Current Trends

A Hands-on Introduction to Hierarchical Pattern-Based Sequence Learning

Patrick Van der Spiegel

Outline

1. Sequence learning
2. Time series anomaly detection
3. Hierarchical piecewise approximation

QUESTION

- Did everyone manage to install *pbsf-lib*?

- **If not:**
  - Install *pbsf-lib* from Github, or
  - Pair up with someone who has it installed

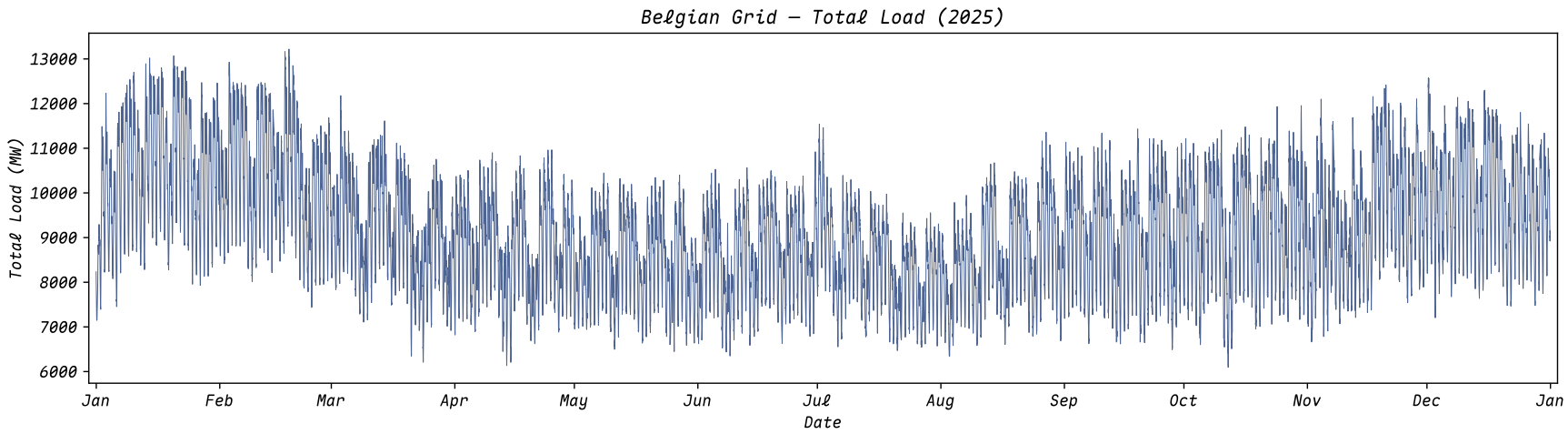
Sequence data

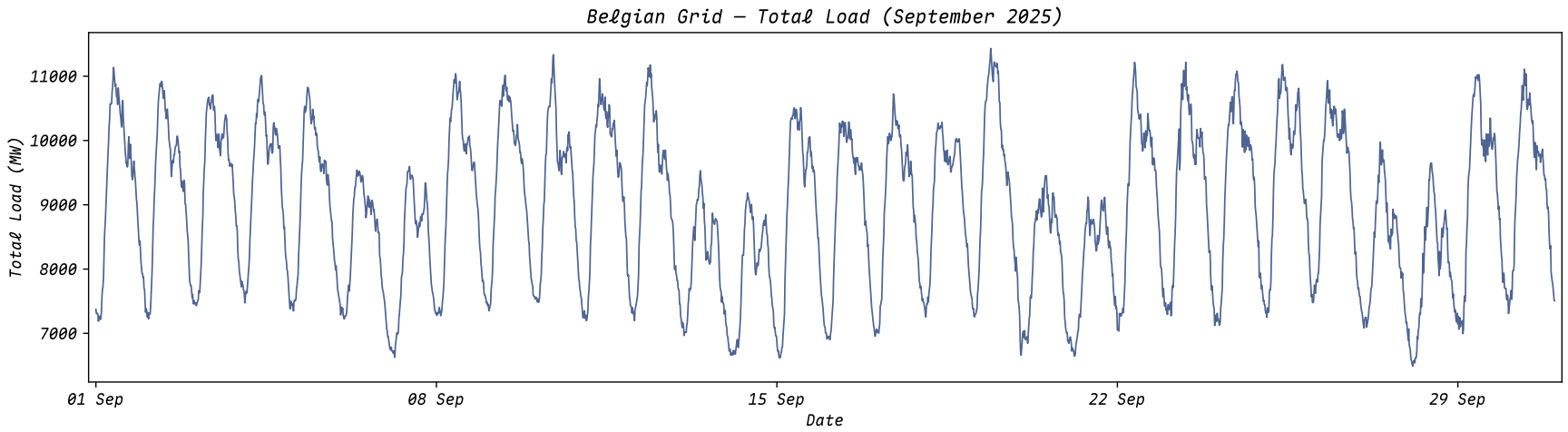
- Ordered sets of elements, may be continuous or discrete.
- Observing processes or systems over time through **sensors** and **event logs**.

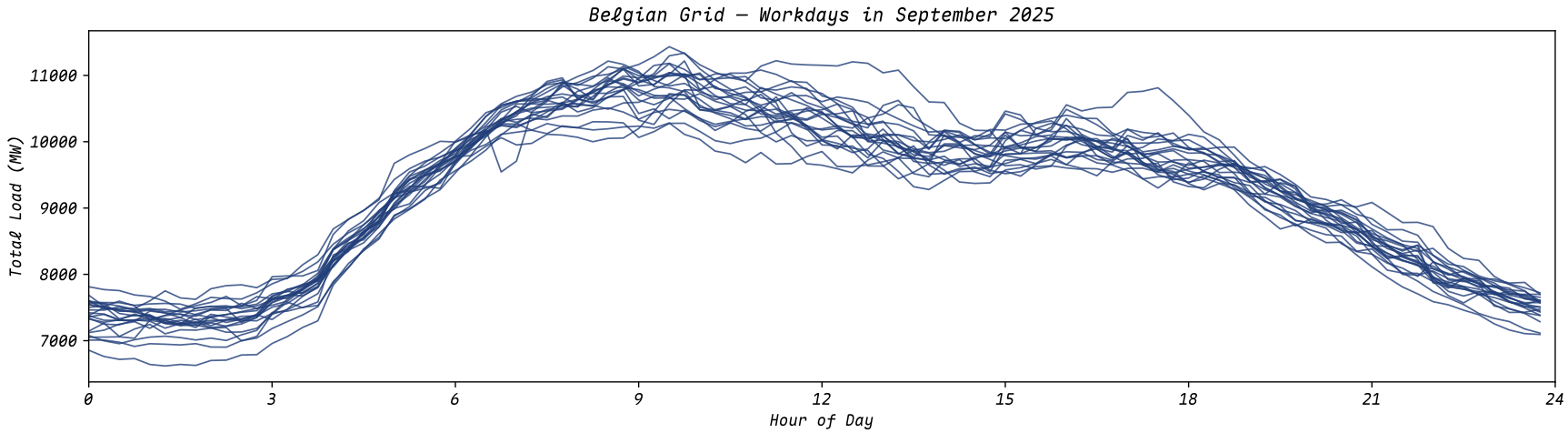
Time series

- Ordered sequence $S = (\boldsymbol{s}_{t_1}, \ldots, \boldsymbol{s}_{t_n})$ of observations $\boldsymbol{s}_{t_i} \in \mathbb{R}^p$.
- Each data point observed at a specific time point.
- Univariate ($p = 1$) vs multivariate ($p > 1$).

Example


Belgian Grid — Total Load (2025)
Total Load (MW)
13000
12000
11000
10000
9000
8000
7000
6000
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
Jan
Date


Belgian Grid — Total Load (September 2025)
Total Load (MW)
11000
10000
9000
8000
7000
01 Sep
08 Sep
15 Sep
22 Sep
29 Sep
Date

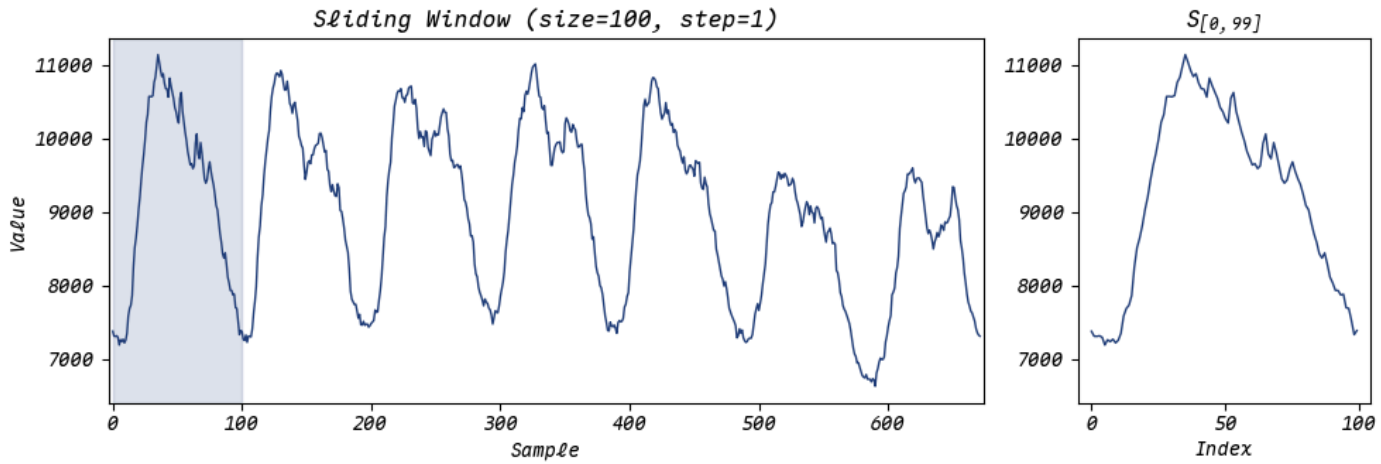

Belgian Grid — Workdays in September 2025
Total Load (MW)
11000
10000
9000
8000
7000
0
3
6
9
12
15
18
21
24
Hour of Day

Contiguous subsequences

- Often not interested in properties of complete S, but in local parts $S_{[i,j]}$.
- Here, $S_{[i,j]}$ is a **contiguous subsequence** from index i up until j.

Sliding windows

Given a sequence S of length n, a **sliding window** of length k extracts all subsequences $S_{[i,i+k-1]}$ with $1 \leq i \leq n - k$.

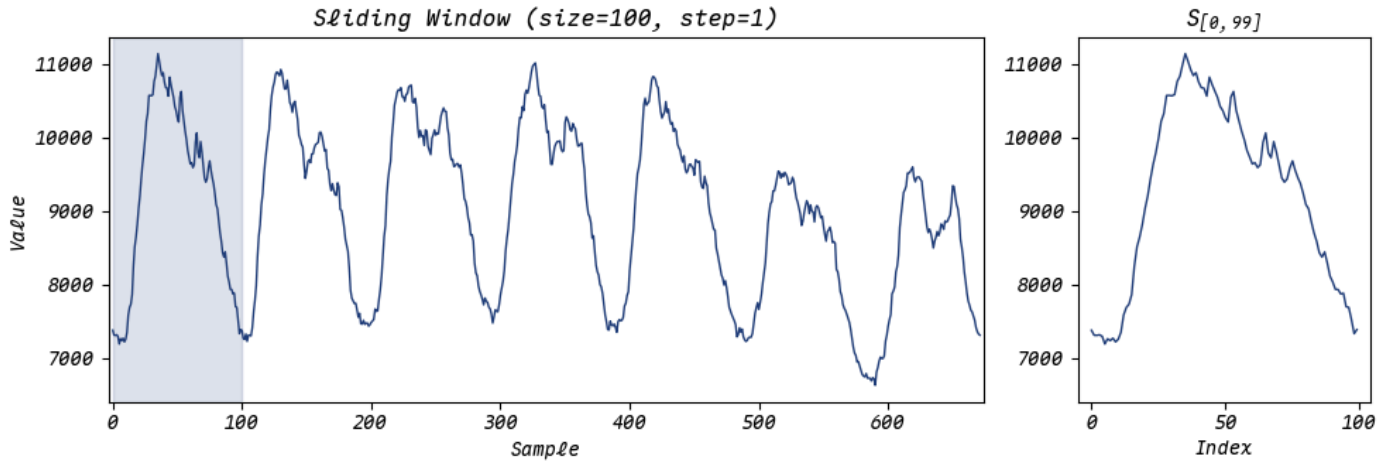


EXERCISE

- Use `pbsf-lib` to create a `SlidingWindow` with fixed window size.
- Extract and plot segments of a NumPy array.

```python
import numpy as np
from pbsf.segmenters import SlidingWindow

# Basic sliding window with fixed size
segmenter = SlidingWindow({'window_size': 100, 'step_size': 1})
segments = segmenter.segment(data)
```

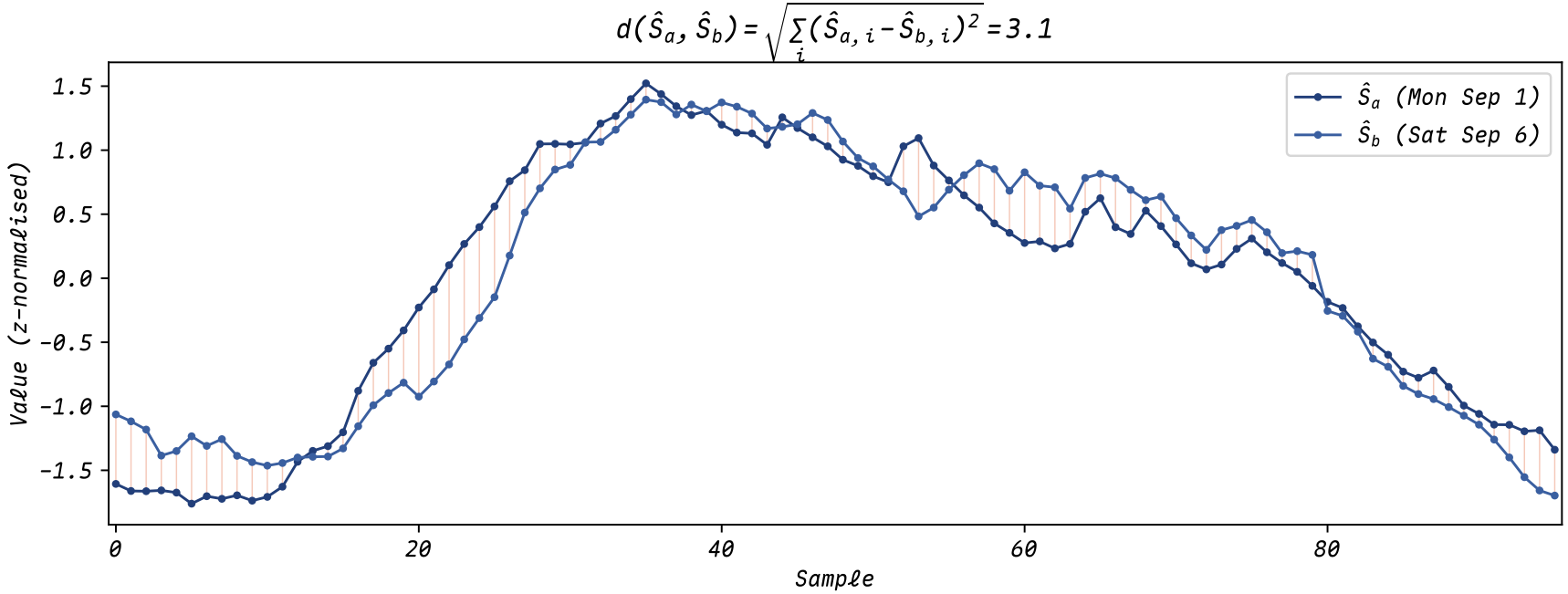


Distances

- Determining **sequence similarity** is crucial for sequence learning.
- Given sequences P and Q, distance $dist(P, Q)$ is a **dissimilarity measure**.
  - **Non-negative:** $dist(P, Q) \geq 0$
  - **Symmetric:** $dist(P, Q) = dist(Q, P)$
- Many possible distances, each with their own specific properties.

Euclidean Distance

Given z-normalised sequences P and Q, the **z-normalised Euclidean distance** is: $\sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$

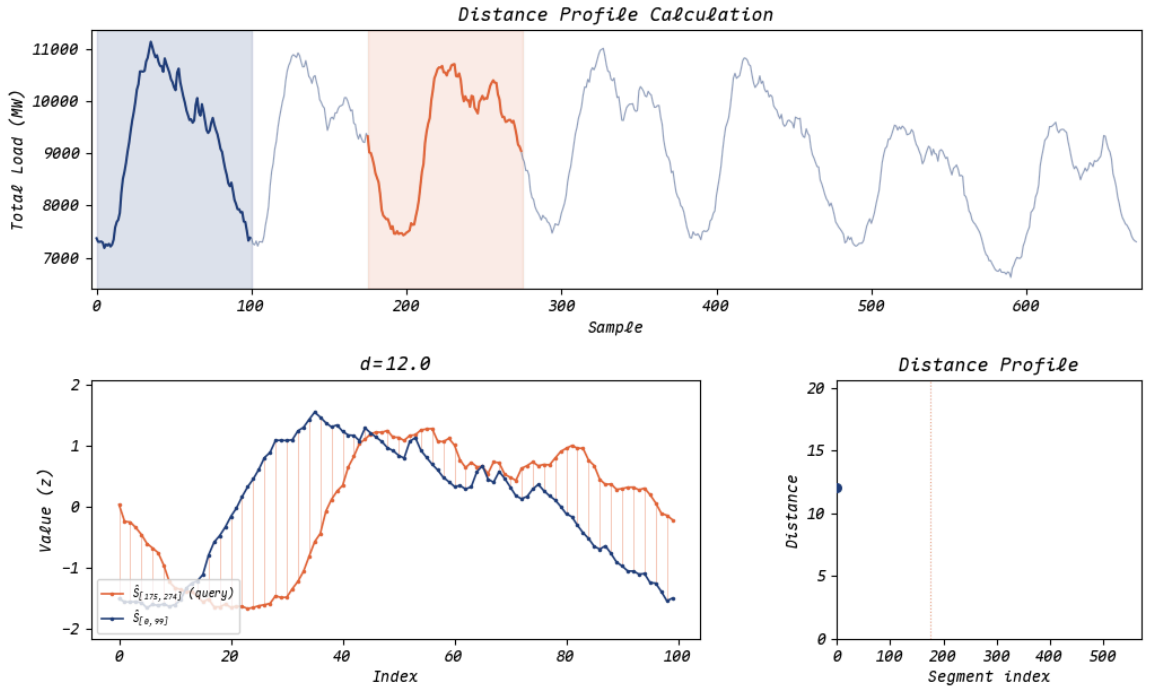


EXERCISE

- Calculate the Euclidean distance between the extracted segments of your `SlidingWindow`.

Distance profile

Given a **query segment**, calculate the z-normalised Euclidean distance between the query and each subsequence of the time series.



EXERCISE

Use the **brute-force algorithm** to calculate the distance profile for a query and your extracted `SlidingWindow` segments.

QUESTION

What kind of results do you get when you change parameters of your `SlidingWindow`?

Motifs and discords

Motifs

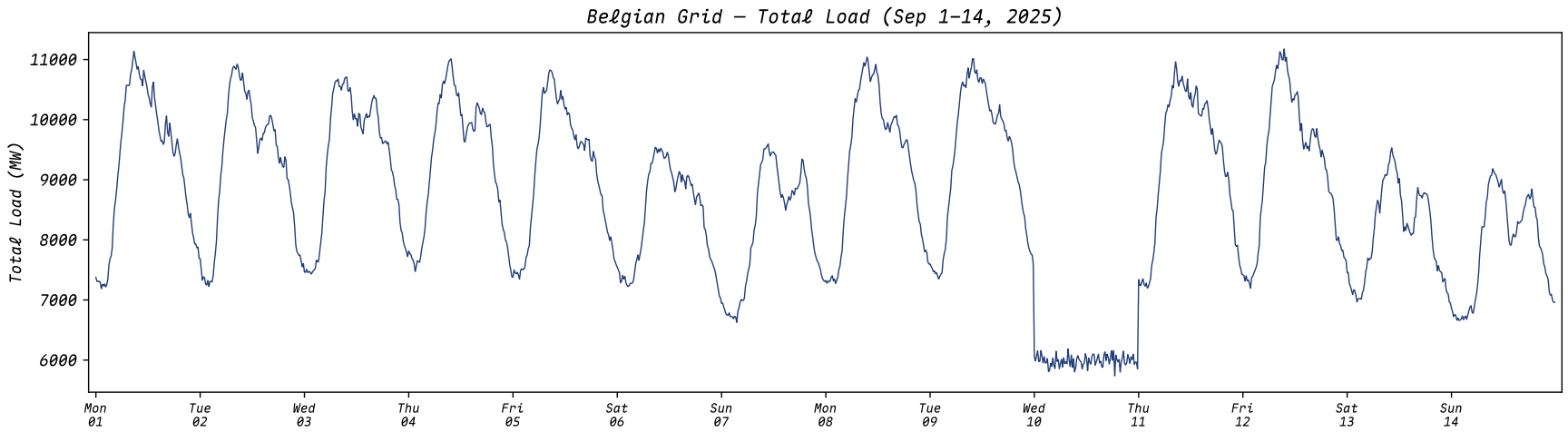
- Recurring subsequences with a **low pairwise distance** to each other.
  - Found within single sequence or across a larger set of sequences.

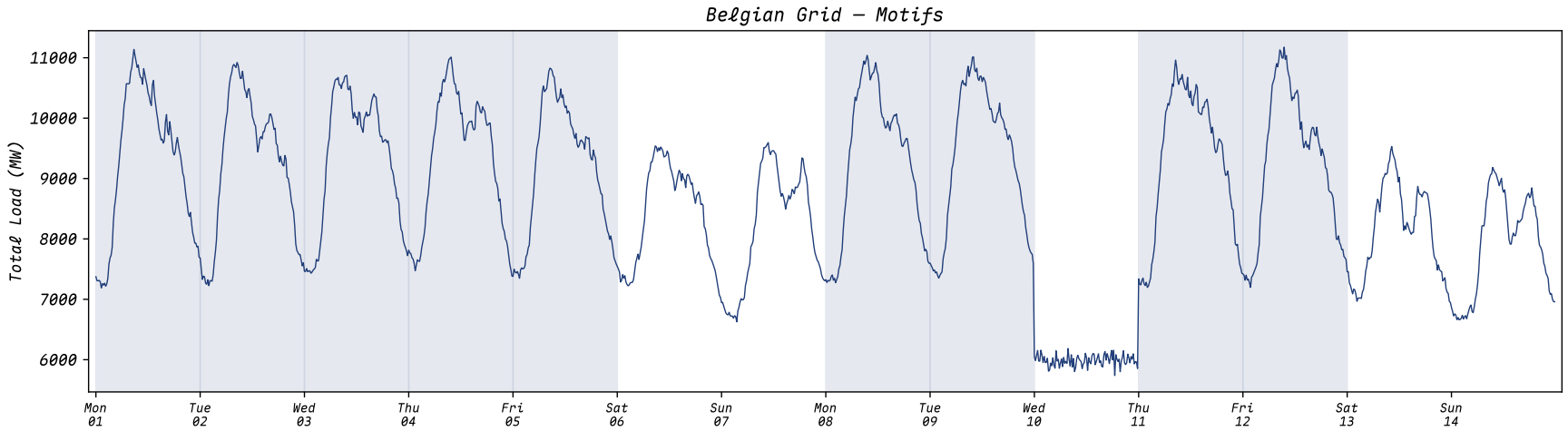
Discord

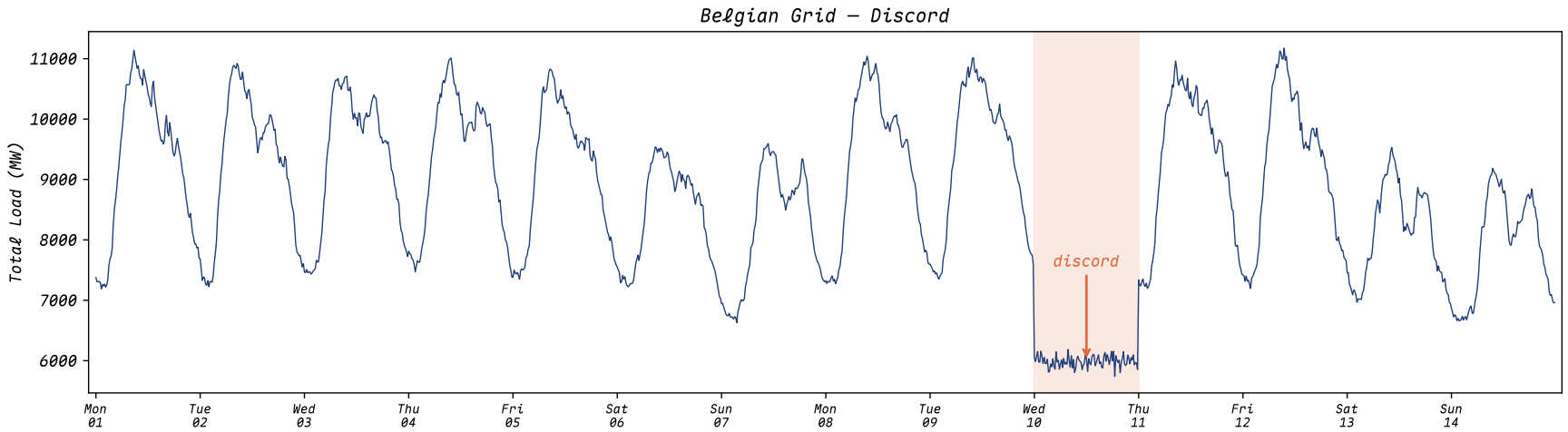
- Subsequence that is **maximally different** from the other subsequences.

EXERCISE

- Point out **motifs** and the **discord**.


Belgian Grid — Total Load (Sep 1–14, 2025)
Total Load (MW)
11000
10000
9000
8000
7000
6000
Mon 01
Tue 02
Wed 03
Thu 04
Fri 05
Sat 06
Sun 07
Mon 08
Tue 09
Wed 10
Thu 11
Fri 12
Sat 13
Sun 14


Belgian Grid — Motifs
Total Load (MW)
11000
10000
9000
8000
7000
6000
Mon 01
Tue 02
Wed 03
Thu 04
Fri 05
Sat 06
Sun 07
Mon 08
Tue 09
Wed 10
Thu 11
Fri 12
Sat 13
Sun 14


Belgian Grid — Discord
Total Load (MW)
11000
10000
9000
8000
7000
6000
discord
Mon 01
Tue 02
Wed 03
Thu 04
Fri 05
Sat 06
Sun 07
Mon 08
Tue 09
Wed 10
Thu 11
Fri 12
Sat 13
Sun 14

Matrix profile

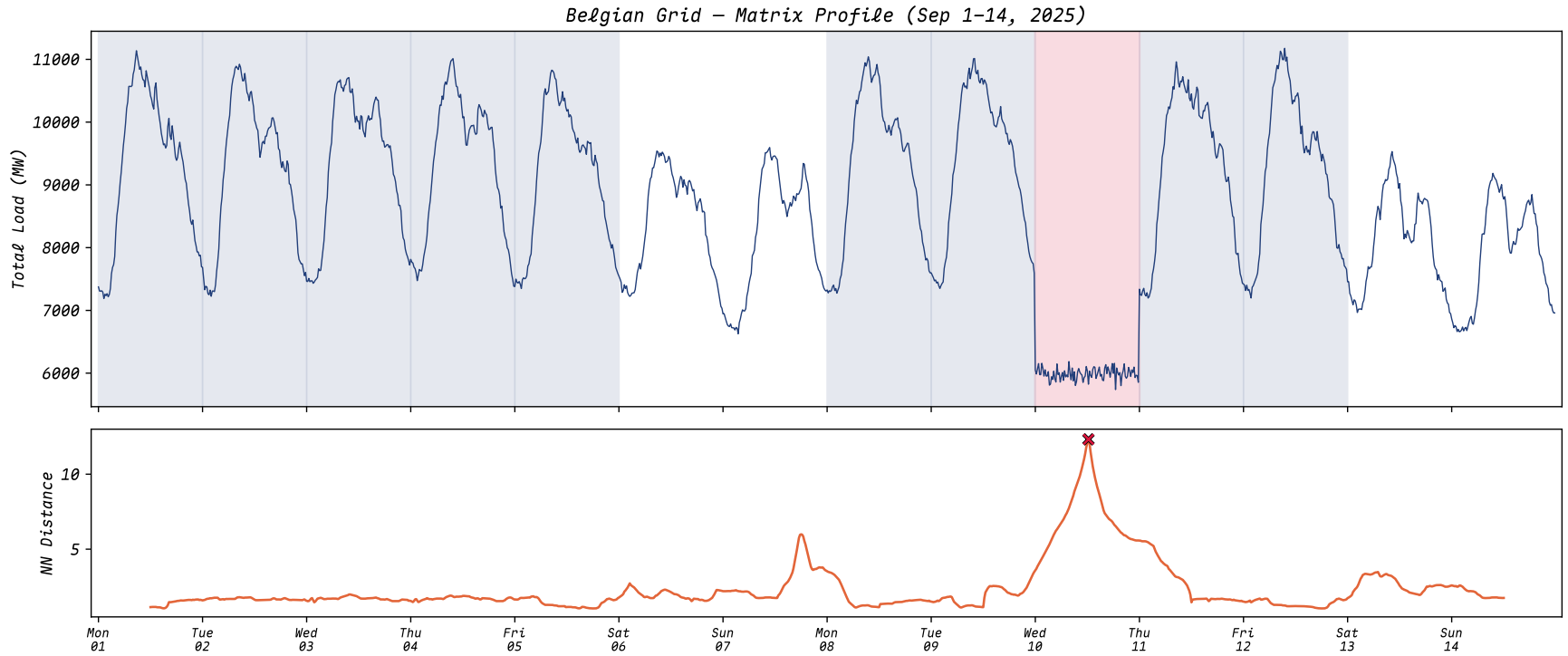
- How can we find **motifs** and **discords**?
  - Distance profile of every segment.
  - Keep track of distance to nearest neighbour.
- **Motifs:** low-distance group of segments.
- **Discord:** segment with highest distance to nearest neighbour.

EXERCISE

Pick a random segment in your time series and replace it with noise.

QUESTION

What does the matrix profile look like?

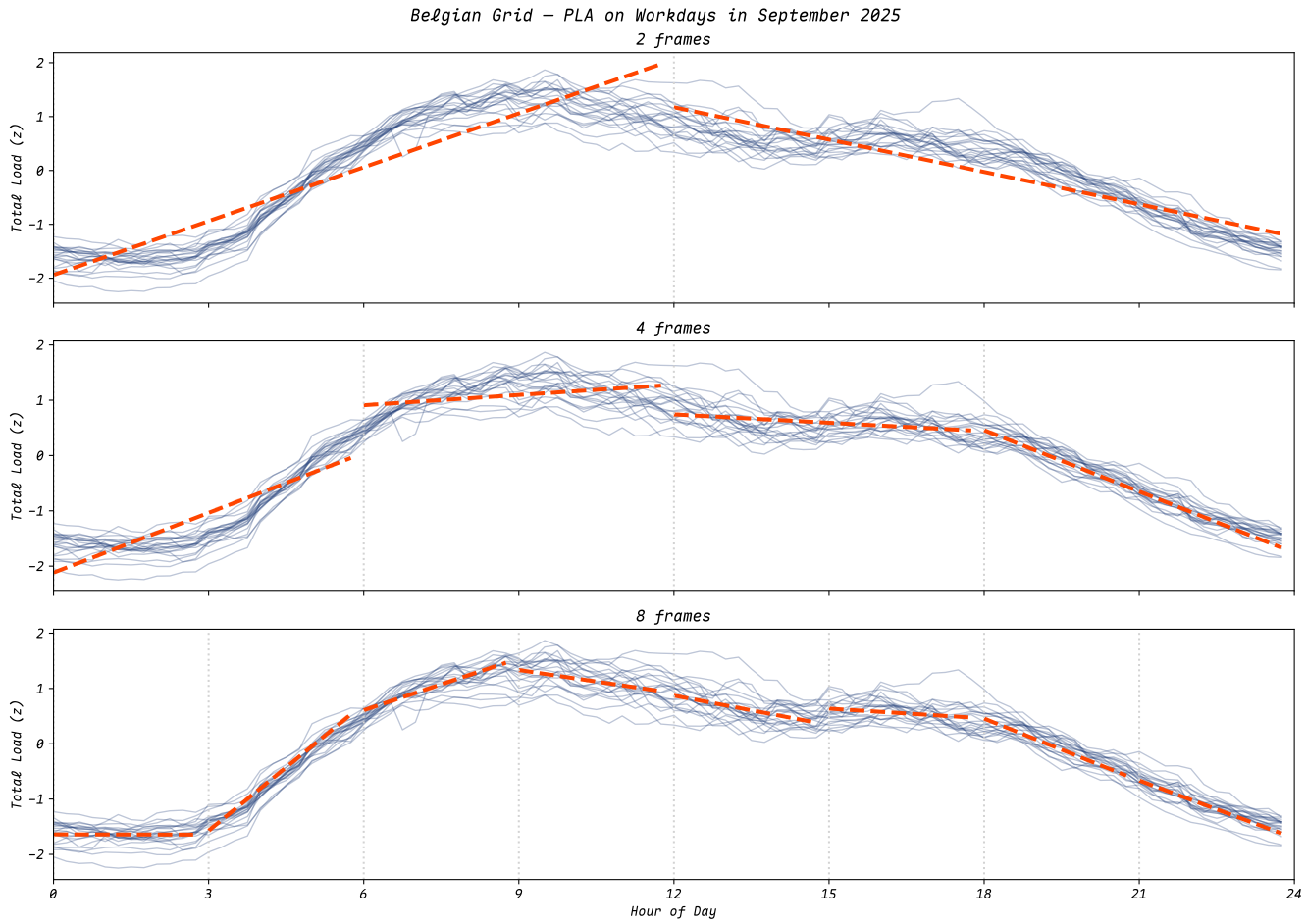

Belgian Grid — Matrix Profile (Sep 1–14, 2025)
Total Load (MW)
11000
10000
9000
8000
7000
6000
NN Distance
10
5
Mon 01
Tue 02
Wed 03
Thu 04
Fri 05
Sat 06
Sun 07
Mon 08
Tue 09
Wed 10
Thu 11
Fri 12
Sat 13
Sun 14

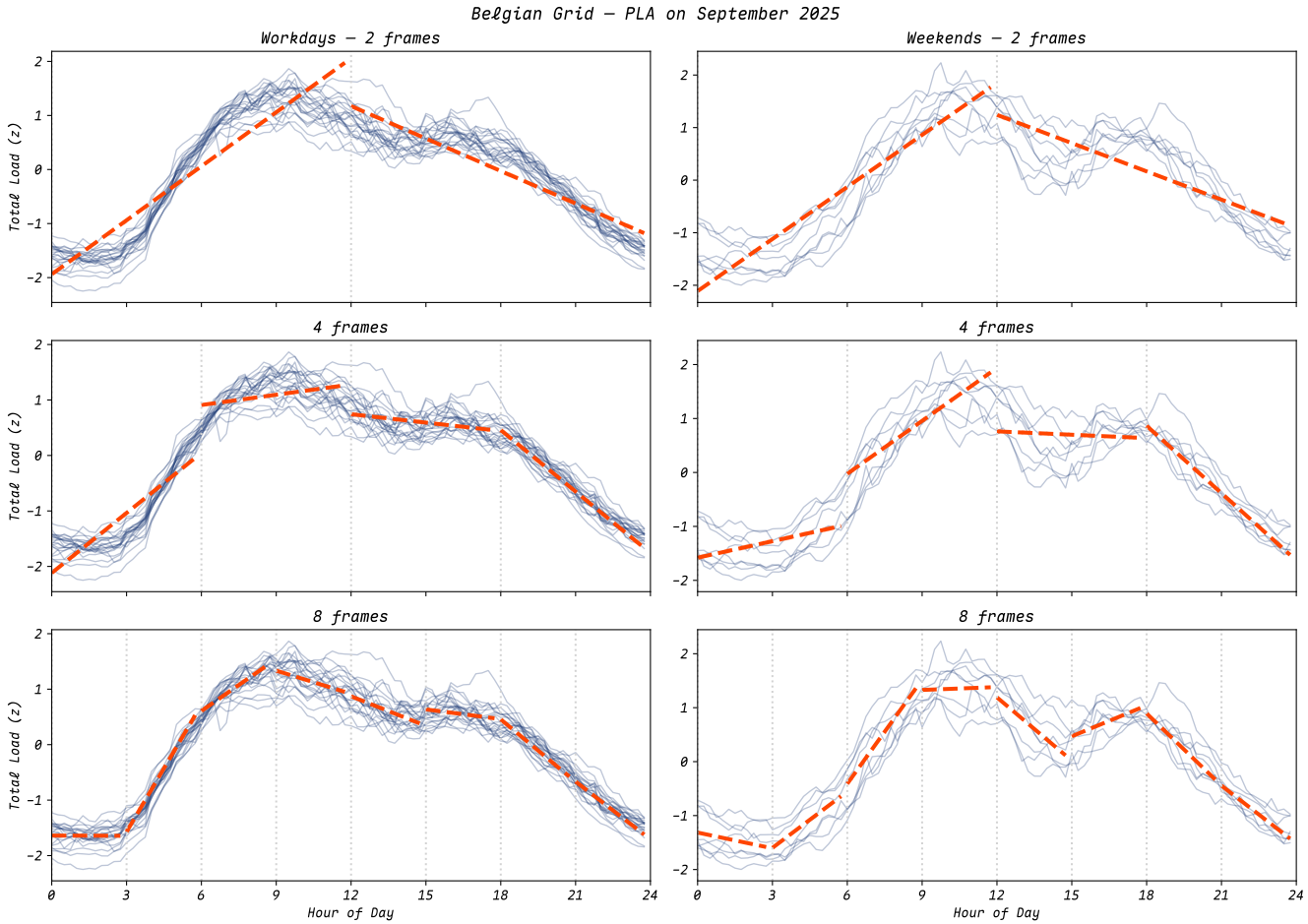
- Distance-based approaches to AD are competitive with state-of-the-art, depending on what your **definition** of an anomaly is!
- Bruteforce calculation of all these pairwise distances is not manageable for large time series, however.

Piecewise Approximations

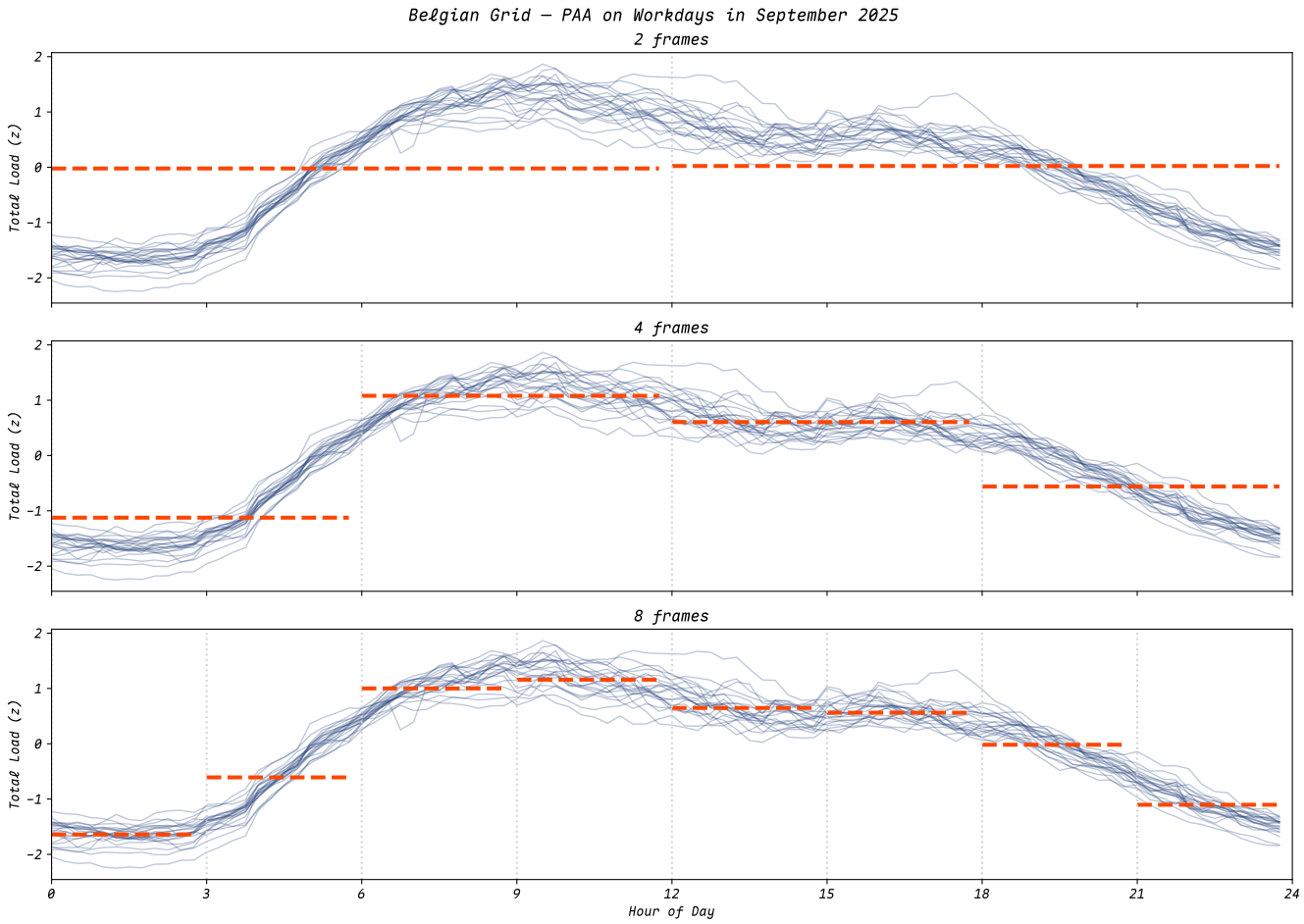
- Creating a sequence representation with strongly reduced dimensionality.
- Interested in set of techniques that:
  - Partition sequence into non-overlapping *frames*,
  - Approximates each frame through a small number of values.

Piecewise Linear Approximation

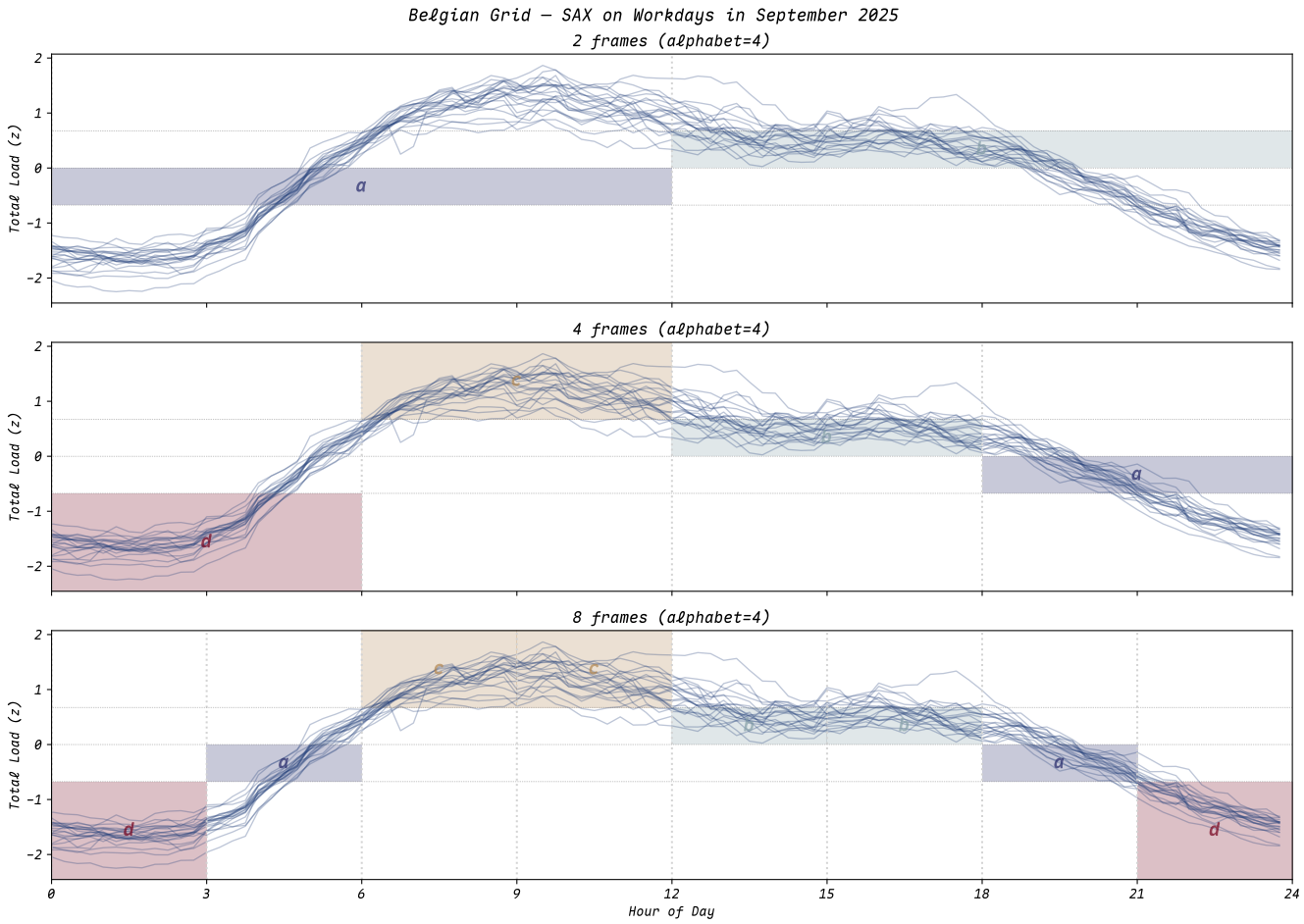



Belgian Grid — PLA on September 2025
Workdays — 2 frames
Weekends — 2 frames
4 frames
4 frames
8 frames
8 frames
Total Load (z)
Hour of Day

Piecewise Aggregate Approximation



Symbolic Aggregate Approximation



EXERCISE

Create three discretisers: *PiecewiseLinear*, *PiecewiseAggregate*, and *SymbolicAggregate*.

For now, apply them with:

```
segmenter = ...({
  "max_depth": lambda data: 1,
  "frames": lambda depth: 8,
  "node_type": ...
})
```

- For *node_type*, use:
  - *PiecewiseLinear* → *PLANode*
  - *PiecewiseAggregate* → *PAANode*
  - *SymbolicAggregate* → *SAXNode*
- Look at the documentation for other required parameters.

Trading accuracy for speed

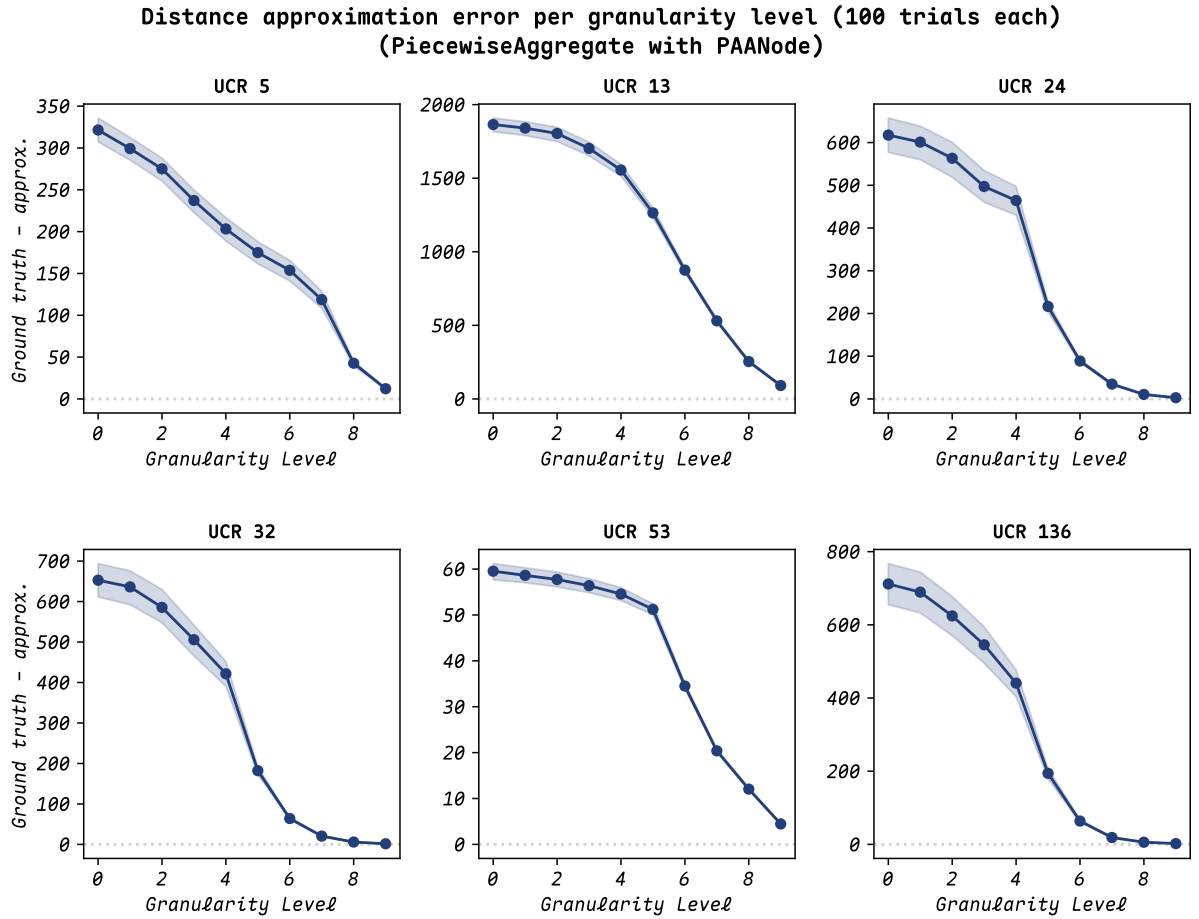
EXERCISE

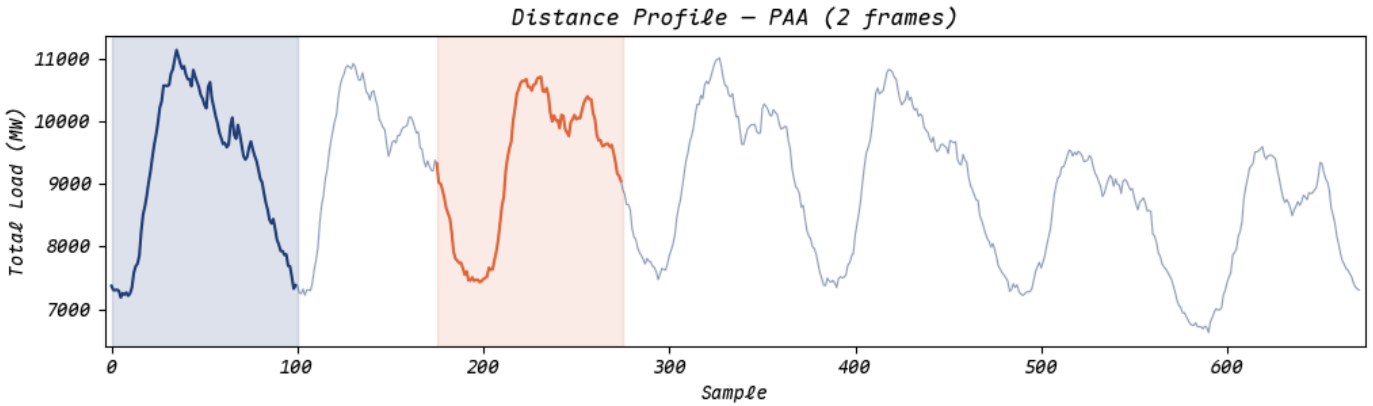
Use your *PLA*, *PAA*, and *SAX* approximations to recalculate the distance profile. How do they compare to the actual distance profile?

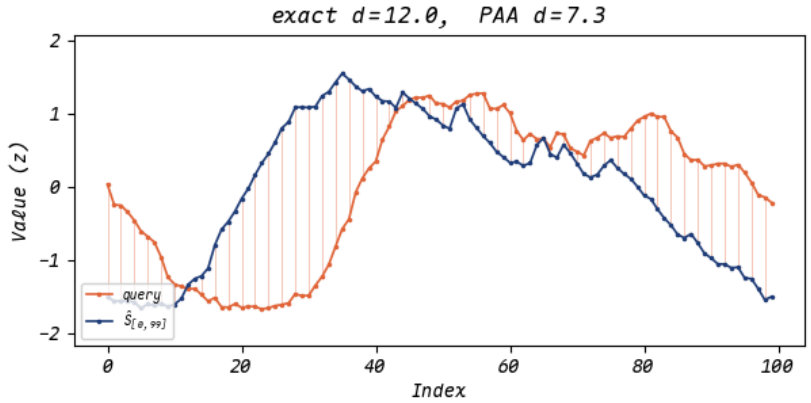
```
distance[i] = query_node.distance(node_i)
```

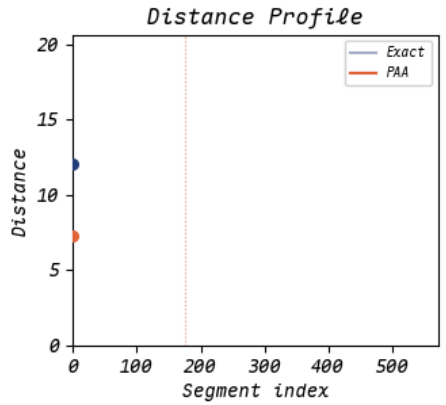
QUESTION

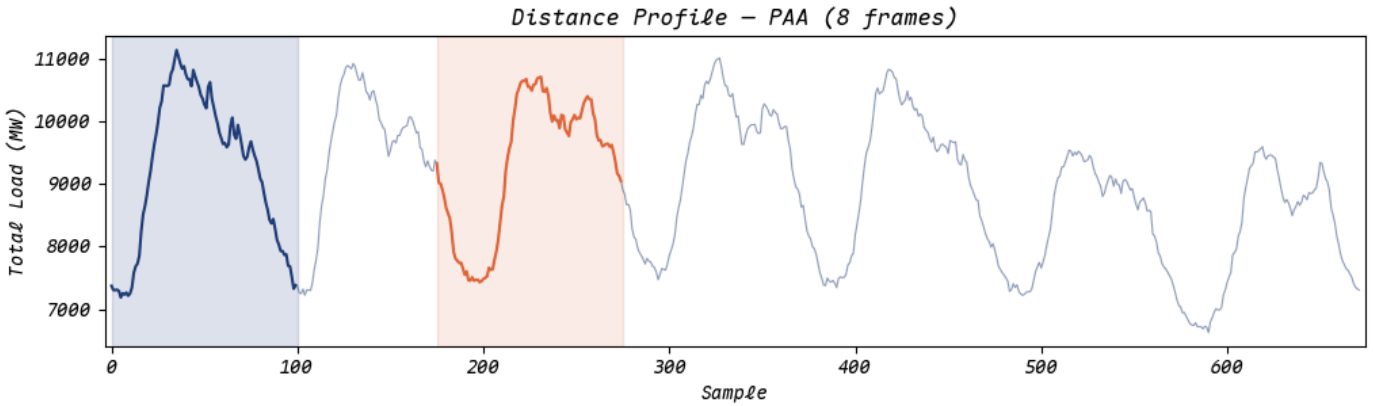
What happens if you vary the number of frames of your approximation?

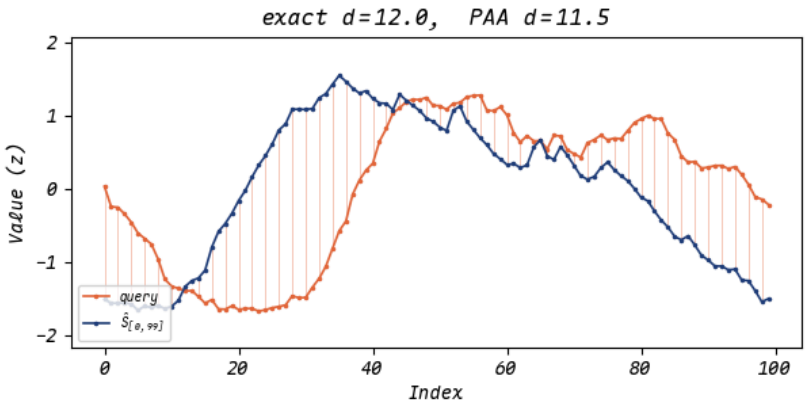

Distance approximation error per granularity level (100 trials each)
(PiecewiseAggregate with PAANode)
UCR 5
UCR 13
UCR 24
UCR 32
UCR 53
UCR 136
Ground truth - approx.
Granularity Level

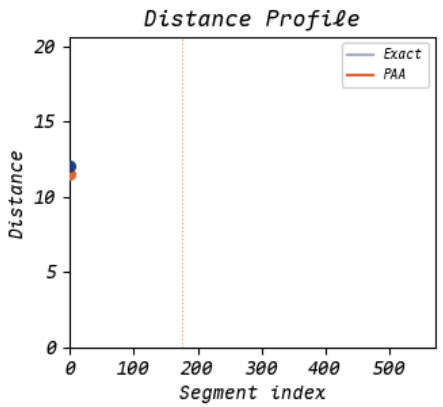

Distance Profile — PAA (2 frames)
Total Load (MW)
11000
10000
9000
8000
7000
0
100
200
300
400
500
600
Sample

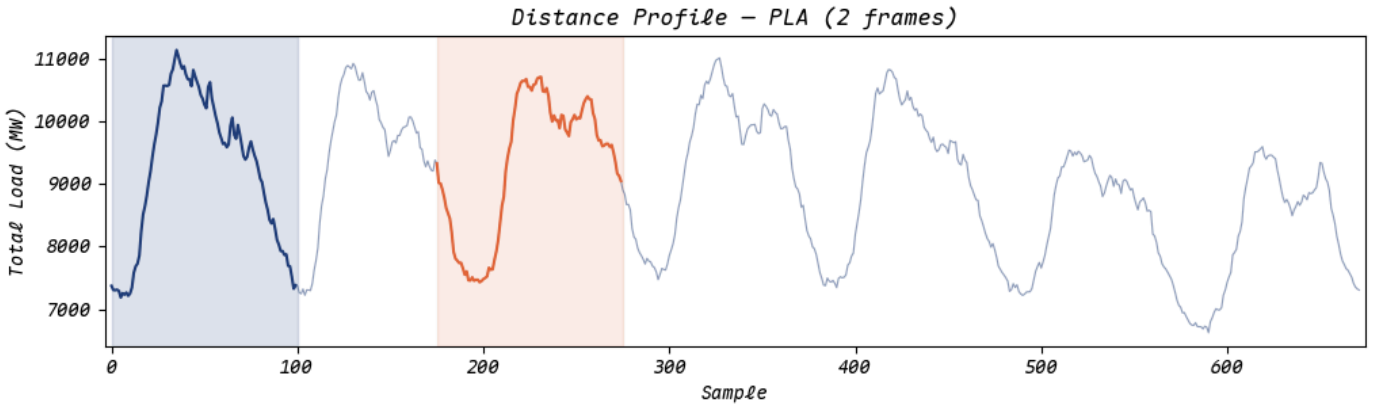

exact d=12.0, PAA d=7.3
Value (z)
2
1
0
−1
−2
query
$\hat{S}_{[0,99]}$
0
20
40
60
80
100
Index

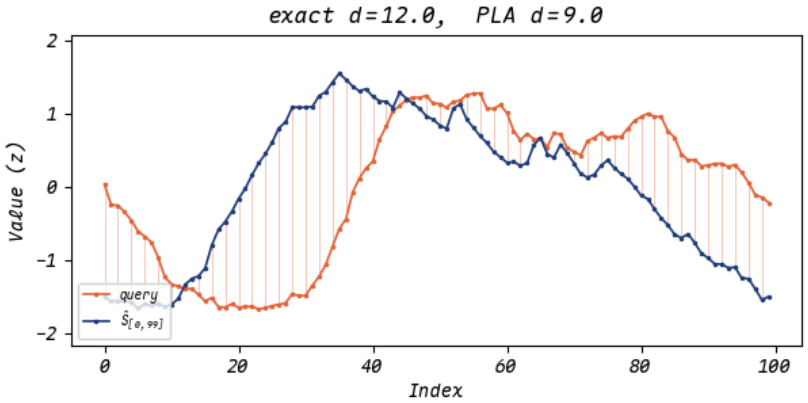

Distance Profile
Exact
PAA
Distance
20
15
10
5
0
0
100
200
300
400
500
Segment index

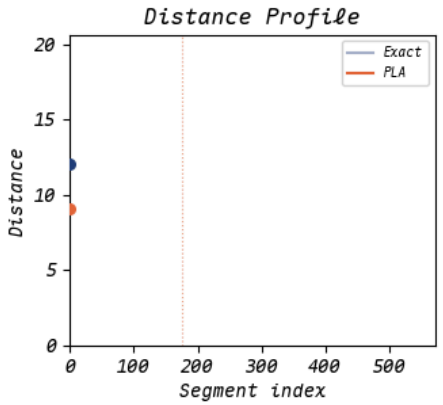

Distance Profile — PAA (8 frames)
Total Load (MW)
11000
10000
9000
8000
7000
0
100
200
300
400
500
600
Sample

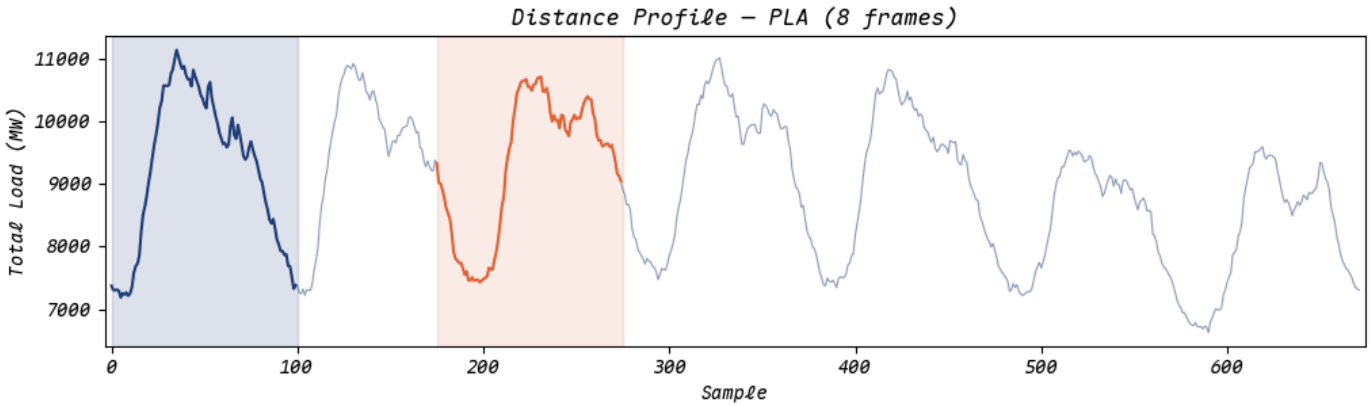

exact d=12.0, PAA d=11.5
Value (z)
2
1
0
−1
−2
0
20
40
60
80
100
Index
query

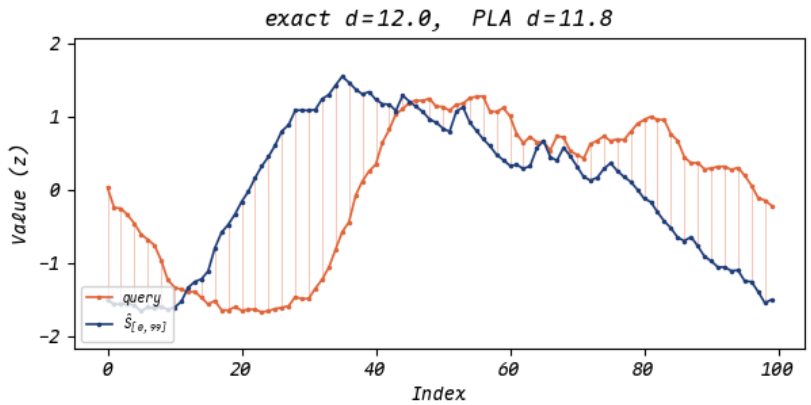

Distance Profile
Exact
PAA
Distance
20
15
10
5
0
0
100
200
300
400
500
Segment index

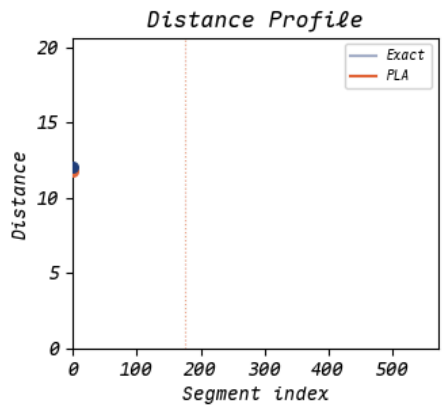

Distance Profile — PLA (2 frames)
Total Load (MW)
11000
10000
9000
8000
7000
0
100
200
300
400
500
600
Sample

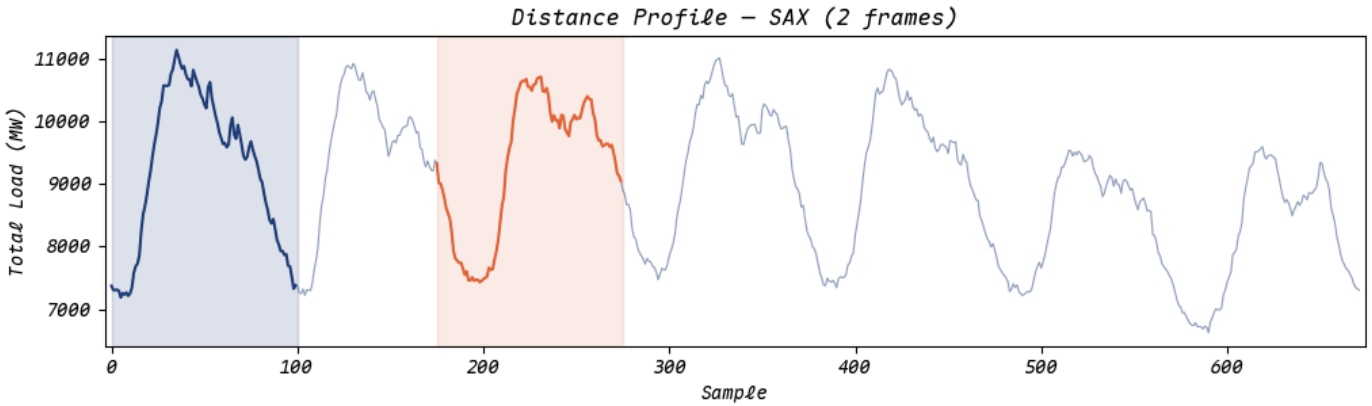

exact d=12.0, PLA d=9.0
Value (z)
2
1
0
−1
−2
query
$\hat{S}_{[0,99]}$
0
20
40
60
80
100
Index

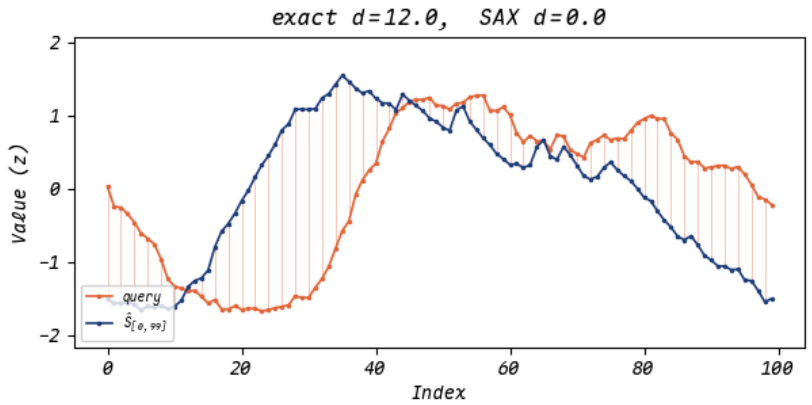

Distance Profile
Exact
PLA
Distance
20
15
10
5
0
0
100
200
300
400
500
Segment index

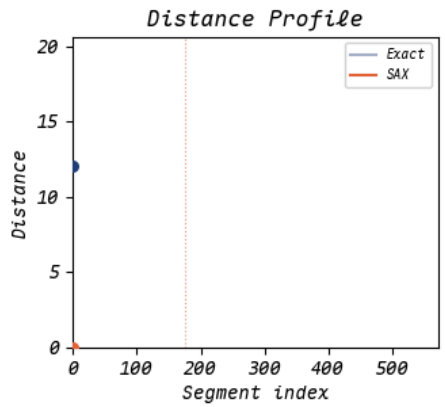

Distance Profile — PLA (8 frames)
Total Load (MW)
11000
10000
9000
8000
7000
0
100
200
300
400
500
600
Sample

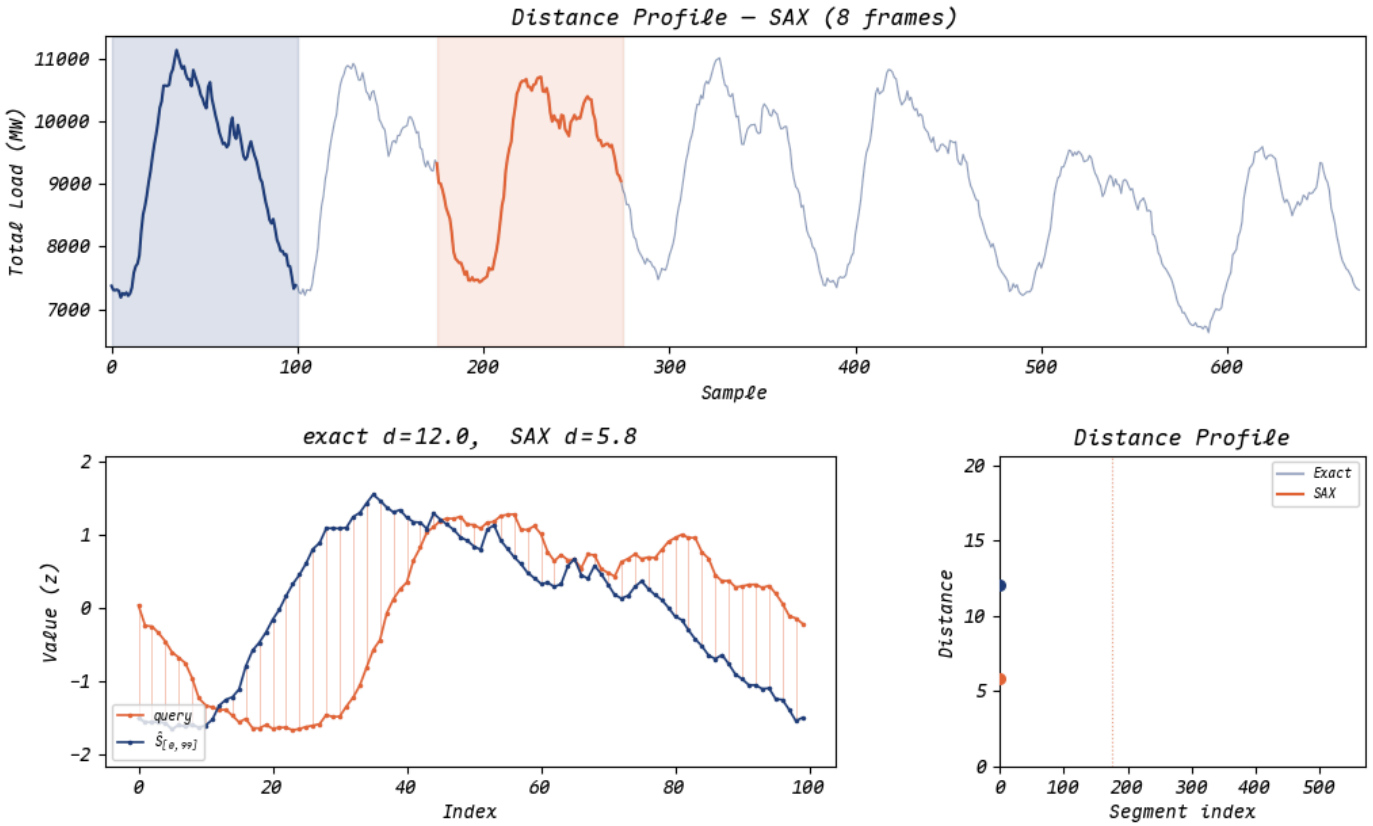

exact d=12.0, PLA d=11.8
Value (z)
2
1
0
−1
−2
query
$\hat{S}_{[0,99]}$
0
20
40
60
80
100
Index


Distance Profile
Exact
PLA
Distance
20
15
10
5
0
0
100
200
300
400
500
Segment index


Distance Profile — SAX (2 frames)
Total Load (MW)
11000
10000
9000
8000
7000
0
100
200
300
400
500
600
Sample

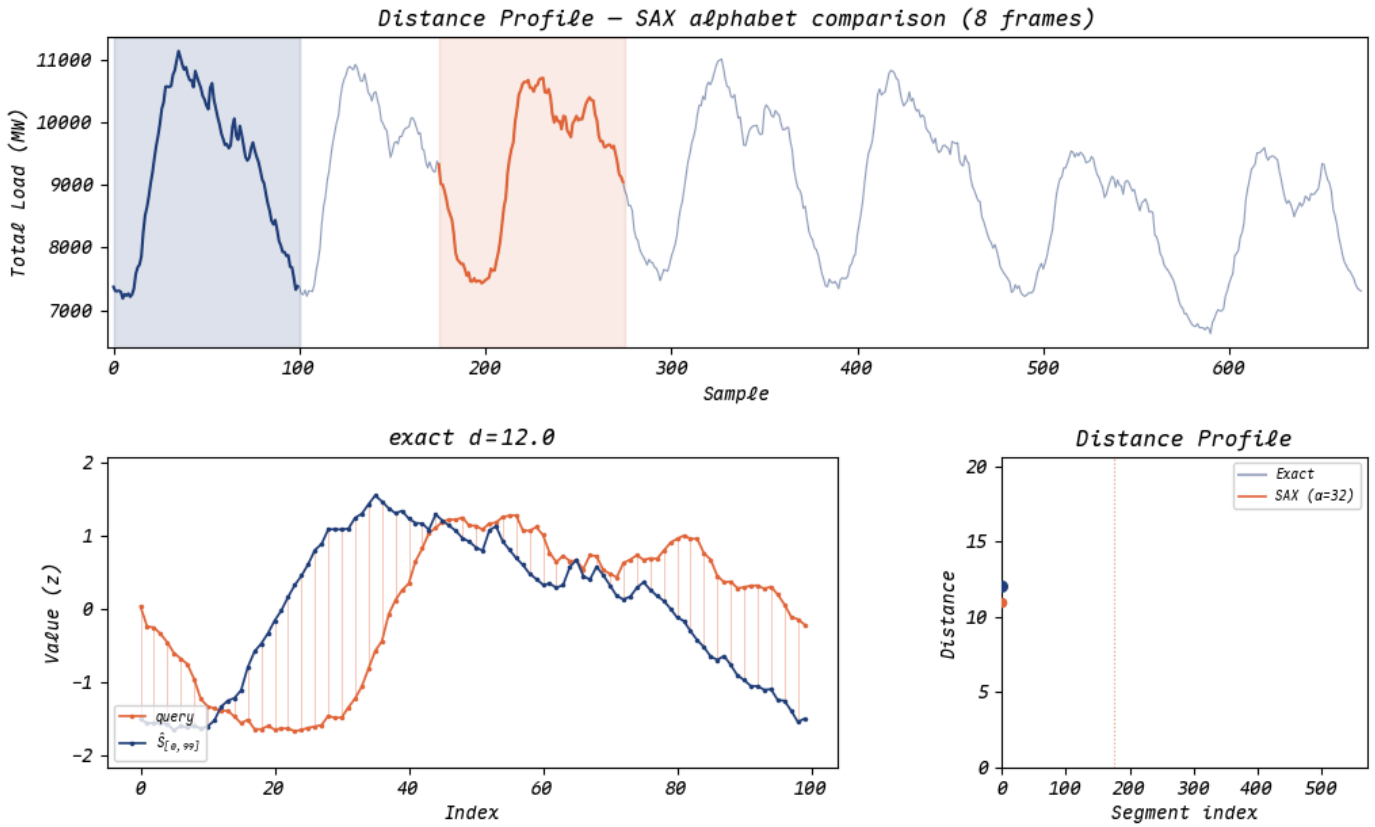

exact d=12.0, SAX d=0.0
Value (z)
2
1
0
−1
−2
0
20
40
60
80
100
Index
query


Distance Profile
Distance
20
15
10
5
0
0
100
200
300
400
500
Segment index
Exact
SAX


Distance Profile — SAX (8 frames)
Total Load (MW)
11000
10000
9000
8000
7000
0
100
200
300
400
500
600
Sample
exact d=12.0, SAX d=5.8
Value (z)
2
1
0
−1
−2
query
$\hat{S}_{[0,99]}$
0
20
40
60
80
100
Index
Distance Profile
Distance
20
15
10
5
0
Exact
SAX
0
100
200
300
400
500
Segment index

QUESTION

SAX seems to underperform when calculating the distance profile: why?


Distance Profile — SAX alphabet comparison (8 frames)
Total Load (MW)
11000
10000
9000
8000
7000
0
100
200
300
400
500
600
Sample
exact d=12.0
Value (z)
2
1
0
−1
−2
query
$\hat{S}_{[0,99]}$
0
20
40
60
80
100
Index
Distance Profile
Distance
20
15
10
5
0
Exact
SAX (a=32)
0
100
200
300
400
500
Segment index

Sequence learning

Create **models** based on legitimate sequences, used for:

- **Recognition**: identify known patterns in new data
- **Prediction**: forecast future values or events
- **Generation**: produce synthetic sequences resembling real ones

Recognition

- Is a sequence legitimate, given a set of known legitimate sequences (or knowledge of the underlying generative process)?

$$(s_i, s_{i+1}, \ldots, s_j) \longrightarrow a \text{ with } a \in \{0, 1\}.$$

Generation and prediction

- Can we generate sequence elements in their natural order?
- Can we predict future sequence elements based on one or more preceding elements of the sequence?

$$(s_i, \ldots, s_j) \longrightarrow (s_{j+1}, \ldots, s_{j+k}) \text{ where } 1 \leq i \leq j < \infty \text{ and } k \geq 1.$$

Sequence learning tasks

- **Novelty detection:**
  - $S = (s_i, s_{i+1}, \ldots, s_j) \longrightarrow \{(k, l) \mid S_{[k,l]} \to 0\}.$
- **Anomaly detection:**
  - $S = (s_i, s_{i+1}, \ldots, s_j) \longrightarrow \{(S_i, a) \mid a \in [0, 1]\}.$
  - $S = (s_i, s_{i+1}, \ldots, s_j) \longrightarrow \{(S_{[k,l]}, a) \mid a \in [0, 1]\}.$
  - ...

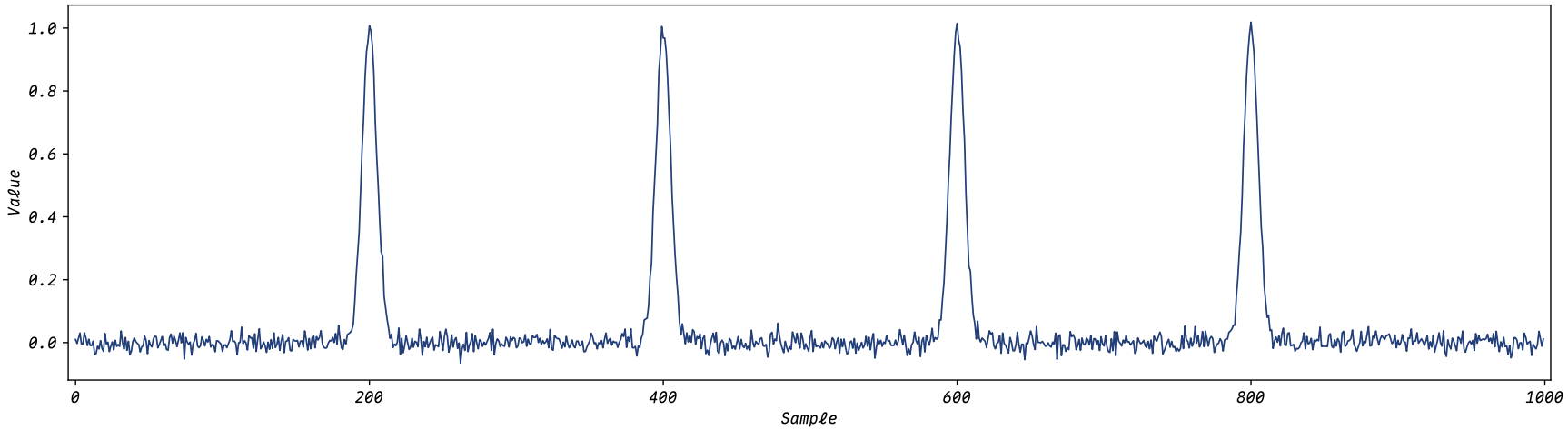
Time series anomaly detection

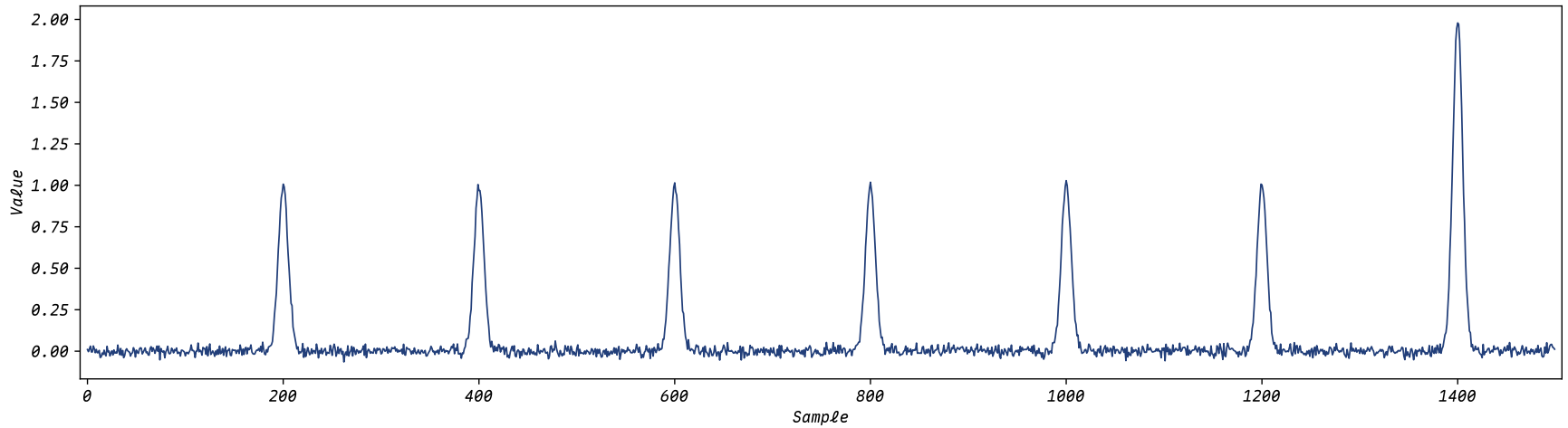
Detecting and flagging data points, subsequences, or sequences that **deviate** from **expected normal behaviour** described in legitimate sequences.

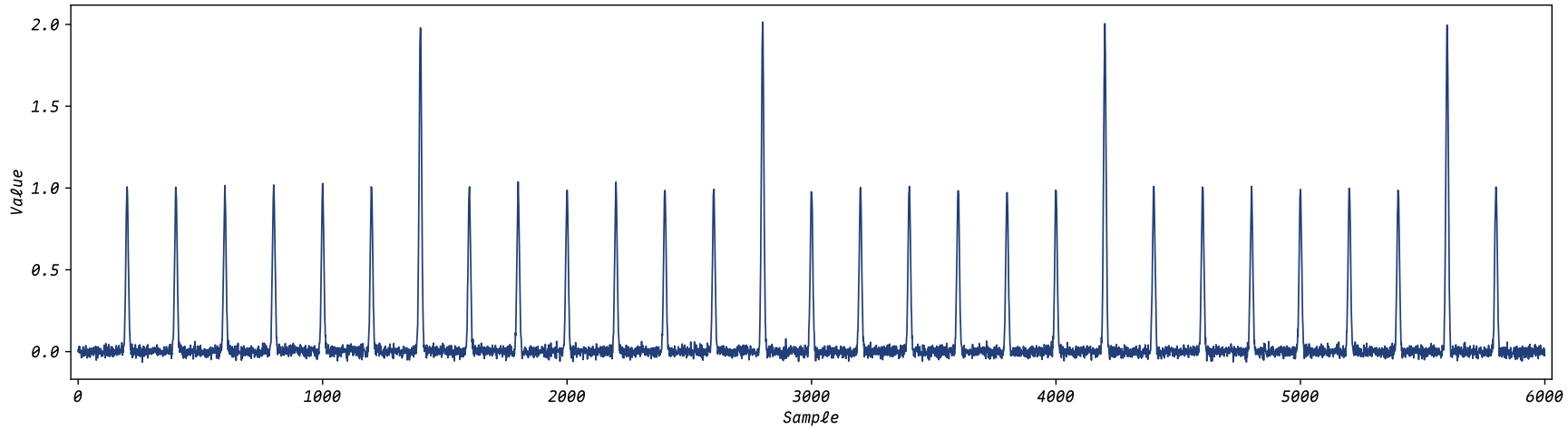
There is no *one-size-fits-all*

No universal definition of normal behaviour and anomaly, so there is no one-size-fits-all detection algorithm.

→ **What counts as an anomaly depends on domain, context, and use case.**


1.0
0.8
0.6
0.4
0.2
0.0
Value
0
200
400
600
800
1000
Sample


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Value
0
200
400
600
800
1000
1200
1400
Sample

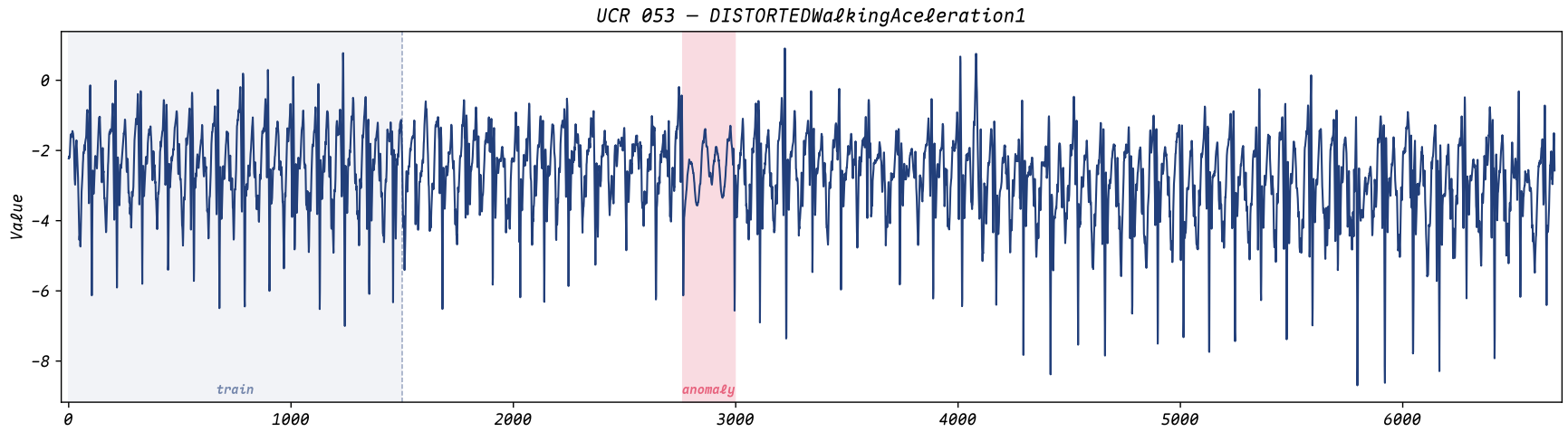

2.0
1.5
1.0
0.5
0.0
Value
0
1000
2000
3000
4000
5000
6000
Sample

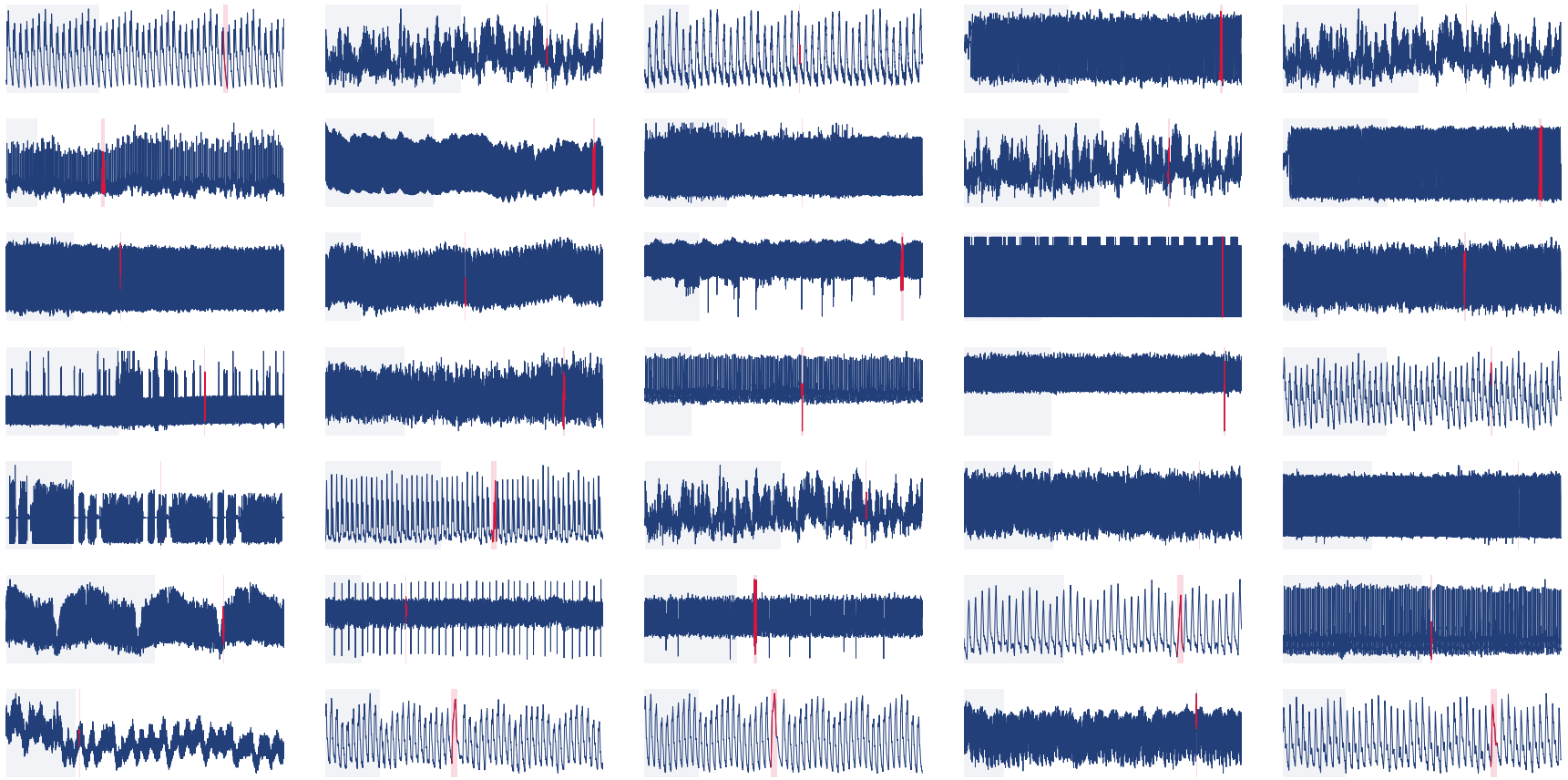
TSAD benchmarks are flawed

- Imperfect ground truth
- Trivial anomalies only: simple spikes, drops, ...
- "Guessable"

UCR time series anomaly archive

- **Dataset of 250 different univariate time series**
  - Data from various domains, differing in length and structure
  - Each of the time series already split into train and test
  - Each of the test sections has exactly **one** anomaly
- **The goal**: find an approach that can point to the anomaly in the test section, using knowledge gathered from the train section

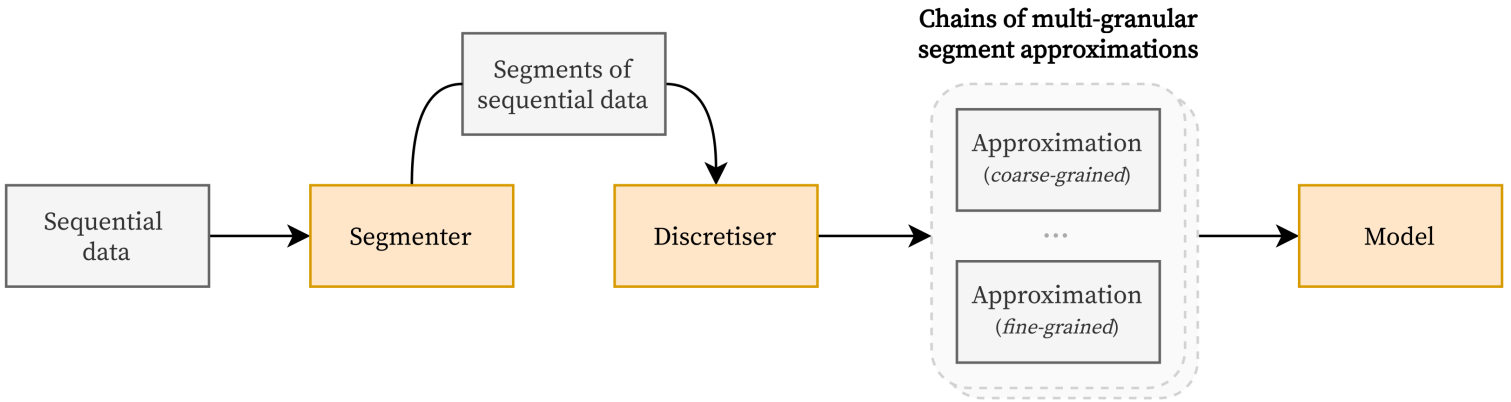

UCR 053 — DISTORTEDWalkingAceleration1
Value
0
−2
−4
−6
−8
train
anomaly
0
1000
2000
3000
4000
5000
6000



Demo: guess the anomaly

pbsf-lib

The components we have used so far can be used for time series anomaly detection.


Chains of multi-granular segment approximations
Segments of sequential data
Sequential data
Segmenter
Discretiser
Approximation (coarse-grained)
...
Approximation (fine-grained)
Model

Segmenters

- Split input sequences into ordered, preprocessed sequences of overlapping or non-overlapping segments.
  - **Example**: `SlidingWindow`

Discretisers

- Take segments as input, return *Chains*: multiple approximations in various granularities, ranging from **coarse- to fine-grained**.
  - **Examples**: *PLA*, *PAA*, *SAX*

Approximations

- A segment representation at one specific level of granularity.
  - Each type of approximation implements their own distance function!

Simplest example: *SlopeSignNode*

- For *PLA*, only store the sign of the slope for each frame.
  - **Equality:** *((self.slopes ≥ 0) == (node.slopes ≥ 0)).all()*

Models

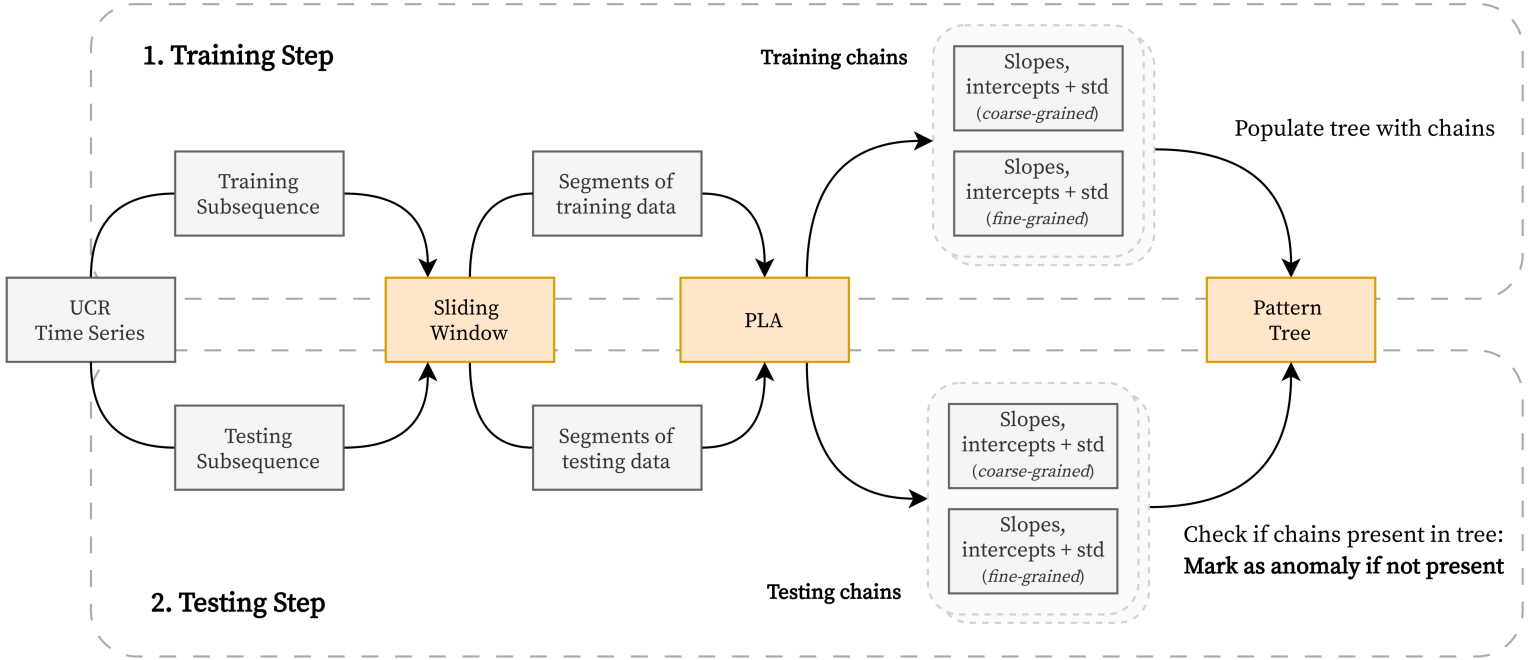
- Aggregation of segment approximations, used to investigate properties of the input data or to compare new data to this model
  - **Examples:** `PatternTree`, `PatternGraph`, `NestedWordSet`

Example: *PatternTree*

- Nodes of the tree are segment approximations.
  - Nodes closer to the root are coarse-grained (= *low number of frames*).
  - Nodes further from the root are fine-grained (= *high number of frames*).
- When new chains come in, we can:
  - Check if their nodes are present in our tree.
  - Match nodes, find their nearest neighbour in the model.
  - Merge nodes if they are similar (based on their equality).
  - ...
- **Can be used for sequence learning tasks:** motif discovery, discord detection, subsequence matching, anomaly detection, ...

Example algorithm: *hpm*

- **"Hierarchical Pattern Matching"**-based anomaly detection.



EXERCISE

- Split your time series in train and test.
- Introduce a synthetic discord in your test set.
- Apply the `hpm` algorithm: does it detect the discord?

Matrix Profile with *pbsf-lib*