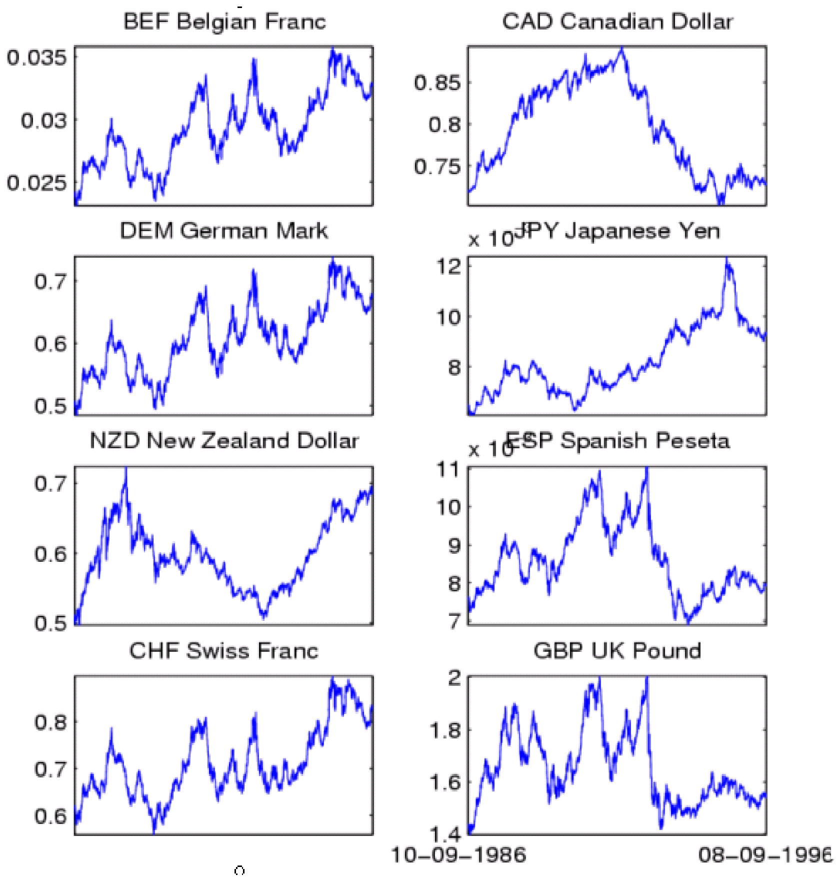
DATA MINING LECTURE 8B

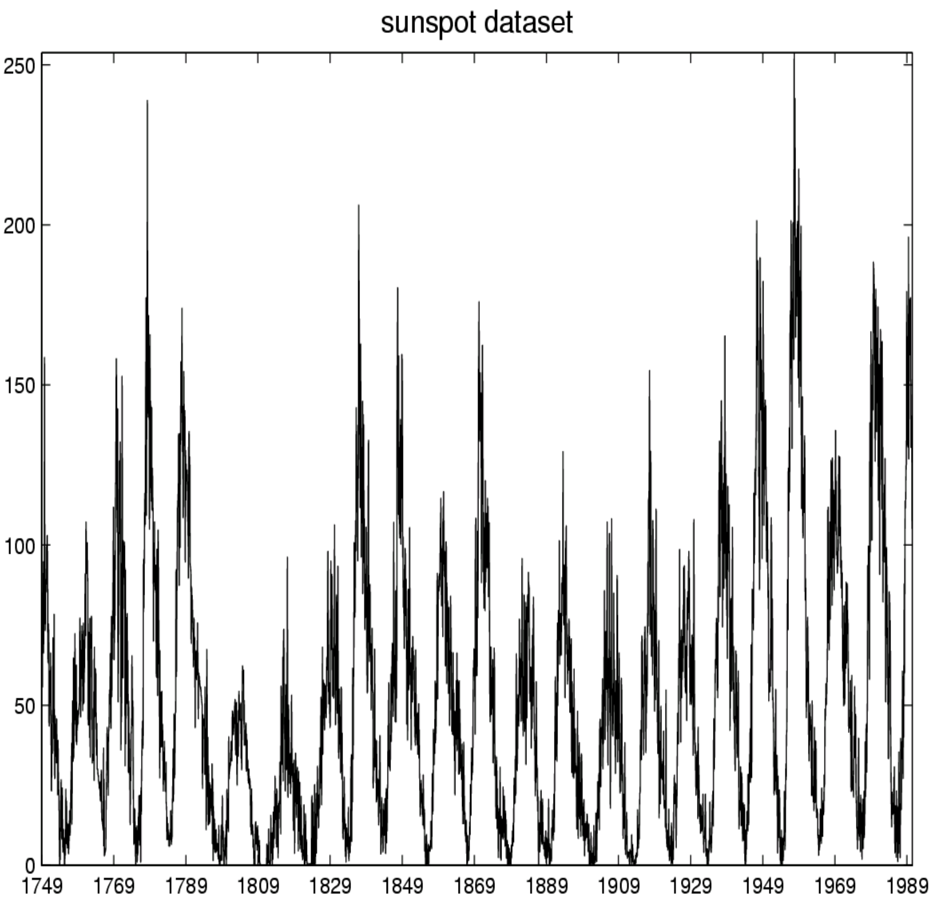
Time series analysis and Sequence Segmentation

Sequential data

- Sequential data (or time series) refers to data that appear in a specific order.
  - The order defines a time axis, that differentiates this data from other cases we have seen so far
- Examples
  - The price of a stock (or of many stocks) over time
  - Environmental data (pressure, temperature, precipitation etc) over time
  - The sequence of queries in a search engine, or the frequency of a query over time
  - The words in a document as they appear in order
  - A DNA sequence of nucleotides
  - Event occurrences in a log over time
  - Etc…
- Time series: usually we assume that we have a vector of numeric values that change over time.

Time-series data





- Financial time series, process monitoring…

Why deal with sequential data?

- Because all data is sequential ☺
  - All data items arrive in the data store in some order
- In some (many) cases the order does not matter
  - E.g., we can assume a bag of words model for a document
- In many cases the order is of interest
  - E.g., stock prices do not make sense without the time information.

Time series analysis

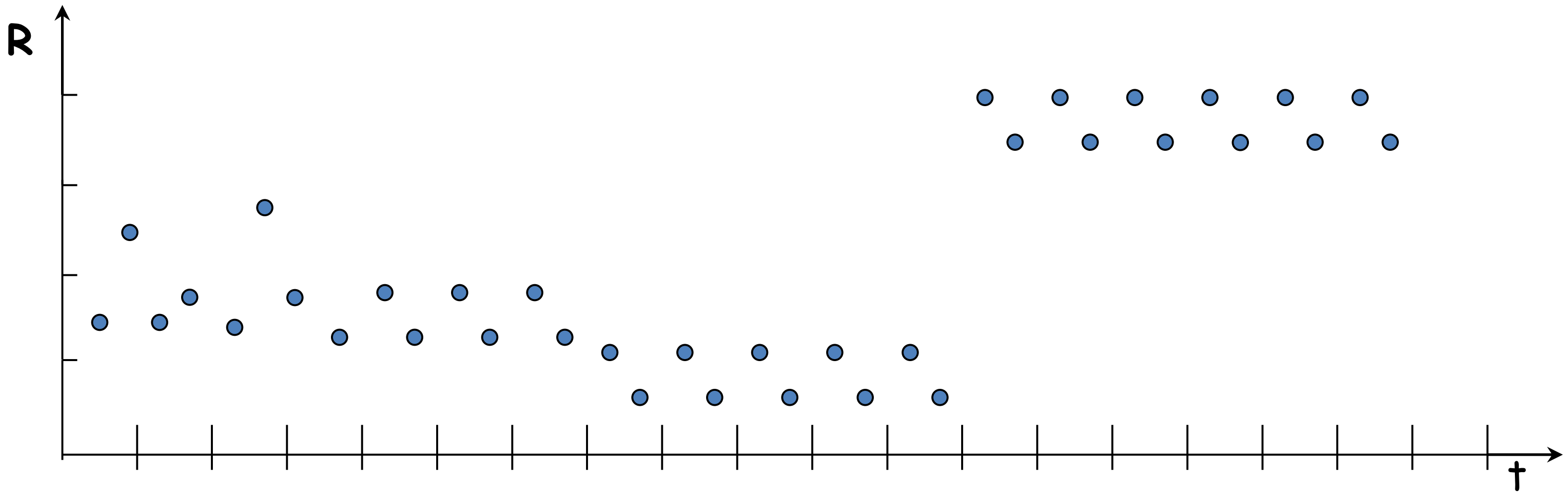
- The addition of the time axis defines new sets of problems
  - Discovering periodic patterns in time series
  - Defining similarity between time series
  - Finding bursts, or outliers

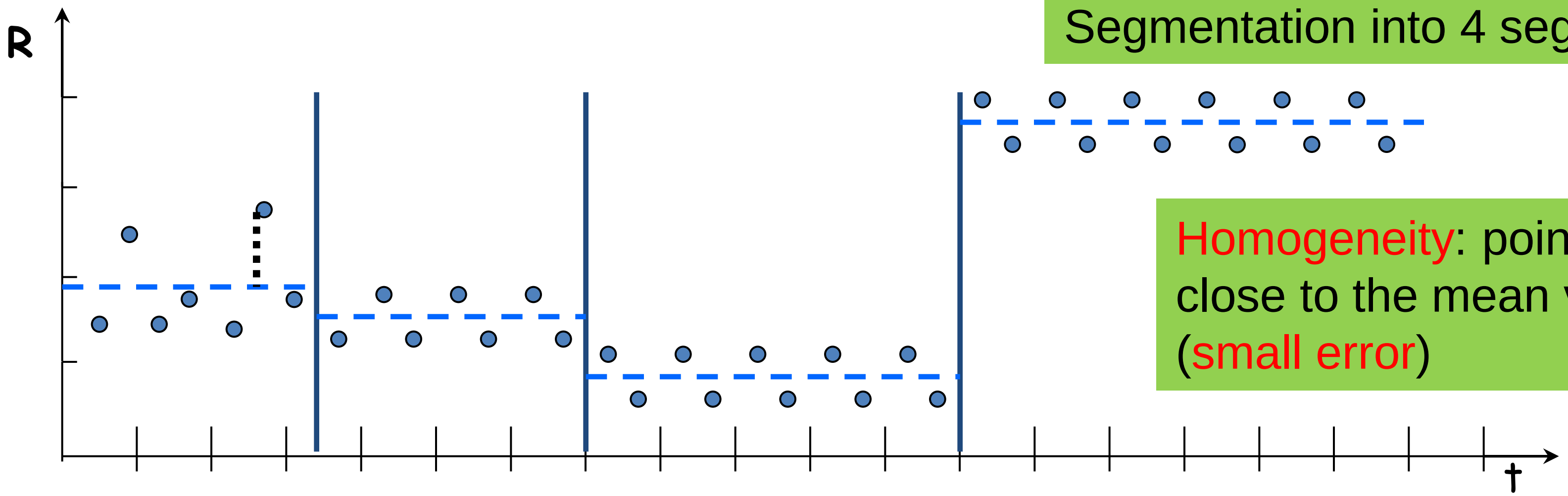
- Also, some existing problems need to be revisited taking sequential order into account
  - Association rules and Frequent Itemsets in sequential data
  - Summarization and Clustering: Sequence Segmentation

Sequence Segmentation

- Goal: discover structure in the sequence and provide a concise summary
- Given a sequence T, segment it into K contiguous segments that are as homogeneous as possible
- Similar to clustering but now we require the points in the cluster to be contiguous
- Commonly used for summarization of histograms in databases

Example





Segmentation into 4 segments

Homogeneity: points are close to the mean value (small error)

Basic definitions

- Sequence $T = \{t_1, t_2, \ldots, t_N\}$: an ordered set of N d-dimensional real points $t_i \in R^d$

- A K-segmentation S: a partition of T into K contiguous segments $\{s_1, s_2, \ldots, s_K\}$.
  - Each segment $s \in S$ is represented by a single vector $\mu_s \in R^d$ (the representative of the segment -- same as the centroid of a cluster)

- Error $E(S)$: The error of replacing individual points with representatives
  - Different error functions, define different representatives.

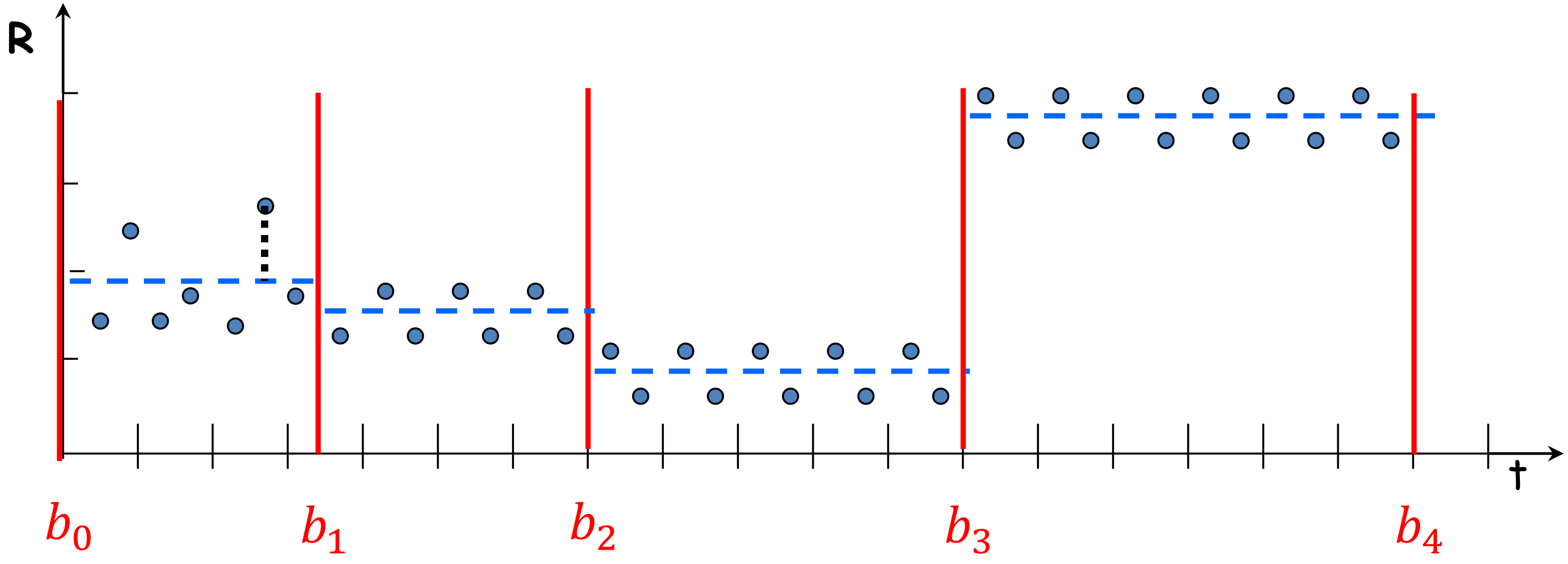
- Sum of Squares Error (SSE):

$$E(S) = \sum_{s \in S} \sum_{t \in s} (t - \mu_s)^2$$

- Representative of segment s with SSE: mean $\mu_s = \frac{1}{|s|} \sum_{t \in s} t$

Basic Definitions

- Observation: a K-segmentation S is defined by K+1 boundary points $b_0, b_1, \dots, b_{K-1}, b_K$.



- $b_0 = 0, b_k = N + 1$ always.
  - We only need to specify $b_1, \dots, b_{K-1}$

The K-segmentation problem

Given a sequence T of length N and a value K, find a K-segmentation $S = \{s_1, s_2, \ldots, s_K\}$ of T such that the SSE error E is minimized.

- Similar to K-means clustering, but now we need the points in the clusters to respect the order of the sequence.
  - This actually makes the problem easier.

Optimal solution for the k-segmentation problem

- Bellman'61: The K-segmentation problem can be solved optimally using a standard dynamic-programming algorithm

- Dynamic Programming:
  - Construct the solution of the problem by using solutions to problems of smaller size
    - Define the dynamic programming recursion
  - Build the solution bottom up from smaller to larger instances
    - Define the dynamic programming table that stores the solutions to the sub-problems

Rule of thumb

- Most optimization problems where order is involved can be solved optimally in polynomial time using dynamic programming.
  - The polynomial exponent may be large though

Dynamic Programming Recursion

- Terminology:
  - $T[1,n]$: subsequence $\{t_1, t_2, \ldots, t_n\}$ for $n \leq N$
  - $E(S[1,n],k)$: error of optimal segmentation of subsequence $T[1,n]$ with k segments for $k \leq K$

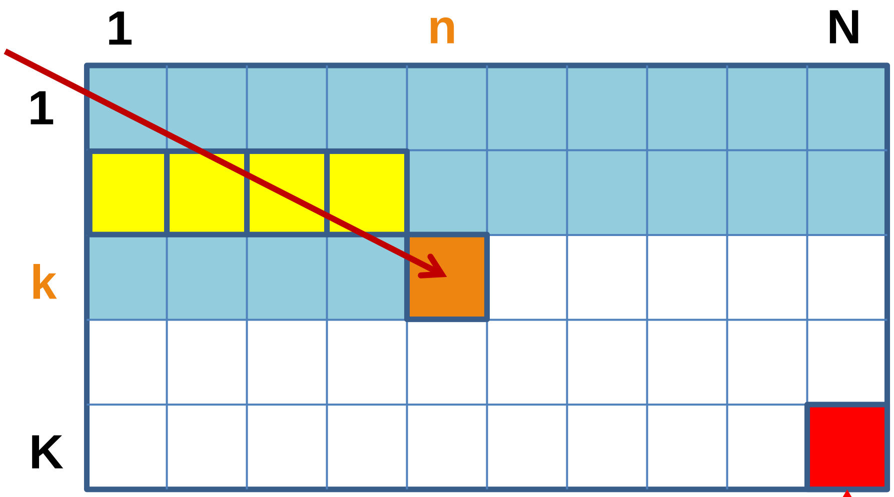
- Dynamic Programming Recursion:

$$E(S[1,n],k) = \underbrace{\min_{k \leq j \leq n-1}}_{\text{Minimum over all possible placements of the last boundary point } b_{k-1}} \left\{ \underbrace{E(S[1,j],k-1)}_{\text{Error of optimal segmentation S[1,j] with k-1 segments}} + \underbrace{\sum_{j+1 \leq t \leq n} \left(t - \mu_{[j+1,n]}\right)^2}_{\text{Error of k-th (last) segment when the last segment is [j+1,n]}} \right\}$$

Dynamic programming table

- Two−dimensional table $A[1 \ldots K, 1 \ldots N]$

$$A[k,n] = E(S[1,n],k)$$

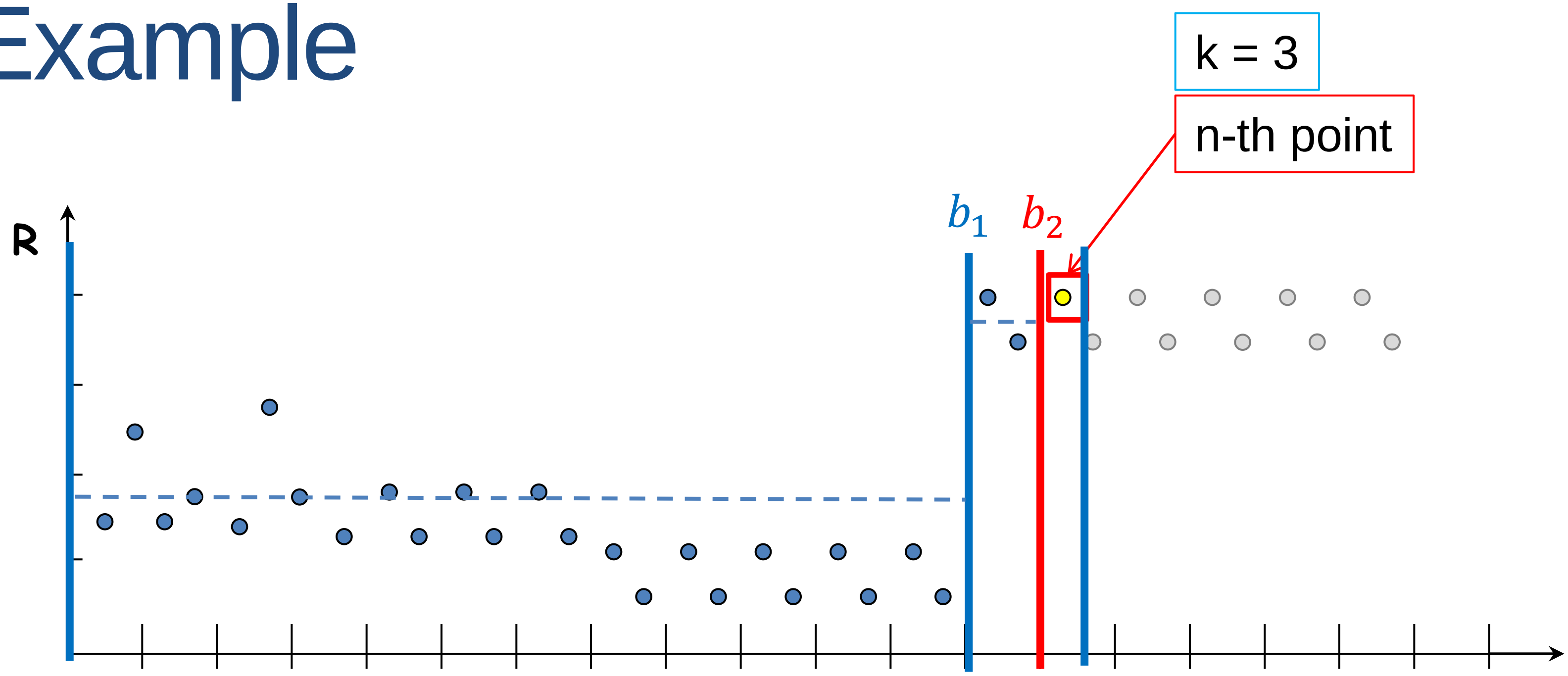


$$E(S[1,n],k) = \min_{k \leq j \leq n-1} \left\{ E(S[1,j],k-1) + \sum_{j+1 \leq t \leq n} \left(t - \mu_{[j+1,n]}\right)^2 \right\}$$

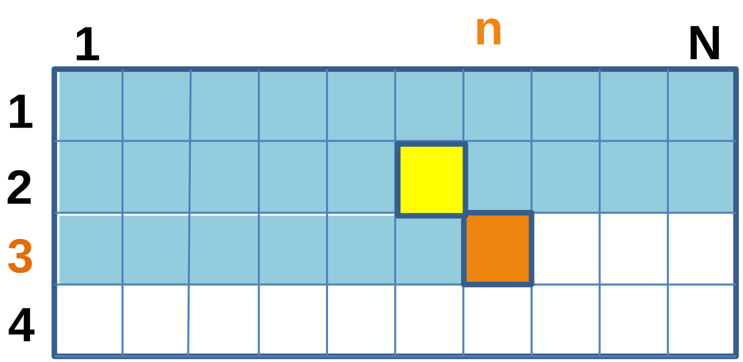
- Fill the table top to bottom, left to right.

Error of optimal K-segmentation

Example

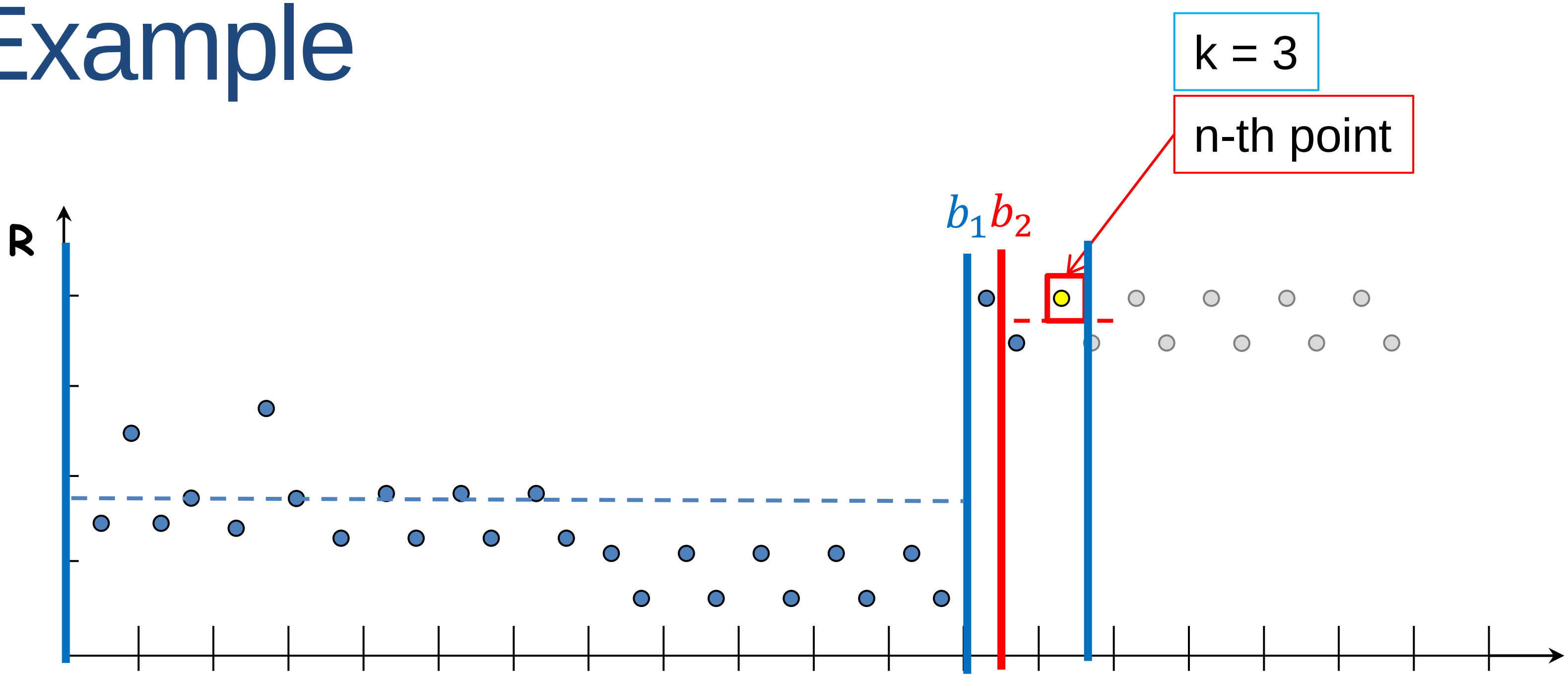


$$E(S[1,n],k) = \min_{k\le j\le n-1}\left\{E(S[1,j],k-1) + \sum_{j+1\le t\le n}\left(t-\mu_{[j+1,n]}\right)^2\right\}$$

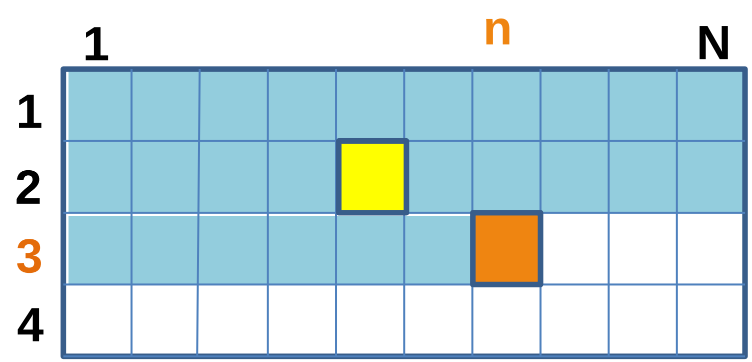


Where should we place boundary b_2 ?

Example

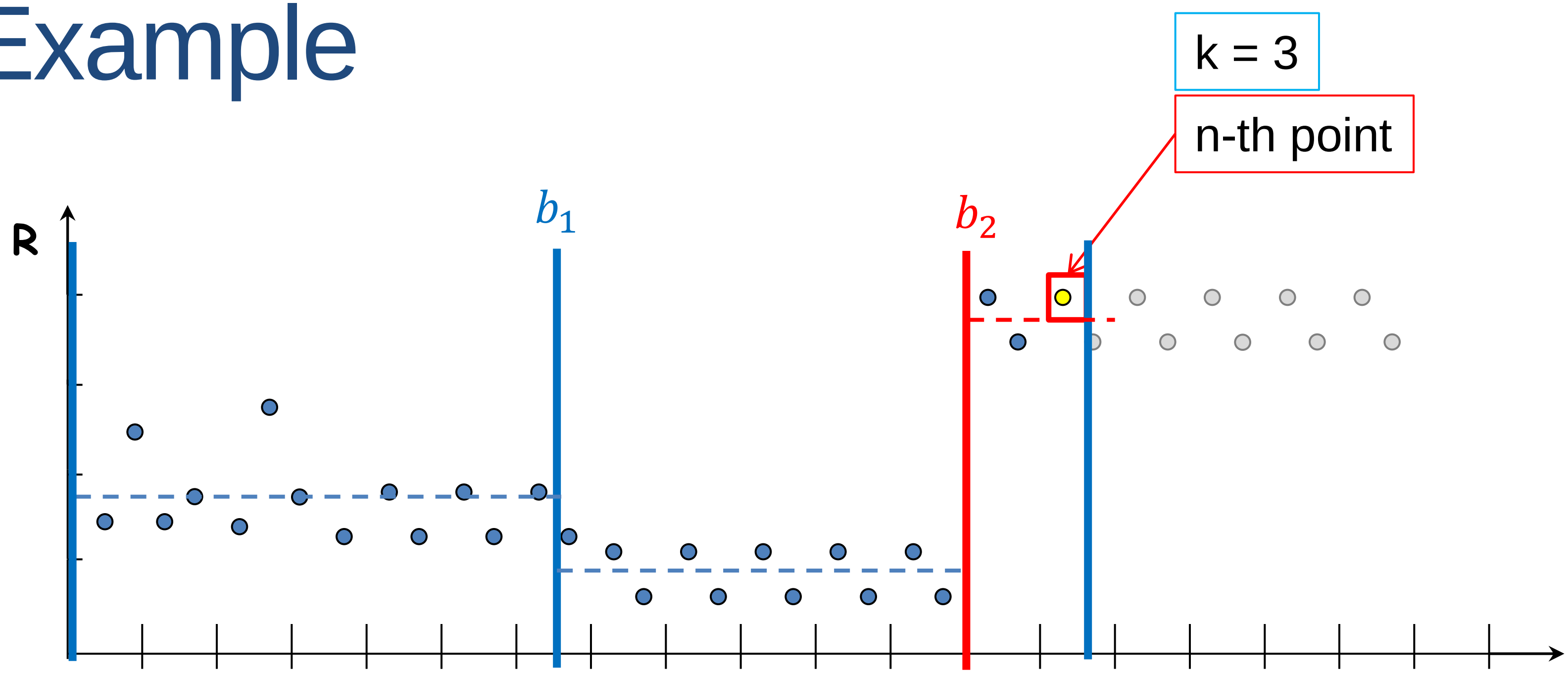


$$E(S[1,n],k) = \min_{k \le j \le n-1} \left\{ E(S[1,j],k-1) + \sum_{j+1 \le t \le n} \left(t - \mu_{[j+1,n]}\right)^2 \right\}$$

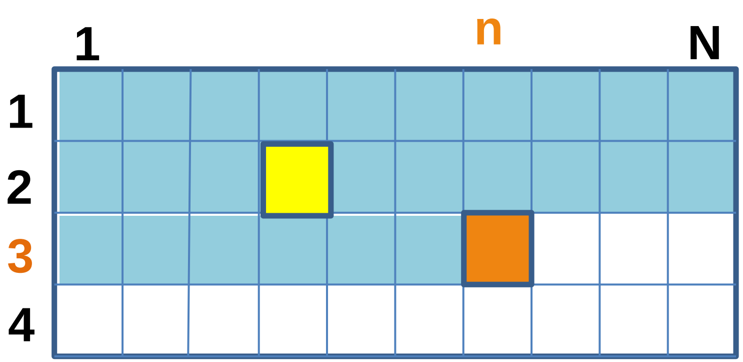


Where should we place boundary b_2 ?

Example

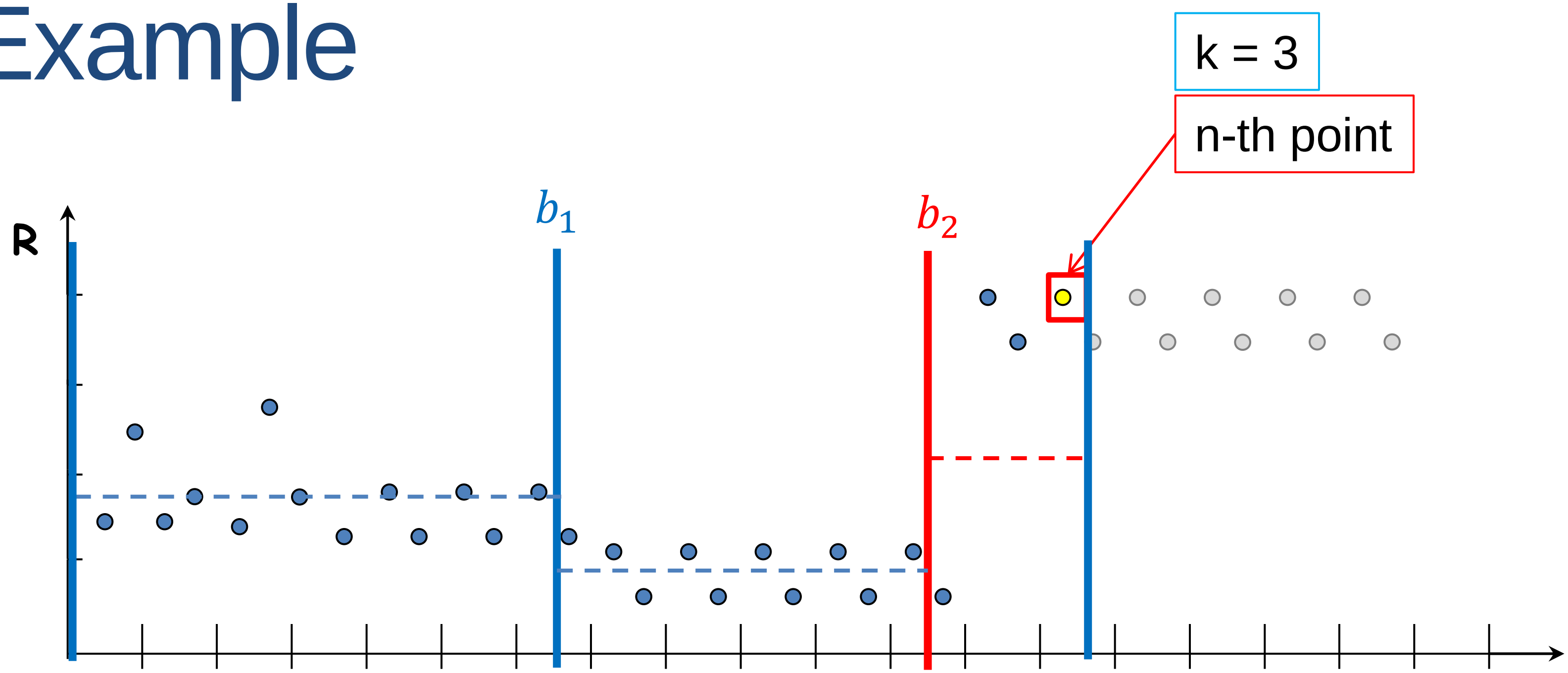


$$E(S[1,n],k) = \min_{k\le j\le n-1}\left\{E(S[1,j],k-1) + \sum_{j+1\le t\le n}\left(t-\mu_{[j+1,n]}\right)^2\right\}$$

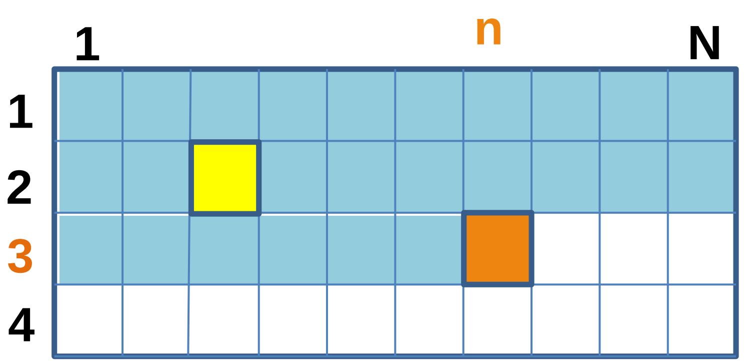


Where should we place boundary b_2 ?

Example

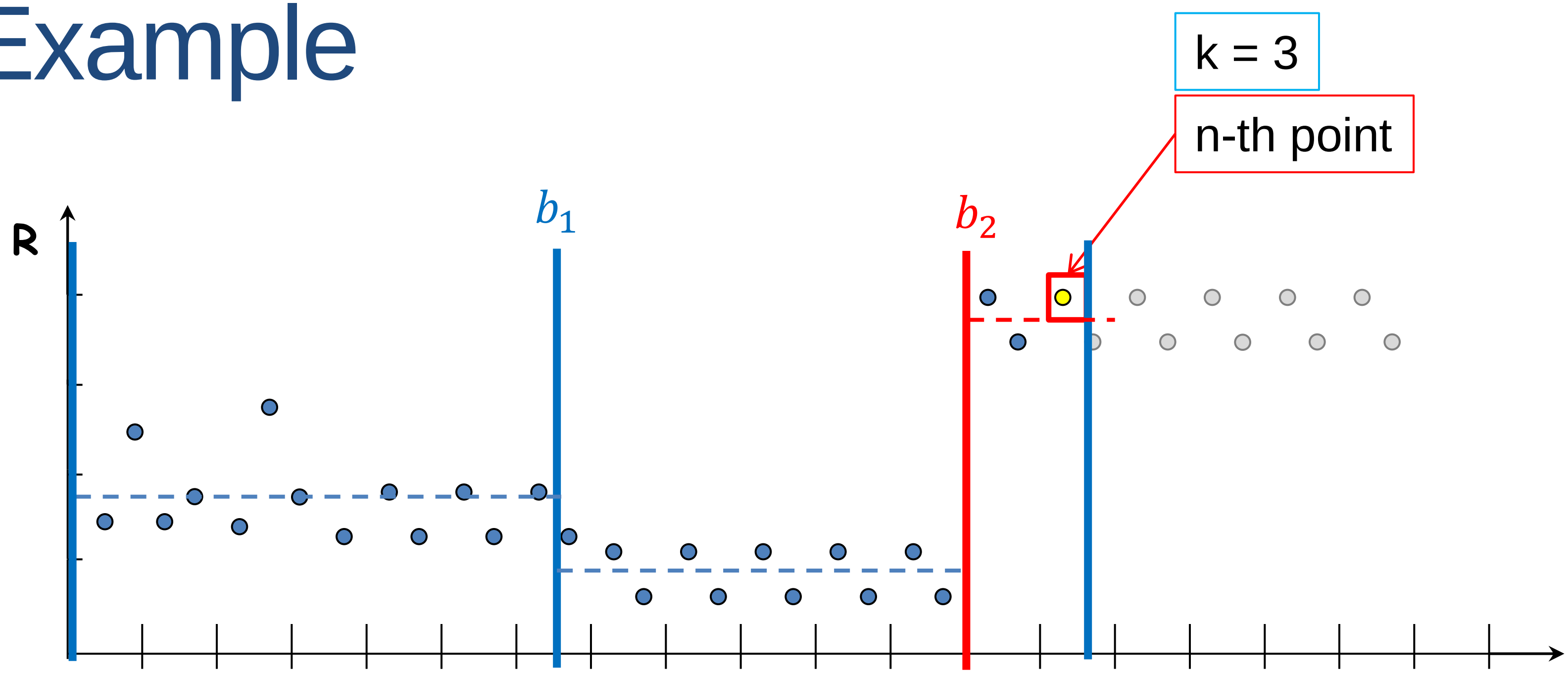


$$E(S[1,n],k) = \min_{k\le j\le n-1}\left\{E(S[1,j],k-1) + \sum_{j+1\le t\le n}\left(t-\mu_{[j+1,n]}\right)^2\right\}$$



Where should we place boundary b_2 ?

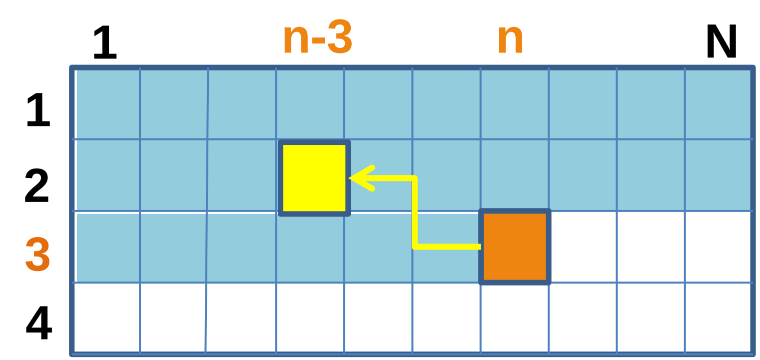
Example



Optimal segmentation S[1:n]

The cell A[3,n] stores the error of the optimal solution 3-segmentation of T[1,n]

In the cell (or in a different table) we also store the position n-3 of the boundary so we can trace back the segmentation



Dynamic-programming algorithm

- Input: Sequence **T**, length **N**, **K** segments, error function **E()**

- For **i=1** to **N** //Initialize first row
  - **A[1,i]=E(T[1…i])** //Error when everything is in one cluster

- For **k=1** to **K** // Initialize diagonal
  - **A[k,k] = 0** // Error when each point in its own cluster

- For **k=2** to **K**
  - For **i=k+1** to **N**
    - $A[k,i] = \min_{j<i}\{A[k-1,j]+E(T[j+1\ldots i])\}$

- To recover the actual segmentation (not just the optimal cost) store also the minimizing values **j**

Algorithm Complexity

- What is the complexity?
- NK cells to fill
- Computation per cell
  $E(S[1,n],k) = \min_{k \le j < n} \left\{ E(S[1,j],k-1) + \sum_{j+1 \le t \le n} \left(t - \mu_{[j+1,n]}\right)^2 \right\}$
  - O(N) boundaries to check per cell
    - O(N) to compute the second term per checked boundary
- $O(N^3K)$ in the naïve computation

- We can avoid the last O(N) factor by observing that

$$\sum_{j+1 \le t \le n} \left(t - \mu_{[j+1,n]}\right)^2 = \sum_{j+1 \le t \le n} t^2 - \frac{1}{n-j} \left(\sum_{j+1 \le t \le n} t \right)^2$$

- We can compute in constant time by precomputing partial sums
  - Precompute $\sum_{1 \le t \le n} t$ and $\sum_{1 \le t \le n} t^2$ for all n = 1..N

- Algorithm Complexity: $O(N^2K)$

Heuristics

- Top-down greedy (TD): O(NK)
  - Introduce boundaries one at the time so that you get the largest decrease in error, until K segments are created.
- Bottom-up greedy (BU): O(NlogN)
  - Merge adjacent points each time selecting the two points that cause the smallest increase in the error until K segments
- Local Search Heuristics: O(NKI)
  - Assign the breakpoints randomly and then move them so that you reduce the error

Other time series analysis

- Using signal processing techniques is common for defining similarity between series
  - Fast Fourier Transform
  - Wavelets

- Rich literature in the field