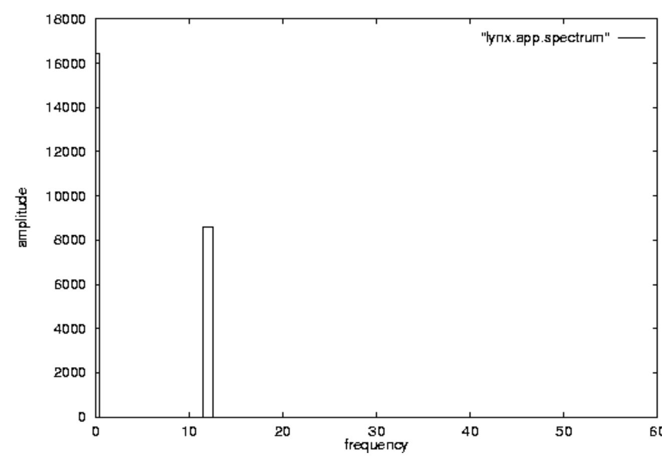


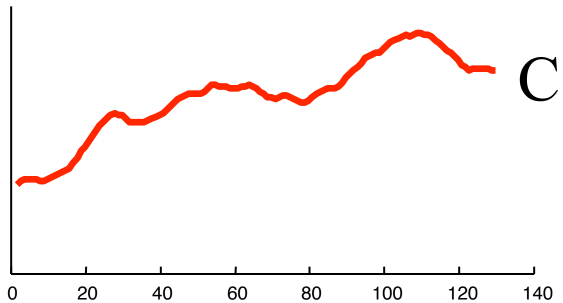


DFT: Amplitude spectrum

- excellent approximation, with only 2 frequencies!
- so what?







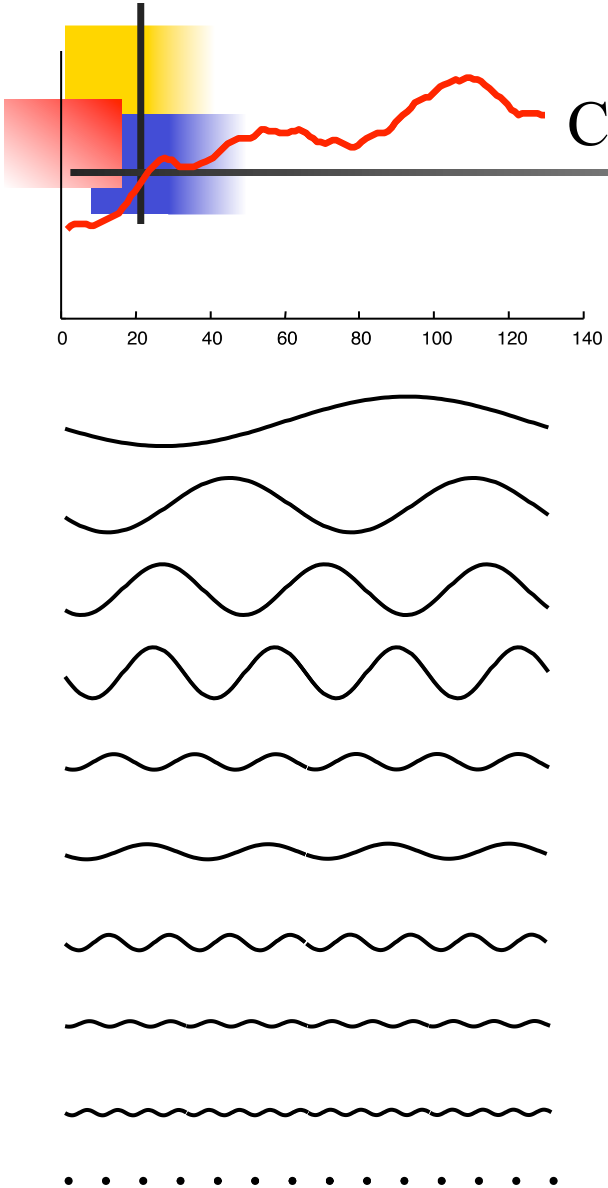
$n = 128$

Raw Data

0.4995
0.5264
0.5523
0.5761
0.5973
0.6153
0.6301
0.6420
0.6515
0.6596
0.6672
0.6751
0.6843
0.6954
0.7086
0.7240
0.7412
0.7595
0.7780
0.7956
0.8115
0.8247
0.8345
0.8407
0.8431
0.8423
0.8387
…

The graphic shows a time series with 128 points.

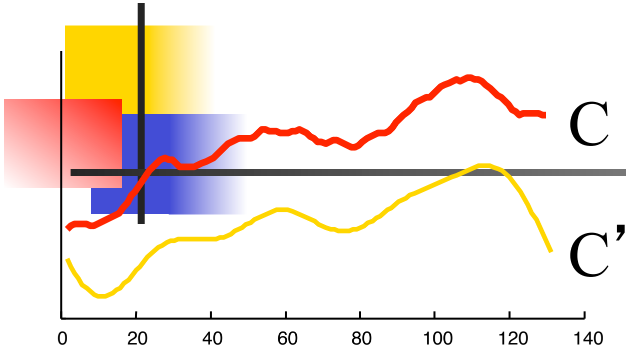
The raw data used to produce the graphic is also reproduced as a column of numbers (just the first 30 or so points are shown).

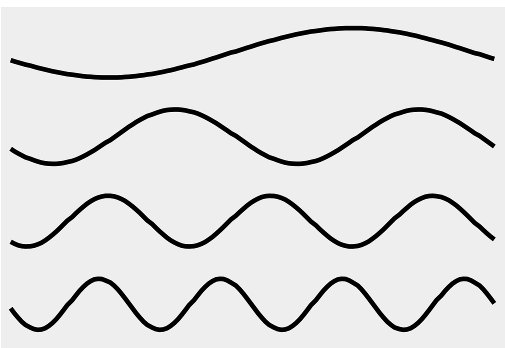


Raw Data	Fourier Coefficients
0.4995	1.5698
0.5264	1.0485
0.5523	0.7160
0.5761	0.8406
0.5973	0.3709
0.6153	0.4670
0.6301	0.2667
0.6420	0.1928
0.6515	0.1635
0.6596	0.1602
0.6672	0.0992
0.6751	0.1282
0.6843	0.1438
0.6954	0.1416
0.7086	0.1400
0.7240	0.1412
0.7412	0.1530
0.7595	0.0795
0.7780	0.1013
0.7956	0.1150
0.8115	0.1801
0.8247	0.1082
0.8345	0.0812
0.8407	0.0347
0.8431	0.0052
0.8423	0.0017
0.8387	0.0002
…	…

We can decompose the data into 64 pure sine waves using the Discrete Fourier Transform (just the first few sine waves are shown).

The Fourier Coefficients are reproduced as a column of numbers (just the first 30 or so coefficients are shown).





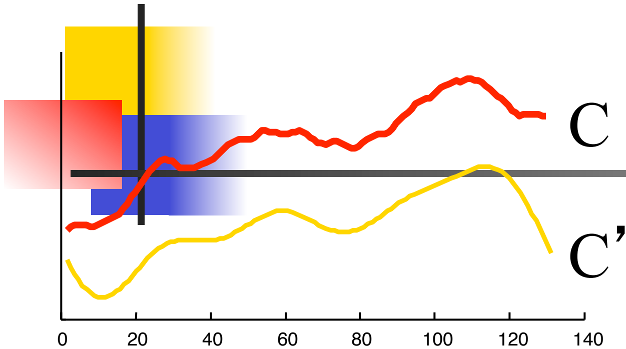
We have discarded $\frac{15}{16}$ of the data.

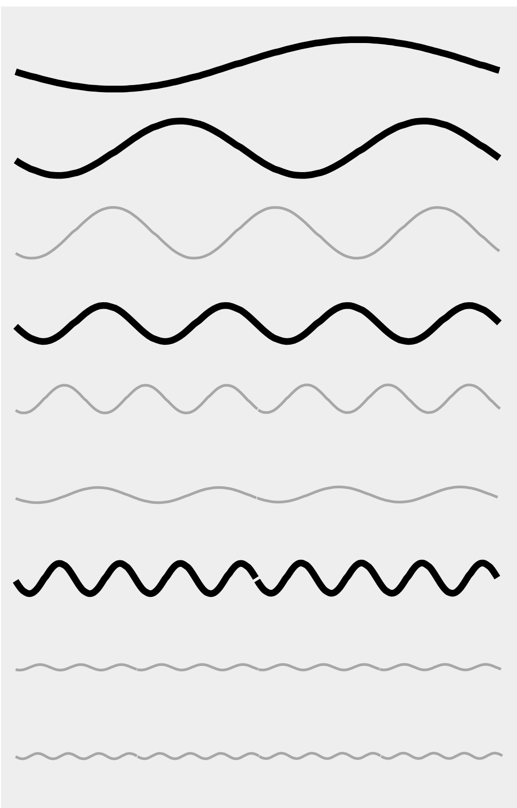
Raw Data	Fourier Coefficients	Truncated Fourier Coefficients
0.4995	1.5698	1.5698
0.5264	1.0485	1.0485
0.5523	0.7160	0.7160
0.5761	0.8406	0.8406
0.5973	0.3709	0.3709
0.6153	0.4670	0.4670
0.6301	0.2667	0.2667
0.6420	0.1928	0.1928
0.6515	0.1635	
0.6596	0.1602	
0.6672	0.0992	
0.6751	0.1282	
0.6843	0.1438	
0.6954	0.1416	
0.7086	0.1400	
0.7240	0.1412	
0.7412	0.1530	
0.7595	0.0795	
0.7780	0.1013	
0.7956	0.1150	
0.8115	0.1801	
0.8247	0.1082	
0.8345	0.0812	
0.8407	0.0347	
0.8431	0.0052	
0.8423	0.0017	
0.8387	0.0002	
…	…	

$n = 128$

$N = 8$

$C_{ratio} = 1/16$





Raw Data	Fourier Coefficients	Sorted Truncated Fourier Coefficients
0.4995	**1.5698**	**1.5698**
0.5264	**1.0485**	**1.0485**
0.5523	**0.7160**	**0.7160**
0.5761	**0.8406**	**0.8406**
0.5973	0.3709	**0.2667**
0.6153	0.1670	**0.1928**
0.6301	**0.4667**	**0.1438**
0.6420	**0.1928**	**0.1416**
0.6515	0.1635	
0.6596	0.1302	
0.6672	0.0992	
0.6751	0.1282	
0.6843	**0.2438**	
0.6954	**0.2316**	
0.7086	0.1400	
0.7240	0.1412	
0.7412	0.1530	
0.7595	0.0795	
0.7780	0.1013	
0.7956	0.1150	
0.8115	0.1801	
0.8247	0.1082	
0.8345	0.0812	
0.8407	0.0347	
0.8431	0.0052	
0.8423	0.0017	
0.8387	0.0002	
…	…	

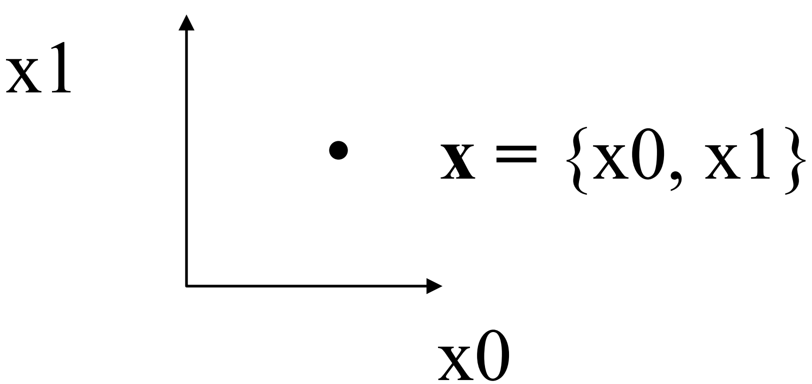
Instead of taking the first few coefficients, we could take the *best* coefficients

DFT: Parseval's theorem

$$\text{sum}(x_t^{\,2}) = \text{sum} (| X_f |^2)$$

Ie., DFT preserves the 'energy'

or, alternatively: it does an axis rotation:



Lower Bounding lemma

- Using Parseval's theorem we can prove the lower bounding property!
- So, apply DFT to each time series, keep first 3-10 coefficients as a vector and use an R-tree to index the vectors
- R-tree works with euclidean distance, OK.

Time series collections

- Fourier and wavelets are the most prevalent and successful “descriptions” of time series.
- Next, we will consider collections of M time series, each of length N.
  - What is the series that is “most similar” to all series in the collection?
  - What is the second “most similar”, and so on…

Time series collections

- Some notation:

$$\mathbb{R}^N \ni \left(x_1^{(i)}, \ldots x_N^{(i)}\right) \equiv \mathbf{x}^{(i)} \quad : i\text{-th time series vector}$$

$$\mathbb{R}^M \ni \left(x_t^{(1)}, \ldots x_t^{(M)}\right) \equiv \mathbf{x}_t \quad : \text{vector of all series at time } t$$

$$x_t^{(i)} \quad : \text{value of the } i\text{-th series at time } t$$

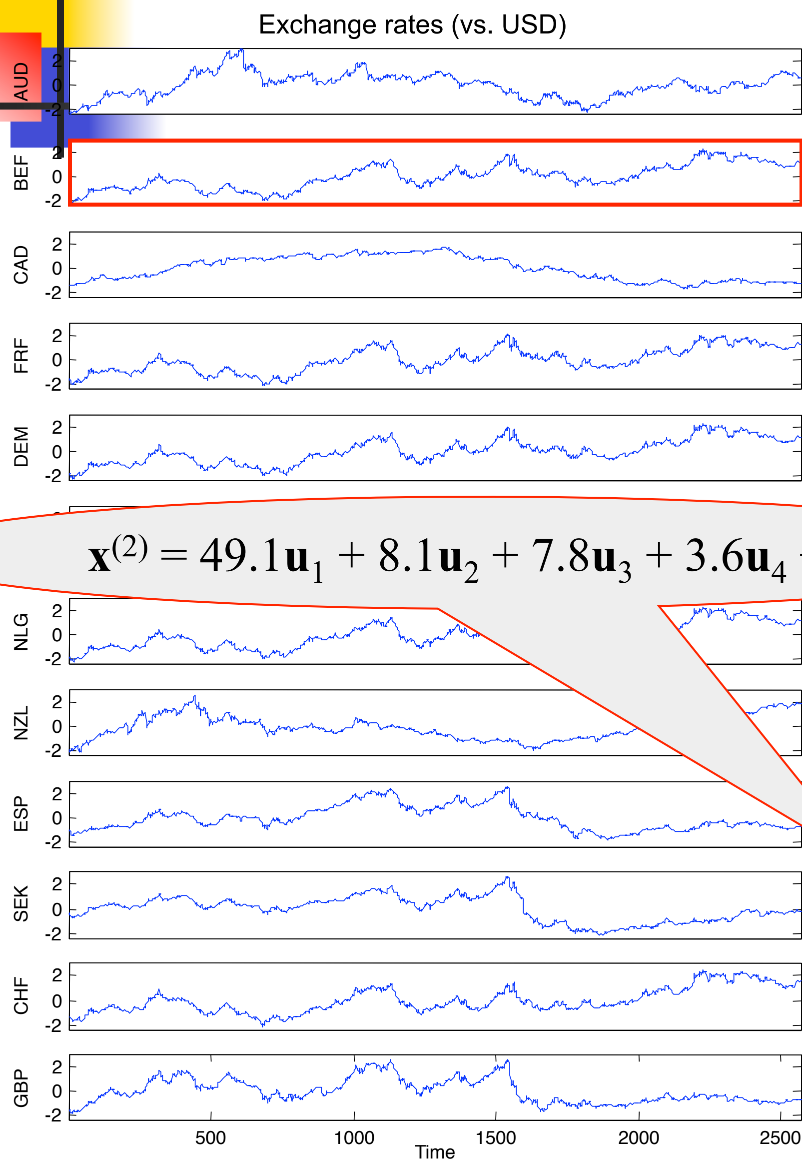
$$\mathbb{R}^{N \times M} \ni \mathbf{X} := \begin{pmatrix} x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(i)} & \cdots & x_1^{(M)} \\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(i)} & \cdots & x_2^{(M)} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ x_t^{(1)} & x_t^{(2)} & \cdots & x_t^{(i)} & \cdots & x_t^{(M)} \\ \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ x_N^{(1)} & x_N^{(2)} & \cdots & x_N^{(i)} & \cdots & x_N^{(M)} \end{pmatrix}$$

values at time t, $\mathbf{x}_t$

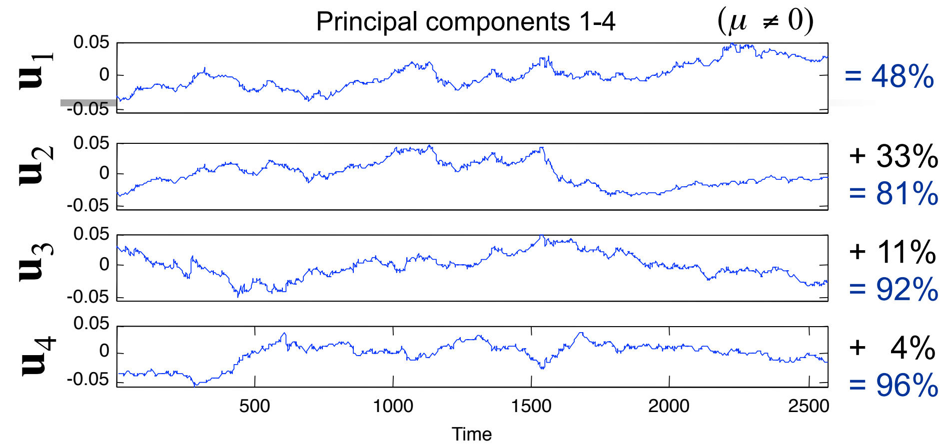
i-th series, $\mathbf{x}^{(i)}$

Principal Component Analysis

Example



$$\mathbf{x}^{(2)} = 49.1\mathbf{u}_1 + 8.1\mathbf{u}_2 + 7.8\mathbf{u}_3 + 3.6\mathbf{u}_4 + \boldsymbol{\varepsilon}_1$$



"Best" basis : $\{ \mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4 \}$

Coefficients of each time series w.r.t. basis $\{ \mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3, \mathbf{u}_4 \}$:

$$\boldsymbol{v}_1 = (-0.2, 31.4, -33.7, 18.0)$$

$$\boxed{\boldsymbol{v}_2 = (49.1, 8.1, 7.8, 3.6)}$$

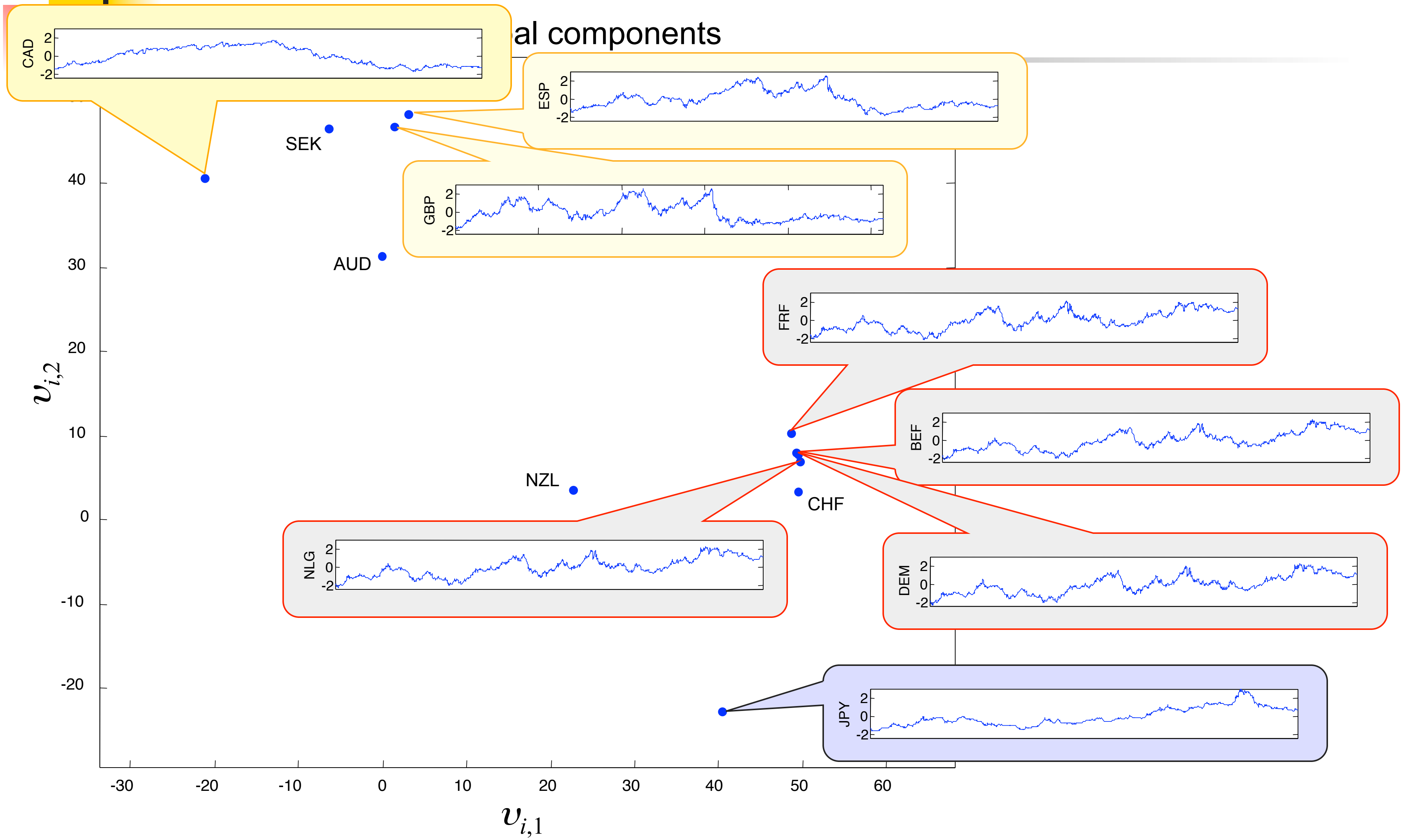
$$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$$

$$\boldsymbol{v}_{11} = (49.3, 3.5, -4.1, -6.7)$$

$$\boldsymbol{v}_{12} = (1.2, 46.8, -2.6, -7.9)$$

Principal component analysis

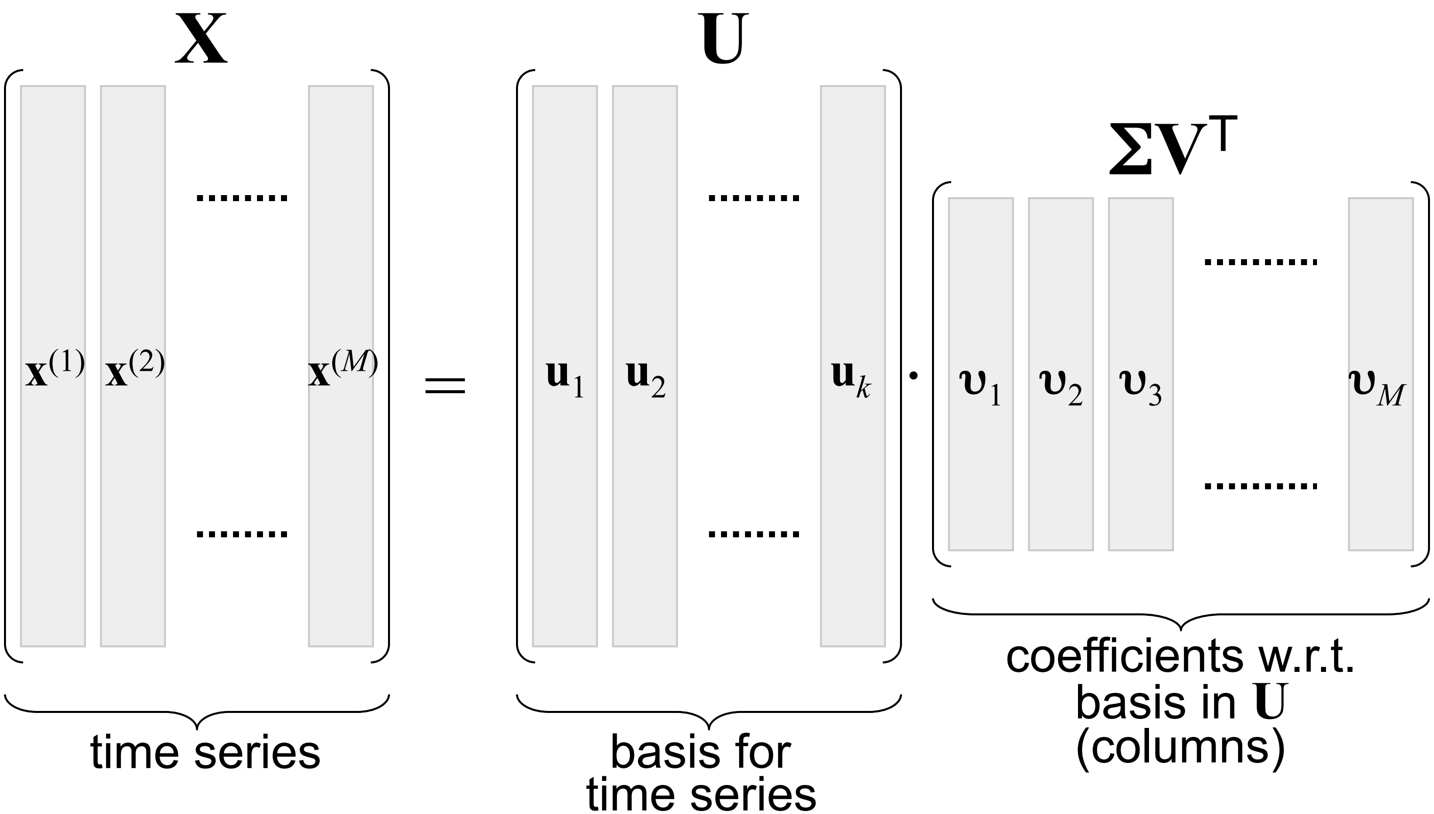
al components



Principal Component Analysis

Matrix notation — Singular Value Decomposition (SVD)

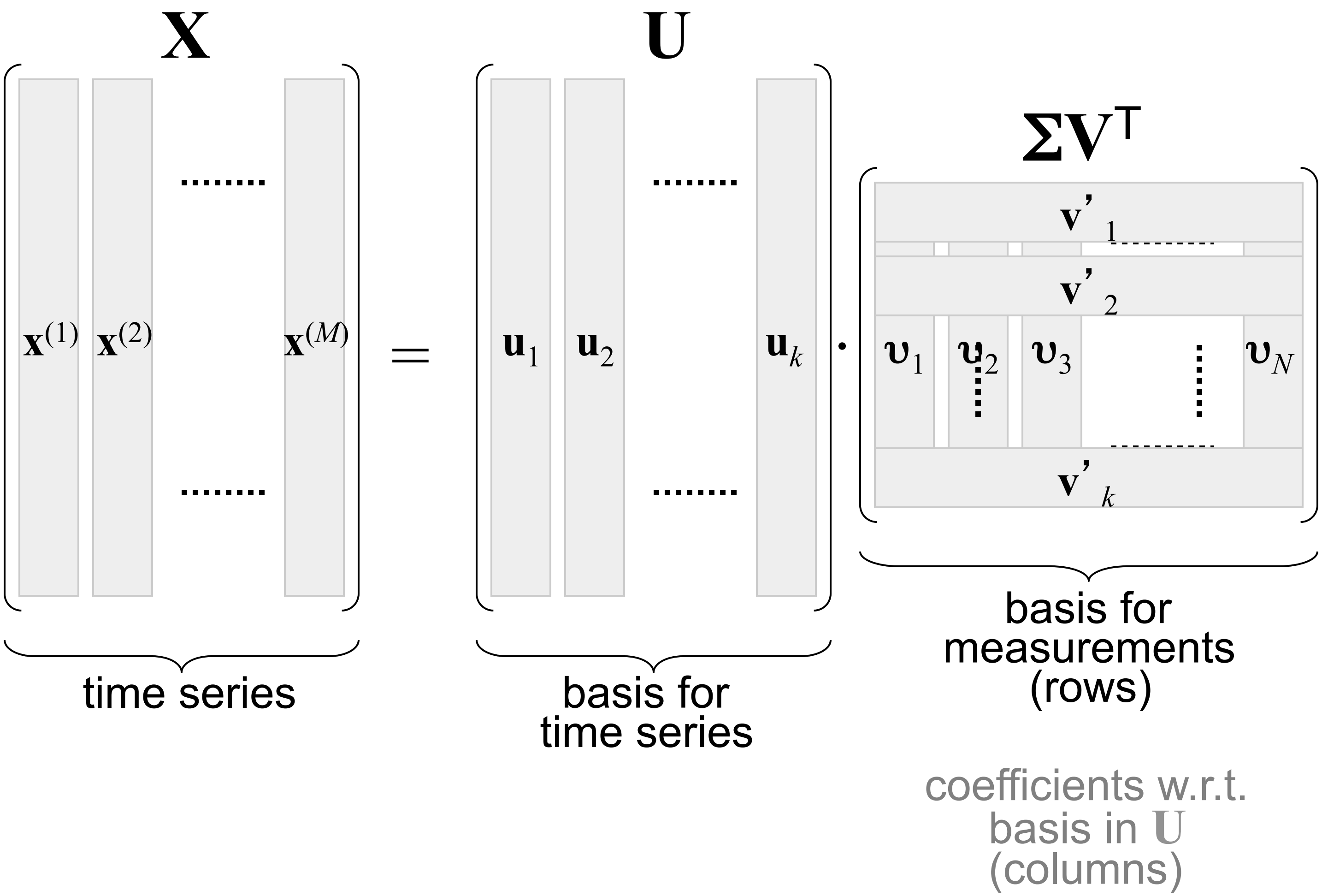
$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T}$$



Principal Component Analysis

Matrix notation — Singular Value Decomposition (SVD)

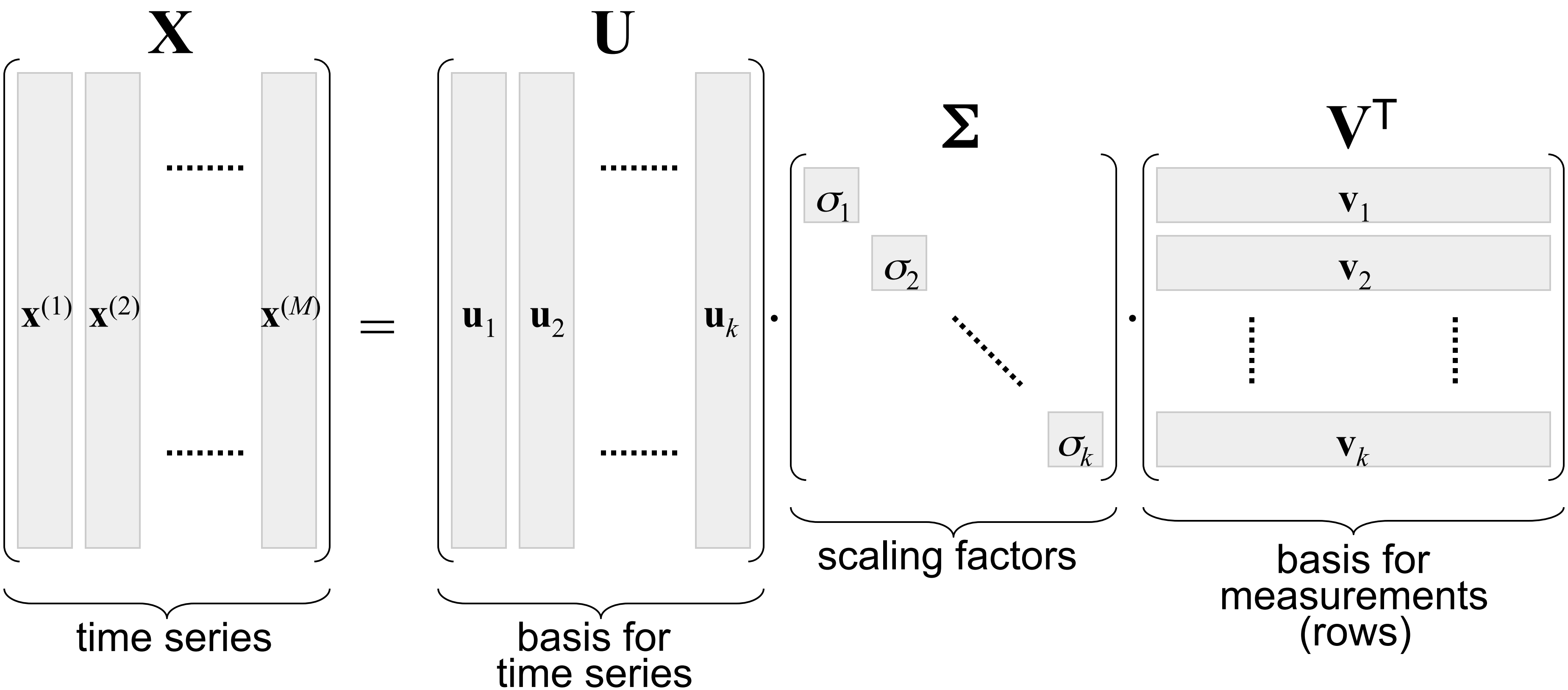
$$\mathbf{X} = \mathbf{U\Sigma V}^{\mathsf{T}}$$



Principal Component Analysis

Matrix notation — Singular Value Decomposition (SVD)

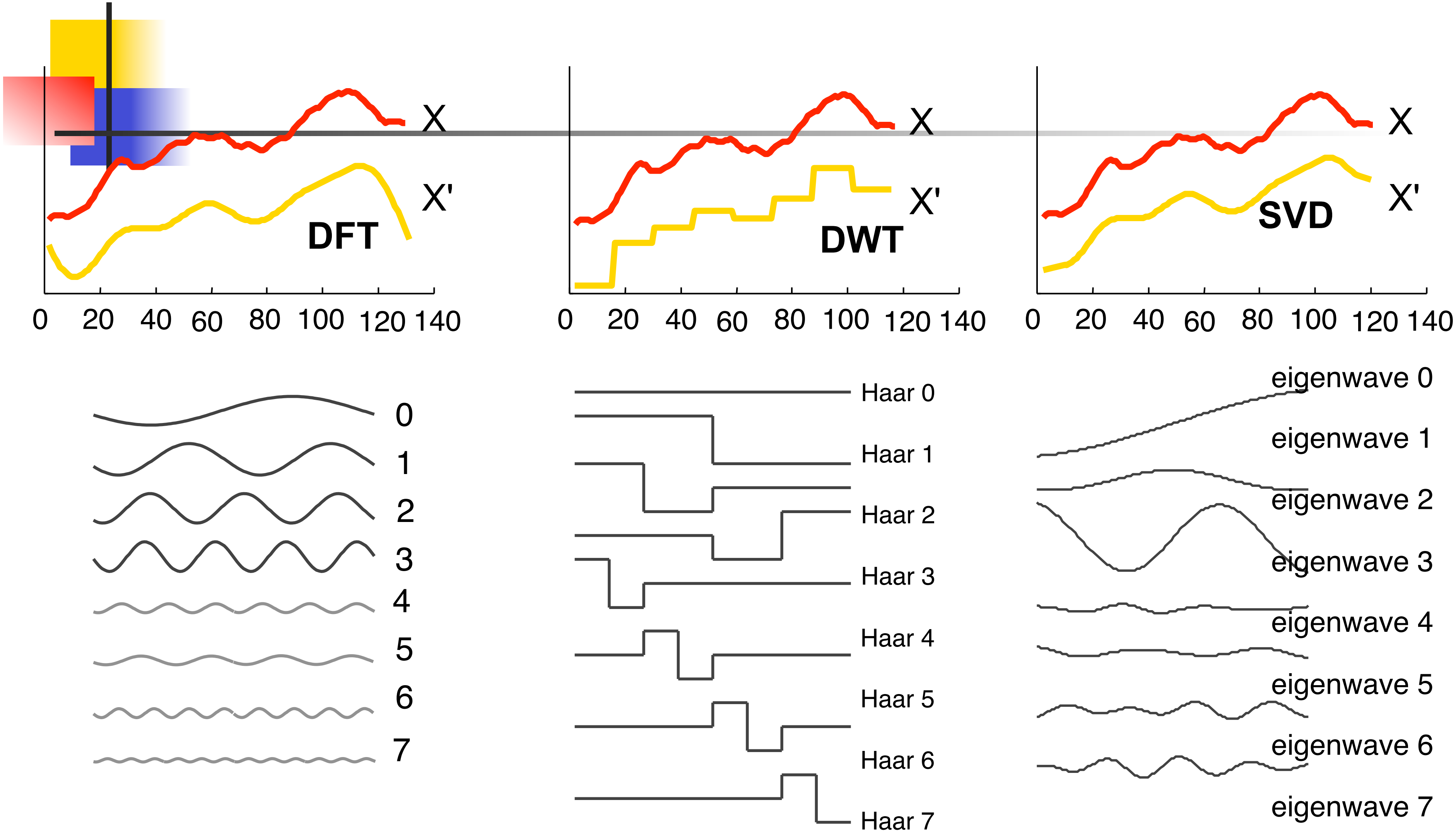
$$\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathsf{T}}$$



- PCA gives another lower dimensional transformation
- Easy to show that the lower bounding lemma holds

- but needs a collection of time series
- and expensive to compute it exactly

Feature Spaces



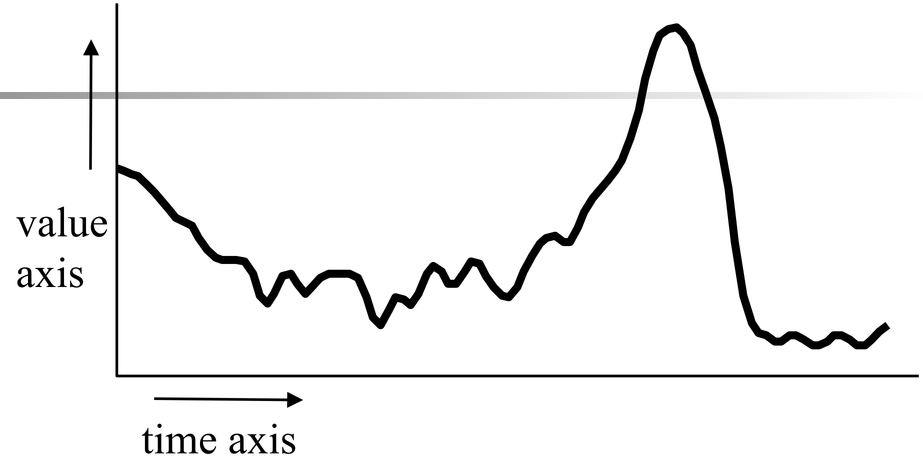
Agrawal, Faloutsos, Swami 1993

Chan & Fu 1999

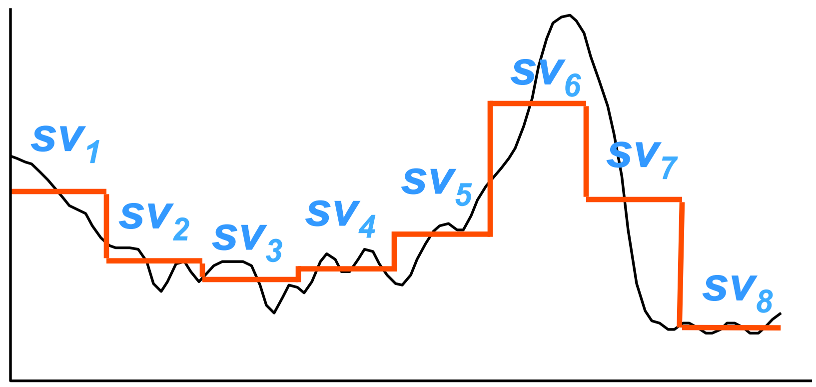
Korn, Jagadish, Faloutsos 1997

Piecewise Aggregate Approximation (PAA)

Original time series
(n-dimensional vector)
$S=\{s_1, s_2, \ldots, s_n\}$



n'-segment PAA representation
(n'-d vector)
$S = \{sv_1, sv_2, \ldots, sv_{n'}\}$



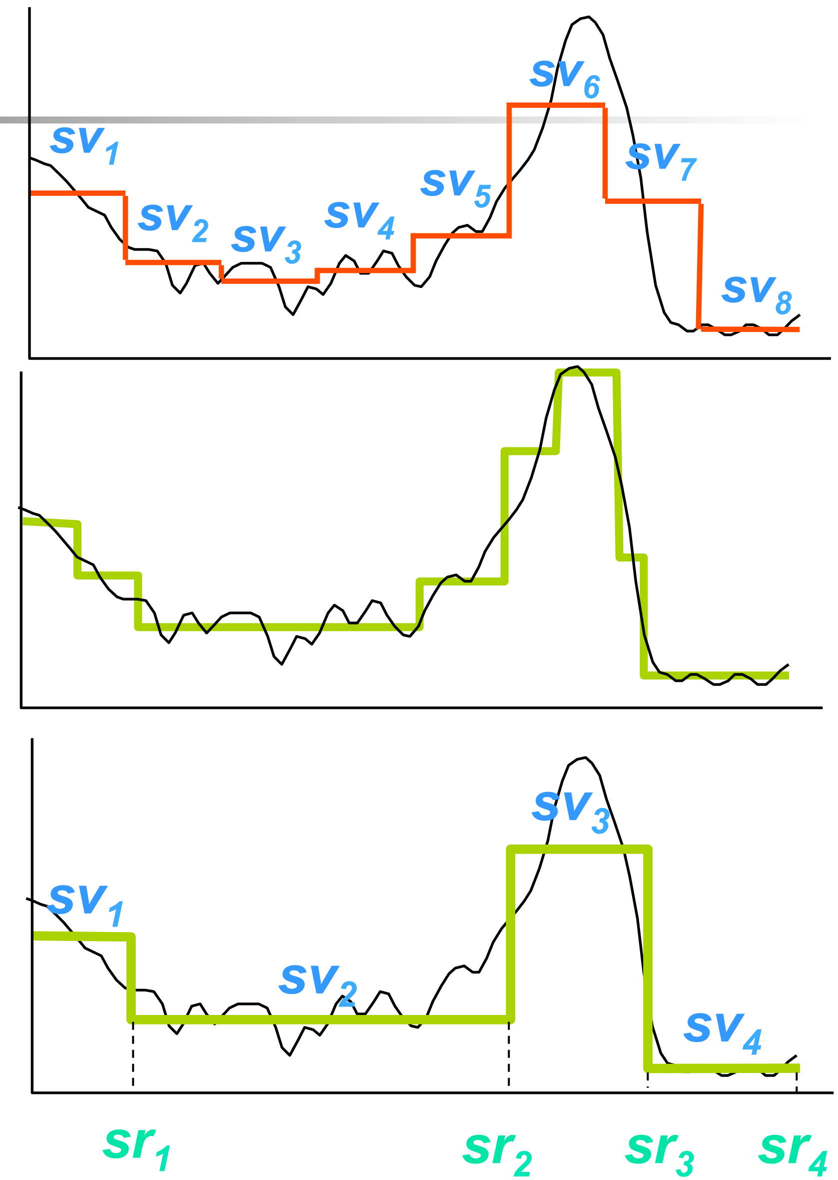
PAA representation satisfies the lower bounding lemma
(Keogh, Chakrabarti, Mehrotra and Pazzani, 2000; Yi and Faloutsos 2000)

Can we improve upon PAA?

n’-segment PAA representation
(n’-d vector)
S = {sv_1, sv_2, …, sv_N}

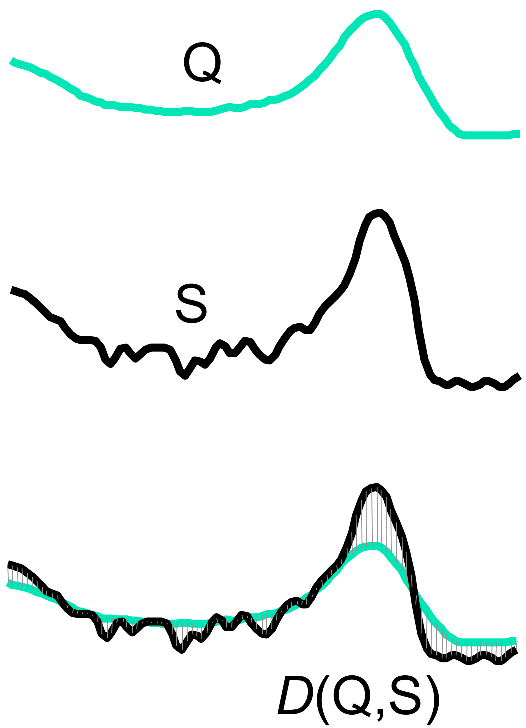
Adaptive Piecewise Constant Approximation (APCA)

n’/2-segment APCA representation
(n’-d vector)
$S = \{ sv_1, sr_1, sv_2, sr_2, \ldots, sv_M, sr_M \}$
(M is the number of segments = n’/2)



Distance Measure

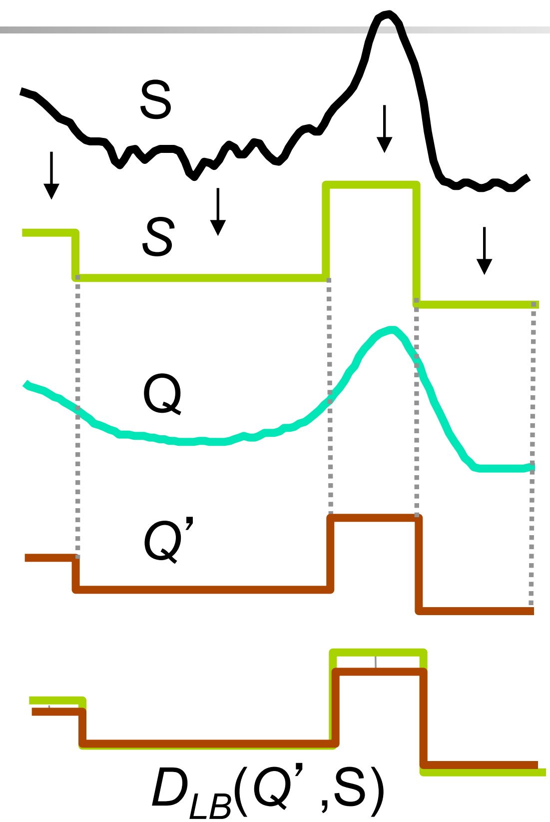
Exact (Euclidean) distance D(Q,S)



D(Q,S)

$$\equiv \sqrt{\sum_{i=1}^{n}\left(q_i - s_i\right)^2}$$

Lower bounding distance D_{LB}(Q,S)



D_{LB}(Q',S)

$$\equiv \sqrt{\sum_{i=1}^{M}(sr_i - sr_{i-1})(qv_i - sv_i)^2}$$

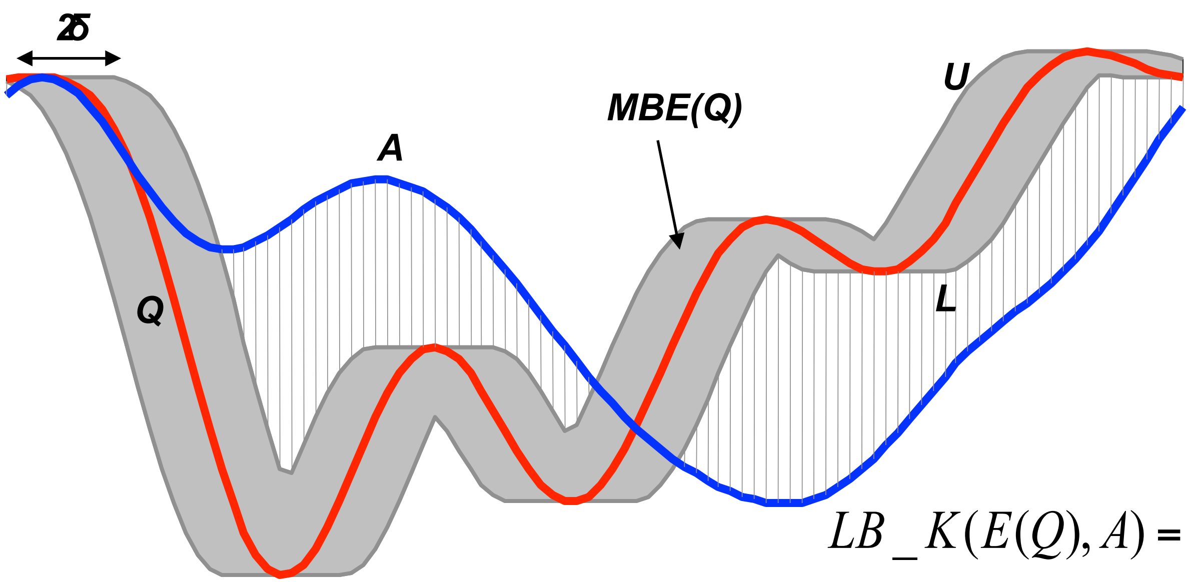
Lower Bounding the Dynamic Time Warping

Recent approaches use the Minimum Bounding Envelope for bounding the ***constrained*** *DTW*

- *Create a δ Envelope of the query Q (U, L)*
- *Calculate distance between MBE of Q and any sequence A*
- *One can show that:* ***$D(MBE(Q)_\delta, A) < DTW(Q,A)$***
- ***δ is the constraint***

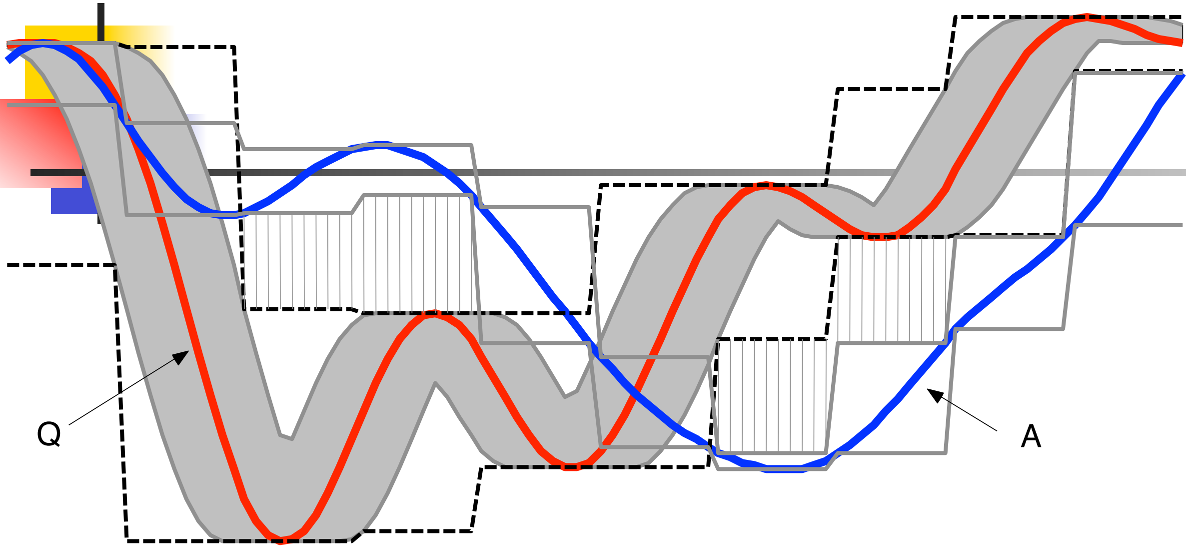
$$U[i] = \max_{-\delta \le r \le \delta}(Q[i+r])$$

$$L[i] = \min_{-\delta \le r \le \delta}(Q[i+r])$$



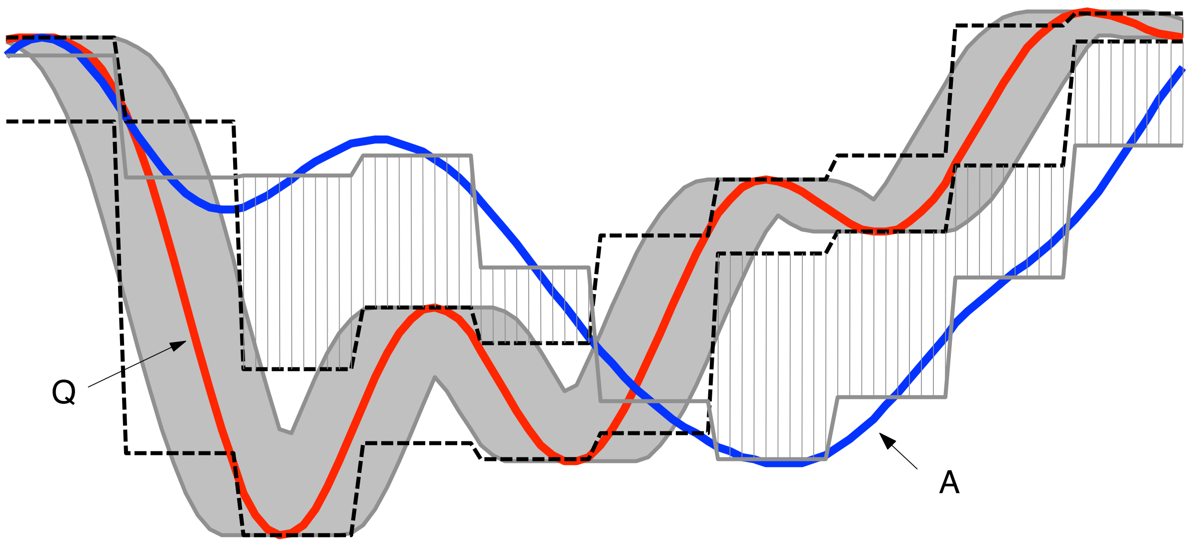
$$LB\_K(E(Q), A) = \sqrt[p]{\sum_{i=1}^{N} \begin{cases} |A[i]-U[i]|^p & if \quad A[i] > U[i] \\ |A[i]-L[i]|^p & if \quad A[i] < L[i] \\ 0 & otherwise \end{cases}}$$

Lower Bounding the Dynamic Time Warping



LB by Keogh approximate MBE and sequence using MBRs

LB = 13.84



LB by Zhu and Shasha approximate MBE and sequence using PAA

LB = 25.41

Computing the LB distance

- Use PAA to approximate each time series A in the sequence and U and L of the query envelop using k segments
- Then the LB_PAA can be computed as follows:

$$LB\_PAA(\overline{E}(Q), \overline{A}) = \sqrt[p]{\sum_{i=1}^{k} \frac{N}{k} \begin{cases} \left|\overline{A}[i] - \overline{U}[i]\right|^p & if \quad \overline{A}[i] > \overline{U}[i] \\ \left|\overline{A}[i] - \overline{L}[i]\right|^p & if \quad \overline{A}[i] < \overline{L}[i] \\ 0 & otherwise \end{cases}}$$

where $\overline{A}[i]$ is the average of the i-th segment of the time series A, i.e.

$$\overline{A}[i] = \frac{k}{N} \sum_{j=\frac{N}{k}(i-1)+1}^{\frac{N}{k}i} A[j]$$

similarly we compute $\overline{U}[i]$ and $\overline{L}[i]$