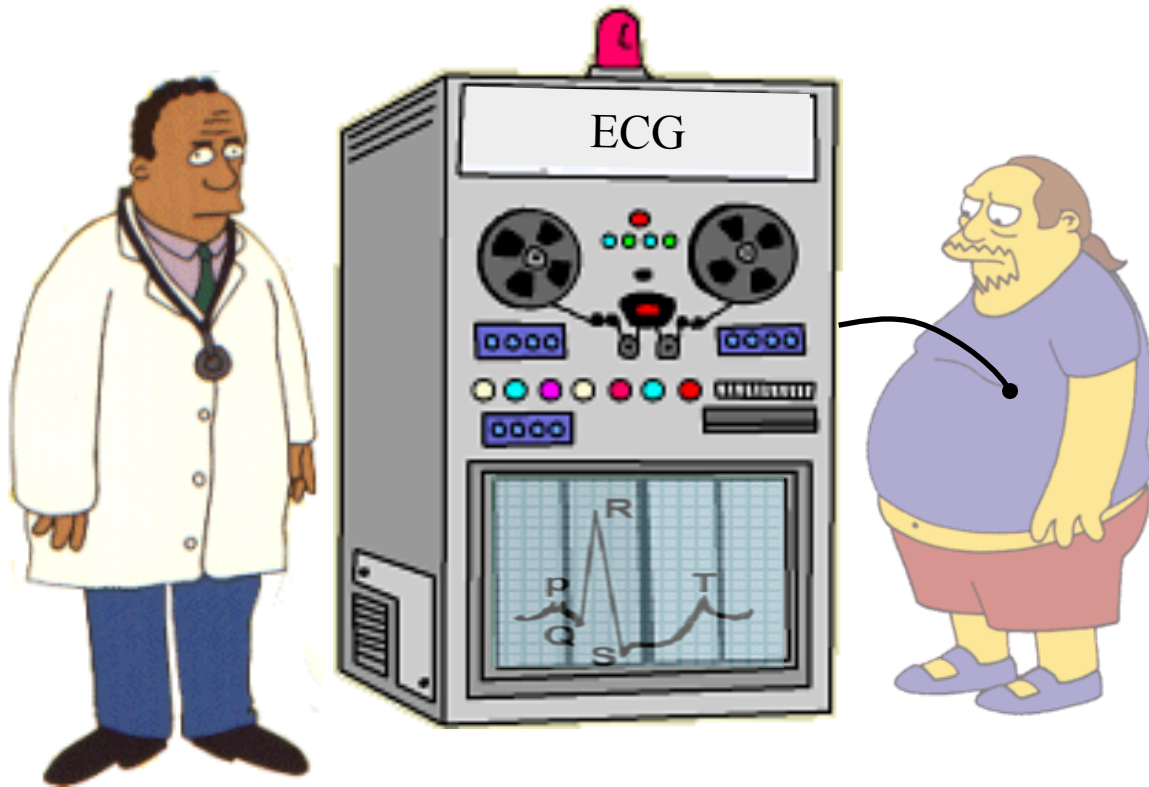


Mining Time Series Data 2

Motivating example revisited...



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

Your 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?**

Indexing Time Series

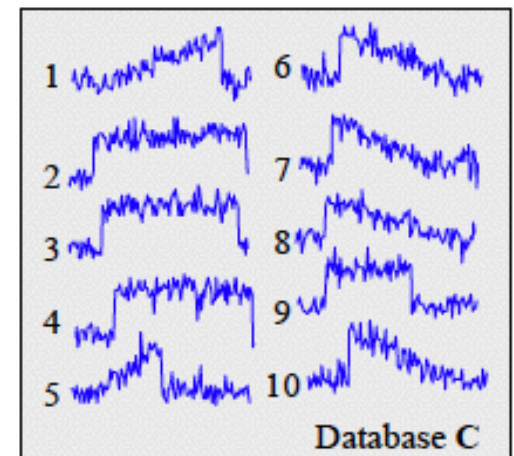
We have seen techniques for assessing the similarity of two time series.

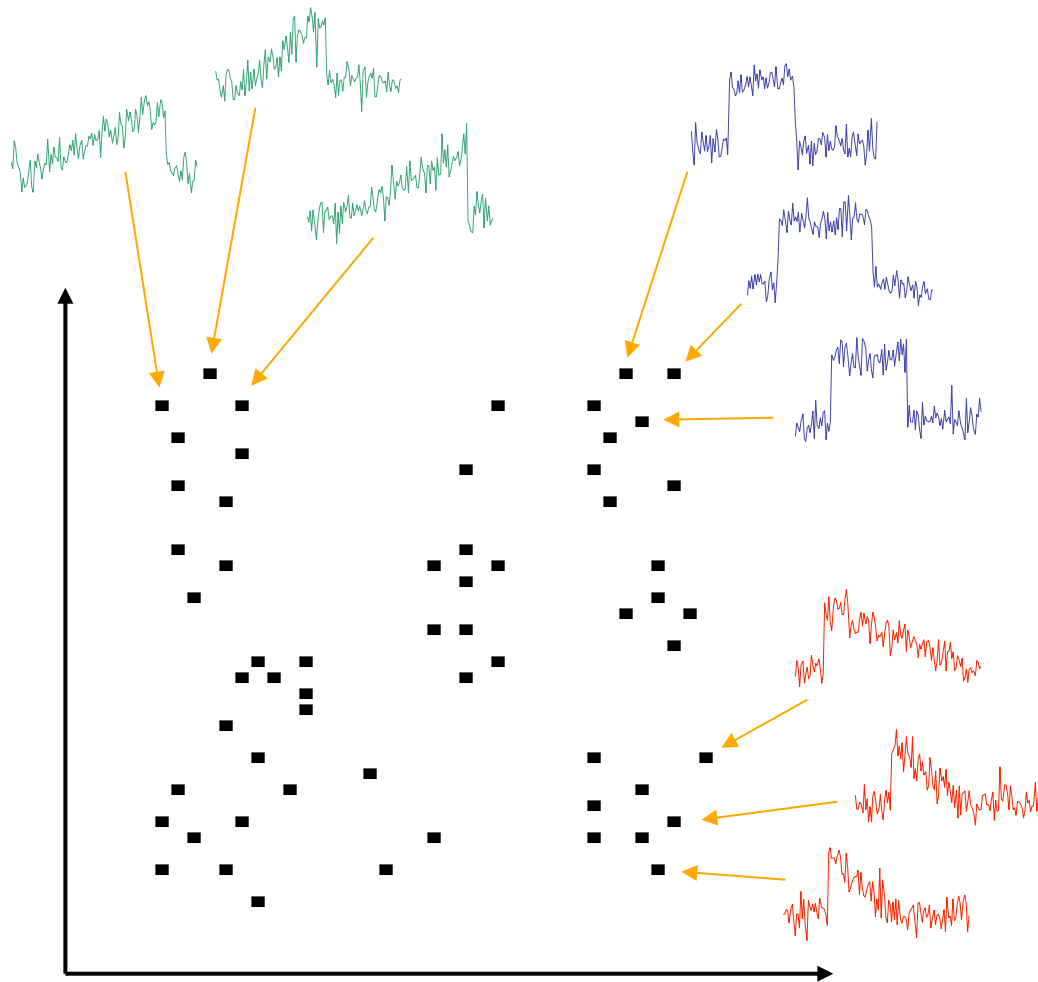
However we have not addressed the problem of finding the best match to a query in a large database

The obvious solution, to retrieve and examine every item (sequential scanning), simply does not scale to large datasets.

We need some way to index the data...

Query Q



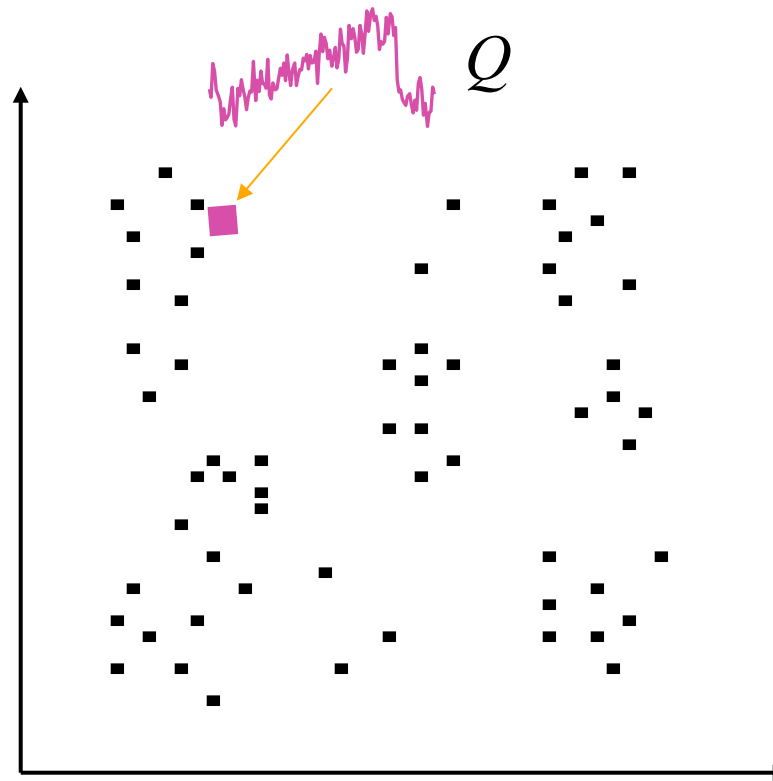


We can project time series of length n into n -dimension space.

The first value in C is the X-axis, the second value in C is the Y-axis etc.

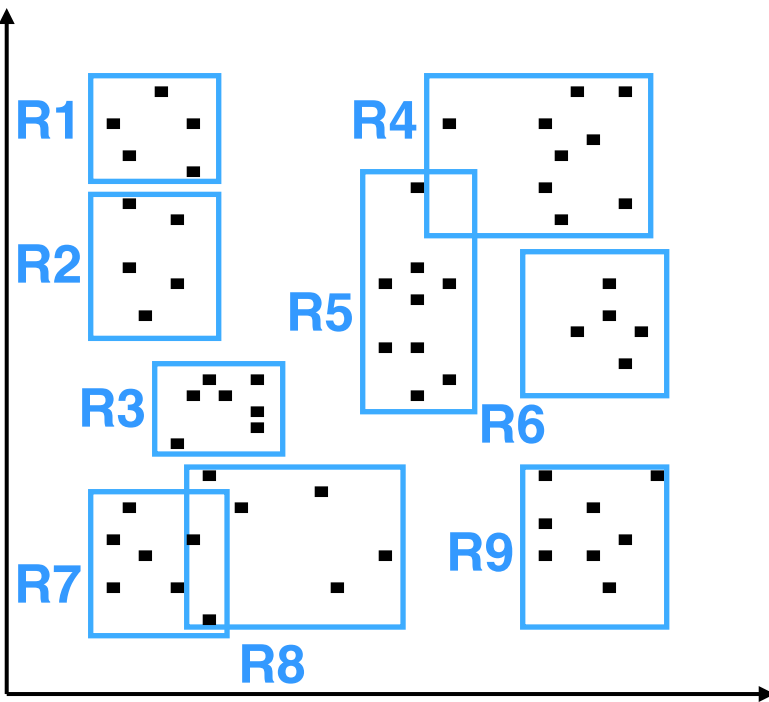
One advantage of doing this is that we have abstracted away the details of “time series”, now all query processing can be imagined as finding points in space...

...we can project the query time series Q into the same n -dimension space and simply look for the nearest points.



...the problem is that we have to look at every point to find the nearest neighbor..

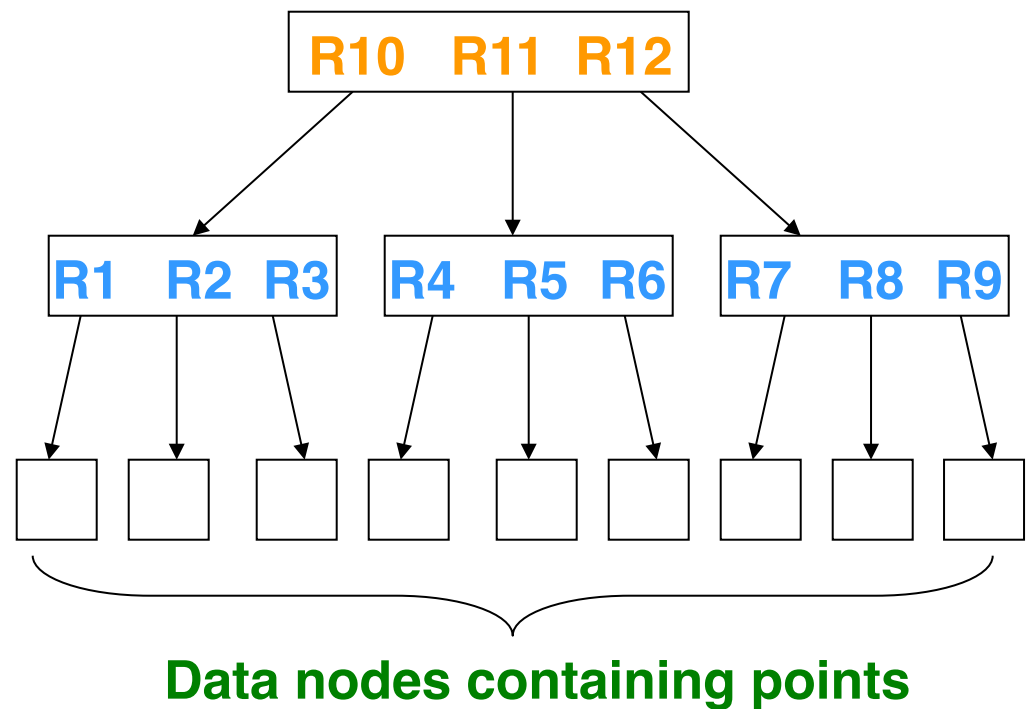
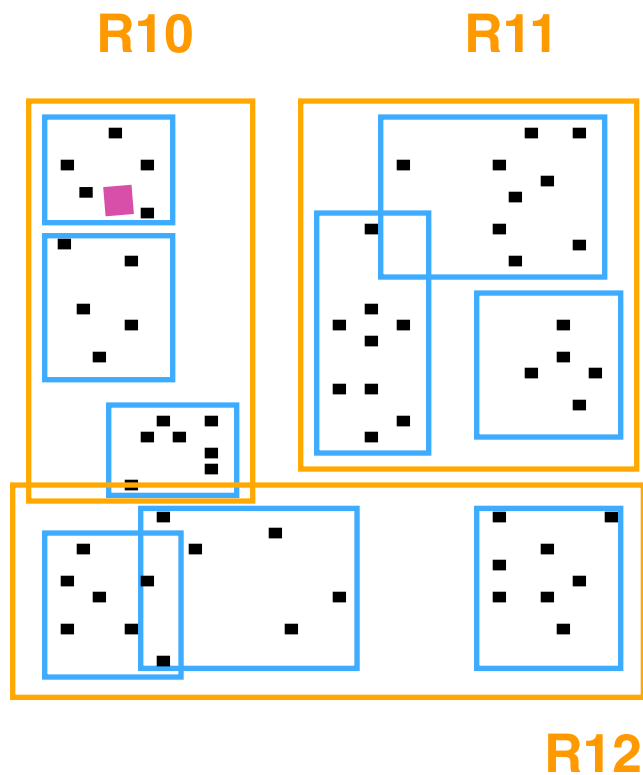
We can group clusters of datapoints with “boxes”, called Minimum Bounding Rectangles (MBR).



We can further recursively group MBRs into larger MBRs....



...these nested MBRs are organized as a tree (called a spatial access tree or a multidimensional tree). Examples include R-tree, Hybrid-Tree etc.



Spatial Access Methods

We can use Spatial Access Methods like the R-Tree to index our data, but...

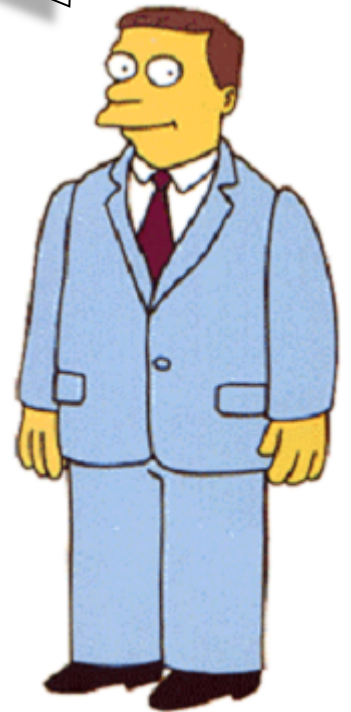
The performance of R-Trees degrade exponentially with the number of dimensions. Somewhere above 6-20 dimensions the R-Tree degrades to linear scanning.

Often we want to index time series with hundreds, perhaps even thousands of features....

Data Mining is Constrained by Disk I/O

For example, suppose you have **one gig** of main memory and want to do K-means clustering...

Clustering $\frac{1}{4}$ gig of data, 100 sec
Clustering $\frac{1}{2}$ gig of data, 200 sec
Clustering 1 gig of data, 400 sec
Clustering 1.1 gigs of data, 20 hours



GEMINI GEneric Multimedia INdexIng

{Christos Faloutsos}

- Establish a distance metric from a domain expert.
- Produce a dimensionality reduction technique that reduces the dimensionality of the data from n to N , where N can be efficiently handled by your favorite SAM.
- Produce a distance measure defined on the N dimensional representation of the data, and prove that it obeys $D_{\text{indexspace}}(A,B) \leq D_{\text{true}}(A,B)$.
i.e. The lower bounding lemma.
- Plug into an off-the-shelf SAM.

Notation for Dimensionality Reduction

For the future discussion of dimensionality reduction we will assume that

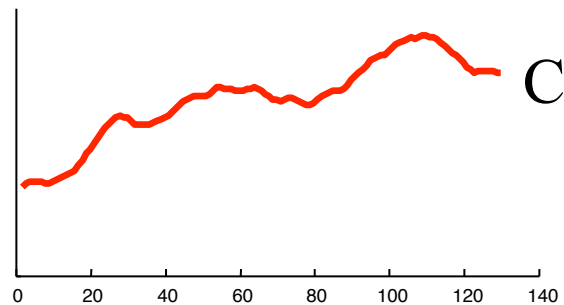
M is the number time series in our database.

n is the original dimensionality of the data.
(i.e. the length of the time series)

N is the reduced dimensionality of the data.

$C_{Ratio} = N/n$ is the compression ratio.

An Example of a Dimensionality Reduction Technique I



$$n = 128$$

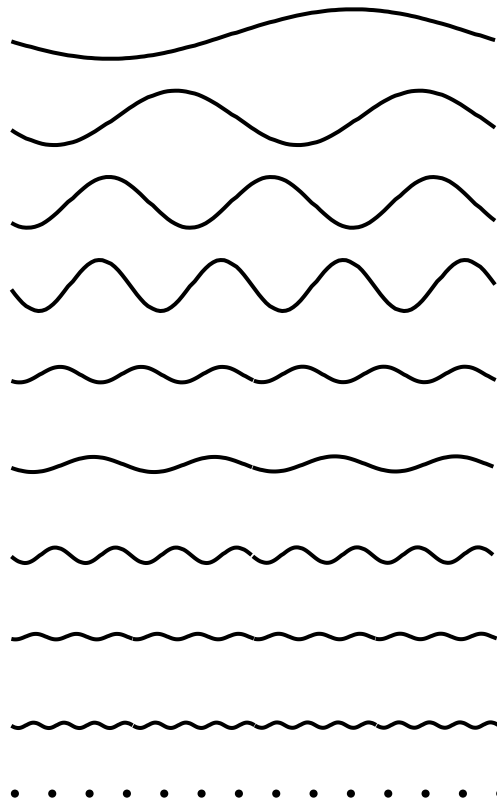
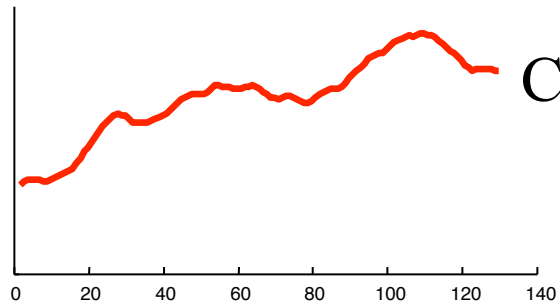
Raw Data

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).

An Example of a Dimensionality Reduction Technique II



Raw Data Fourier Coefficients

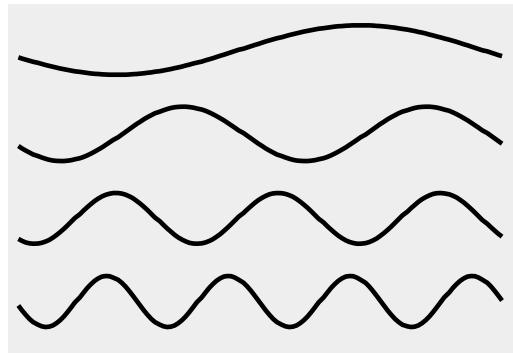
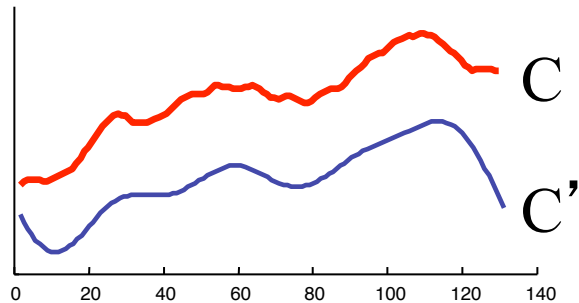
0.4995	1.5698
0.5264	<u>1.0485</u>
0.5523	0.7160
0.5761	<u>0.8406</u>
0.5973	0.3709
0.6153	<u>0.4670</u>
0.6301	0.2667
0.6420	<u>0.1928</u>
0.6515	0.1635
0.6596	<u>0.1602</u>
0.6672	0.0992
0.6751	<u>0.1282</u>
0.6843	0.1438
0.6954	<u>0.1416</u>
0.7086	0.1400
0.7240	<u>0.1412</u>
0.7412	0.1530
0.7595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

Note that at this stage we have **not** done dimensionality reduction, we have merely changed the representation...

An Example of a Dimensionality Reduction Technique III



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

**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
...

**Fourier
Coefficients**

1.5698
1.0485
0.7160
0.8406
0.3709
0.4670
0.2667
0.1928
0.1635
0.1602
0.0992
0.1282
0.1438
0.1416
0.1400
0.1412
0.1530
0.0795
0.1013
0.1150
0.1801
0.1082
0.0812
0.0347
0.0052
0.0017
0.0002
...

**Truncated
Fourier
Coefficients**

1.5698
1.0485
0.7160
0.8406
0.3709
0.4670
0.2667
0.1928

$$n = 128$$

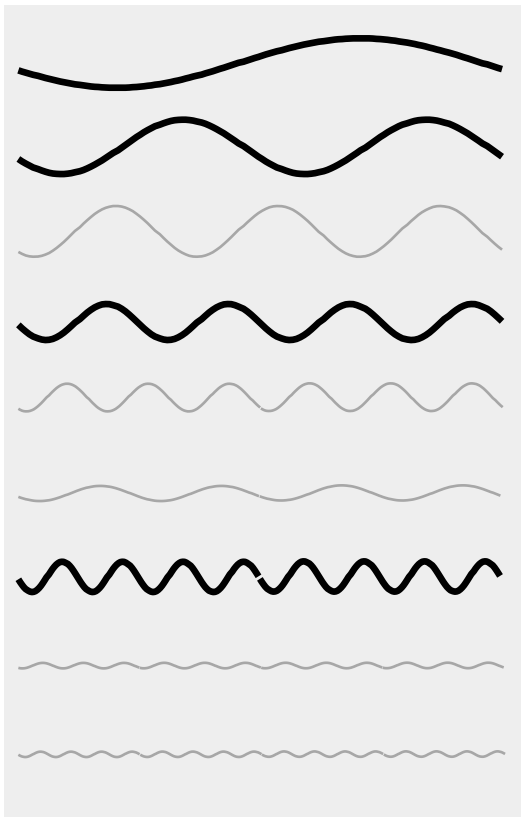
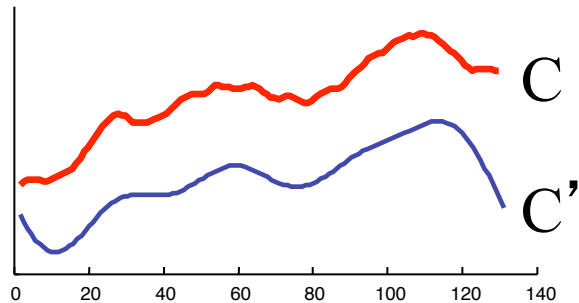
$$N = 8$$

$$C_{\text{ratio}} = 1/16$$

... however, note that the first few sine waves tend to be the largest (equivalently, the magnitude of the Fourier coefficients tend to decrease as you move down the column).

We can therefore truncate most of the small coefficients with little effect.

An Example of a Dimensionality Reduction Technique IIII



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
...

Fourier Coefficients

1.5698
1.0485
0.7160
0.8406
0.3709
0.1670
0.4667
0.1928
0.1635
0.1302
0.0992
0.1282
0.2438
0.2316
0.1400
0.1412
0.1530
0.0795
0.1013
0.1150
0.1801
0.1082
0.0812
0.0347
0.0052
0.0017
0.0002
...

Sorted Truncated Fourier Coefficients

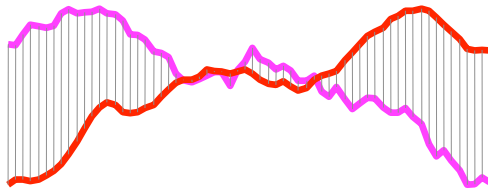
1.5698
1.0485
0.7160
0.8406
0.2667
0.1928
0.1438
0.1416

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

This can help greatly in terms of approximation quality, but makes indexing hard (impossible?).

Note this applies also to Wavelets

An Example of a Dimensionality Reduction Technique IIII



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

Raw Data 1 **Raw Data 2**

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

Truncated Fourier Coefficients 1

Truncated Fourier Coefficients 2

≥

<u>1.5698</u>	-	<u>1.1198</u>
<u>1.0485</u>	-	<u>1.4322</u>
<u>0.7160</u>	-	<u>1.0100</u>
<u>0.8406</u>	-	<u>0.4326</u>
<u>0.3709</u>	-	<u>0.5609</u>
<u>0.4670</u>	-	<u>0.8770</u>
<u>0.2667</u>	-	<u>0.1557</u>
<u>0.1928</u>	-	<u>0.4528</u>

The Euclidean distance between the two truncated Fourier coefficient vectors is always less than or equal to the Euclidean distance between the two raw data vectors*.

So DFT allows lower bounding!

*Parseval's Theorem

Mini Review for the Generic Data Mining Algorithm

We *cannot* fit all that raw data in main memory.

We *can* fit the dimensionally reduced data in main memory.

So we will solve the problem at hand on the dimensionally reduced data, making a few accesses to the raw data were necessary, and, if we are careful, the lower bounding property will insure that we get the right answer!

Raw Data 1	Raw Data 2	Raw Data n
.4995	0.7412	0.8115
.5264	0.7595	0.8247
.5523	0.7780	0.8345
.5761	0.7956	0.8407
.5973	0.8115	0.8431
.6153	0.8247	0.8423
.6301	0.8345	0.8387
.6420	0.8407	0.4995
.6515	0.8431	0.7412
.6596	0.8423	0.7595
.6672	0.8387	0.7780
.6751	0.4995	0.7956
.6843	0.5264	0.5264
.6954	0.5523	0.5523
.7086	0.5761	0.5761
.7240	0.5973	0.5973
.7412	0.6153	0.6153
...

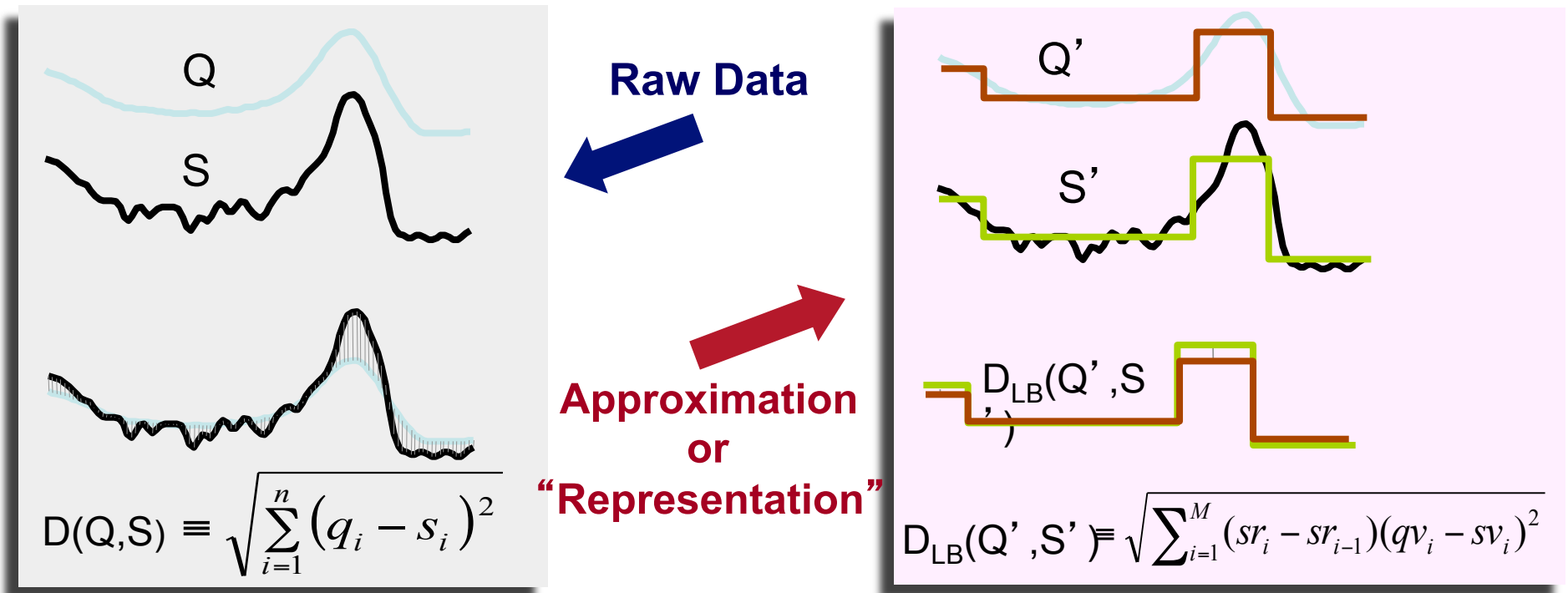
← Disk

Main
Memory →

Truncated Fourier Coefficients 1	Truncated Fourier Coefficients 2	Truncated Fourier Coefficients n
1.5698	1.1198	1.3434
1.0485	1.4322	1.4343
0.7160	1.0100	1.4643
0.8406	0.4326	0.7635
0.3709	0.5609	0.5448
0.4670	0.8770	0.4464
0.2667	0.1557	0.7932
0.1928	0.4528	0.2126

Lower Bounding Revisited

- Lower bounding means the estimated distance in the reduced space is always less than or equal to the distance in the original space.

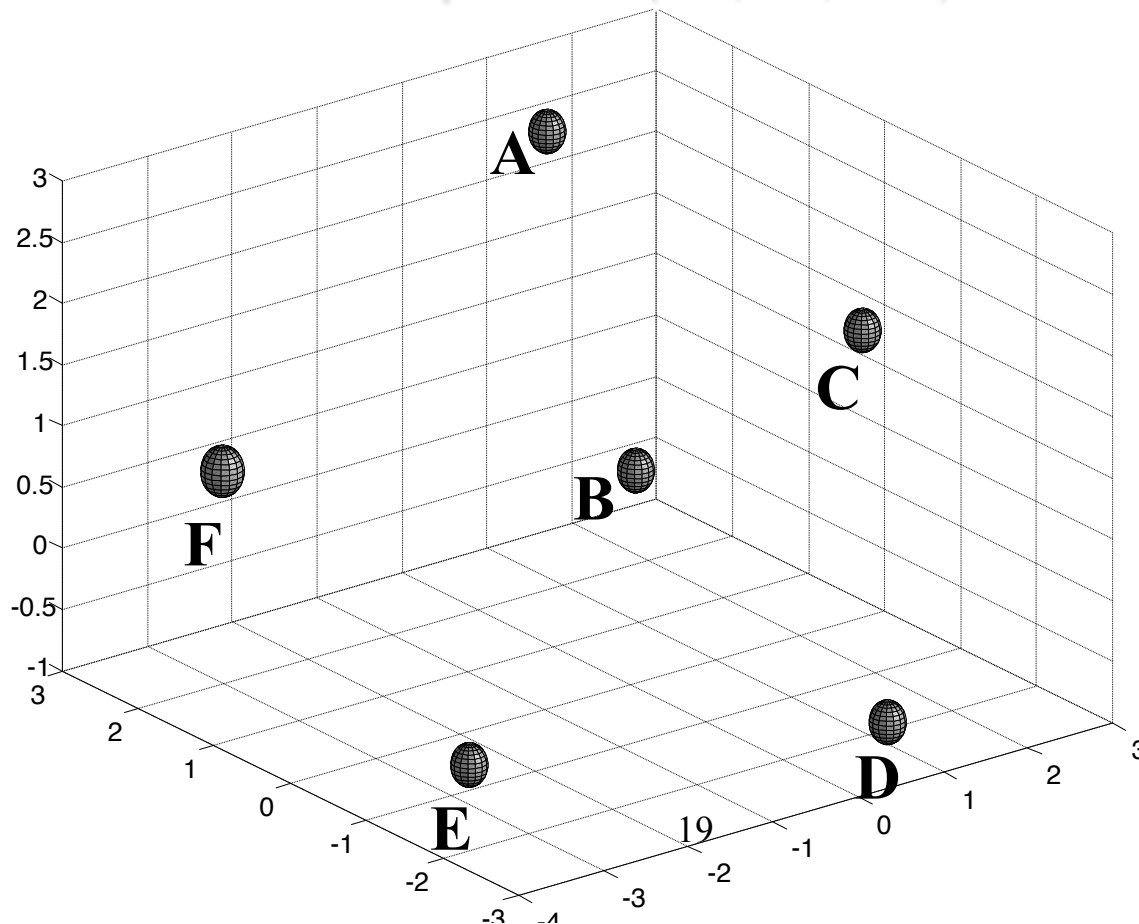


Lower bounding means that for all Q and S, we have:

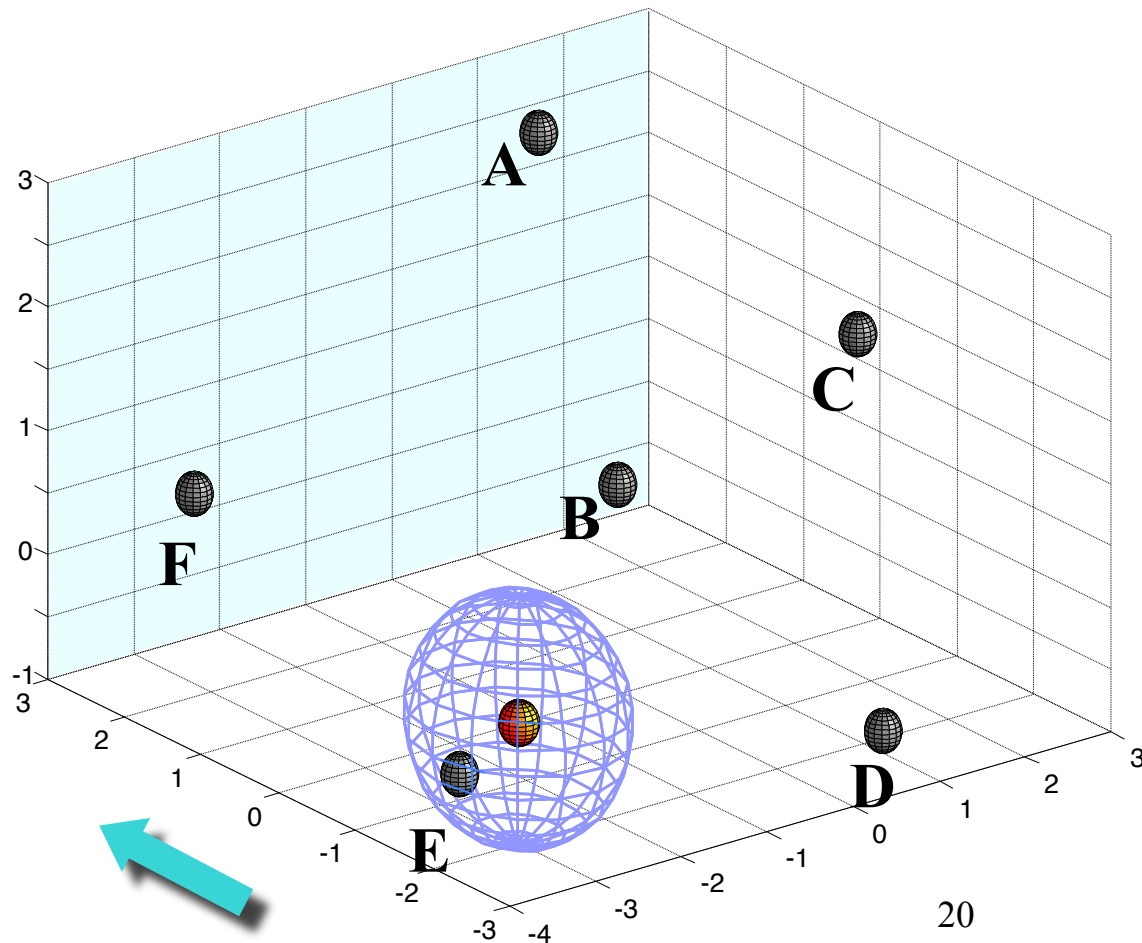
$$D_{LB}(Q', S') \leq D(Q, S)$$

Why is Lower Bounding So Important?

We have 6 objects in 3-D space. We issue a query to find all objects within 1 unit of the point $(-3, 0, -2)$...



Why is Lower Bounding So Important?



The **query** successfully finds the object E.

Consider what would happen if we issued the same query after reducing the dimensionality to 2, assuming the dimensionality technique obeys the lower bounding lemma...

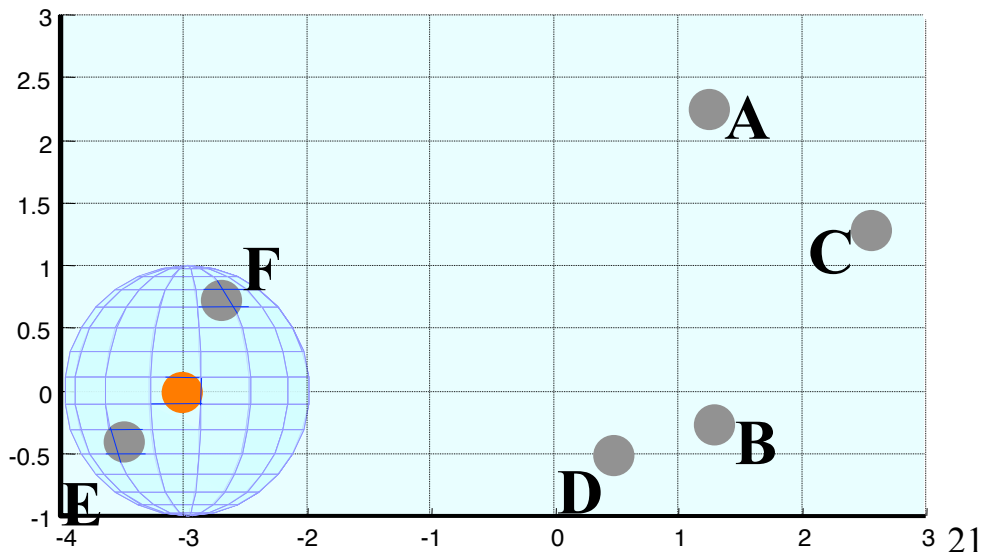
Why is Lower Bounding So Important?

Example of a dimensionality reduction technique in which the lower bounding lemma is satisfied

Informally, it's OK if objects appear closer in the dimensionality reduced space, than in the true space.

Note that because of the dimensionality reduction, object F appears to less than one unit from the query (it is a *false alarm*).

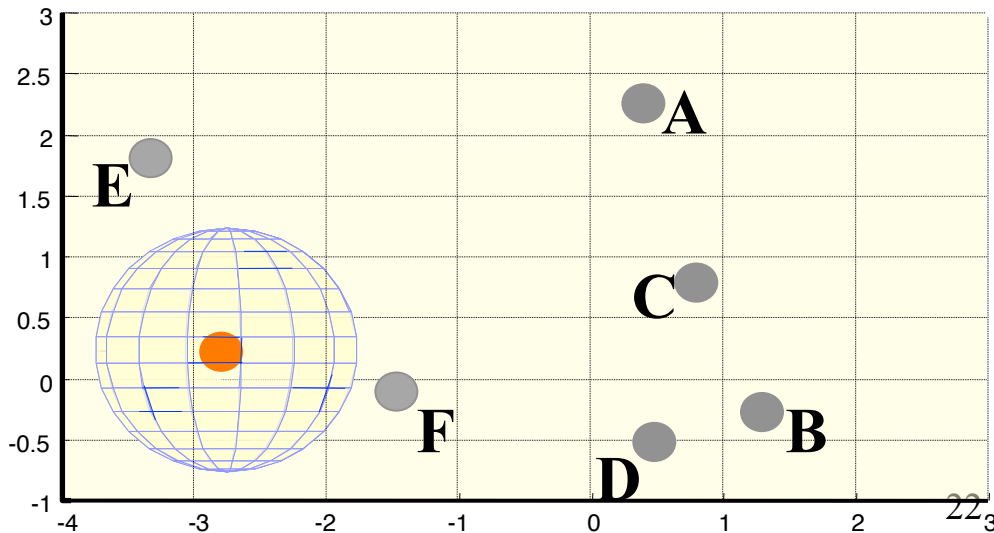
This is OK so long as it does not happen too much, since we can always retrieve it, then test it in the true, 3-D space. This would leave us with just E, the correct answer.



Why is Lower Bounding So Important?

Example of a dimensionality reduction technique in which the lower bounding lemma is not satisfied

Informally, some objects appear further apart in the dimensionality reduced space than in the true space.

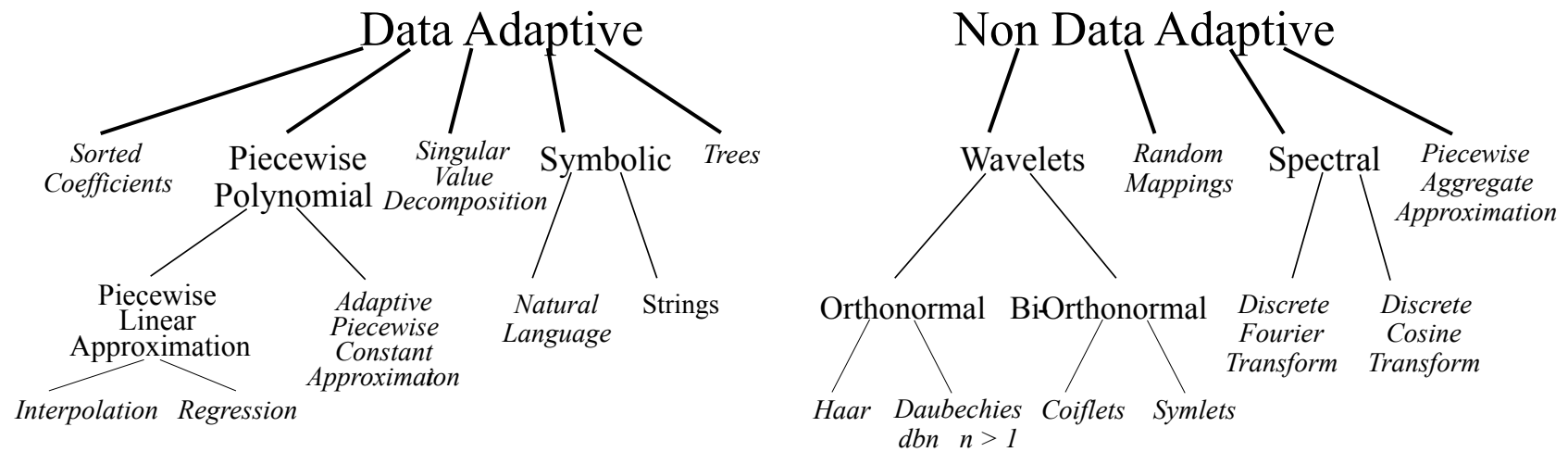


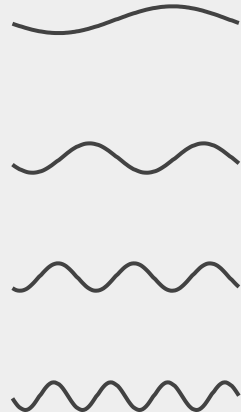
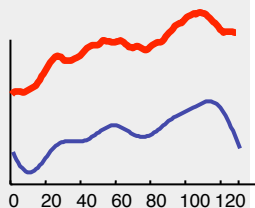
Note that because of the dimensionality reduction, object E appears to be more than one unit from the query (it is a *false dismissal*).

This is unacceptable.

We have failed to find the true answer set to our query.

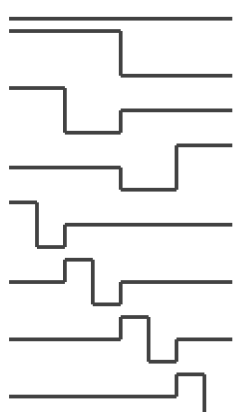
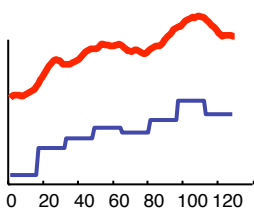
Time Series Representations





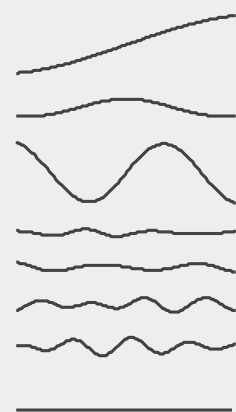
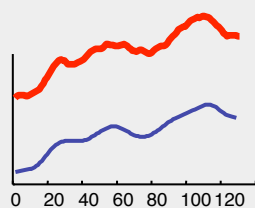
DFT

Agrawal, Faloutsos, &
1993
Faloutsos, Ranganathan, &
Manolopoulos. SIGMOD 1994



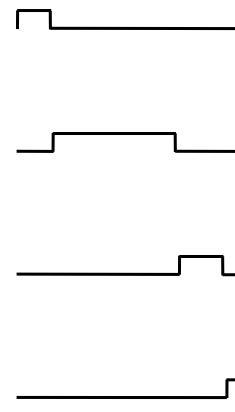
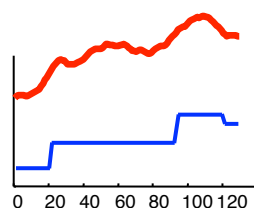
DWT

Chan & Fu. ICDE 1999



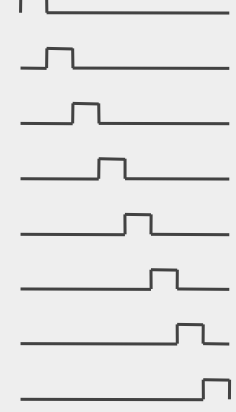
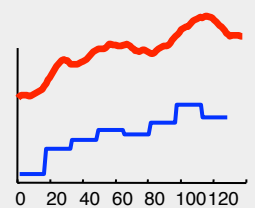
SVD

Korn, Jagadish & Faloutsos.
SIGMOD 1997



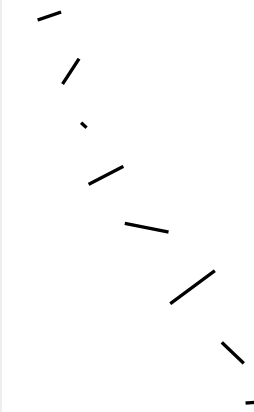
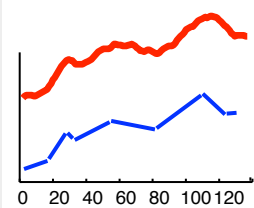
APCA

Keogh, Chakrabarti, Pazzani &
Mehrotra SIGMOD 2001



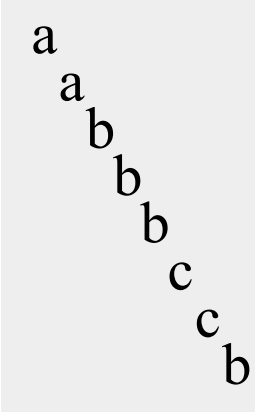
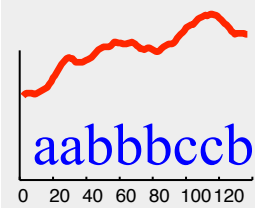
PAA

Keogh, Chakrabarti, Pazzani &
Mehrotra KAIS 2000
Yi & Faloutsos VLDB 2000



PLA

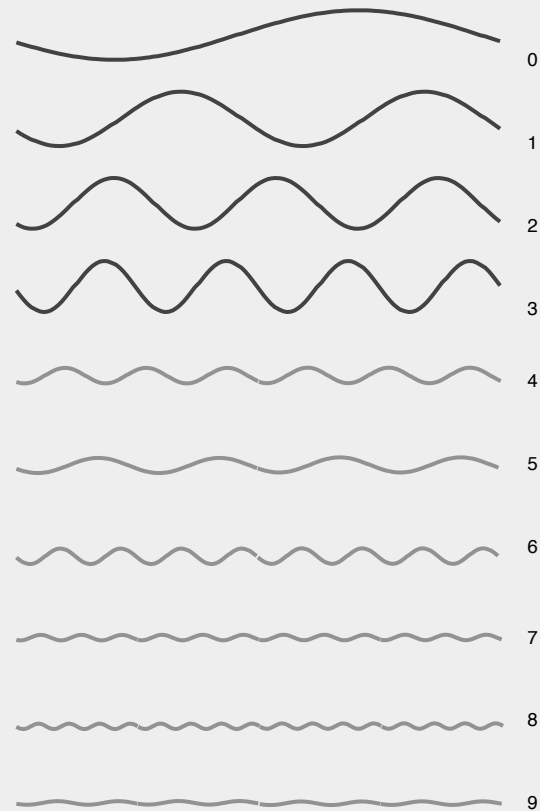
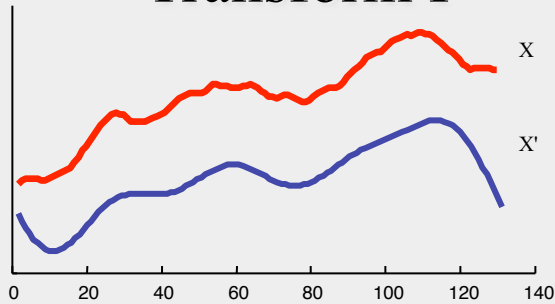
Morinaka,
Yoshikawa, Amagasa, &
Uemura, PAKDD 2001



SAX

Lin, J., Keogh, E.,
Lonardi, S. & Chiu, B.
DMKD 2003

Discrete Fourier Transform I



Basic Idea: Represent the time series as a linear combination of sines and cosines, but keep only the first $n/2$ coefficients.

Why $n/2$ coefficients? Because the coefficients are symmetric: the 2nd half is the repeat of the first half in reverse order

DFT does a good job concentrating energy in the first few coefficients

$$C(t) = \sum_{k=1}^n (A_k \cos(2\pi w_k t) + B_k \sin(2\pi w_k t))$$



Jean Fourier

1768-1830

Excellent free Fourier Primer

Hagit Shatkay, "The Fourier Transform - a Primer", Technical Report CS-95-37, Department of Computer Science, Brown University, 1995.

<http://www.ncbi.nlm.nih.gov/CBBresearch/Postdocs/Shatkay/>

Fourier Decomposition

Decompose a time-series into sum of sine waves

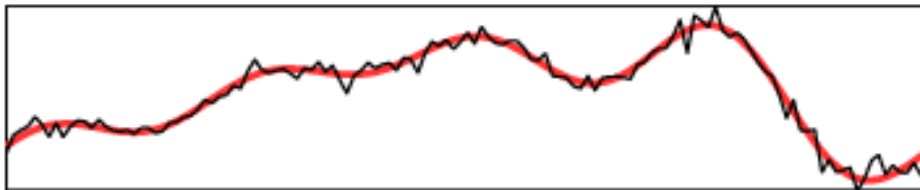
DFT: $X(f_{k/N}) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x(n) e^{-\frac{j2\pi kn}{N}}, \quad k = 0, 1 \dots N-1$

IDFT: $x(n) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X(f_{k/N}) e^{\frac{j2\pi kn}{N}}, \quad k = 0, 1 \dots N-1$

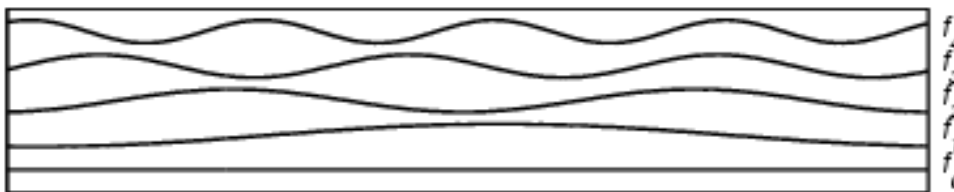
$$e^{ia} = \cos(a) + i \sin(a)$$

$$X(f_{k/N}) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x(n) \left(\cos\left(\frac{2\pi kn}{N}\right) - i \sin\left(\frac{2\pi kn}{N}\right) \right)$$

Signal & Reconstruction



Fourier Coefficients



“Every signal can be represented as a superposition of sines and cosines”

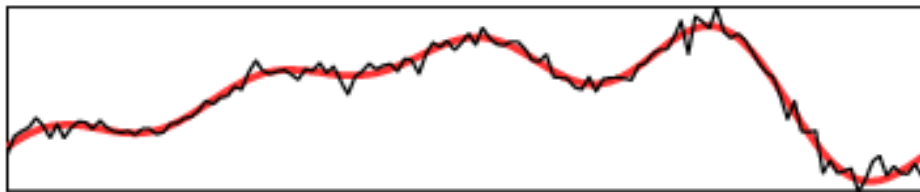


Fourier Decomposition

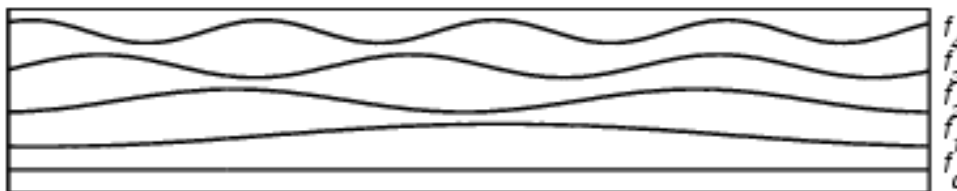
Decompose a time-series into sum of sine waves

DFT:
$$X(f_{k/N}) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x(n) e^{-\frac{j2\pi kn}{N}}, \quad k = 0, 1 \dots N-1$$

IDFT:
$$x(n) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X(f_{k/N}) e^{\frac{j2\pi kn}{N}}, \quad k = 0, 1 \dots N-1$$



Fourier Coefficients



$X(f)$ $x(n)$

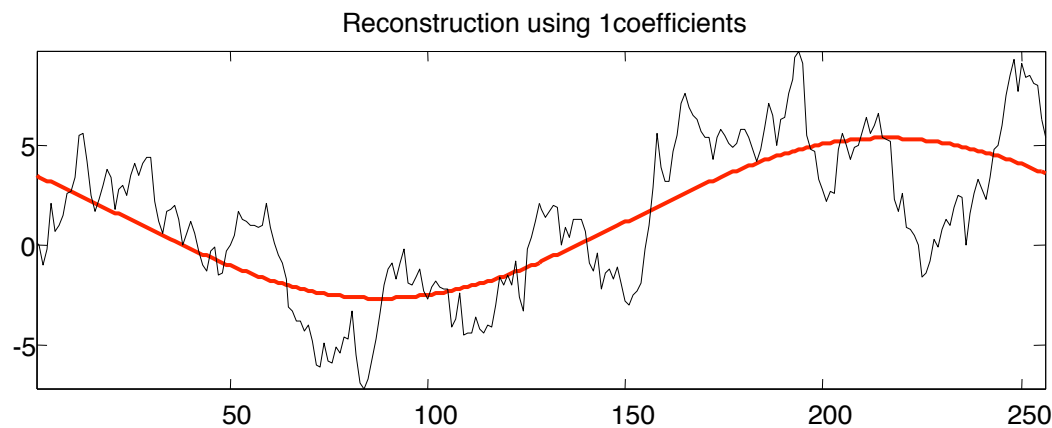
-0.3633	-0.4446
-0.6280 + 0.2709i	-0.9864
-0.4929 + 0.0399i	-0.3254
-1.0143 + 0.9520i	-0.6938
0.7200 - 1.0571i	-0.1086
-0.0411 + 0.1674i	-0.3470
-0.5120 - 0.3572i	0.5849
0.9860 + 0.8043i	1.5927
-0.3680 - 0.1296i	-0.9430
-0.0517 - 0.0830i	-0.3037
-0.9158 + 0.4481i	-0.7805
1.1212 - 0.6795i	-0.1953
<u>0.2667 + 0.1100i</u>	-0.3037
0.2667 - 0.1100i	0.2381
1.1212 + 0.6795i	2.8389
-0.9158 - 0.4481i	-0.7046
-0.0517 + 0.0830i	-0.5529
-0.3680 + 0.1296i	-0.6721
0.9860 - 0.8043i	0.1189
-0.5120 + 0.3572i	0.2706
-0.0411 - 0.1674i	-0.0003
0.7200 + 1.0571i	1.3976
-1.0143 - 0.9520i	-0.4987
-0.4929 - 0.0399i	-0.2387
-0.6280 - 0.2709i	-0.7588

```
fa = fft(a); % Fourier decomposition
fa(5:end) = 0; % keep first 5 coefficients (low frequencies)
reconstr = real(ifft(fa)); % reconstruct signal
```



Fourier Decomposition

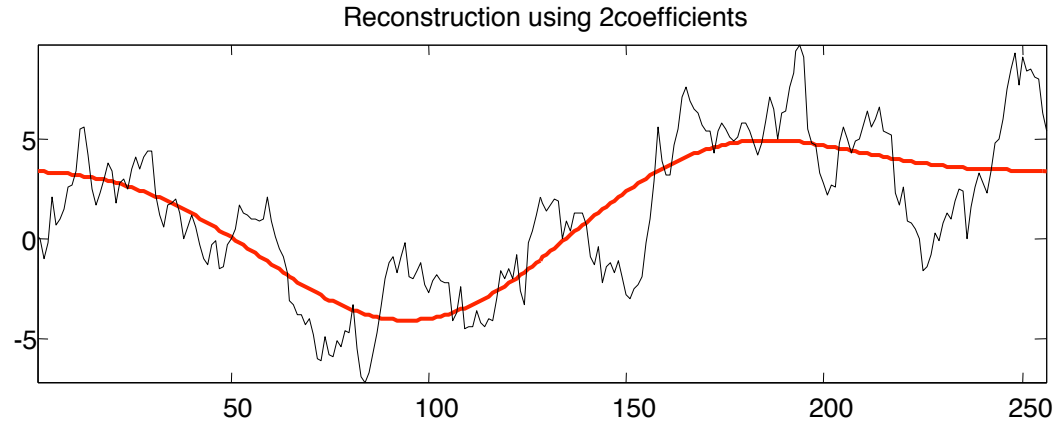
How much space we gain by compressing random walk data?



- 1 coeff > 60% of energy
- 10 coeff > 90% of energy

Fourier Decomposition

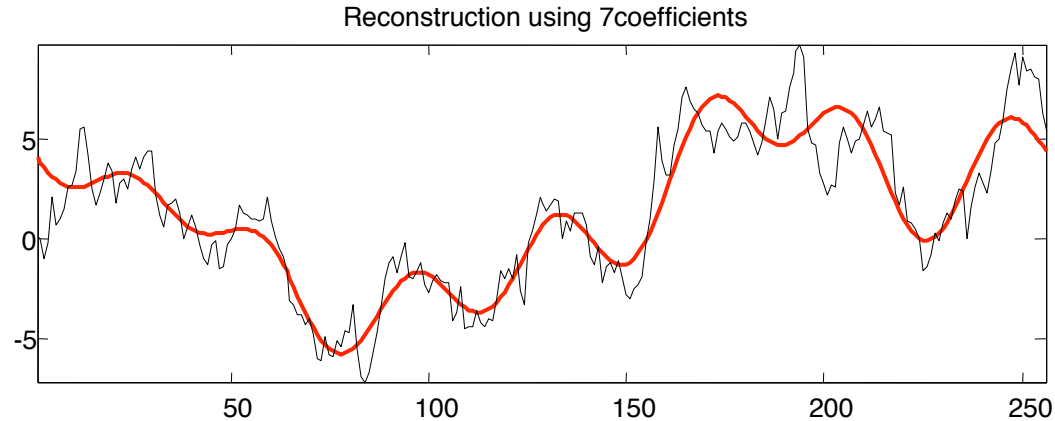
How much space we gain by compressing random walk data?



- 1 coeff > 60% of energy
- 10 coeff > 90% of energy

Fourier Decomposition

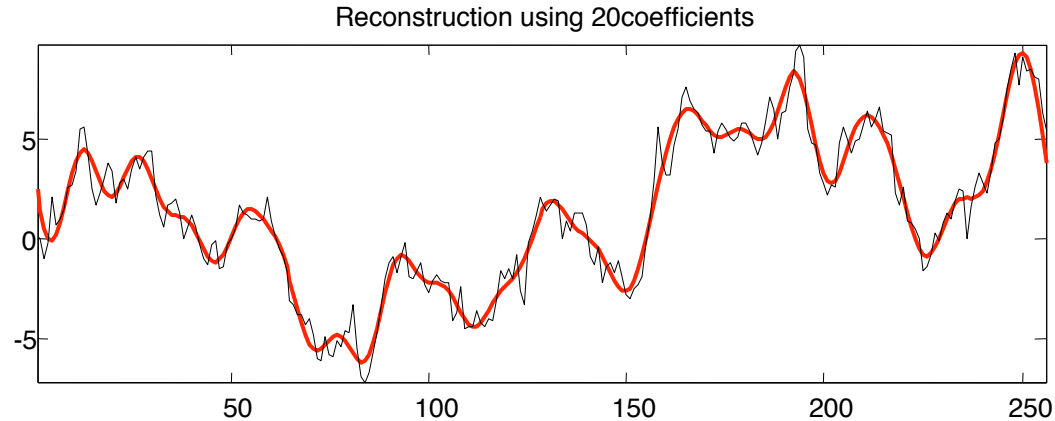
How much space we gain by compressing random walk data?



- 1 coeff > 60% of energy
- 10 coeff > 90% of energy

Fourier Decomposition

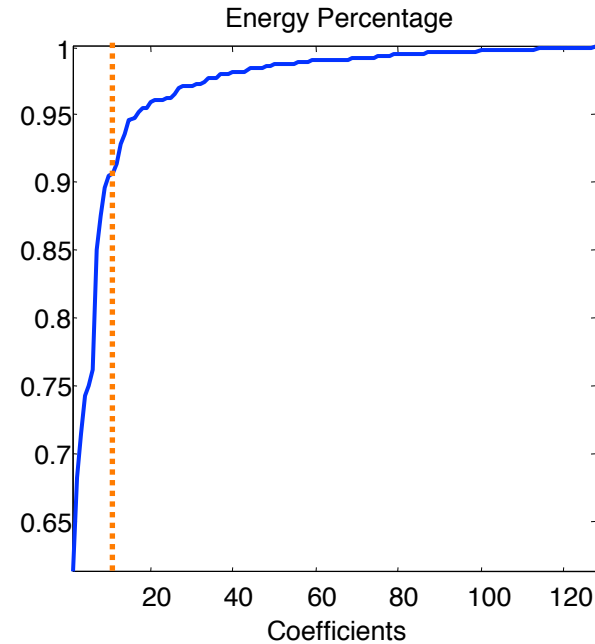
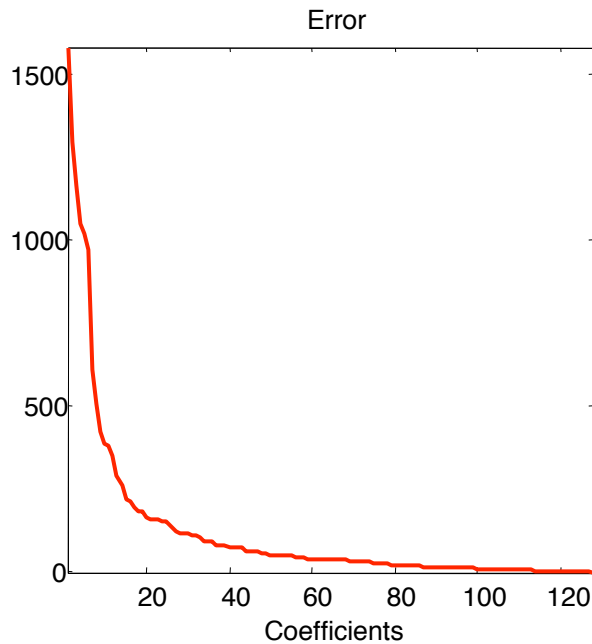
How much space we gain by compressing random walk data?



- 1 coeff > 60% of energy
- 10 coeff > 90% of energy

Fourier Decomposition

How much space we gain by compressing random walk data?

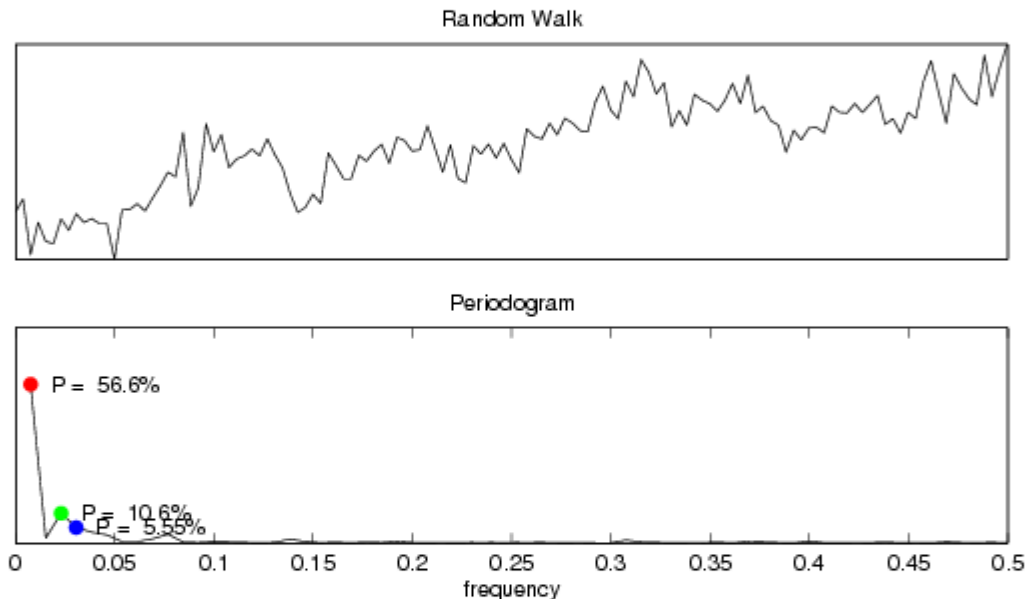


- 1 coeff > 60% of energy
- 10 coeff > 90% of energy

Fourier Decomposition

Which coefficients are important?

- We can measure the ‘energy’ of each coefficient
- $\text{Energy} = \text{Real}(X(f_k))^2 + \text{Imag}(X(f_k))^2$



```
fa = fft(a); % Fourier decomposition
N = length(a); % how many?
fa = fa(1:ceil(N/2)); % keep first half only
mag = 2*abs(fa).^2; % calculate energy
```

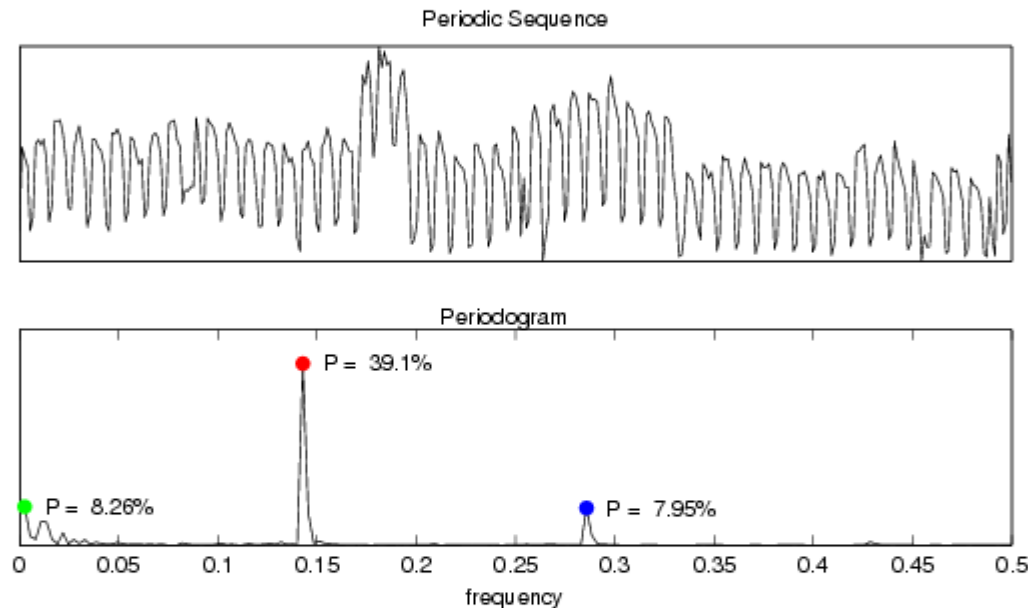
Most of data-mining research uses first k coefficients:

- Good for random walk signals (eg stock market)
- Easy to ‘index’
- Not good for general signals

Fourier Decomposition

Which coefficients are important?

- We can measure the ‘energy’ of each coefficient
- $\text{Energy} = \text{Real}(X(f_k))^2 + \text{Imag}(X(f_k))^2$



Usage of the coefficients with highest energy:

- Good for all types of signals
- Believed to be difficult to index
- CAN be indexed using *metric trees*

Code for Reconstructed Sequence $x(f)$

```
a = load('randomWalk.dat');
a = (a-mean(a))/std(a);           % z-normalization

fa = fft(a);

maxInd = ceil(length(a)/2);      % until the middle
N = length(a);

energy = zeros(maxInd-1, 1);
E = sum(a.^2);                   % energy of a

for ind=2:maxInd,

    fa_N = fa;                   % copy fourier
    fa_N(ind+1:N-ind+1) = 0;     % zero out unused
    r = real(ifft(fa_N));         % reconstruction

    plot(r, 'r', 'LineWidth', 2); hold on;
    plot(a, 'k');
    title(['Reconstruction using ' num2str(ind-1) ' coefficients']);
    set(gca, 'plotboxaspecratio', [3 1 1]);
    axis tight
    pause;                       % wait for key
    cla;                         % clear axis
end
```

keep

Ignore

keep

$x(f)$
-0.6280 + 0.2709i
-0.4929 + 0.0399i
-1.0143 + 0.9520i
0.7200 - 1.0571i
-0.0411 + 0.1674i
-0.5120 - 0.3572i
0.9860 + 0.8043i
-0.3680 - 0.1296i
-0.0517 - 0.0830i
-0.9158 + 0.4481i
1.1212 - 0.6795i
<u>0.2667 + 0.1100i</u>
0.2667 - 0.1100i
1.1212 + 0.6795i
-0.9158 - 0.4481i
-0.0517 + 0.0830i
-0.3680 + 0.1296i
0.9860 - 0.8043i
-0.5120 + 0.3572i
-0.0411 - 0.1674i
0.7200 + 1.0571i
-1.0143 - 0.9520i
-0.4929 - 0.0399i
-0.6280 - 0.2709i

Code for Plotting the Error

```
a = load('randomWalk.dat');
a = (a-mean(a))/std(a);           % z-normalization
fa = fft(a);
maxInd = ceil(length(a)/2);      % until the middle
N = length(a);
energy = zeros(maxInd-1, 1);
E = sum(a.^2);                   % energy of a

for ind=2:maxInd,
    fa_N = fa;                   % copy fourier
    fa_N(ind+1:N-ind+1) = 0;     % zero out unused
    r = real(ifft(fa_N));        % reconstruction

    energy(ind-1) = sum(r.^2);    % energy of reconstruction
    error(ind-1) = sum(abs(r-a).^2); % error
end

E = ones(maxInd-1, 1)*E;
error = E - energy;
ratio = energy ./ E;

subplot(1,2,1);                  % left plot
plot([1:maxInd-1], error, 'r', 'LineWidth',1.5);
subplot(1,2,2);                  % right plot
plot([1:maxInd-1], ratio, 'b', 'LineWidth',1.5);
```

This is the same

Lower Bounding using Fourier coefficients

Parseval's Theorem states that energy in the frequency domain equals the energy in the time domain:

$$\sum_{t=0}^{N-1} \|x(t)\|^2 = \sum_{k=0}^{N-1} \|X(f_{k/N})\|^2$$

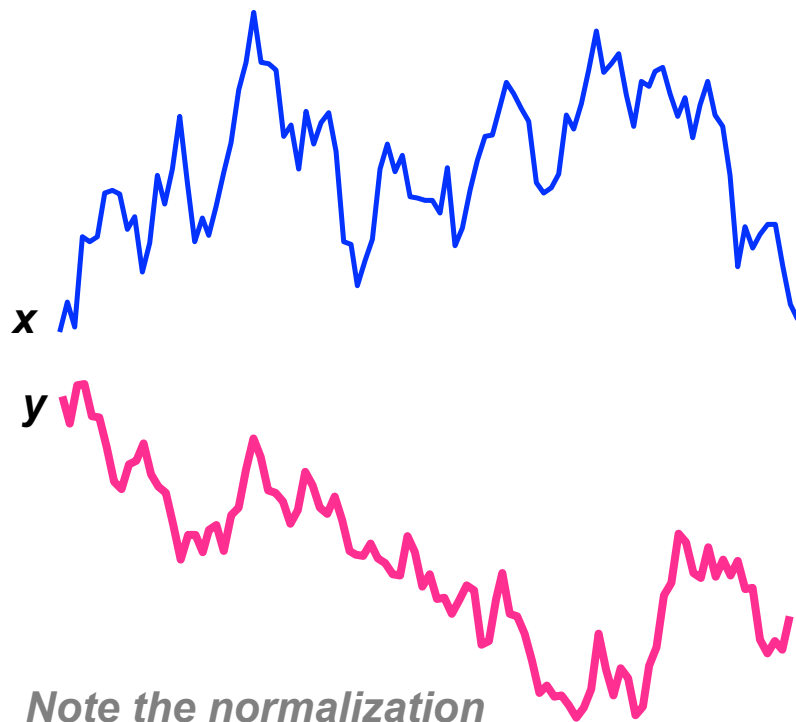
or, that

$$\sum_{t=0}^{N-1} \|x(t) - y(t)\|^2 = \sum_{k=0}^{N-1} \|X(f_{k/N}) - Y(f_{k/N})\|^2 \quad \text{Euclidean distance}$$

If we just keep some of the coefficients, their sum of squares always underestimates (ie lower bounds) the Euclidean distance:

$$\sum_{k=0}^m \|X(f_{k/N}) - Y(f_{k/N})\|^2 \leq \sum_{k=0}^{N-1} \|x(t) - y(t)\|^2, \quad m \leq N - 1$$

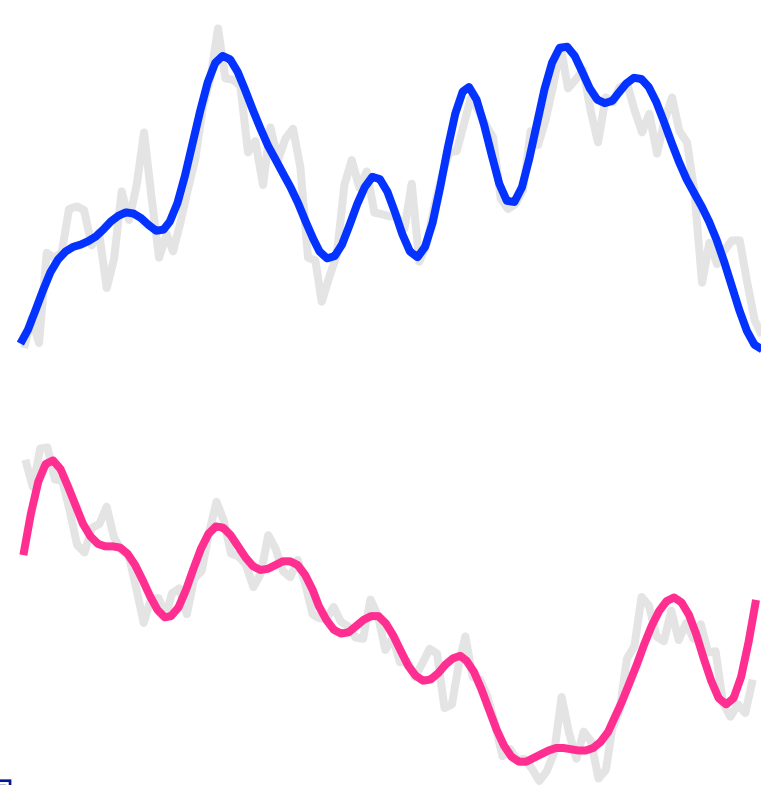
Lower Bounding using Fourier coefficients -Example



```
x = cumsum(randn(100,1));  
y = cumsum(randn(100,1));  
euclid_Time = sqrt(sum((x-y).^2));  
  
fx = fft(x)/sqrt(length(x));  
fy = fft(y)/sqrt(length(x));  
euclid_Freq = sqrt(sum(abs(fx - fy).^2));
```

120.9051

120.9051



*Keeping 10 coefficients
the distance is:
 $115.5556 < 120.9051$*

Fourier Decomposition

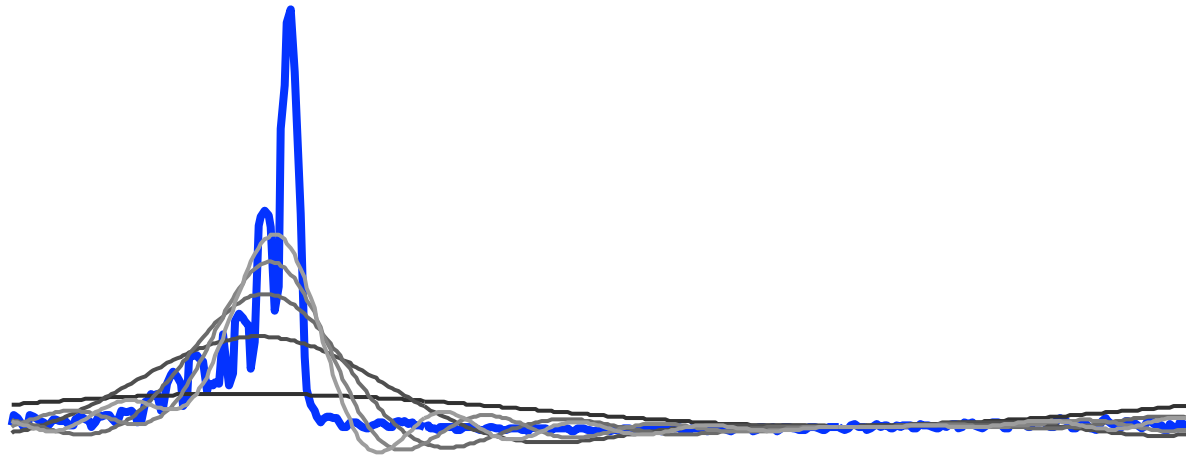


- **$O(n \log n)$ complexity**
- **Tried and tested**
- **Hardware implementations**
- **Many applications:**
 - compression
 - smoothing
 - periodicity detection

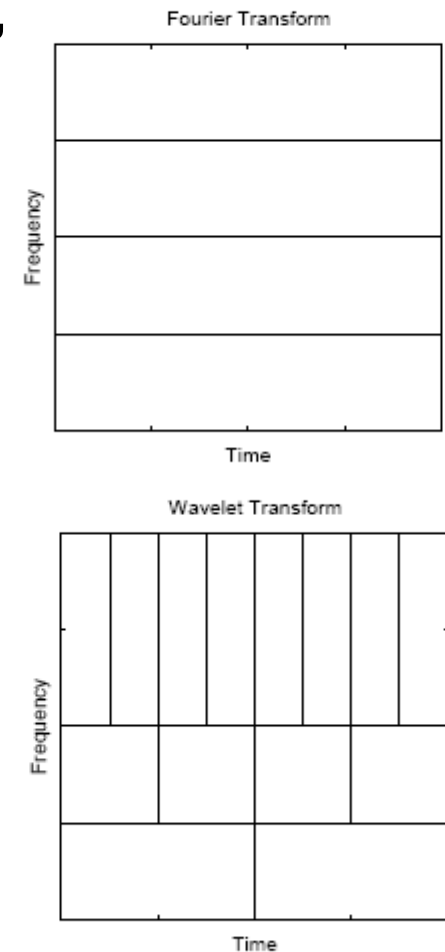
- **Not good approximation for *bursty* signals**
- **Not good approximation for signals with flat and busy sections**
(requires many coefficients)

Wavelets – Why exist?

- Similar concept with Fourier decomposition
- Fourier coefficients represent global contributions, wavelets are localized

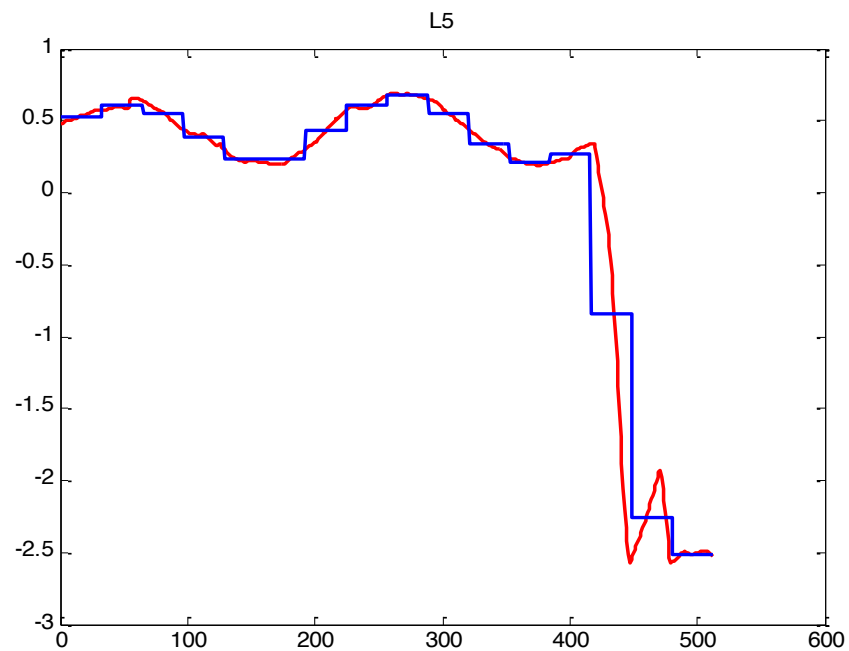


***Fourier is good for smooth, random walk data,
but not for bursty data or flat data***



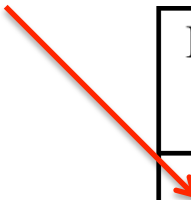
HAAR Wavelets and Time Series

- Multiresolution
- Fast to compute: $O(n)$



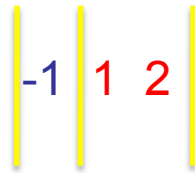
HAAR Wavelet Example

Full resolution



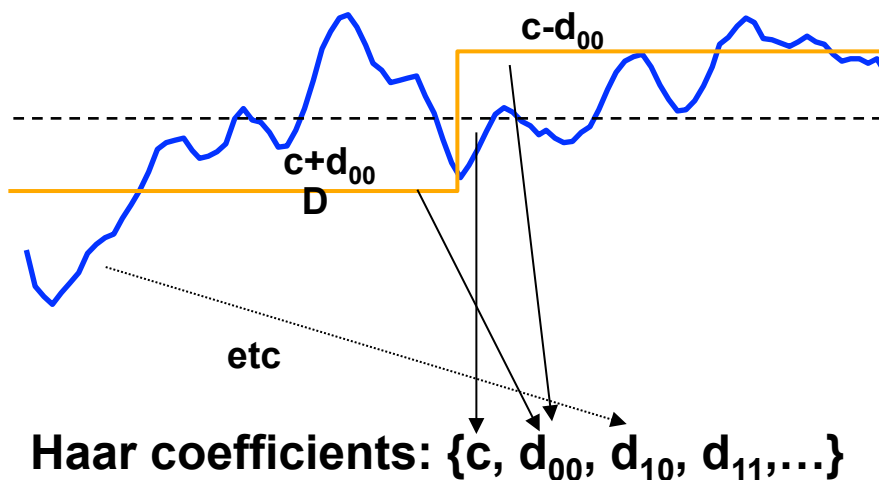
Resolution	Averages	Differences (coefficients)
8	(2 8 1 5 9 7 2 6)	
4	(5 3 8 4)	(-3 -2 1 -2)
2	(4 6)	(1 2)
1	5	-1

HAAR coefficients: (5 -1 1 2 -3 -2 1 -2)



Wavelets (Haar) - Intuition

- Wavelet coefficients, still represent an inner product (projection) of the signal with some basis functions.
- These functions have lengths that are powers of two (full sequence length, half, quarter etc)



An arithmetic example

$$X = [9, 7, 3, 5]$$

$$\text{Haar} = [6, 2, 1, -1]$$

$$c = 6 = (9+7+3+5)/4$$

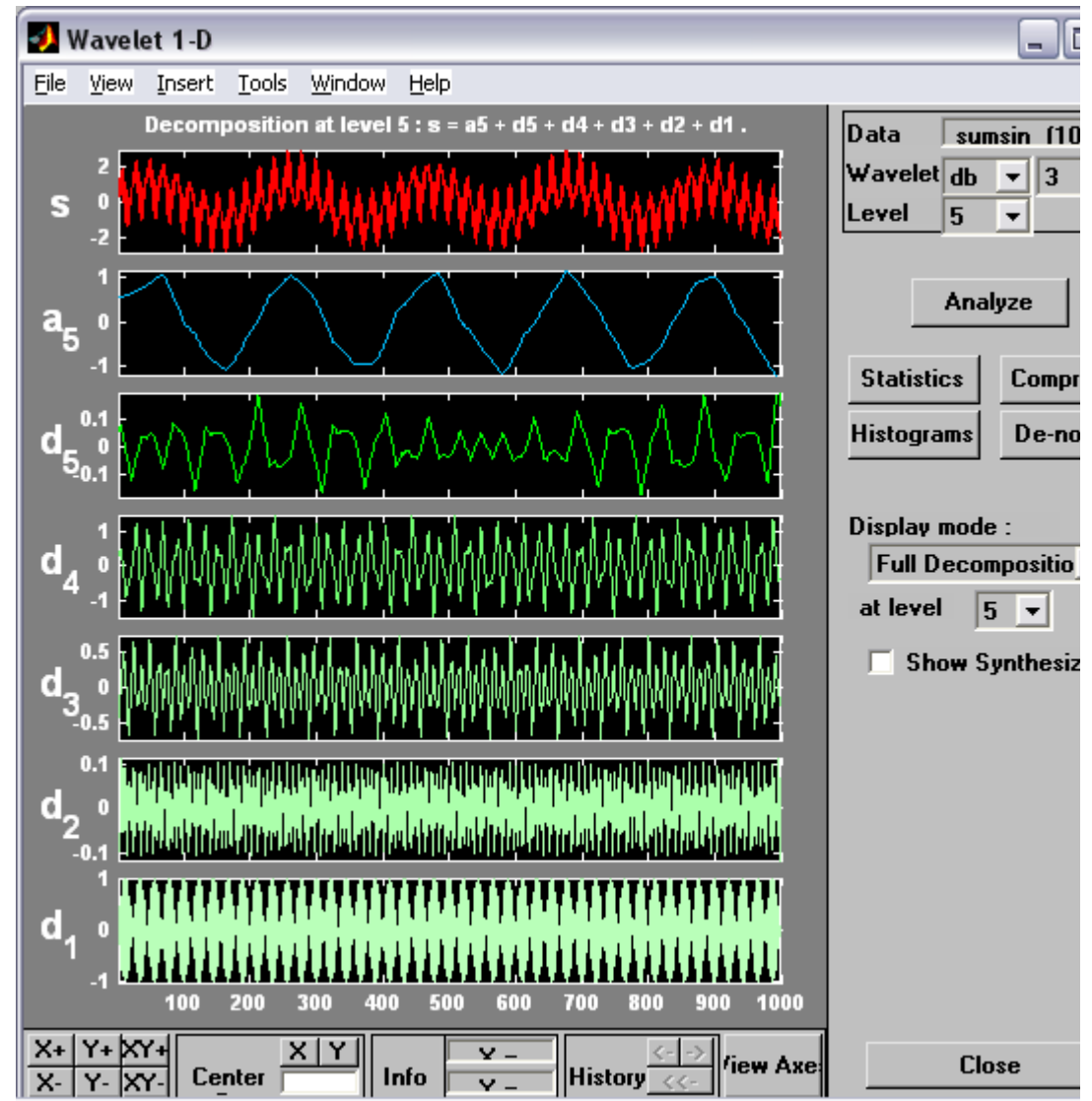
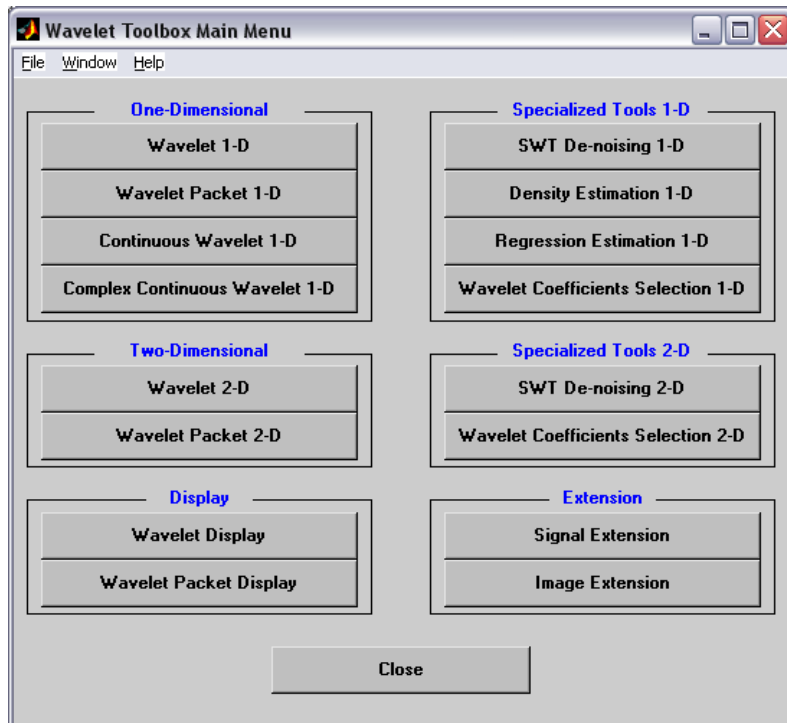
$$c + d_{00} = 6+2 = 8 = (9+7)/2$$

$$c - d_{00} = 6-2 = 4 = (3+5)/2$$

etc

Wavelets in Matlab

*Specialized Matlab interface
for wavelets*



See also: wavemenu

Code for Haar Wavelets

```
a = load('randomWalk.dat');
a = (a-mean(a))/std(a);           % z-normalization
maxlevels = wmaxlev(length(a),'haar');
[Ca, La] = wavedec(a,maxlevels,'haar');

% Plot coefficients and MRA
for level = 1:maxlevels
    cla;
    subplot(2,1,1);
    plot(detcoef(Ca,La,level)); axis tight;
    title(sprintf('Wavelet coefficients - Level %d',level));
    subplot(2,1,2);
    plot(wrcoef('d',Ca,La,'haar',level)); axis tight;
    title(sprintf('MRA - Level %d',level));
    pause;
end

% Top-20 coefficient reconstruction
[Ca_sorted, Ca_sortind] = sort(Ca);
Ca_top20 = Ca; Ca_top20(Ca_sortind(1:end-19)) = 0;
a_top20 = waverec(Ca_top20,La,'haar');
figure; hold on;
plot(a, 'b'); plot(a_top20, 'r');
```

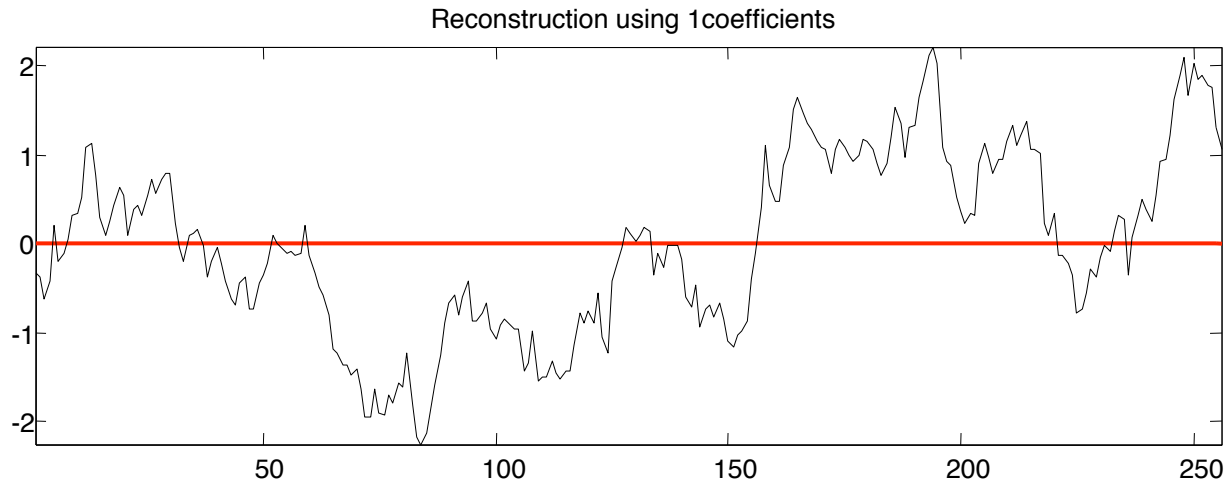
Wavelet Decomposition

- **$O(n)$ complexity**
 - **Hierarchical structure**
 - **Progressive transmission**
 - **Better localization**
 - **Good for bursty signals**
 - **Many applications:**
 - compression
 - periodicity detection
- **Most data-mining research still utilizes Haar wavelets because of their simplicity.**

PAA (Piecewise Aggregate Approximation)

also featured as **Piecewise Constant Approximation**

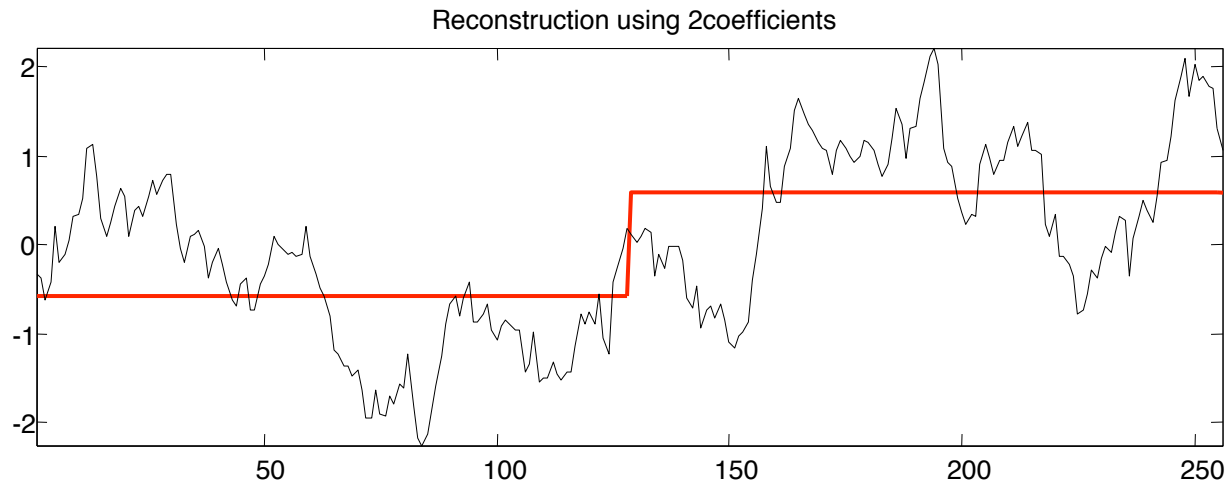
- **Represent time-series as a sequence of segments**
- **Essentially a projection of the Haar coefficients in time**



PAA (Piecewise Aggregate Approximation)

also featured as **Piecewise Constant Approximation**

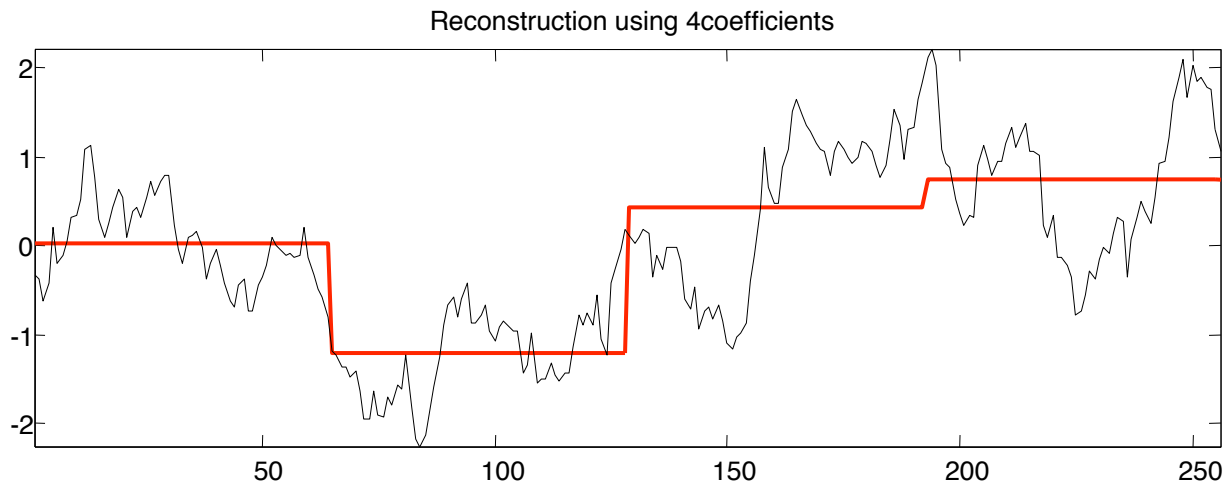
- **Represent time-series as a sequence of segments**
- **Essentially a projection of the Haar coefficients in time**



PAA (Piecewise Aggregate Approximation)

also featured as **Piecewise Constant Approximation**

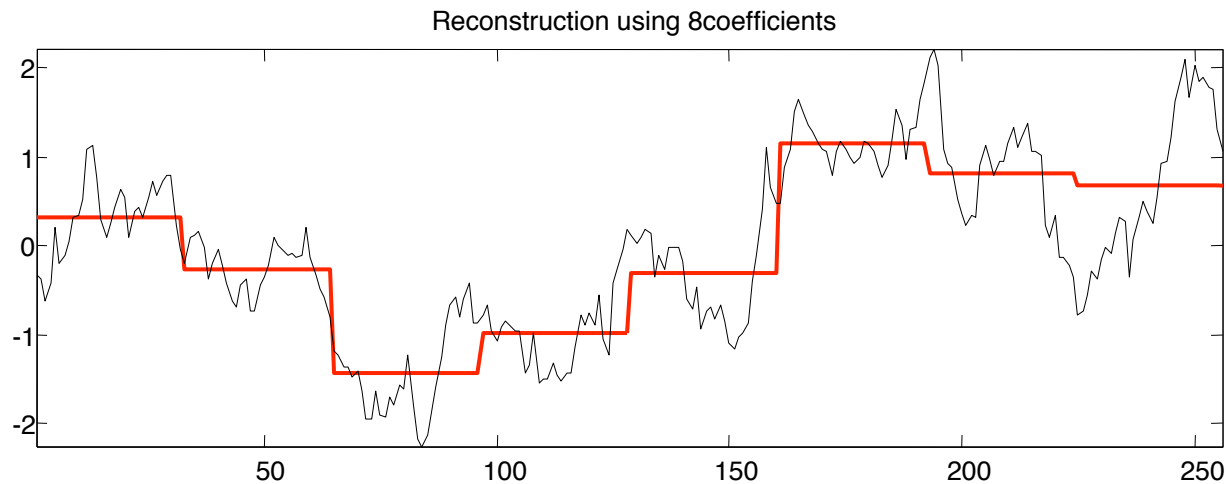
- **Represent time-series as a sequence of segments**
- **Essentially a projection of the Haar coefficients in time**



PAA (Piecewise Aggregate Approximation)

also featured as **Piecewise Constant Approximation**

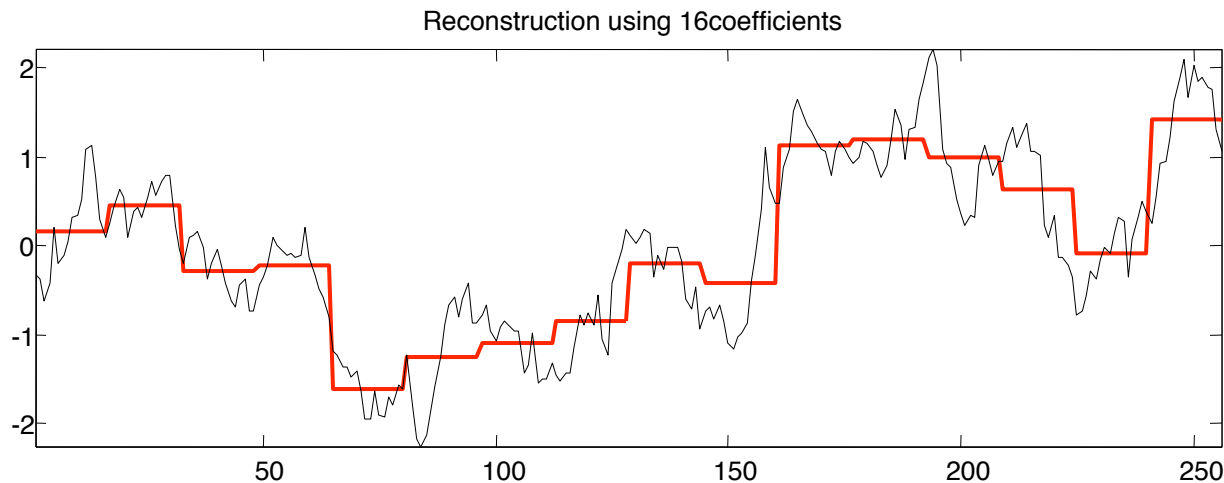
- **Represent time-series as a sequence of segments**
- **Essentially a projection of the Haar coefficients in time**



PAA (Piecewise Aggregate Approximation)

also featured as **Piecewise Constant Approximation**

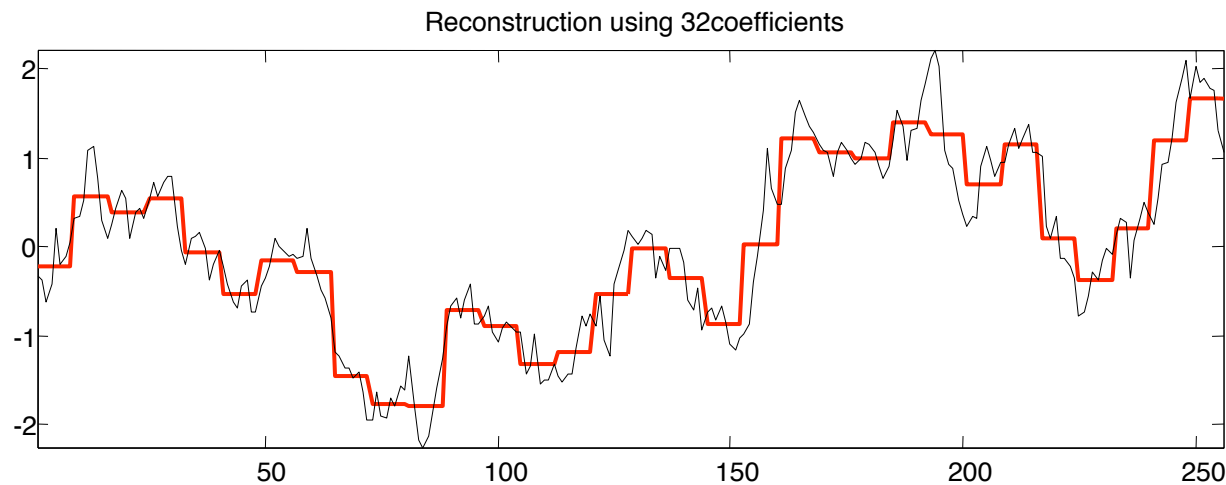
- **Represent time-series as a sequence of segments**
- **Essentially a projection of the Haar coefficients in time**



PAA (Piecewise Aggregate Approximation)

also featured as **Piecewise Constant Approximation**

- **Represent time-series as a sequence of segments**
- **Essentially a projection of the Haar coefficients in time**



PAA Matlab Code

```
function data = paa(s, numCoeff)
% PAA(s, numcoeff)
% s: sequence vector (Nx1 or Nx1)
% numCoeff: number of PAA segments
% data: PAA sequence (Nx1)

N = length(s);           % length of sequence
segLen = N/numCoeff;      % assume it's integer

sN = reshape(s, segLen, numCoeff); % break in segments
avg = mean(sN);           % average segments
data = repmat(avg, segLen, 1);    % expand segments
data = data(:);           % make column
```

N=8
segLen = 2

s

1

2

3

4

5

6

7

8

numCoeff

4

PAA Matlab Code

```
function data = paa(s, numCoeff)
% PAA(s, numcoeff)
% s: sequence vector (Nx1 or Nx1)
% numCoeff: number of PAA segments
% data: PAA sequence (Nx1)

N = length(s);           % length of sequence
segLen = N/numCoeff;      % assume it's integer

sN = reshape(s, segLen, numCoeff); % break in segments
avg = mean(sN);           % average segments
data = repmat(avg, segLen, 1); % expand segments
data = data(:);           % make column
```

N=8
segLen = 2

s 1 2 3 4 5 6 7 8

numCoeff 4

sN

1	3	5	7
2	4	6	8

PAA Matlab Code

```
function data = paa(s, numCoeff)
% PAA(s, numcoeff)
% s: sequence vector (Nx1 or Nx1)
% numCoeff: number of PAA segments
% data: PAA sequence (Nx1)

N = length(s);           % length of sequence
segLen = N/numCoeff;      % assume it's integer

sN = reshape(s, segLen, numCoeff); % break in segments
avg = mean(sN);           % average segments
data = repmat(avg, segLen, 1); % expand segments
data = data(:);           % make column
```

N=8
segLen = 2

s 1 2 3 4 5 6 7 8 **numCoeff** 4

sN

1	3	5	7
2	4	6	8

avg

1.5	3.5	5.5	7.5
-----	-----	-----	-----

PAA Matlab Code

```
function data = paa(s, numCoeff)
% PAA(s, numcoeff)
% s: sequence vector (1xN)
% numCoeff: number of PAA segments
% data: PAA sequence (1xN)

N = length(s);           % length of sequence
segLen = N/numCoeff;      % assume it's integer

sN = reshape(s, segLen, numCoeff); % break in segments
avg = mean(sN);           % average segments
data = repmat(avg, segLen, 1); % expand segments
data = data(:)';          % make row
```

N=8
segLen = 2

s 1 2 3 4 5 6 7 8 **numCoeff** 4

sN 1 3 5 7
 2 4 6 8

data 1.5 3.5 5.5 7.5
 1.5 3.5 5.5 7.5

avg 1.5 3.5 5.5 7.5

PAA Matlab Code

```
function data = paa(s, numCoeff)
% PAA(s, numcoeff)
% s: sequence vector (1xN)
% numCoeff: number of PAA segments
% data: PAA sequence (1xN)

N = length(s);           % length of sequence
segLen = N/numCoeff;     % assume it's integer

sN = reshape(s, segLen, numCoeff); % break in segments
avg = mean(sN);          % average segments
data = repmat(avg, segLen, 1);    % expand segments
data = data(:)';         % make row
```

N=8
segLen = 2

s 1 2 3 4 5 6 7 8 **numCoeff** 4

sN

1	3	5	7
2	4	6	8

data

1.5	3.5	5.5	7.5
1.5	3.5	5.5	7.5

avg

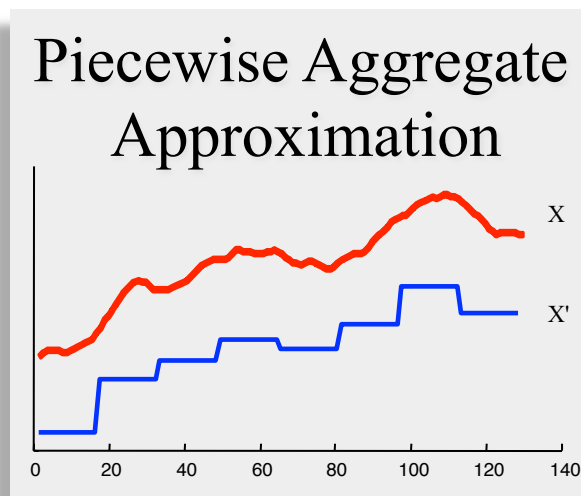
1.5	3.5	5.5	7.5
-----	-----	-----	-----

data

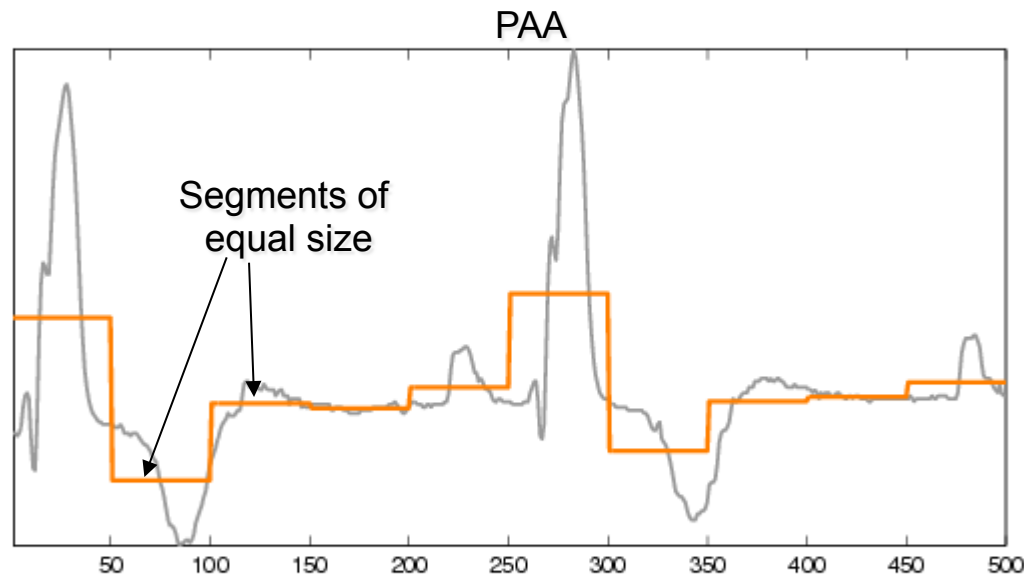
1.5	1.5	3.5	3.5	5.5	5.5	7.5	7.5
-----	-----	-----	-----	-----	-----	-----	-----

A Completely Pointless Slide

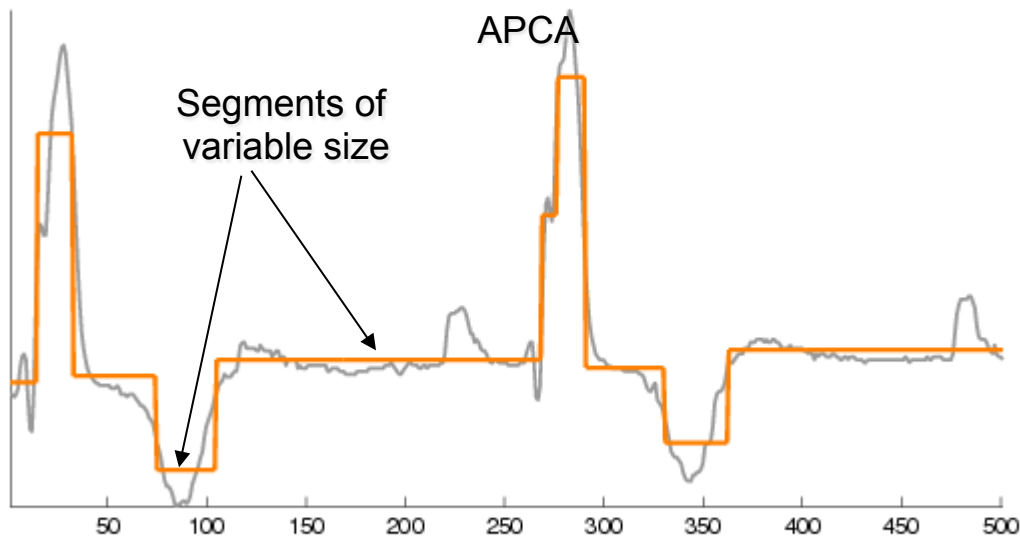
A piecewise constant
approximate of a time series,
and a piecewise constant
approximation of me!



APCA (Adaptive Piecewise Constant Approximation)



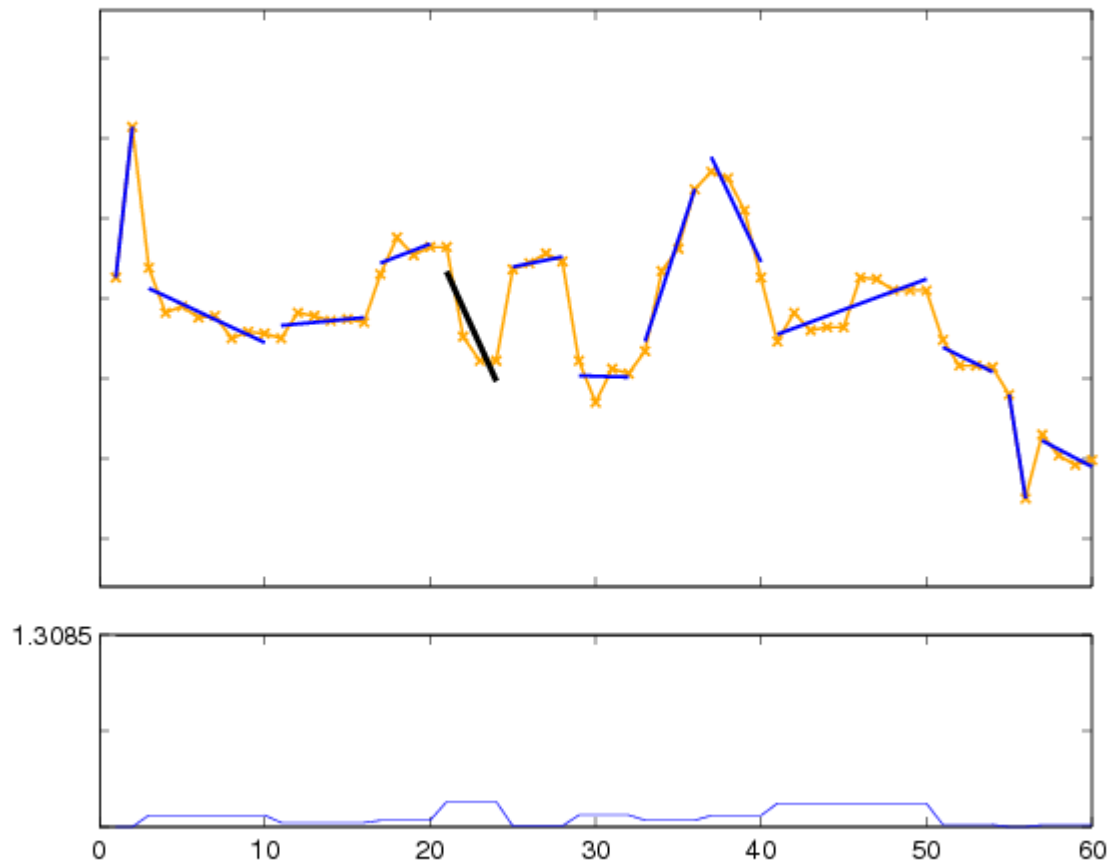
- Not all haar/PAA coefficients are equally important
- Intuition: Keep ones with the highest energy
- Segments of variable length



- APCA is good for bursty signals
- PAA requires **1 number** per segment, APCA requires **2: [value, length]**

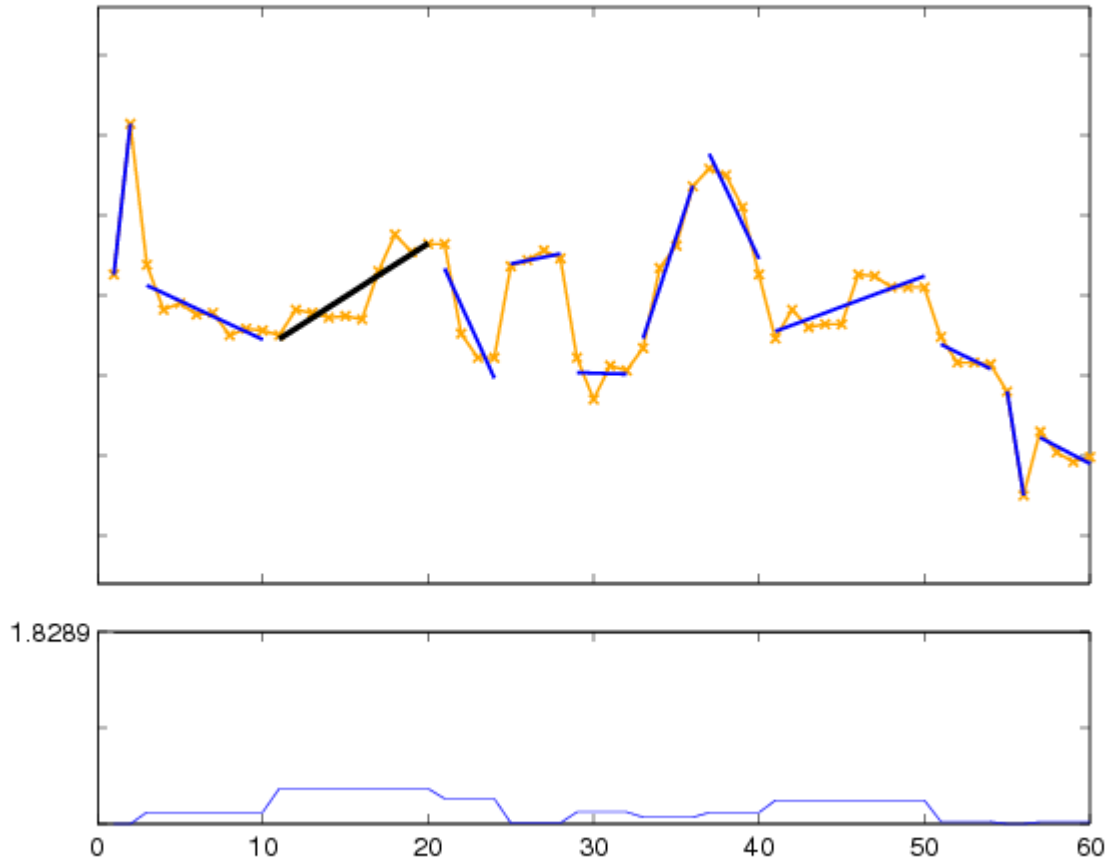
↓
E.g. 10 bits for a sequence of 1024 points

Piecewise Linear Approximation (PLA)



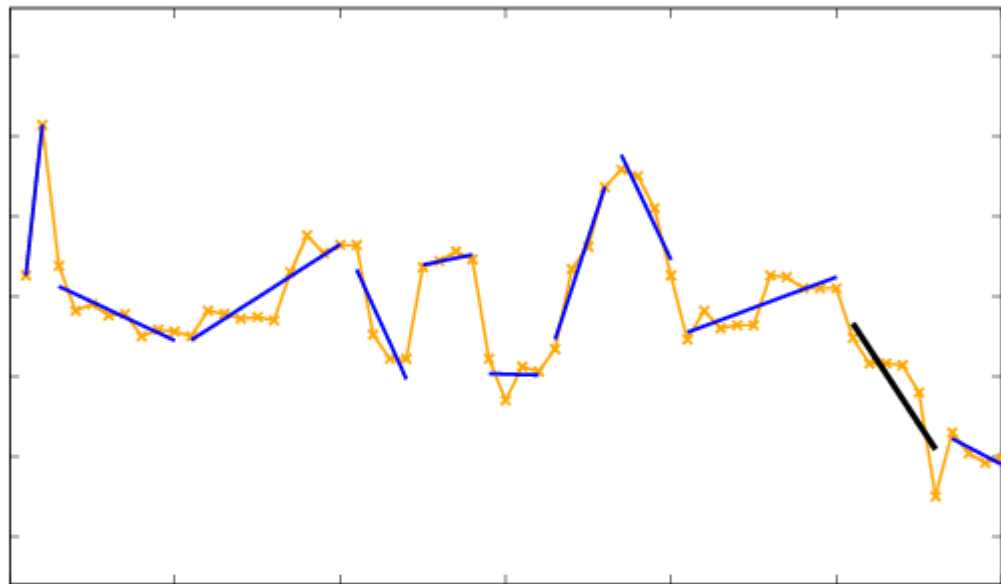
- Approximate a sequence with multiple linear segments
- First such algorithms appeared in *cartography* for map approximation
- Many implementations
 - Optimal
 - Greedy Bottom-Up
 - Greedy Top-down
 - Genetic, etc

Piecewise Linear Approximation (PLA)

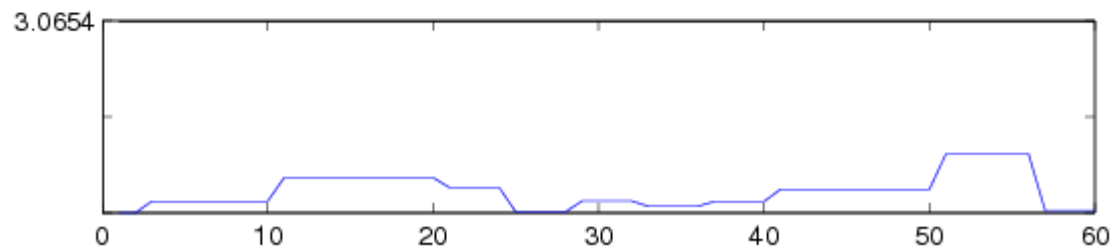


- Approximate a sequence with multiple linear segments
- First such algorithms appeared in *cartography* for map approximation

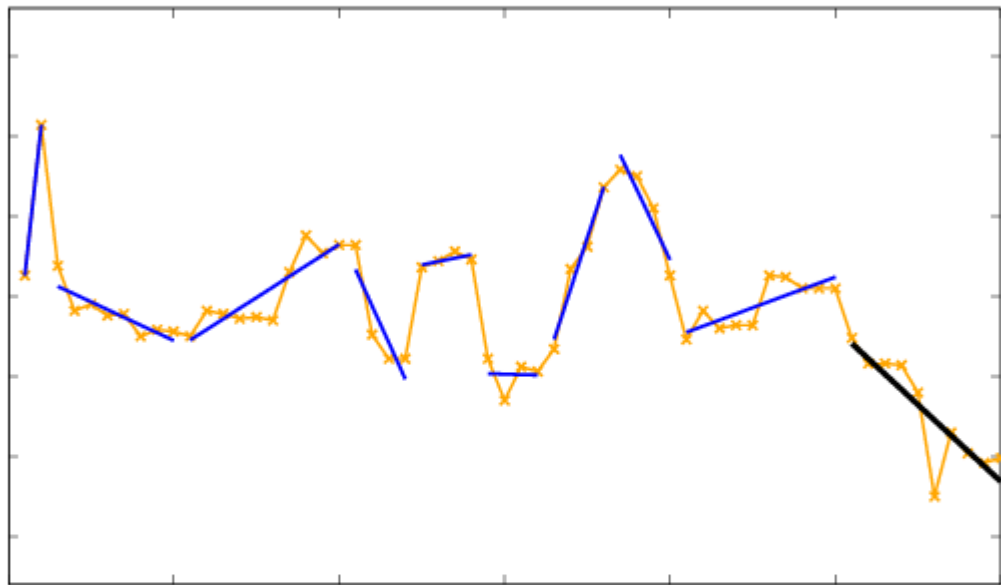
Piecewise Linear Approximation (PLA)



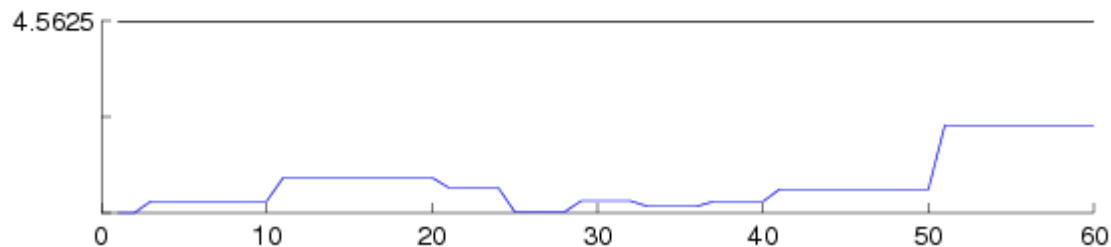
- Approximate a sequence with multiple linear segments
- First such algorithms appeared in *cartography* for map approximation



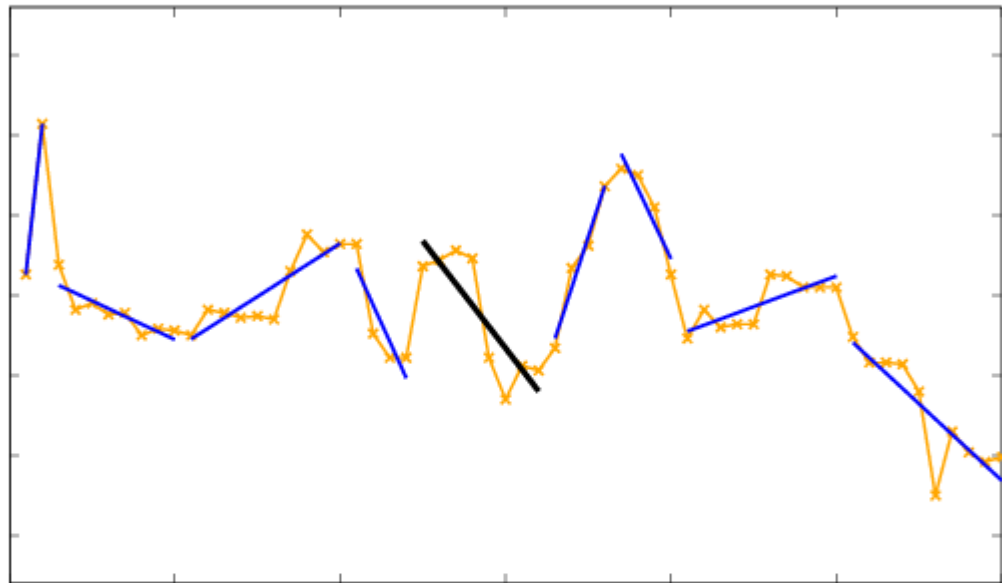
Piecewise Linear Approximation (PLA)



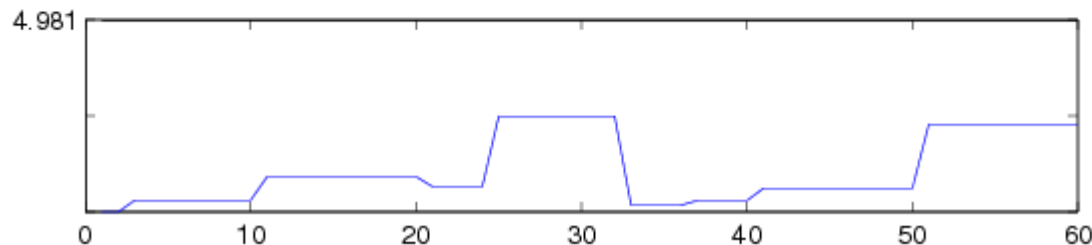
- Approximate a sequence with multiple linear segments
- First such algorithms appeared in *cartography* for map approximation



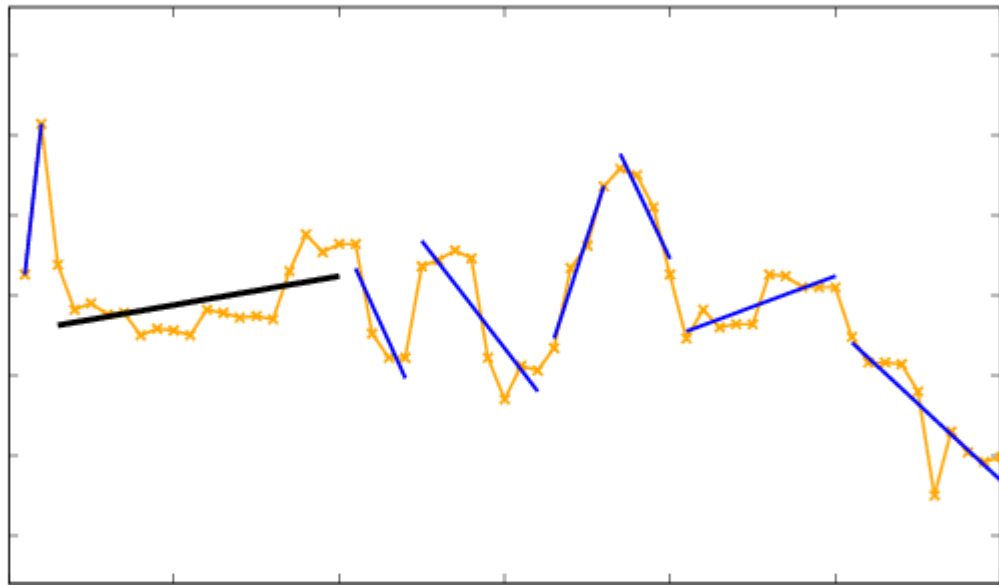
Piecewise Linear Approximation (PLA)



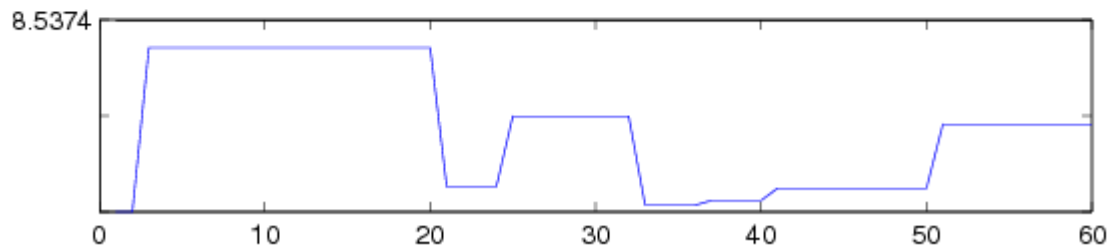
- Approximate a sequence with multiple linear segments
- First such algorithms appeared in *cartography* for map approximation



Piecewise Linear Approximation (PLA)



- Approximate a sequence with multiple linear segments
- First such algorithms appeared in *cartography* for map approximation



Piecewise Linear Approximation (PLA)



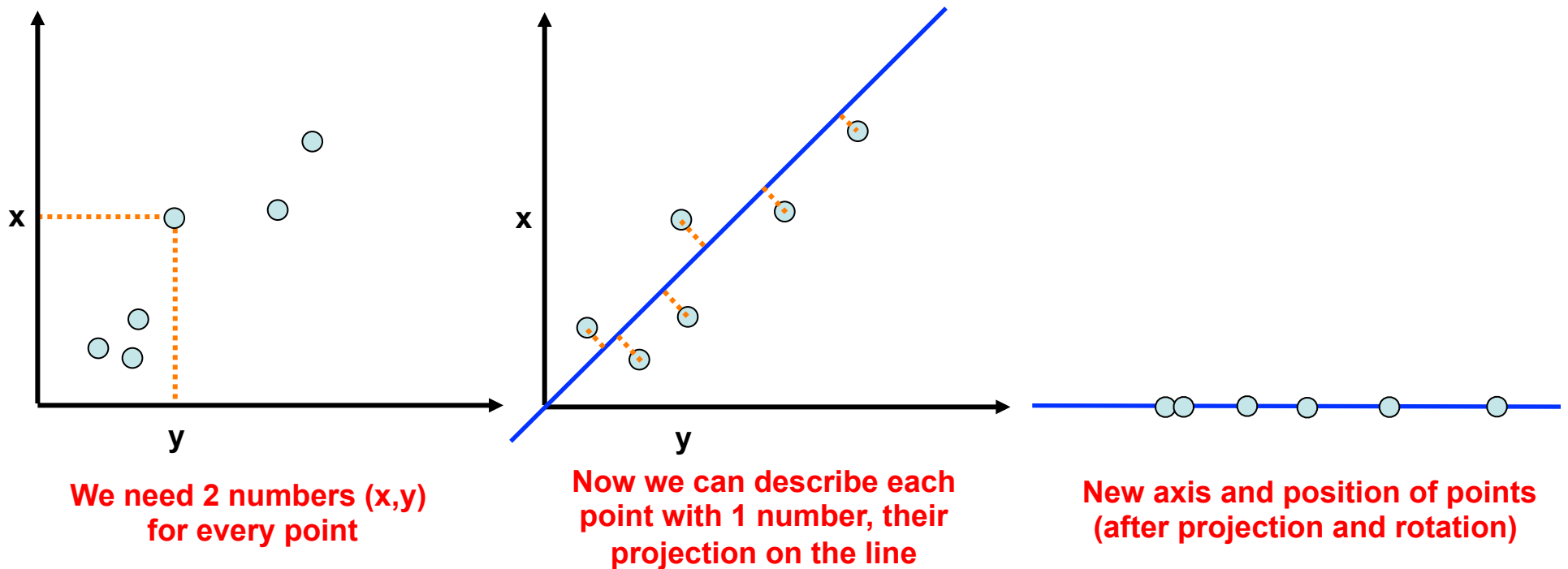
- **$O(n \log n)$ complexity for “bottom up” algorithm**
- **Incremental computation possible**
- **Provable error bounds**
- **Applications for:**
 - Image / signal simplification
 - Trend detection



- **Visually not very smooth or pleasing.**

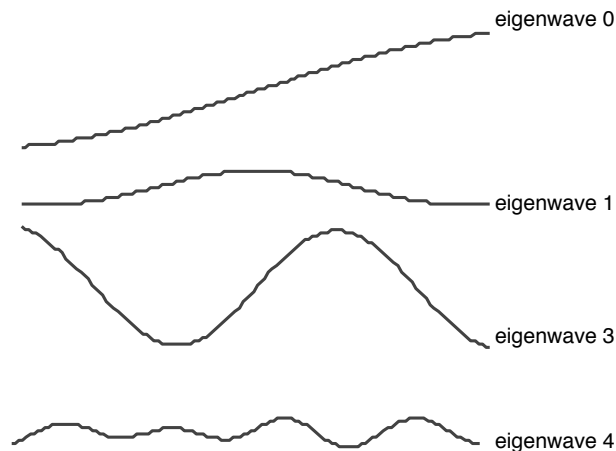
Singular Value Decomposition (SVD)

- SVD attempts to find the ‘optimal’ basis for describing a set of multidimensional points
- Objective: Find the axis (‘directions’) that describe better the data variance

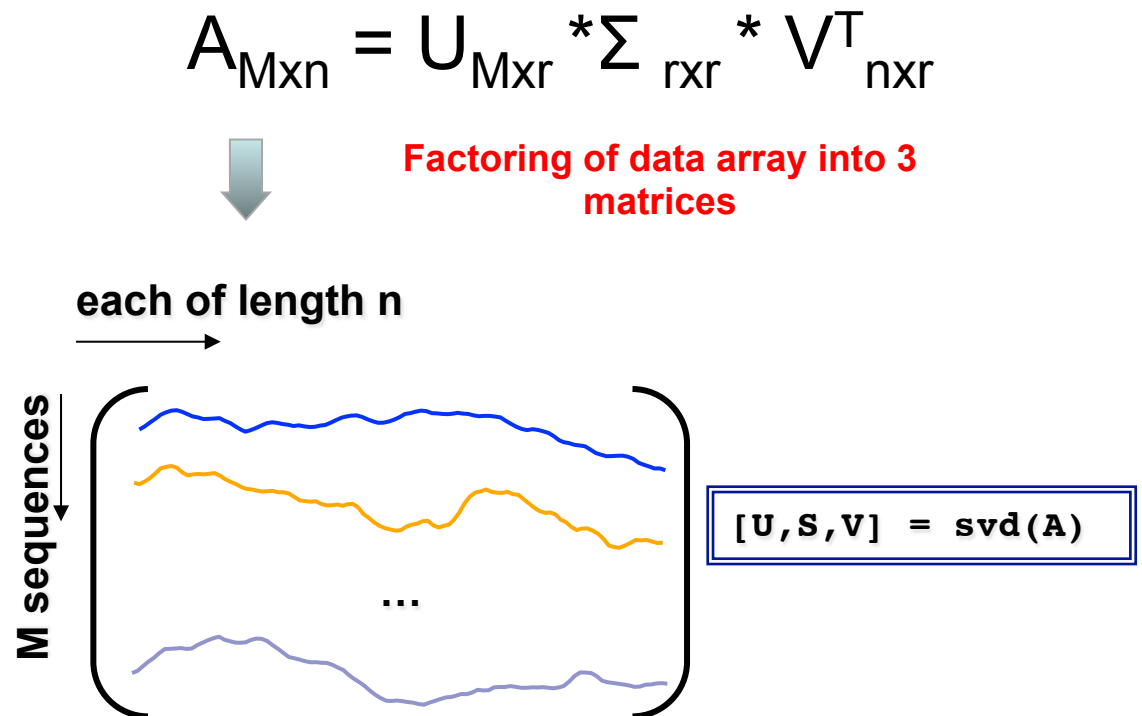


Singular Value Decomposition (SVD)

- Each time-series is essentially a multidimensional point
- Objective: Find the ‘eigenwaves’ (basis) whose linear combination describes best the sequences. Eigenwaves are data-dependent.



A linear combination of the eigenwaves can produce any sequence in the database



Code for SVD / PCA

```
A = cumsum(randn(100,10));  
% z-normalization  
A = (A-repmat(mean(A),size(A,1),1))./repmat(std(A),size(A,1),1);  
[U,S,V] = svd(A,0);  
  
% Plot relative energy  
figure; plot(cumsum(diag(S).^2)/norm(diag(S))^2);  
set(gca, 'YLim', [0 1]); pause;  
  
% Top-3 eigenvector reconstruction  
A_top3 = U(:,1:3)*S(1:3,1:3)*V(:,1:3)';  
  
% Plot original and reconstruction  
figure;  
for i = 1:10  
    cla;  
    subplot(2,1,1);  
    plot(A(:,i));  
    title('Original'); axis tight;  
    subplot(2,1,2);  
    plot(A_top3(:,i));  
    title('Reconstruction'); axis tight;  
    pause;  
end
```

Singular Value Decomposition



- **Optimal dimensionality reduction in Euclidean distance sense**
- **SVD is a very powerful tool in many domains:**
 - Websearch (PageRank)

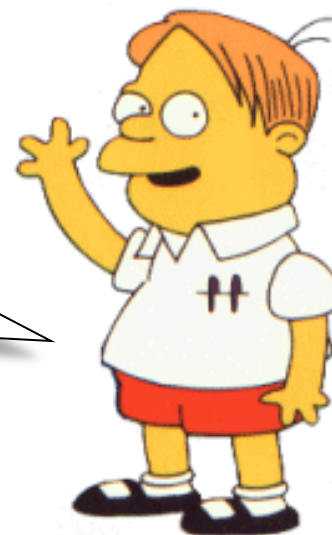


- **Cannot be applied for just one sequence. A set of sequences is required.**
- **Addition of a sequence in database requires recomputation**
- **Very costly to compute.**
Time: $\min\{O(M^2n), O(Mn^2)\}$
Space: $O(Mn)$
M sequences of length n

Lower Bounding functions are known for wavelets, Fourier, SVD, piecewise polynomials, and Chebyshev Polynomials. They are all real-valued representations

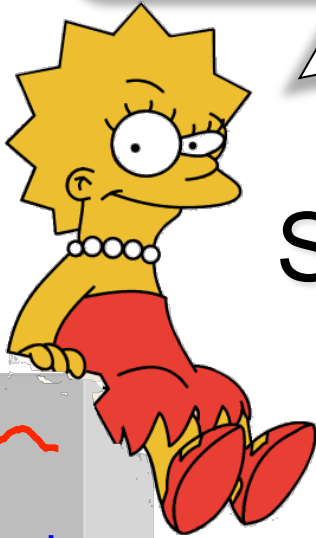


While there are more than 200 different symbolic or discrete ways to approximate time series, none except for one: SAX



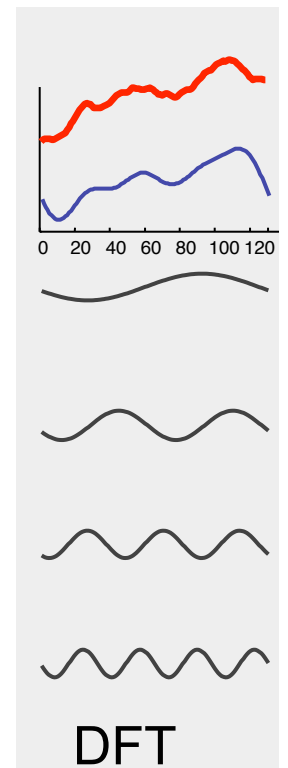
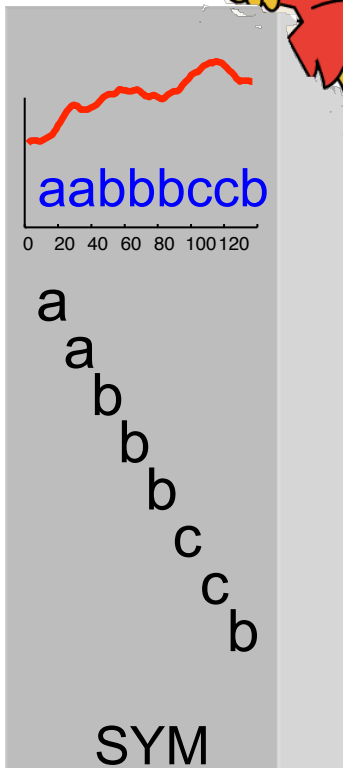
We'll spend some time on SAX, since many recent time series pattern discovery algorithms use it

Why do we care so much about symbolic representations?



Symbolic Representations Allow:

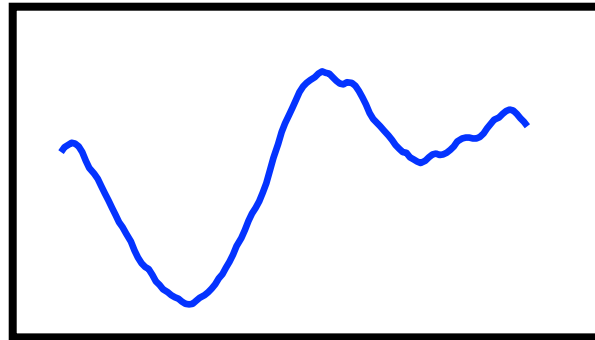
- Hashing
- Suffix Trees
- Markov Models
- Utilize ideas from text processing/statistical language processing/bioinformatics community
- Much less space requirement
- etc



There is *one* symbolic representation of time series, that allows...

- Lower bounding of Euclidean distance
- Lower bounding of the DTW distance
- Dimensionality Reduction
- Numerosity Reduction

SAX: Symbolic Aggregate approXimation

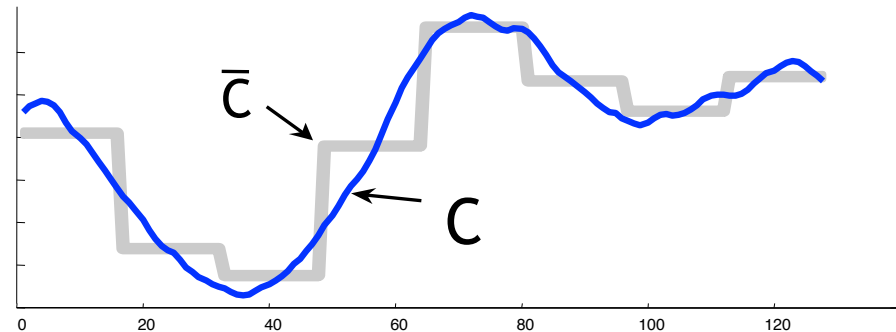


baabccbc

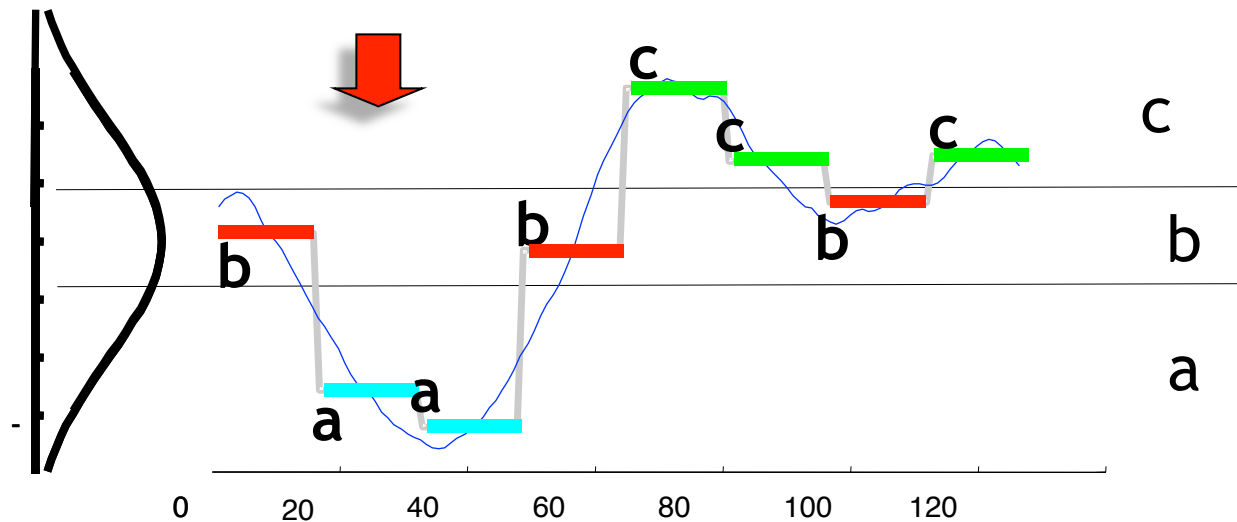
- Lower bounds Euclidean distance
- Achieves dimensionality reduction

How do we obtain SAX?

(normalized) Time Series \rightarrow PAA
(Piecewise Aggregate Approximation)



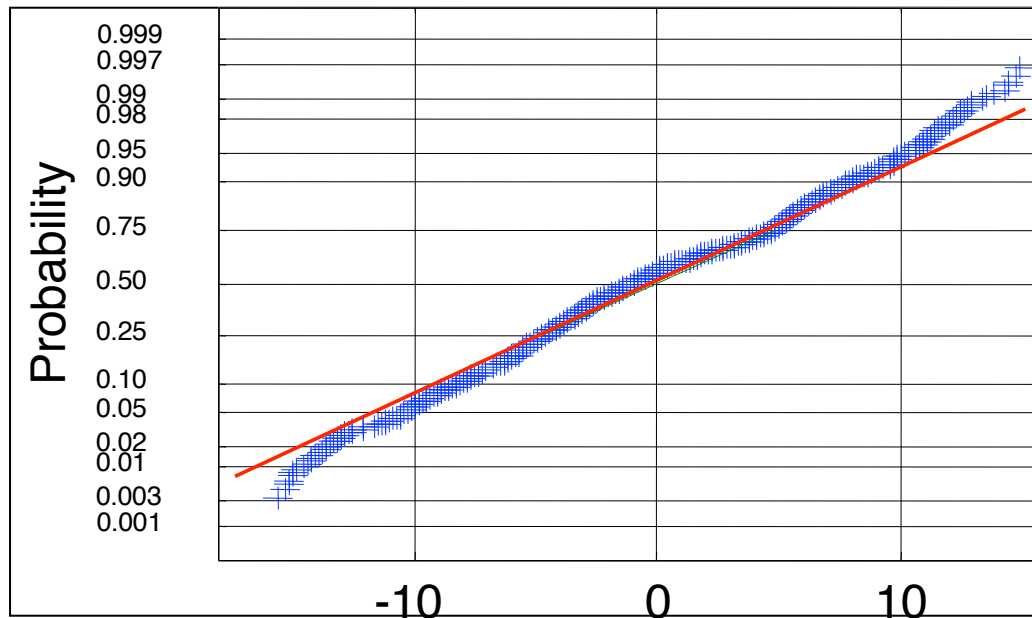
PAA \rightarrow Symbols



baabccbc

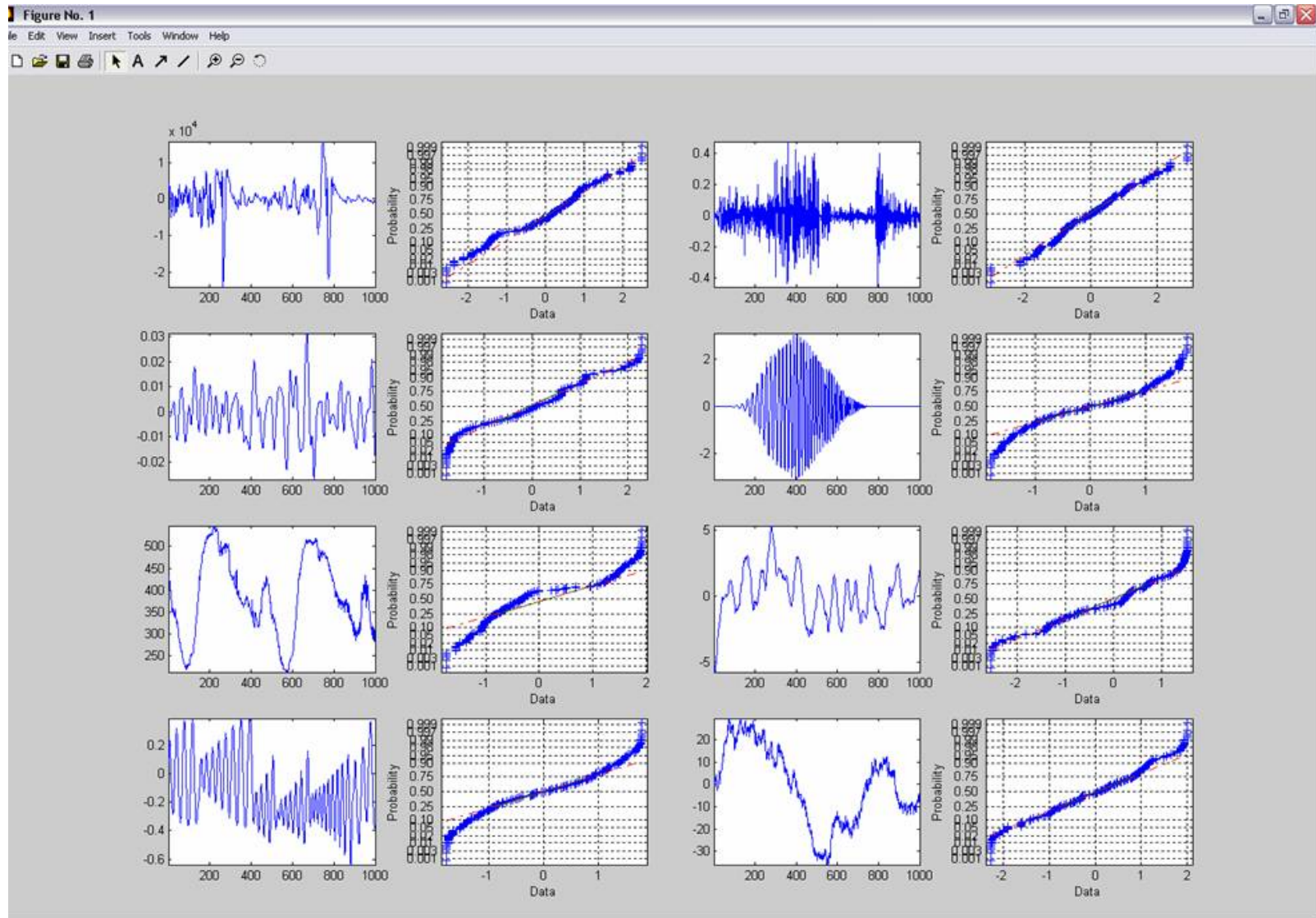
Why a Gaussian?

Short time series subsequences tend to have a highly Gaussian distribution

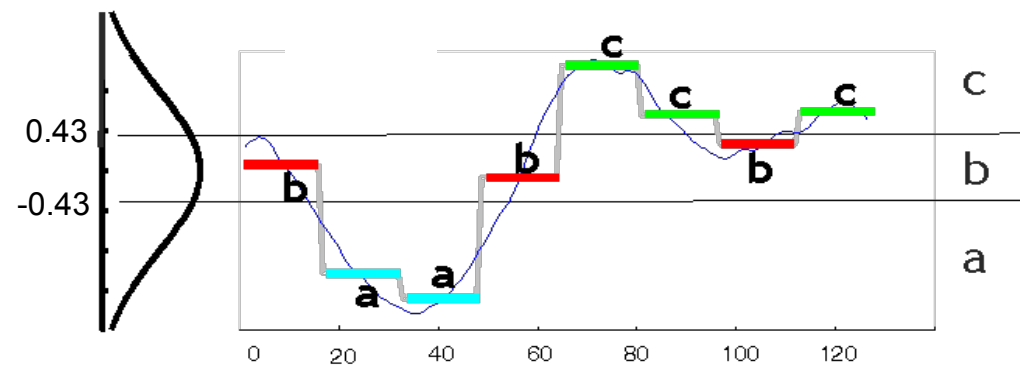


A normal probability plot of the (cumulative) distribution of values from subsequences of length 128.

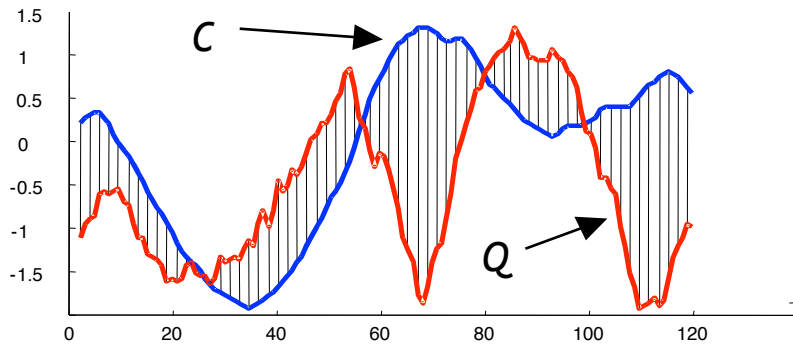
Normality Plots



Determining Breakpoints

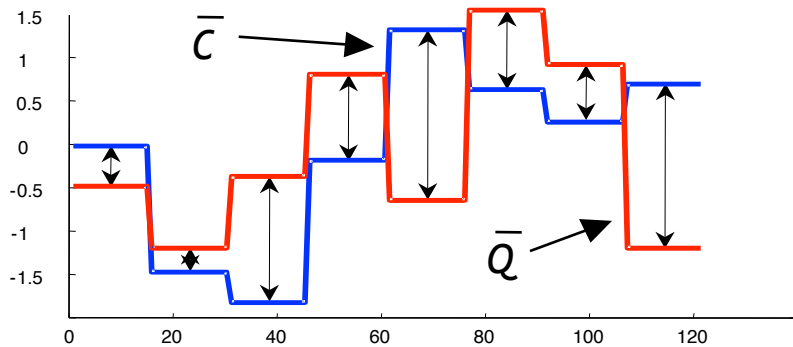


		Alphabet size							
Breakpoints		3	4	5	6	7	8	9	10
	β_1	-0.43	-0.67	-0.84	-0.97	-1.07	-1.15	-1.22	-1.28
	β_2	0.43	0	-0.25	-0.43	-0.57	-0.67	-0.76	-0.84
	β_3		0.67	0.25	0	-0.18	-0.32	-0.43	-0.52
	β_4			0.84	0.43	0.18	0	-0.14	-0.25
	β_5				0.97	0.57	0.32	0.14	0
	β_6					1.07	0.67	0.43	0.25
	β_7						1.15	0.76	0.52
	β_8							1.22	0.84
	β_9								1.28



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

Euclidean Distance



$$DR(\bar{Q}, \bar{C}) \equiv \sqrt{\frac{n}{w}} \sqrt{\sum_{i=1}^w (\bar{q}_i - \bar{c}_i)^2}$$

PAA distance lower-bounds the Euclidean Distance

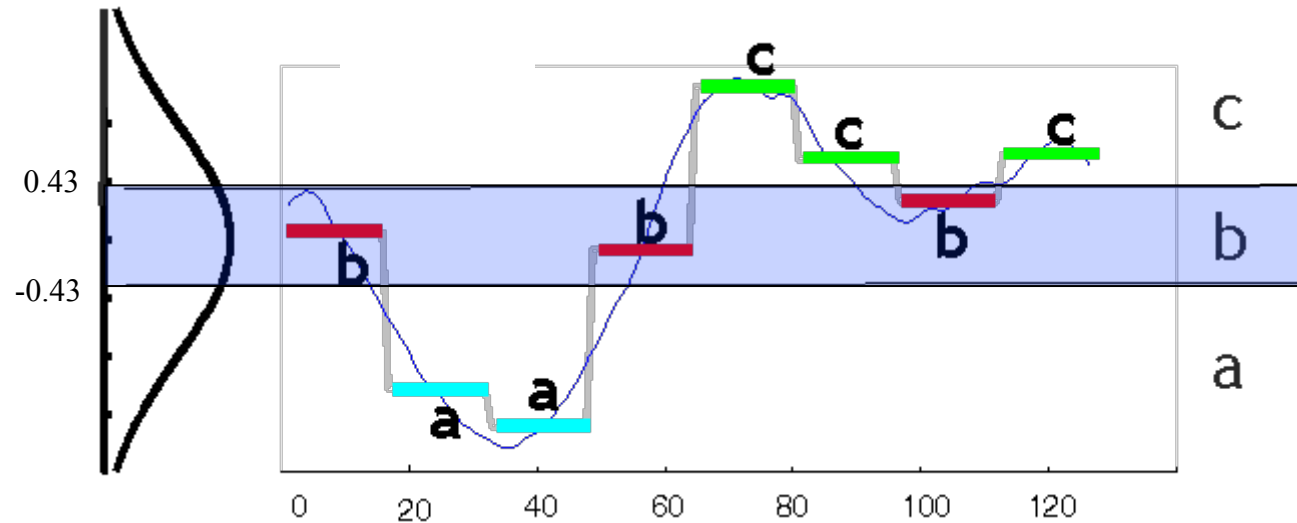
\hat{C} = baabccbc
 \hat{Q} = babcacca

$\updownarrow \updownarrow \updownarrow \updownarrow \updownarrow \updownarrow \updownarrow \updownarrow$

$$MINDIST(\hat{Q}, \hat{C}) \equiv \sqrt{\frac{n}{w}} \sqrt{\sum_{i=1}^w (dist(\hat{q}_i, \hat{c}_i))^2}$$

dist() can be implemented using a table lookup.

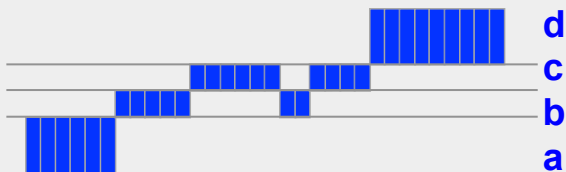
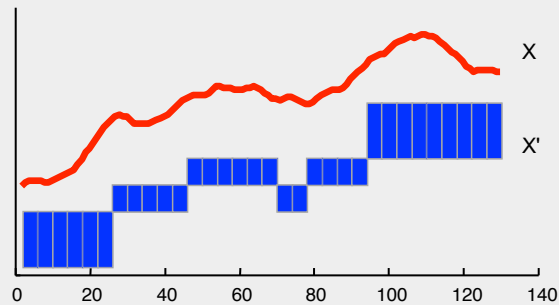
Computing String Distances



	a	b	c
a	0	0	0.86
b	0	0	0
c	0.86	0	0

Distance table

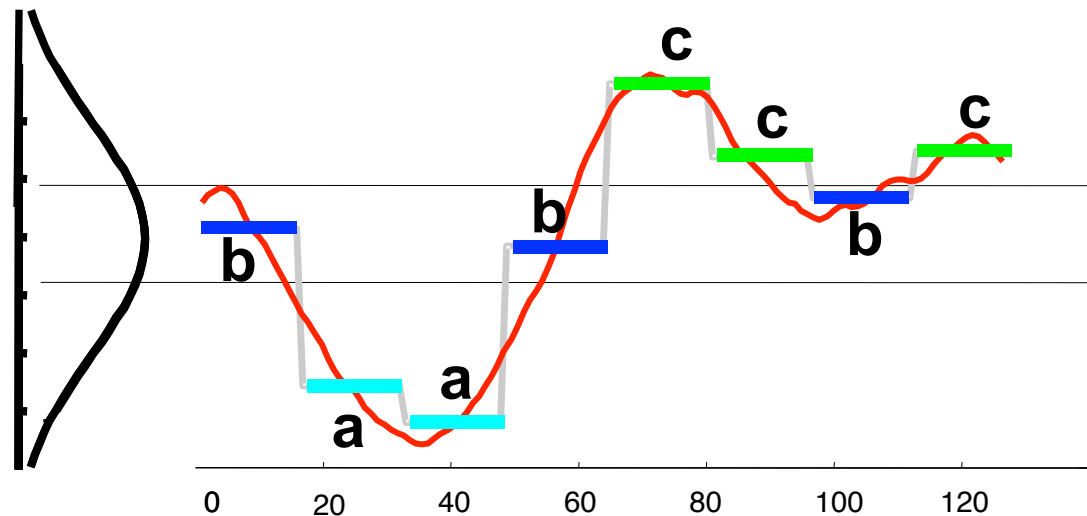
SAX: Symbolic Aggregate approXimation



aaaaaabbbbbccccccbbccccddddd

SAX is (for the first time) a symbolic representation that allows

- Lower bounding of Euclidean distance
- Dimensionality Reduction
- Numerosity Reduction



baabccbc

Symbolic Approximations

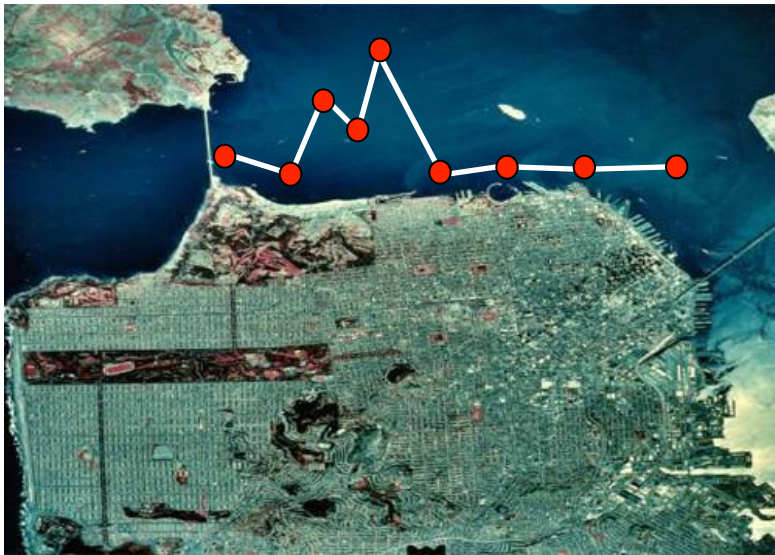


- **Linear complexity**
- **After ‘symbolization’ many tools from bioinformatics can be used**
 - Markov models
 - Suffix-Trees, etc

- **Number of regions (alphabet length) can affect the quality of result**

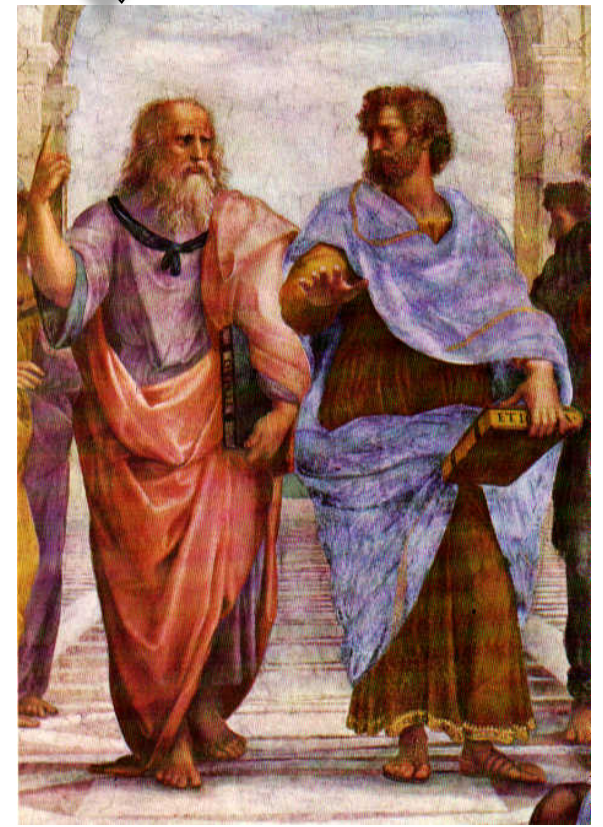
Multidimensional Time-Series

- Catching momentum in the last decade
- Applications for *mobile trajectories*, *sensor networks*, *epidemiology*, etc



- Let's see how to approximate 2D trajectories with Minimum Bounding Rectangles

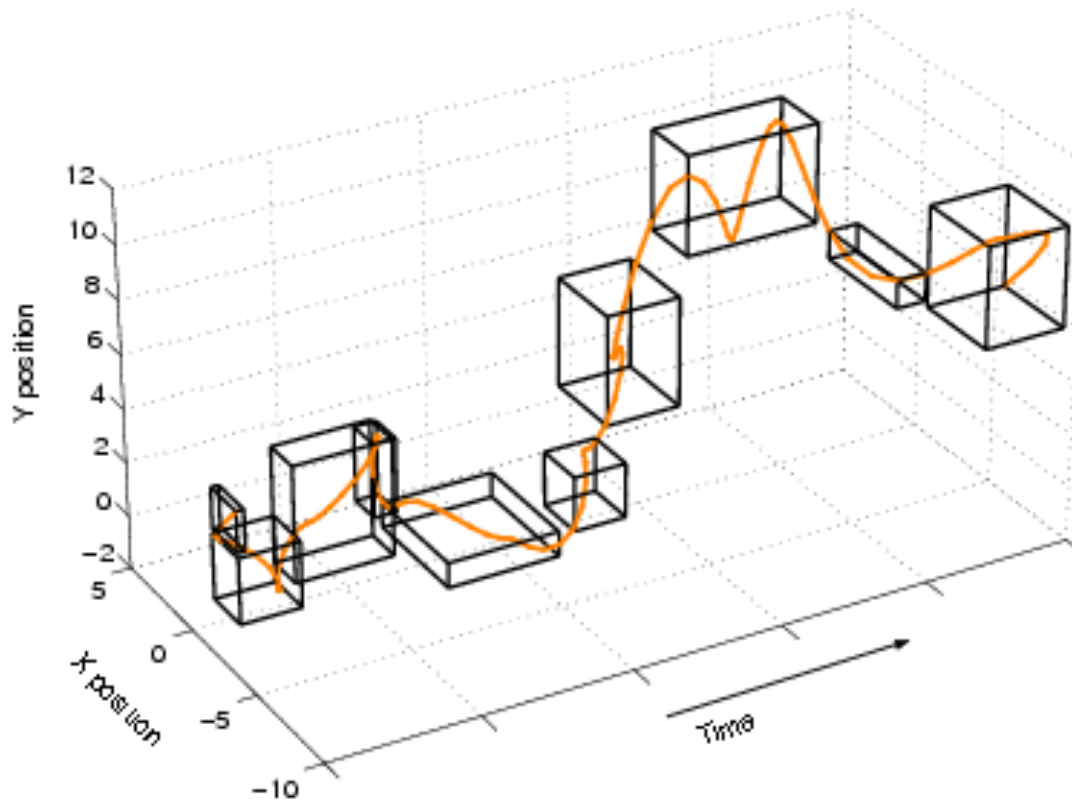
Ari, are you sure the world is not 1D?



Aristotle

Multidimensional MBRs

Find Bounding rectangles that completely contain a trajectory given some optimization criteria (eg minimize volume)



On my income tax 1040 it says "Check this **box** if you are blind." I wanted to put a check mark about three inches away.

- Tom Lehrer, lecturing in "The Nature of Math"

So which dimensionality reduction is the best?



Fourier is
good...



1993

PAA!



2000

APCA is
better
than
PAA!



2001

Chebyshev
is better
than
APCA!



2004

The
future is
symbolic!



2005

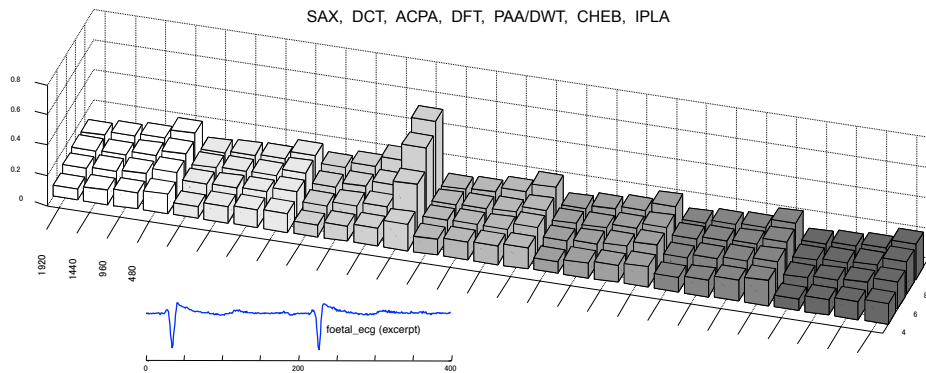
Absence of proof is no proof of absence.

- [Michael Crichton](#)

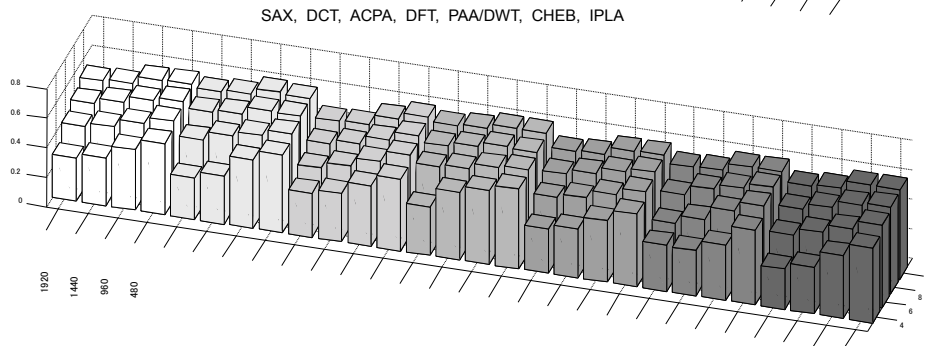
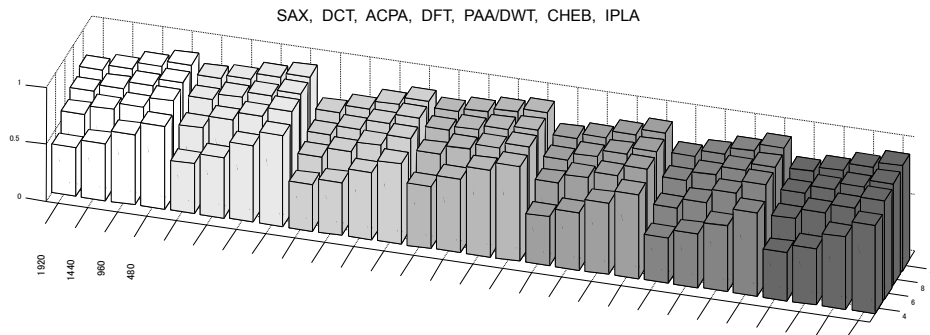
Comparison of all Dimensionality Reduction Techniques

- We have already compared features (Does representation X allow weighted queries, queries of arbitrary lengths, is it simple to implement...)
- We can compare the indexing efficiency: How long does it take to find the best answer to the query.
- It turns out that the fairest way to measure this is to measure the number of times we have to retrieve an item from disk.

Comparison of Time Series Representation Methods



TLB on an ECG data set



- 8 representation methods:
SAX, DFT, DWT, DCT, PAA, CHEB, APCA, IPLA
- Use tightness of lower bounds (TLB) as a metric for comparison:
 - $$TLB = \text{LowerBoundDist} / \text{TrueEuclideanDist}$$
- The tightness of lower bounding (\Rightarrow pruning power, \Rightarrow effectiveness of the indexing) of different representation methods, for the most part, makes little difference on various data sets

TLB on a bursty data set

TLB on a periodic data set

We have seen different distance measures and time series representations

- Many time series data mining tasks are really about
 - Choosing the right representations, and/or
 - Choosing the right distance measures