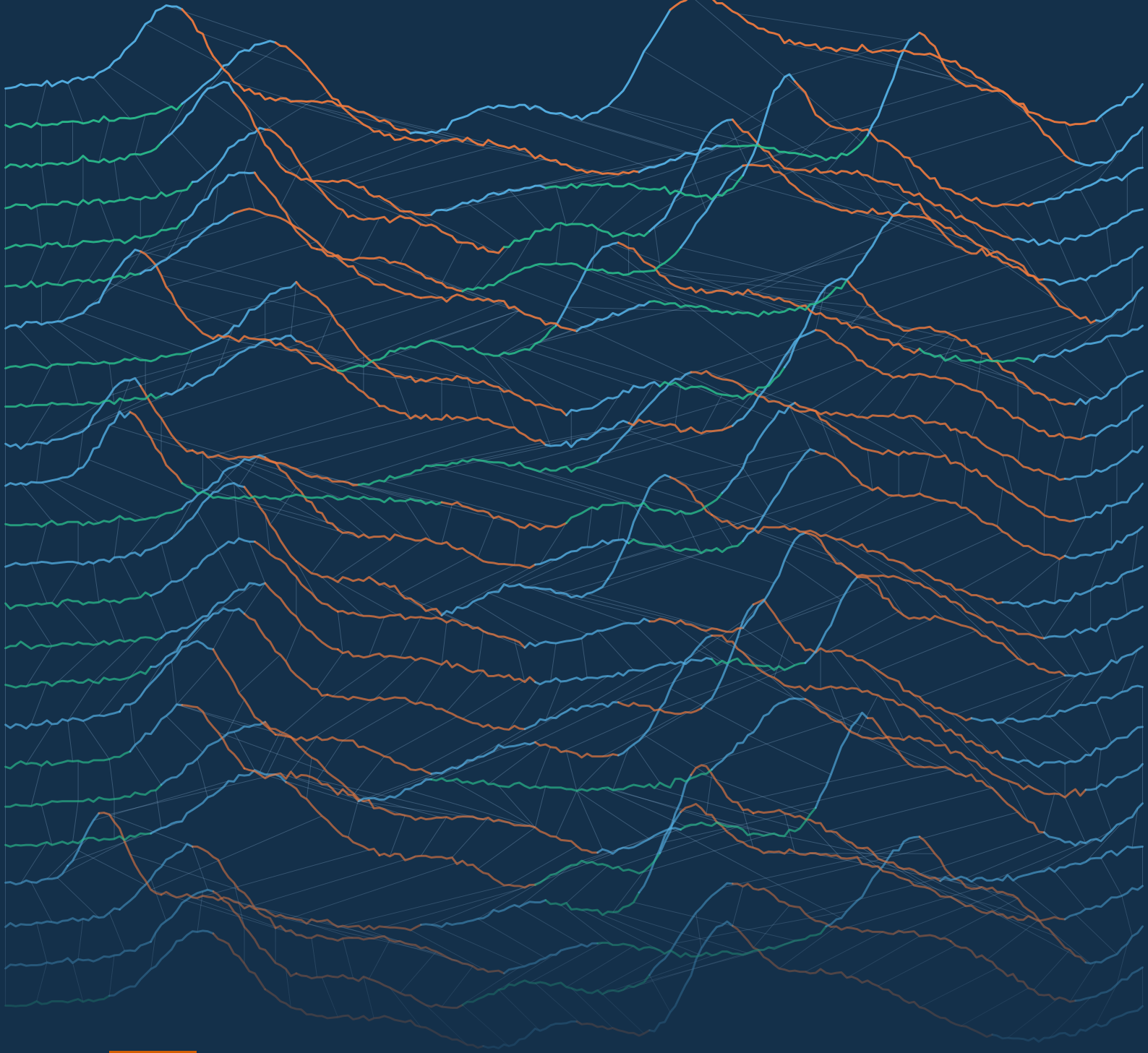


Segmentation, Warping and Clustering of Multivariate Time Series in Python

The `mvtsa` toolbox

János Abonyi

University of Pannonia, Veszprém



Segmentation, Warping and Clustering of Multivariate Time Series in Python

The `mvtsa` toolbox

János Abonyi

University of Pannonia, Veszprém

Version 1.0 · November 2026

Segmentation, Warping and Clustering of Multivariate Time Series in Python. The `mvtsa` toolbox

János Abonyi

Version 1.0, November 2026

© 2026 János Abonyi. This book is licensed under the Creative Commons Attribution 4.0 International License (CC BY 4.0, https://creativecommons.org/licenses/by/4.0/): it may be copied, redistributed, and adapted for any purpose, provided the source is credited. The `mvtsa` software described in the book is released under the MIT License.

Software: https://github.com/ComplexSystemsIntelligence/mvtsa
Documentation: https://complexsystemsintelligence.github.io/mvtsa
Installation: `pip install mvtsa`

The bundled BasicMotions, JapaneseVowels and GunPoint collections come from the UEA & UCR time series classification archives; all other data are generated or simulated by the package. The research summarized in this book was supported by the National Research, Development and Innovation Office of Hungary (NKFIH) through the OTKA 143482 project.

Typeset with LaTeX in Libertinus Serif, Raleway, and IBM Plex Mono. Every figure that shows a computed result was produced by an example script of the package. The cover shows twenty-four warped copies of one signal computed with `mvtsa`: every copy is cut into linear segments coloured by their trend, and the faint lines between neighbouring copies are their dynamic time warping alignments.

Preface

Most of the data that a chemical plant, a power station or a production line records are multivariate time series: dozens or hundreds of variables sampled every second, minute or hour for years. The information an engineer needs is rarely in a single variable. It is in how the variables move *together*: a feed and a temperature that stop following each other, a correlation structure that changes when the reactor switches to a new product grade, a sequence of operating regimes that repeats every campaign. Three questions return again and again. Where does the behaviour of the process change? How similar are two records whose time axes do not agree? Which records, segments or units belong together?

This book and the `mvtsa` package answer these questions. They grew out of two decades of work of my group at the University of Pannonia (University of Veszprém until 2006). It started with the observation that segmenting a multivariate series is a constrained clustering problem, and that principal component models measure the homogeneity of a segment much better than the means of the variables: a segment is homogeneous if one PCA model describes all of its samples. The fuzzy segmentation algorithm built on this idea, a modified Gath–Geva clustering with probabilistic PCA models and Gaussian memberships in time, was published with Balázs Feil, Sándor Németh and Péter Árva [1, 2]; Kalman filtering made it applicable to noisy process transitions [3]. Zoltán Bankó turned the PCA costs into a bottom-up segmentation and, from there, into correlation-based dynamic time warping, which compares multivariate series by the sequence of their correlation structures [4, 5]; his doctoral work added dynamic PCA, a mixed dissimilarity measure for piecewise linear approximations and the tightness of warping [6, 7, 8]. Balázs Balaskó compared process trends by aligning their qualitative episodes [9, 10]. László Dobos developed the on-line, recursive dynamic-PCA segmentation and the segmentation of historical data by the Fisher information they carry for parameter estimation [11, 12, 13, 14, 15]. Model-based segmentation found its way into energy monitoring [16], decision trees into the description of operating regimes [17], fuzzy Sammon mapping into the visualization of clusters of trajectories [18]; more recent work clusters trajectories by neighbourhood preservation [19] and gives visibility graphs the meaning of statistical process control zones [20].

These methods were published in journals and conference proceedings and implemented in MATLAB. The field around them has grown enormously in the meantime: exact penalized change point detection, Bayesian on-line detection, self-supervised segmentation, soft and differentiable alignment, shape-based clustering, large benchmarks and statistical comparison protocols. This book places our methods among the best of these, implements them all in one Python package, and adds what a practitioner needs around them: evaluation, validation, explanation, simplification and the operation of the models in production. Several methods are new: merge persistence and the segmentogram of the bottom-up hierarchy, predictive homogeneity with any scikit-learn model, consensus segmentation, Riemannian and mixed local distances for correlation-based warping, and episode warping.

How the book is organized. Chapters 1 and 2 introduce the questions, the package and the common foundations: notation, segmentations as partitions, warping paths, PCA models and their statistics. Chapter 3 collects the representations that the later methods build on. Chapters 4 to 6 are about segmentation: cost-based methods and their search strategies, latent-variable methods – the core of our own work – and the probabilistic, profile-based and state models of the recent literature. Chapters 7 and 8 are about warping: the family of elastic measures, then correlation-based and segment-level warping. Chapter 9 groups series, segments and trajectories. Chapters 10 to 12 turn methods into trustworthy tools: evaluation and validation, visualization and explanation, model operations. Chapter 13 brings everything together in three case studies. The appendices list the API, the example scripts, and every method with its origin.

How to read it. Every chapter starts with a short roadmap and ends its theory sections with a summary box; algorithms are stated in boxes, concepts are drawn as diagrams. Every figure that shows a computed result is produced by one of the forty-eight example scripts of the package, and the code printed in the book is cut from these scripts, so what you read is what runs. The scripts are also available as Jupyter notebooks in the online documentation. A reader interested in a single method can go directly to its section and its example; the notation is the same everywhere (table 2.1).

Acknowledgements. The methods of this book are the work of many people. I am grateful to my former and present doctoral students and co-authors – Zoltán Bankó, Balázs Feil, Balázs Balaskó, László Dobos, Sándor Németh, Péter Árva, Ferenc Szeifert, Tamás Varga, Tibor Kulcsár, Bálint Levente Tarcsay, Martin Ferenczi and Alex Kummer – for the ideas, the experiments and the many discussions behind them. The research summarized in this book was supported by the National Research, Development and Innovation Office of Hungary (NKFIH) through the OTKA 143482 project.

Veszprém, November 2026 János Abonyi

Contents

Preface iii

1 Introduction 1

1.1 Three questions about multivariate time series 1
1.2 Why the variables have to be analysed together 2
1.3 Two decades of methods 3
1.4 The package 4
1.5 A first session 4
1.6 Related software 6

2 Foundations 7

2.1 Series, collections and notation 7
2.2 Segmentations 7
2.3 Warping paths 9
2.4 PCA models and their statistics 9
2.5 Preprocessing 12
2.6 Conventions of the API 12

3 Representations 15

3.1 Piecewise approximations 15
3.2 Symbolic representations 16
3.3 Qualitative episodes 16
3.4 Features and path signatures 17
3.5 Networks and images 18
3.6 Matrix profiles 18

4 Cost-based segmentation 21

4.1 Segment costs 21
4.2 Search strategies 22
4.3 The merge hierarchy 24

4.4 How many segments? 25

5 Latent-variable segmentation 29

5.1 PCA segmentation 29
5.2 Dynamic PCA segmentation 31
5.3 On-line segmentation with recursive PCA 32
5.4 Fuzzy segmentation 35
5.5 Segmentation by Fisher information 36
5.6 Model-based segmentation of energy indicators 37

6 Probabilistic, profile-based and state segmentation 41

6.1 Bayesian on-line change point detection 41
6.2 States: segments that recur 43
6.3 Changes of shape: ClaSP and FLUSS 45
6.4 Predictive homogeneity 45
6.5 Consensus segmentation 47
6.6 Streams 48

7 Elastic similarity 51

7.1 Dynamic time warping 51
7.2 Variants 53
7.3 Edit-type measures 53
7.4 Soft alignment 55
7.5 Averaging 56
7.6 Diagnosing and learning warping 57

8 Correlation-based and segment-level warping 59

8.1 Correlation-based dynamic time warping 59
8.2 Process-dynamics-aware warping 62
8.3 Riemannian and mixed segment distances 63
8.4 Warping piecewise linear approximations 65
8.5 Aligning qualitative episodes 67

9 Clustering 69

9.1 Partitional clustering 69
9.2 Relational clustering 71
9.3 Maps of distance matrices 72
9.4 Features, segments and trajectories 73

10 Evaluation and validation **77**

10.1 Segmentation metrics . . . 77
10.2 Clustering metrics . . . 78
10.3 Retrieval . . . 79
10.4 Stability . . . 79
10.5 Comparing methods . . . 80
10.6 Tuning . . . 81

11 Visualization and explanation **85**

11.1 A segmentation report . . . 85
11.2 What changed? . . . 86
11.3 Rules for states and transition networks . . . 87
11.4 Six-sigma visibility graphs . . . 87
11.5 Alignments and collections . . . 88

12 Model operations **91**

12.1 The life cycle . . . 91
12.2 Configurations and pipelines . . . 91
12.3 Model files, registry and tracking . . . 92
12.4 Contracts and drift . . . 93
12.5 Model cards and the command line . . . 94

13 Case studies **97**

13.1 Grade transitions of a polymerization reactor . . . 97
13.2 Operating regions of a pH process . . . 99
13.3 Explaining operating regimes . . . 100
13.4 A workflow . . . 101

A API reference **103**

A.1 `mvtsa` . . . 103
A.2 `mvtsa.preprocessing` . . . 104
A.3 `mvtsa.representation` . . . 105
A.4 `mvtsa.segmentation` . . . 108
A.5 `mvtsa.warping` . . . 111
A.6 `mvtsa.distances` . . . 113
A.7 `mvtsa.clustering` . . . 114
A.8 `mvtsa.evaluation` . . . 114
A.9 `mvtsa.simplify` . . . 116
A.10 `mvtsa.explain` . . . 116
A.11 `mvtsa.mlops` . . . 117

A.12 `mvtsa.io` . . . 118
A.13 `mvtsa.viz` . . . 118
A.14 `mvtsa.datasets` . . . 121

B The example gallery **123**

B.1 First steps: series . . . 123
B.2 Cost-based segmentation: costs . . . 124
B.3 Latent-variable segmentation: PCA . . . 124
B.4 Probabilistic . . . 124
B.5 Elastic similarity: DTW . . . 125
B.6 Correlation-based . . . 125
B.7 Clustering series . . . 126
B.8 Evaluation, validation and benchmarking . . . 126
B.9 Visualization and explanation . . . 126
B.10 Model operations . . . 127
B.11 Case studies . . . 127

C Method index and migration **129**

Migration from the MATLAB toolboxes . . . 131

Bibliography **133**

List of Figures

1.1 The three questions of the book . . . 2
1.2 Methods of the group and milestones of the literature . . . 3
1.3 Structure of the package . . . 4
1.4 A first segmentation . . . 5

2.1 A segmentation and the three ingredients of a segmentation method . . . 9
2.2 Dynamic time warping . . . 10
2.3 The T^2 and Q statistics of a PCA model . . . 10
2.4 T^2 and Q statistics of a reference model . . . 11

3.1 Piecewise linear approximation . . . 16
3.2 The seven qualitative episode primitives . . . 17
3.3 Qualitative episodes of a signal . . . 17
3.4 Recurrence plot . . . 19

4.1 PELT with the Gaussian cost . . . 24
4.2 Segmentogram and merge persistence . . . 25
4.3 Cost curve and relative reduction rate . . . 26

5.1 The two PCA costs of a segment . . . 30
5.2 Loadings of the segment models . . . 31
5.3 Dynamic PCA segmentation of a second-order system . . . 32
5.4 Dynamic PCA segmentation of the pH process . . . 33
5.5 On-line segmentation with recursive PCA . . . 34
5.6 Fuzzy segmentation with the modified Gath–Geva algorithm . . . 36
5.7 Informative segments of a polymerization record . . . 38
5.8 Model-based segmentation of an energy indicator . . . 39

6.1 Bayesian on-line change point detection . . . 42
6.2 Recurring operating states . . . 44
6.3 ClaSP and FLUSS . . . 46

6.4 Predictive homogeneity . . . 47
6.5 Consensus of five segmenters . . . 48

7.1 A DTW alignment . . . 52
7.2 Elastic measures on Cylinder–Bell–Funnel . . . 54
7.3 Expected alignments of soft-DTW . . . 55
7.4 Arithmetic mean, DBA and soft-DTW barycenter . . . 56

8.1 Correlation-based dynamic time warping . . . 60
8.2 Segment-level alignment of two signatures . . . 61
8.3 Precision–recall curves of correlation-based and sample-level warping . . . 62
8.4 A series of the dynamic-sequence collection . . . 63
8.5 Local distances on JapaneseVowels . . . 64
8.6 Learning the weights of MDPLA . . . 66
8.7 Qualitative episodes of a noisy signal . . . 67

9.1 Cluster prototypes of DBA k-means . . . 70
9.2 Sammon map of BasicMotions . . . 72
9.3 Sammon and fuzzy Sammon maps . . . 73
9.4 Segments of a batch coloured by pattern . . . 74

10.1 F1 score and margin . . . 79
10.2 Stability of a segmentation . . . 81
10.3 Critical difference diagram . . . 82

11.1 A one-page segmentation report . . . 86
11.2 Six-sigma visibility graphs . . . 88
11.3 An alignment and its uncertainty . . . 89

12.1 Life cycle of a time-series model . . . 92
12.2 Drift monitoring . . . 94

13.1 Grades of a polymerization campaign . . . 98
13.2 Information content of the campaign . . . 99
13.3 Local gain of the pH process . . . 100
13.4 Regimes and their transition network . . . 101

CHAPTER

1



Introduction

This chapter states the three questions the book is about – where a multivariate time series changes, how similar two series are when their time axes do not agree, and which series belong together – and explains why the answers have to look at how the variables move *together*. It then traces the methods of the book back to their origins, introduces the structure of the `mvtsa` package, and walks through a first session.

1.1 Three questions about multivariate time series

A multivariate time series is a matrix $\mathbf{X} \in \mathbb{R}^{T \times n}$ whose row $\mathbf{x}_t$ holds the values of n variables at time t. Process historians, sensor networks, wearable devices and pen tablets produce such matrices in large numbers. Whatever the domain, three tasks recur (fig. 1.1).

Segmentation. Where does the behaviour change? A segmentation cuts $\mathbf{X}$ into contiguous pieces that are internally homogeneous: an operating regime, a product grade, an activity, a stroke of a signature. Segmentation compresses a long record into a short sequence of interpretable episodes, localizes faults and transitions, and selects the parts of a record that are worth modelling.

Warping. How similar are two series whose time axes do not agree? Two batches of the same recipe, two signatures of the same writer or two utterances of the same word follow the same pattern at different speeds. An elastic measure aligns them before comparing them; the alignment itself – the warping path – tells which phases correspond.

Clustering. Which series, segments or objects belong together? Grouping records reveals recurring operating modes, typical batch trajectories, classes of motion and groups of units that behave alike.

The three tasks support each other. Segments are the natural units of comparison, so segmentation makes warping faster and more robust; warping distances are the input of clustering; clustering the segments of many series yields a vocabulary of recurring regimes, which in turn labels new data. The methods of this book therefore form one toolbox rather than three.

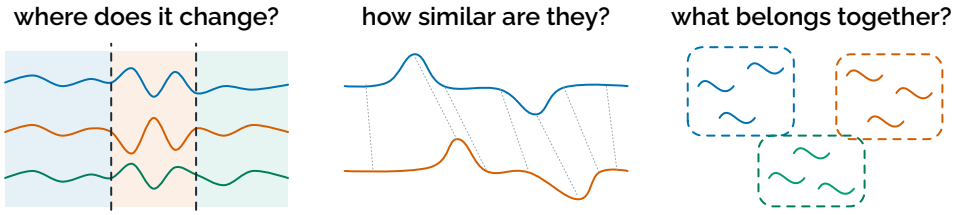


Figure 1.1. The three questions of the book. (a) A segmentation cuts a series into homogeneous pieces; here the correlation of the three variables changes in the middle segment while their levels stay the same. (b) An elastic measure aligns two series that follow the same pattern at different speeds before comparing them. (c) Clustering groups series (or segments) by a distance.

1.2 Why the variables have to be analysed together

The most common way to segment, compare or cluster multivariate series is to treat the variables one by one, or to stack them into a long vector and use the Euclidean distance. Both ignore what is often the most informative property of process data: the *correlation* of the variables. A heat exchanger whose fouling changes, a reactor that switches between product grades, a control loop whose tuning changes – all of these alter how the variables move together, sometimes without moving any of their means or variances. Principal component analysis (PCA) is the classical tool to describe this structure: a PCA model of a segment is a low-dimensional subspace in which the samples live, and two segments differ if their subspaces differ [21, 22, 23].

Figure 1.4 shows the simplest example. Three variables share one latent factor whose loadings change three times; the levels and the variances stay the same. A segmentation by the means sees nothing but noise, while a segmentation by the reconstruction error of local PCA models finds the four regimes exactly. This observation is the starting point of the latent-variable methods of chapter 5 and of correlation-based warping in chapter 8. It does not make the other methods obsolete: when the information lies in levels, shapes or recurring patterns, the

costs of chapter 4, the subsequence methods of chapter 6 and the elastic measures of chapter 7 are the better tools, and the package treats all of them alike.

1.3 Two decades of methods

Figure 1.2 places the methods developed by the group (right) among milestones of the literature (left). The first line of work, fuzzy and PCA-based segmentation of process data [1, 2, 3], produced two branches: correlation-based warping and its extensions in the doctoral work of Zoltán Bankó [4, 5, 6, 7, 8], and on-line and information-driven segmentation in the work of László Dobos [11, 12, 13, 14]. Qualitative episodes [9], decision-tree descriptions of operating regimes [17], model-based segmentation for energy monitoring [16], trajectory clustering [19] and zone-labelled visibility graphs [20] complete the picture. Over the same period, the literature brought exact penalized segmentation [24], Bayesian on-line change point detection [25], the matrix profile and self-supervised segmentation [26, 27, 28], differentiable alignment [29] and shape-based clustering [30]. Recent benchmarks agree that no segmenter dominates on all kinds of data [31, 32]; a toolbox therefore has to offer several, and the means to choose among them.

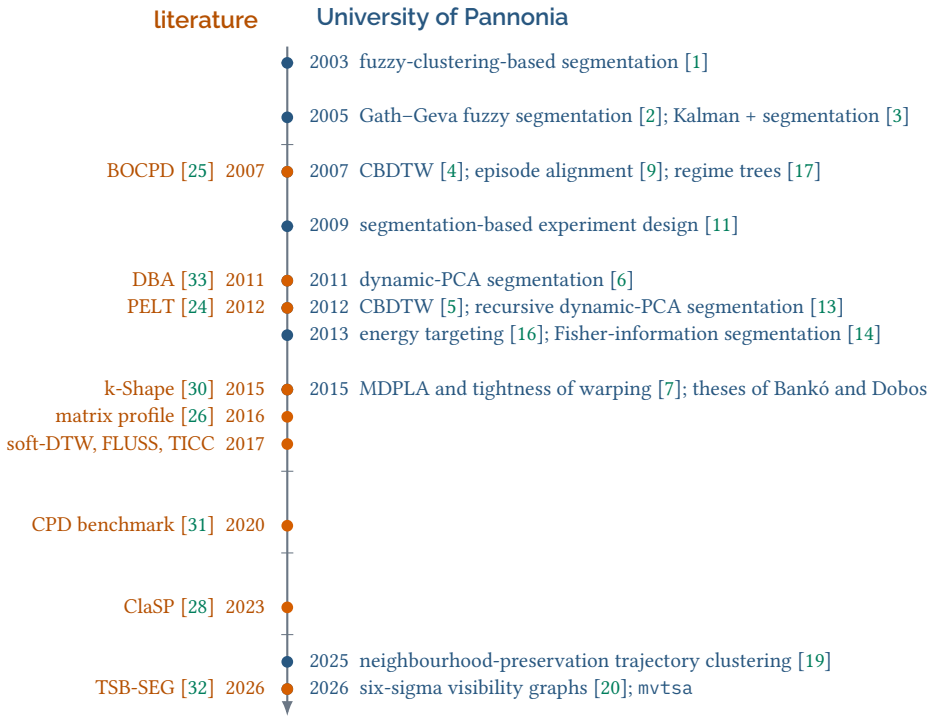


Figure 1.2. Methods developed by the group (blue, right) and milestones of the literature (vermilion, left). All are implemented in `mvtsa`; chapter C lists them with their references.

1.4 The package

`mvtsa` is organized by task (fig. 1.3). The core objects are deliberately few: a series is a NumPy array of shape (T, n), a collection is a list of such arrays of possibly different lengths, and every segmenter returns a `Segmentation` – the change points, optionally state labels, memberships and scores, with conversions to labels, tables and the formats of other libraries. Segmenters, warping models and clusterers follow the conventions of scikit-learn [34]: parameters are set in the constructor, `fit` learns, fitted attributes end with an underscore, and `get_params`/`set_params` make every object configurable. A segmentation cost can be combined with any search strategy, a distance registry makes every elastic measure available to every clusterer, and every object can be written to a configuration file and rebuilt from it. The compute-heavy dynamic programs run as compiled kernels when Numba [35] is installed and fall back to identical pure-Python code otherwise.

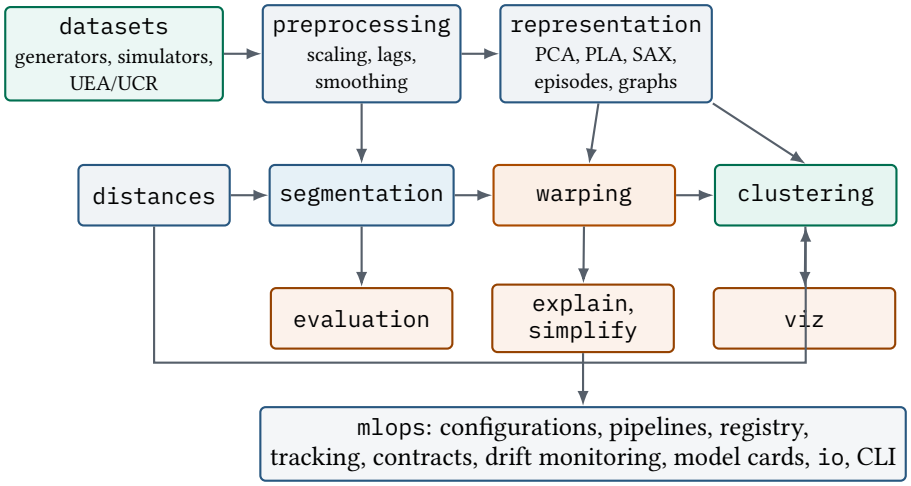


Figure 1.3. Structure of the package. Data are preprocessed and represented; segmentation, warping and clustering use each other (segment models are the units of correlation-based warping, distances feed every clusterer); evaluation, explanation, simplification and visualization act on their results; the model-operations layer makes analyses reproducible and deployable.

1.5 A first session

The example behind fig. 1.4 generates a series whose correlation structure switches three times, segments it with PCA models and evaluates the result against the known change points:

```
X, true_cps, states = mv.datasets.make_correlation_switching(
    n_segments=4, length=150, random_state=0
)
seg = mv.segment(
    X, "pca", n_segments=4, n_components=1
)  # Q-statistic cost, bottom-up search
print(seg)
print(evaluate_segmentation(true_cps, seg, len(X), margin=5))
```

The one-line function `segment` looks the method up in a registry; the same result comes from `PCASegmenter(n_segments=4, n_components=1).segment(X)`. `evaluate_segmentation` returns the metrics of chapter 10: the F1 score of the detected change points within a margin, the covering of the true segments, the Hausdorff distance of the boundaries and others. A segmentation by the means, in contrast, misses the changes:

```
seg_l2 = mv.segment(X, "bottomup", cost="l2", n_segments=4)
print(
    "L2 change points:",
    seg_l2.change_points,
    " PCA change points:",
    seg.change_points,
)
```

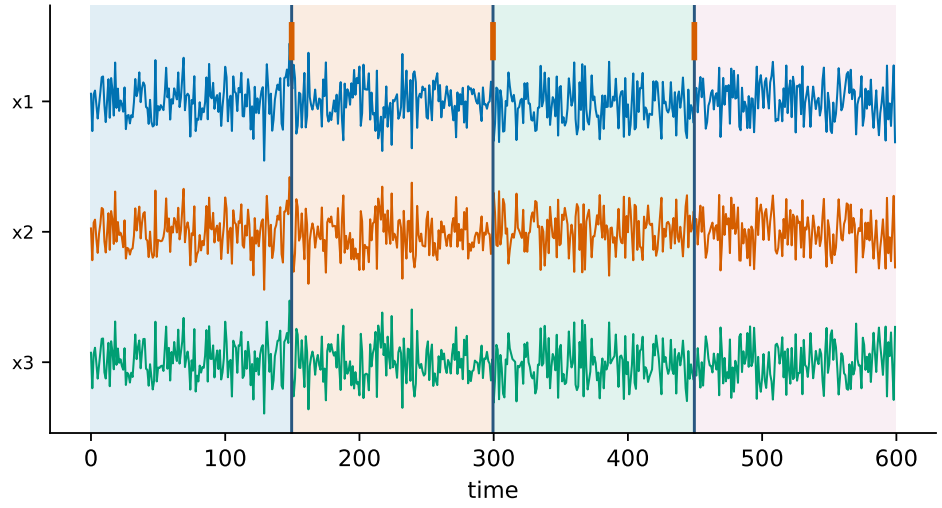


Figure 1.4. A first segmentation. Three variables share one latent factor whose loadings switch three times while the levels and variances stay the same. The PCA segmentation (blue lines, shaded segments) finds the true change points (orange marks at the top) exactly. Produced by `examples/01_basics/plot_01_first_steps.py`.

The figures of the book, like fig. 1.4, come from such scripts; each ends with a short study of its main parameter, so that the reader sees not only that a method works but also when it stops working.

In a nutshell The book in one paragraph

Multivariate time series carry their information in how the variables move together. The book answers three questions about them – where they change (segmentation), how similar they are under time distortions (warping) and which belong together (clustering) – with methods developed at the University of Pannonia and the best methods of the literature, all implemented in `mvtsa` with a shared `Segmentation` object, interchangeable costs and searches, a distance registry, and layers for evaluation, explanation and model operations.

1.6 Related software

Several Python libraries cover parts of the field. `ruptures` implements the cost-and-search framework of offline change point detection reviewed by Truong, Oudre and Vayatis [36]; `tslearn` [37] and `aeon` [38] provide elastic distances, barycenters and clustering; `stumpy` [39] computes matrix profiles. `mvtsa` differs in three ways: it implements the latent-variable segmentation and correlation-based warping methods that are not available elsewhere, it treats segmentation, warping and clustering as one connected toolbox with a shared segmentation object and distance registry, and it adds the evaluation, explanation and operation layers that turn an analysis into a maintained model. Segmentations convert to and from the formats of `ruptures`, and the optional `interop` extra installs these libraries for comparison.

CHAPTER

2



Foundations

This chapter fixes the notation and the three objects that the rest of the book builds on. A *segmentation* is a partition of the time axis into contiguous segments, found by minimizing a sum of segment costs; a *warping path* is a monotone alignment of two time axes, found by dynamic programming; a *PCA model* describes the correlation structure of a set of samples and measures, by the T^2 and Q statistics, how well a sample fits it. The chapter closes with the preprocessing steps and the conventions of the API.

2.1 Series, collections and notation

A multivariate time series is a matrix $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_T]^\mathsf{T} \in \mathbb{R}^{T \times n}$; row $\mathbf{x}_t$ is the sample at time t, column j the trajectory of variable j. In the package a series is a NumPy array of shape (T, n); a one-dimensional array is a univariate series and is treated as $(T, 1)$. A *collection* $C = \{\mathbf{X}^{(1)}, \dots, \mathbf{X}^{(N)}\}$ is a list of series with the same variables and possibly different lengths T_i. Table 2.1 summarizes the notation, which is the same in the docstrings, the documentation and this book.

2.2 Segmentations

A segmentation S of a series of length T into c segments is defined by the change points $0 = \tau_0 < \tau_1 < \cdots < \tau_{c-1} < \tau_c = T$; segment k holds the samples $\mathbf{x}_t$, $\tau_{k-1} \le t < \tau_k$. Change points are thus the first samples of new segments, and the boundaries 0 and T are never listed. Most segmentation methods solve, exactly or approximately, the problem

$$\min_{c,\ \tau_1 < \cdots < \tau_{c-1}} \sum_{k=1}^{c} \ell(\tau_{k-1}, \tau_k) \ + \ \beta\,(c-1), \tag{2.1}$$

Table 2.1. Notation of the book and the package.

Symbol	Meaning	Code
$\mathbf{X} \in \mathbb{R}^{T \times n}$, $\mathbf{x}_t$	series, sample at time t	`X`, `X[t]`
T, n	length, number of variables	`n_samples`, `n_variables`
C, N	collection, number of series	list of arrays
$\tau_1 < \cdots < \tau_{c-1}$	change points (first sample of a new segment)	`change_points`
c, $[\tau_{k-1}, \tau_k)$	number of segments, segment k ($\tau_0 = 0$, $\tau_c = T$)	`n_segments`, `segments`
$\ell(a, b)$, β	cost of segment $[a, b)$, penalty per change point	`cost.error(a, b)`, `penalty`
p	number of principal components	`n_components`
$\mathbf{U}_p$, $\mathbf{\Lambda}_p$	loadings and eigenvalues of a PCA model	`components_`, `explained_variance_`
T_t^2, Q_t	Hotelling statistic, reconstruction error of $\mathbf{x}_t$	`t2_statistic`, `q_statistic`
s_{PCA}	Krzanowski similarity of two PCA models	`krzanowski_similarity`
L, λ	lags of dynamic PCA, forgetting factor	`lags`, `forgetting`
$\mathbf{C}$, π, $D(i, j)$	local cost matrix, warping path, accumulated cost	`cost_matrix`, `path`, `accumulated`
w, γ	warping window, soft-minimum temperature	`window`, `gamma`
$\mathbf{D}$, $\mathbf{U}$	distance matrix, fuzzy memberships	`pairwise_distances`, `memberships_`

where the cost $\ell(a, b)$ measures the inhomogeneity of the segment $[a, b)$ and the penalty β prevents the trivial solution of one segment per sample [36]. With a fixed number of segments the penalty is dropped. Three ingredients define a method: the *cost*, which decides what kind of change is detected (levels, variances, correlations, dynamics, model parameters); the *search*, which decides how the minimum is found (exactly by dynamic programming, or greedily by merging or splitting); and the *selection* of c or β (fig. 2.1). Chapter 4 discusses all three; chapter 5 contributes costs based on latent-variable models; chapter 6 adds methods that do not fit the form (2.1), such as Bayesian on-line detection and state models.

Segmentation objects. Every segmenter of the package returns an instance of the `Segmentation` class. It holds the change points, the length of the series, and optionally state labels of the segments (when segments recur as states), fuzzy memberships of the samples, scores of the change points, the total cost and metadata. Its methods convert between representations: `segments` and `lengths`, `to_labels` (the segment or state index of every sample) and `from_labels`, `to_frame` (a ta-

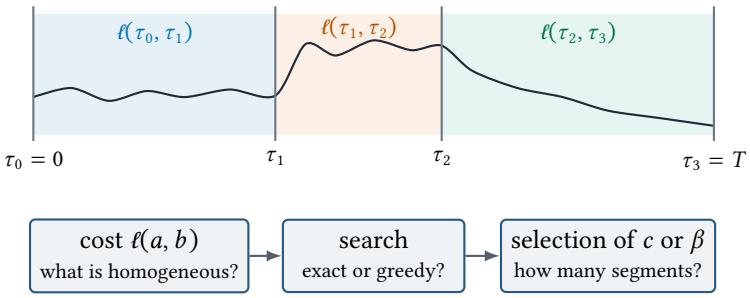


Figure 2.1. A segmentation into $c = 3$ segments by the change points τ_1, τ_2, and the three ingredients that define a segmentation method: the segment cost, the search strategy and the rule that selects the number of segments.

ble of the segments), `to_dict` and `from_dict` for files, and `to_ruptures`/`from_ruptures` for the breakpoint convention of `ruptures`.

2.3 Warping paths

Two series $\mathbf{X} \in \mathbb{R}^{T_x \times n}$ and $\mathbf{Y} \in \mathbb{R}^{T_y \times n}$ are compared under time distortions by aligning their samples. A *warping path* is a sequence of index pairs $\pi = ((i_1, j_1), \dots, (i_K, j_K))$ with $(i_1, j_1) = (1, 1)$, $(i_K, j_K) = (T_x, T_y)$ (boundary condition) and steps $(i_{k+1} - i_k,\ j_{k+1} - j_k) \in \{(1, 0), (0, 1), (1, 1)\}$ (continuity and monotonicity). With the local cost matrix $\mathbf{C}$, $c_{ij} = \|\mathbf{x}_i - \mathbf{y}_j\|^2$, dynamic time warping (DTW) finds the path of minimal total cost by the recursion

$$D(i, j) = c_{ij} + \min\{D(i-1, j),\ D(i, j-1),\ D(i-1, j-1)\}, \tag{2.2}$$

with $D(0, 0) = 0$ and infinite values outside the matrix; the distance is $\mathrm{DTW}(\mathbf{X}, \mathbf{Y}) = D(T_x, T_y)$, computed in $O(T_x T_y)$ time [40]. Figure 2.2 shows the cost matrix, the path and the alignment it induces. A global constraint restricts the path to a band around the diagonal [40, 41]; other step patterns, penalties, soft minima and costs between segments instead of samples give the family of elastic measures of chapters 7 and 8.

2.4 PCA models and their statistics

PCA describes the correlation structure of a set of samples. For standardized samples $\mathbf{x}_t$ with covariance matrix $\Sigma = \mathbf{U}\Lambda\mathbf{U}^\top$, the first p eigenvectors $\mathbf{U}_p$ span the subspace of the largest variance. The projection $\mathbf{y}_t = \mathbf{U}_p^\top \mathbf{x}_t$ gives the scores and $\hat{\mathbf{x}}_t = \mathbf{U}_p \mathbf{y}_t$ the reconstruction. Two statistics measure how well a sample fits the model:

$$T_t^2 = \mathbf{y}_t^\top \Lambda_p^{-1} \mathbf{y}_t, \qquad Q_t = \|\mathbf{x}_t - \hat{\mathbf{x}}_t\|^2 = \mathbf{x}_t^\top (\mathbf{I} - \mathbf{U}_p \mathbf{U}_p^\top) \mathbf{x}_t. \tag{2.3}$$

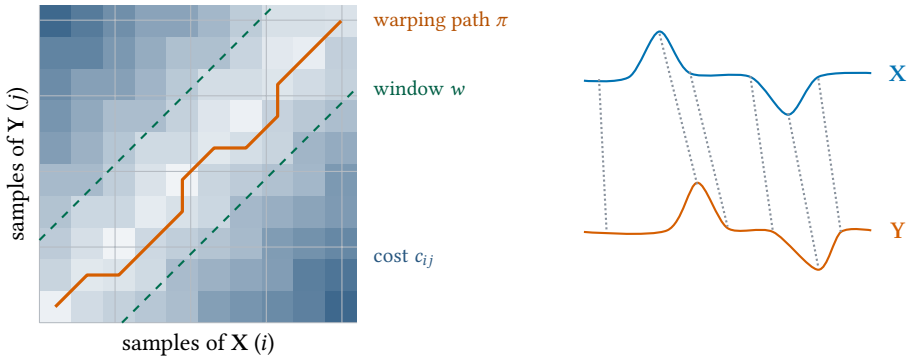


Figure 2.2. Dynamic time warping. Left: the local cost matrix (darker = larger cost), the optimal warping path and a Sakoe–Chiba window of half-width w. Right: the alignment of the samples induced by the path; the peaks and the dips are matched although they occur at different times.

The Hotelling statistic T_t^2 is the Mahalanobis distance of the sample inside the model subspace; the Q statistic, or squared prediction error, is its distance from the subspace (fig. 2.3). In multivariate statistical process control, T^2 detects unusual operation within the normal correlation structure, Q a broken correlation structure [22, 23]. The control limit of T^2 follows from an F distribution and that of Q from the approximation of Jackson and Mudholkar [42]; `PCAModel.control_limits` computes both. Figure 2.4 shows the statistics of a model fitted on the first regime of a series whose correlation structure changes in the middle: Q exceeds its limit for most samples after the change.

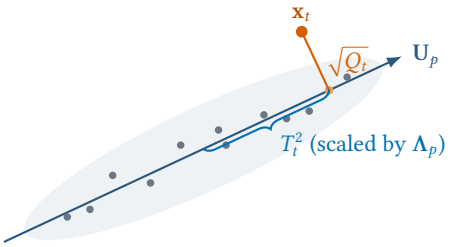


Figure 2.3. The two statistics of a PCA model. T_t^2 measures the position of the projection of $\mathbf{x}_t$ inside the model subspace (scaled by the variances of the components), Q_t the squared distance from the subspace. A change of the correlation structure shows up mainly in Q.

```
model = PCAModel(n_components=1).fit(X[:300])  # first regime as reference
t2_lim, q_lim = model.control_limits(alpha=0.01)
q = model.q_statistic(X)
print(
    f"Q limit {q_lim:.3f}; exceedance before/after the change: "
    f"{np.mean(q[:300] > q_lim):.2f} / {np.mean(q[300:] > q_lim):.2f}"
)
```

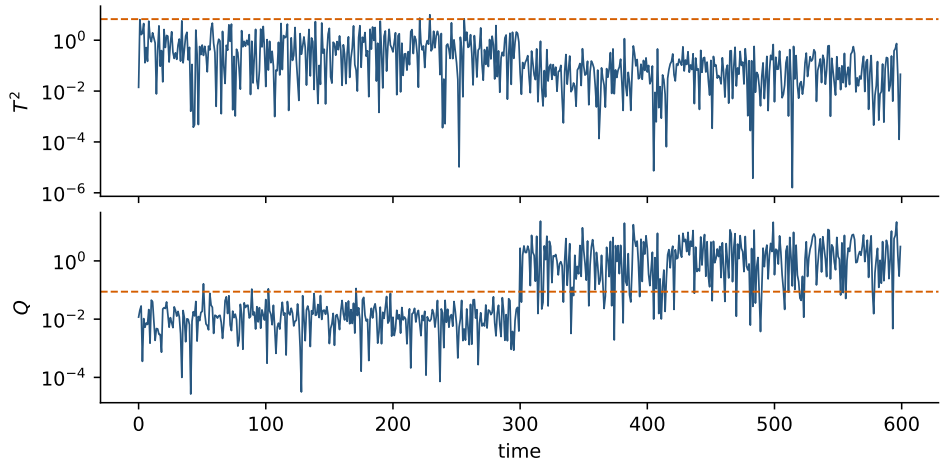


Figure 2.4. The T^2 and Q statistics of a PCA model fitted on the first 300 samples, with their 99 % control limits. The correlation structure changes at sample 300; Q exceeds its limit for most of the later samples, while T^2 hardly reacts. Produced by `examples/01_basics/plot_03_pca_models.py`.

Comparing PCA models.

Two models with loadings $\mathbf{U}_p$ and $\mathbf{V}_p$ are compared by the Krzanowski similarity factor [21]

$$s_{\mathrm{PCA}}(\mathbf{U}_p, \mathbf{V}_p) = \frac{1}{p}\operatorname{tr}\left(\mathbf{U}_p^{\mathsf{T}}\mathbf{V}_p\mathbf{V}_p^{\mathsf{T}}\mathbf{U}_p\right) = \frac{1}{p}\sum_{i=1}^{p}\sum_{j=1}^{p}\cos^2\theta_{ij}, \tag{2.4}$$

where θ_{ij} are the angles between the loading vectors; $s_{\mathrm{PCA}} = 1$ for identical subspaces and 0 for orthogonal ones. Variants weight the angles by the eigenvalues (the λ-similarity of Johannesmeyer, Singhal and Seborg [43, 44]), use the eigenvalue-weighted Eros measure of Yang and Shahabi [45], or turn the principal angles into the Grassmann and chordal distances. The package implements all of them (`krzanowski_similarity`, `lambda_similarity`, `eros_similarity`, `grassmann_distance`, `chordal_distance`); the dissimilarity $1 - s_{\mathrm{PCA}}$ is the local cost of correlation-based warping (chapter 8).

```
m1, m2 = PCAModel(1).fit(X[:300]), PCAModel(1).fit(X[300:])
print(
    "s_PCA(regime 1, regime 1 half) =",
    round(m1.similarity(PCAModel(1).fit(X[:150])), 3),
)
print("s_PCA(regime 1, regime 2)      =", round(m1.similarity(m2), 3))
print(
    "chordal distance               =",
    round(chordal_distance(m1.components_, m2.components_), 3),
)
```

In a nutshell PCA models

A PCA model with p components describes the correlation structure of a set of samples by a subspace. The Hotelling statistic T_t^2 measures variation inside the subspace, the Q statistic the squared distance from it; their control limits come from the F distribution and the Jackson–Mudholkar approximation. Two models are compared by the Krzanowski similarity, the mean squared cosine of the angles between their loadings. In `mvtsa`: `PCAModel`, `PPCAModel`, `RecursivePCA`, `krzanowski_similarity` and relatives.

2.5 Preprocessing

Most methods of the book assume that the variables are comparable in scale. `zscore`, `robust_scale` and `minmax` standardize a series, `Standardizer` learns the statistics on one series and applies them to others (which matters on line). `resample` and `resample_collection` bring series to a common length for lock-step measures; `fill_missing` interpolates gaps; `moving_average`, `smooth` and `kalman_smoother` reduce noise – the Kalman smoother was the key to segmenting noisy process transitions in [3]. `derivative` and `keogh_derivative` produce the derivatives used by derivative DTW, and `detrend` removes trends.

Lag matrices. Dynamic relations among variables become static relations among lagged variables. The lag matrix with L lags,

$$\mathbf{X}_L = \begin{bmatrix} \mathbf{x}_{L+1}^\mathsf{T} & \mathbf{x}_L^\mathsf{T} & \cdots & \mathbf{x}_1^\mathsf{T} \\ \vdots & \vdots & & \vdots \\ \mathbf{x}_T^\mathsf{T} & \mathbf{x}_{T-1}^\mathsf{T} & \cdots & \mathbf{x}_{T-L}^\mathsf{T} \end{bmatrix} \in \mathbb{R}^{(T-L)\times n(L+1)}, \tag{2.5}$$

is the input of dynamic PCA [46] (`dynamize`); `delay_embedding` builds the analogous embedding of a univariate series, with the delay and the dimension chosen by the mutual information [47] and the false nearest neighbours [48] (`ami_lag`, `false_nearest_neighbors`).

2.6 Conventions of the API

All estimators follow the same rules. Parameters are constructor arguments; `fit` learns and returns the estimator; learned attributes end with an underscore. Segmenters have `fit(X)`, which sets `segmentation_`, and the shortcut `segment(X)`, which returns it; the registry `SEGMENTERS` and the function `segment(X, method, **params)` make every segmenter available by name. Costs are objects with `fit(X)` and `error(a, b)`; a cost can be passed to any search by name (`"l2"`, `"normal"`, `"pca"`, …) or as an instance. Distances are functions `f(x, y, **params)` collected in the registry `DISTANCES`; `pairwise_distances` computes

the distance matrix of a collection with any registered measure or with a fitted segment-level model such as `CBDTW`. Clusterers have `fit` and `fit_predict` and set `labels_`. Every object supports `get_params`, `set_params` and the configuration functions `to_config` and `from_config` of chapter 12. Random behaviour is controlled by `random_state`.

CHAPTER

3



Representations

> Segmentation, warping and clustering rarely work on the raw samples alone. This chapter collects the representations the later chapters build on: piecewise approximations that compress a series into segments, symbolic words, the qualitative episodes of trend analysis, feature vectors and path signatures that summarize a whole series, networks and images that expose its structure, and matrix profiles that index its repeating subsequences. Each representation answers a different question about a series, and each has a role in a later method.

3.1 Piecewise approximations

A piecewise approximation replaces a series by c segments and one simple model per segment. The piecewise aggregate approximation (PAA) uses frames of equal length and their means [49, 50]; the adaptive piecewise constant approximation (APCA) lets the segment lengths vary [51]; the piecewise linear approximation (PLA) fits a line to every segment, which here is found by bottom-up segmentation with the linear cost of chapter 4; perceptually important points (PIP) keep the samples that deviate most from the line between their already selected neighbours [52, 53]. PLA is the representation of the MPLA, SPLA and MDPLA measures of section 8.4: every segment is described by its mean and by the angle of its slope [7].

```
p = rp.pla(x, n_segments=12)
print("slope angles (degrees):", np.degrees(p.angles[:, 0]).round(0))
```

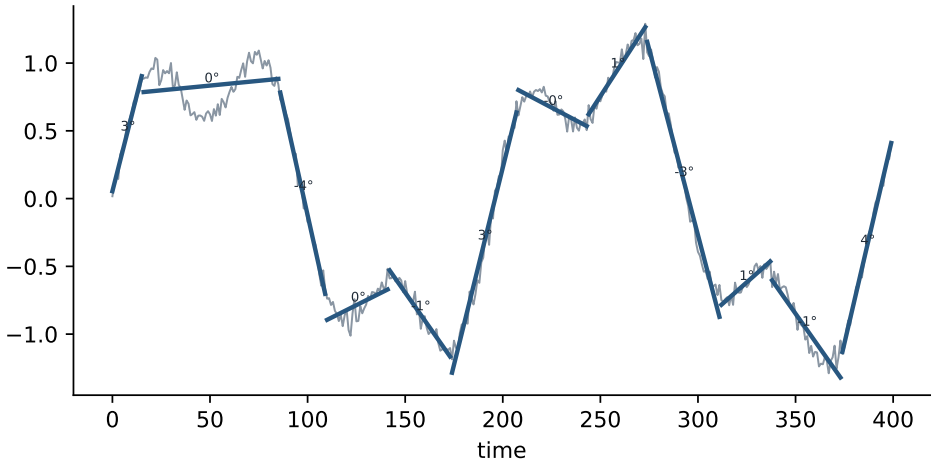


Figure 3.1. Piecewise linear approximation of a noisy two-tone signal with twelve segments found by bottom-up segmentation. Produced by `examples/01_basics/plot_02_representations.py`.

3.2 Symbolic representations

Symbolic representations turn a series into a word over a small alphabet, which makes string algorithms, hashing and indexing applicable. SAX z-normalizes the series, computes its PAA, and maps every frame mean to a letter by breakpoints that cut the standard normal distribution into equiprobable intervals [54, 55]; the MINDIST between two words lower-bounds the Euclidean distance of the series. ABBA compresses the series into linear pieces with a tolerance and clusters the pieces by their length and increment, so that the alphabet adapts to the data [56]. SFA quantizes the leading Fourier coefficients instead of frame means [57]. `sax`, `abba` and `sfa` implement the three.

```
print("SAX word:", "".join(rp.sax(x, n_segments=16, alphabet_size=5)))
```

3.3 Qualitative episodes

Process operators describe trends qualitatively: *the temperature rose faster and faster, then levelled off*. Cheung and Stephanopoulos formalized such descriptions with seven triangular primitives defined by the signs of the first and second derivatives [58]: A (increasing, concave), B (decreasing, concave), C (decreasing, convex), D (increasing, convex), E (linear increase), F (linear decrease) and G (steady) (fig. 3.2). A signal is smoothed, its derivatives are estimated, and maximal runs of equal sign patterns become episodes, each with a duration and a magnitude. Balaskó, Bankó and Abonyi compared process trends by aligning such episode

strings [9, 10]; section 8.5 shows the alignment. `triangular_episodes` extracts the episodes, `episode_string` writes them as letters (fig. 3.3).



Figure 3.2. The seven triangular primitives of qualitative trend analysis [58], defined by the signs of the first and second derivatives of the smoothed signal.

```
eps = rp.triangular_episodes(x, sigma=6)
print("episodes:", rp.episode_string(eps))
```

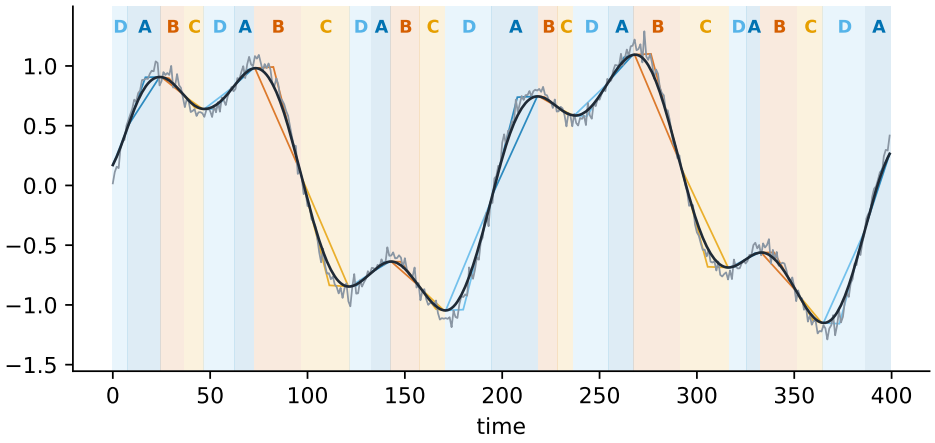


Figure 3.3. Qualitative episodes of the signal of fig. 3.1: the smoothed signal, the episodes coloured by primitive, and their triangles. The smoothing width decides how many wiggles survive (parameter study of the script). Produced by `examples/01_basics/plot_02_representations.py`.

3.4 Features and path signatures

A feature vector summarizes a whole series: moments, autocorrelations, spectral and entropy measures, trend and crossing statistics. Feature-based methods are fast, robust to different lengths and interpretable; large feature libraries and their compact selections, such as hctsa and catch22, show how far this goes [59, 60]. `extract_features` computes seventeen features per variable, in the spirit of catch22 – moments, autocorrelation at the first lag and at its first zero crossing, the first minimum of the autocorrelation, permutation entropy [61], spectral centroid and entropy, trend, mean crossings, longest monotone run and others – and

the pairwise correlations of the variables; `feature_matrix` stacks them for a collection.

The *signature* of a path is the sequence of its iterated integrals,

$$S(\mathbf{X})^{i_1 \cdots i_k} = \int_{0<t_1<\cdots<t_k<T} \mathrm{d}x_{t_1}^{i_1} \cdots \mathrm{d}x_{t_k}^{i_k}, \tag{3.1}$$

which characterizes the path up to reparametrization and is the natural feature map of multivariate streams in rough path theory [62, 63]. The first level holds the increments, the second level the signed areas between pairs of variables – a measure of which variable leads which. For piecewise linear paths the signature is computed exactly from Chen's identity, segment by segment; `signature_features` returns the levels up to depth three, optionally with time as an extra channel and with the lead–lag transformation.

3.5 Networks and images

A series can be turned into a network whose structure reflects its dynamics. In the natural visibility graph two samples are linked if the straight line between them passes above all samples in between [64]; in the horizontal version the line must be horizontal [65]. The degree distribution of the horizontal visibility graph of an uncorrelated series is known exactly, so departures from it measure correlation and nonlinearity. Labelling the nodes with the zones of a Shewhart control chart gives the six-sigma visibility graph of section 11.4 [20]. The recurrence network links samples whose delay vectors are closer than ε [66]; its adjacency matrix is the recurrence plot [67]. The ordinal partition network has the ordinal patterns of the delay vectors as nodes and their transitions as edges [68, 69]. Gramian angular and Markov transition fields encode a series as an image [70]. Figure 3.4 shows a recurrence plot of a series that switches from a periodic to a random-walk regime.

3.6 Matrix profiles

The matrix profile of a series stores, for every subsequence of length w, the distance to its nearest neighbour elsewhere in the series and the index of that neighbour [26]; the STOMP algorithm computes it in $O(T^2)$ time by updating dot products along diagonals [71]. Low values mark motifs, high values discords, and the arcs between subsequences and their neighbours reveal the regime structure: few arcs cross a regime boundary. FLUSS turns this observation into a segmentation [27], and ClaSP uses the k-nearest-neighbour profile [28]; both are discussed in section 6.3. `matrix_profile` and `knn_profile` compute the z-normalized (shape) or plain Euclidean (level-sensitive) profiles; `auto_window` estimates the subsequence width from the autocorrelation.

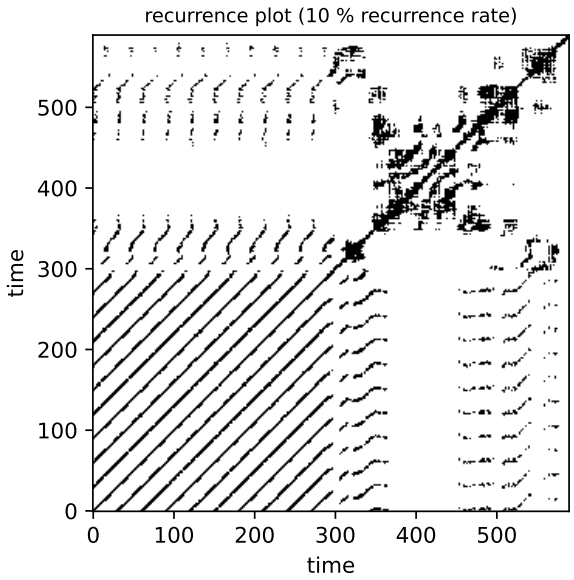


Figure 3.4. Recurrence plot (10 % recurrence rate) of a series that is periodic in its first half and a random walk in its second half: diagonal lines mark the periodic regime, a blocky texture the drifting one. Produced by `examples/09_visualization/plot_03_recurrence_images.py`.

In a nutshell Representations

Piecewise approximations (PAA, APCA, PLA, PIP) compress a series into segments; symbolic words (SAX, ABBA, SFA) make string methods applicable; qualitative episodes describe trends in the language of operators; features and path signatures summarize whole series; visibility, recurrence and ordinal networks and image encodings expose dynamics; matrix profiles index repeating subsequences. PLA and episodes are the representations of the segment-level warping methods of chapter 8, features and signatures of feature-based clustering, matrix profiles of profile-based segmentation.

CHAPTER

4



Cost-based segmentation

Most segmentation methods minimize a sum of segment costs, eq. (2.1). This chapter takes the three ingredients apart. The *cost* decides which changes are seen: shifts of the level, the variance, the correlation, the trend, the dynamics or the relation between variables. The *search* decides how the minimum is found: exactly by dynamic programming, or greedily by merging or splitting segments. The *selection* decides how many segments there are. Bottom-up merging receives special attention: one run produces a whole hierarchy of segmentations, which the package exposes as a merge tree with a segmentogram, merge persistence and the relative reduction rate of the cost.

4.1 Segment costs

A cost $\ell(a, b)$ measures how badly one model fits the samples of the segment $[a, b)$. Table 4.1 lists the costs of the package. Every cost precomputes what it can on the whole series in `fit`, so that `error(a, b)` is cheap: the L2 and linear costs need only cumulative sums of the samples and of their products with time, the Gaussian and PCA costs cumulative sums of the outer products $\mathbf{x}_t \mathbf{x}_t^\mathsf{T}$, from which the covariance of any segment, and hence its eigenvalues, follows in $O(n^2)$ plus an eigendecomposition.

The kernel cost with a Gaussian kernel compares segment distributions in a reproducing kernel Hilbert space [72, 73]; for long series it can use random Fourier features. The PCA cost, which sums the Q or the (unscaled) T^2 statistics of the samples under the local PCA model of the segment, is the subject of chapter 5; its two parts add up to the L2 cost. The regression cost is the model-based cost of the energy-targeting segmentation of Abonyi, Kulcsár, Balaton and Nagy [16]: a segment is homogeneous if one linear model predicts the key indicator from the operating variables. The example below compares five costs on a series with

Table 4.1. Segment costs. $\bar{\mathbf{x}}_{ab}$ and $\hat{\Sigma}_{ab}$ are the mean and covariance of the segment, λ_i its eigenvalues in decreasing order, k a kernel.

Cost	$\ell(a, b)$	detects
`"l2"`	$\sum_{t=a}^{b-1} \lVert \mathbf{x}_t - \bar{\mathbf{x}}_{ab} \rVert^2$	level
`"l1"`	$\sum_t \lVert \mathbf{x}_t - \mathrm{med}_{ab} \rVert_1$	level, robustly
`"normal"`	$(b-a) \log\det \hat{\Sigma}_{ab}$	level, variance, correlation
`"linear"`	residual sum of squares of a line in t per variable	trend
`"ar"`	residual sum of squares of an AR(q) model per variable	dynamics (univariate)
`"kernel"`	$\sum_t k(\mathbf{x}_t, \mathbf{x}_t) - \frac{1}{b-a} \sum_{s,t} k(\mathbf{x}_s, \mathbf{x}_t)$	distribution
`"pca"`	$(b-a) \sum_{i>p} \lambda_i \, (Q)$, $(b-a) \sum_{i \le p} \lambda_i \, (T^2)$	correlation structure
`"regression"`	residual of regressing the targets on the other variables	relation between variables

changes of the level, the variance and the correlation: the Gaussian and kernel costs see every change, the L2 and linear costs only the level shifts; the PCA cost reacts to the correlation changes and finds every boundary, because each of them changes the correlations as well.

```
rows = []
for cost in ("l2", "normal", "linear", "kernel", "pca"):
    seg = sg.BottomUp(cost, n_segments=5, initial_size=10).segment(X)
    rows.append(
        {"cost": cost, **evaluate_segmentation(cps, seg, len(X),
    margin=10)}
    )
print(pd.DataFrame(rows)[["cost", "f1", "covering", "hausdorff"]])
```

4.2 Search strategies

Exact search. For a fixed number of segments, (2.1) is solved exactly by the dynamic program of segment neighbourhoods [74]: with $F_k(t)$ the minimal cost of $[0, t)$ in k segments, $F_k(t) = \min_{s<t}\{F_{k-1}(s) + \ell(s, t)\}$, which costs $O(cT^2)$ evaluations of ℓ (`OptimalPartition`). With a penalty instead of a fixed c, optimal partitioning computes $F(t) = \min_{s<t}\{F(s) + \ell(s, t) + \beta\}$; PELT prunes every candidate s that can never become optimal again, $F(s) + \ell(s, t) > F(t)$, and runs in linear expected time when the number of change points grows with T [24] (`PELT`). The default penalty is of BIC type, the number of segment parameters times $\log T$ (scaled by a robust noise variance for least-squares costs); `penalty_path` traces the number of change points over a range of penalties, in the spirit of CROPS [75].

Greedy searches. Top-down or binary segmentation splits recursively at the point that decreases the cost most [76] (`TopDown`). Bottom-up segmentation starts

Algorithm 4.1 Bottom-up segmentation with a merge tree

Input: series $\mathbf{X}$, cost ℓ, initial segment length m, stopping rule (number of segments c, penalty β or maximal cost)
Output: segmentation $\mathcal{S}$ and merge tree (removed change points, merge criteria, total costs)

cut $[0, T)$ into segments of length m;
for *every pair of adjacent segments* S_k, S_{k+1} **do**
  $\Delta_k \leftarrow \ell(S_k \cup S_{k+1}) - \ell(S_k) - \ell(S_{k+1})$; `// merge cost`
while *more than one segment remains* **do**
  $k^* \leftarrow \arg\min_k \Delta_k$; record the removed change point, Δ_{k^*} and the new total cost;
  merge S_{k^*} and S_{k^*+1}; recompute Δ_{k^*-1} and Δ_{k^*};
  if *the stopping rule holds and no full tree is requested* **then break**;
return the segmentation at the requested level and the merge tree;

from many small segments and repeatedly merges the adjacent pair whose union increases the cost least [77, 78] (`BottomUp`, algorithm 4.1). The sliding window grows a segment until its cost exceeds a threshold (`SlidingWindow`); SWAB combines a sliding buffer with bottom-up merging inside it for on-line use [77] (`SWAB`). The window discrepancy method scores every position by the cost reduction of splitting a window centred on it and picks the peaks (`WindowDiscrepancy`). Figure 4.1 compares the strategies with the Gaussian cost.

```
searches = {
    "bottom-up": sg.BottomUp("normal", n_segments=5, initial_size=10),
    "top-down": sg.TopDown("normal", n_segments=5, jump=5),
    "optimal": sg.OptimalPartition("normal", n_segments=5, jump=10),
    "PELT": sg.PELT("normal", penalty=250, jump=5, min_size=20),
    "window": sg.WindowDiscrepancy("normal", width=40, n_segments=5),
}
for name, s in searches.items():
    seg = s.segment(X)
    print(
        f"{name:10s}",
        seg.change_points,
        round(evaluate_segmentation(cps, seg, len(X), 10)["covering"], 3),
    )
```

Bottom-up merging is the workhorse of this book. It is simple, fast with a heap of merge costs, and it works with *any* cost, including costs that have no cumulative-sum shortcut and dissimilarities between adjacent segments that are not costs at all (section 6.4). Its initial segment length matters: segments must hold enough samples for the model of the cost – for a PCA model of n variables with L lags, at least a few times $n(L+1)$.

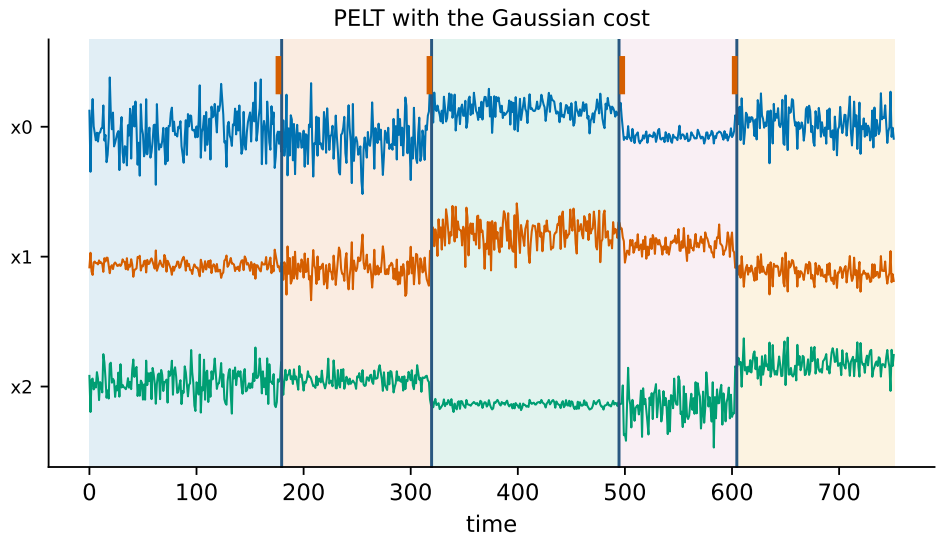


Figure 4.1. A series with changes of the level, the variance and the correlation, segmented by PELT with the Gaussian cost; true change points as orange marks at the top. Produced by `examples/02_cost_segmentation/plot_01_costs_and_search.py`.

4.3 The merge hierarchy

A bottom-up run that continues to a single segment produces every segmentation from the finest to the coarsest, each obtained from the previous one by removing one change point. The package keeps this hierarchy as a `MergeTree`: the initial change points, the change point removed by every merge, the merge criteria and the total cost after every merge. `cut(c)` returns the change points of the level with c segments, `cost_curve` the total cost as a function of c, `levels` all segmentations. Two new tools make the hierarchy readable (fig. 4.2).

Segmentogram. The segmentogram draws the levels of the hierarchy as rows – the finest at the bottom, the coarsest at the top – with the change points of every level as ticks. A change point that survives many merges forms a long vertical line; spurious boundaries disappear after a few merges.

Merge persistence. In analogy with topological persistence, every initial change point is *born* at level zero and *dies* when it is removed. Measuring the death on the scale of the merge criterion – the running maximum of the merge costs, so that the levels form a filtration – makes the persistence comparable across change points: a boundary between genuinely different segments is removed only at a high merge cost. The change points ranked by persistence give the most important boundaries first, and a large gap in the logarithmic persistences suggests the number of segments.

```
bu = sg.BottomUp("pca", initial_size=10, cost_params={"n_components": 1})
tree = bu.full_tree(X)
print("levels:", tree.n_levels, "| 5 segments:", tree.cut(5), "| true:",
    cps)
pers = tree.persistence()
print(
    "most persistent change points:",
    sorted(pers, key=pers.get, reverse=True)[:4],
)
```

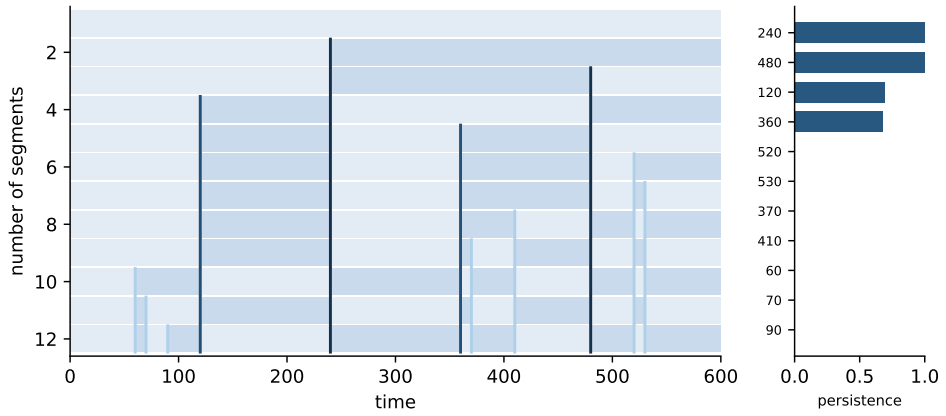


Figure 4.2. Segmentogram of a bottom-up run with the PCA cost on a series with five correlation regimes (rows: levels with 2 to 12 segments) and the persistence of the change points (right). The four true boundaries survive to the top; the others vanish within a few merges. Produced by `examples/02_cost_segmentation/plot_02_segmentogram.py`.

4.4 How many segments?

The total cost decreases monotonically with the number of segments, so it cannot choose c by itself. The package offers five rules (`select_n_segments`):

- *knee*: the point of the normalized cost curve farthest from the chord between its end points;
- *relative reduction rate*: Bankó's $RR(c) = \left(\ell_{c-1} - \ell_c\right)/\ell_{c-1}$, the relative decrease of the total cost when a segment is added [8]; the number of segments is the last c before the reduction falls below a threshold (fig. 4.3);
- *BIC*: $T \log(\ell_c/T) + k\, c \log T$ for Gaussian errors with k parameters per segment;
- *permutation test*: the cost reduction from $c-1$ to c segments is compared with the reductions after permuting the samples inside the segments of the coarser level, which keeps the coarse structure and destroys the finer one; c grows while the reduction is significant [79];

- *persistence*: the largest gap of the logarithmic merge persistences (section 4.3).

They agree when the segments are clearly different and disagree otherwise, which is itself informative; the stability analysis of section 10.4 provides an independent check.

```
for method, kw in [
    ("knee", {}),
    ("rr", {"rr_threshold": 0.05}),
    ("bic", {}),
    ("persistence", {}),
    ("permutation", {"n_permutations": 19, "random_state": 0}),
]:
    n, info = sg.select_n_segments(
        X, "l2", method, max_segments=12, initial_size=5, **kw
    )
    print(f"{method:12s} -> {n} segments")
print("true number of segments:", len(cps) + 1)
```

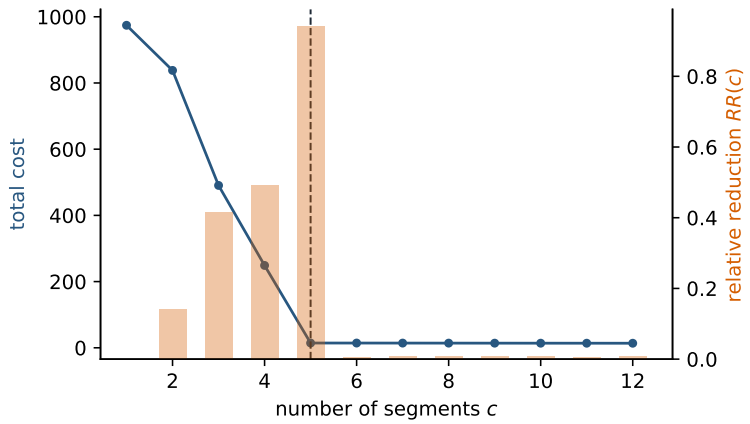


Figure 4.3. Total cost (line) and relative reduction rate $RR(c)$ (bars) of the hierarchy of fig. 4.2. The reduction rate collapses after $c = 5$, the true number of segments. Produced by `examples/02_cost_segmentation/plot_02_segmentogram.py`.

In a nutshell **Cost-based segmentation**

A segmentation method combines a cost, a search and a selection rule. Costs decide what is homogeneous: levels (L2, L1), distributions (Gaussian, kernel), trends (linear), dynamics (AR), correlation structure (PCA), relations between variables (regression). Exact searches (optimal partitioning, PELT) minimize (2.1); greedy ones (top-down, bottom-up, sliding window, SWAB) approximate it quickly and work with any cost. A full bottom-up run yields a merge tree: its segmentogram, the persistence of its change points and the relative reduction rate of its cost curve support the choice of the number of segments, together with the knee, BIC and permutation-test rules. In `mvtsa`: `get_cost`, `BottomUp`, `TopDown`, `OptimalPartition`, `PELT`, `SWAB`, `MergeTree`, `select_n_segments`, `penalty_path`.

CHAPTER

5



Latent-variable segmentation

This chapter is the core of the group's contribution to segmentation. A segment is homogeneous if one latent-variable model – a PCA model, a dynamic PCA model of the lagged variables, or a probabilistic PCA model inside a fuzzy cluster – describes all of its samples. We start with the PCA costs of bottom-up segmentation, extend them to dynamics with dynamic PCA, run them on line with recursive PCA and a variable forgetting factor, make the segments fuzzy with the modified Gath–Geva algorithm, and finally segment historical data by the information they carry for parameter estimation. Each method answers a question that the costs of chapter 4 cannot.

5.1 PCA segmentation

Abonyi, Feil, Németh and Árva observed that the homogeneity of a segment of process data is best measured by the hidden structure of the variables rather than by their means [1, 2]: a segment is homogeneous if a PCA model with few components reconstructs its samples well. Bankó made this the cost of bottom-up segmentation [8]. For a segment S with covariance eigenvalues $\lambda_1 \geq \cdots \geq \lambda_n$ and a local PCA model of p components,

$$\ell_Q(S) = \sum_{t \in S} Q_t = |S| \sum_{i=p+1}^{n} \lambda_i, \qquad \ell_{T^2}(S) = \sum_{t \in S} \tilde{T}_t^2 = |S| \sum_{i=1}^{p} \lambda_i, \tag{5.1}$$

where $\tilde{T}_t^2 = \|\mathbf{y}_t\|^2$ is the unscaled Hotelling statistic used by Bankó, the variation inside the model. The Q cost reacts to changes of the correlation structure, the T^2 cost to drifts within it, and their sum is the L2 cost (fig. 5.1). The aggregate can be the sum (long segments weigh more) or the mean (segments compete on equal terms). With cumulative outer-product sums, the covariance of any segment is available in $O(n^2)$ and the cost of a merge in one eigendecomposition. The number

of components is a fixed integer or a fraction of explained variance, resolved once on the whole series and then kept fixed, so that all segments are compared with models of equal size.

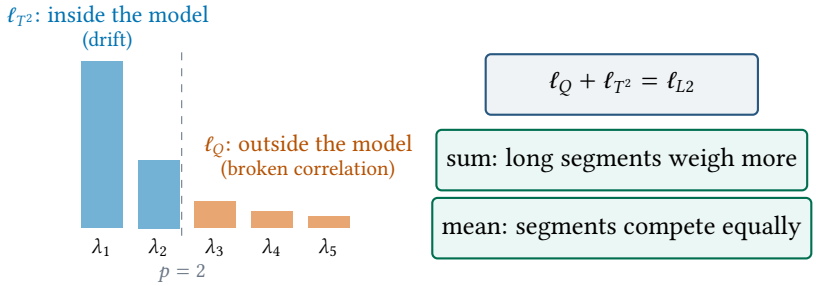


Figure 5.1. The two PCA costs of a segment are sums of the eigenvalues of its covariance: those outside the p-dimensional model (ℓ_Q) and those inside (ℓ_{T^2}). Their sum is the L2 cost.

```
rows = []
for stat in ("Q", "T2", "both"):
    seg = sg.PCASegmenter(
        n_segments=5, n_components=1, statistic=stat
    ).segment(X)
    rows.append(
        {
            "statistic": stat,
            **evaluate_segmentation(cps, seg, len(X), margin=6),
        }
    )
print(pd.DataFrame(rows)[["statistic", "f1", "covering", "hausdorff"]])
```

A closely related criterion compares adjacent segments directly instead of costing them one by one: `KrzanowskiSegmenter` merges neighbouring segments in the order of the dissimilarity $1 - s_{\mathrm{PCA}}$ of their PCA models, optionally adding a term for the distance of their means – the bottom-up form of the similarity-driven merging that Dobos and Abonyi used for Fisher matrices (section 5.5). On a series whose correlation structure switches while the variances stay equal, Q finds every change and T^2 none, as expected. The loadings of the segment models (fig. 5.2) explain the segmentation: every segment has its own first principal direction.

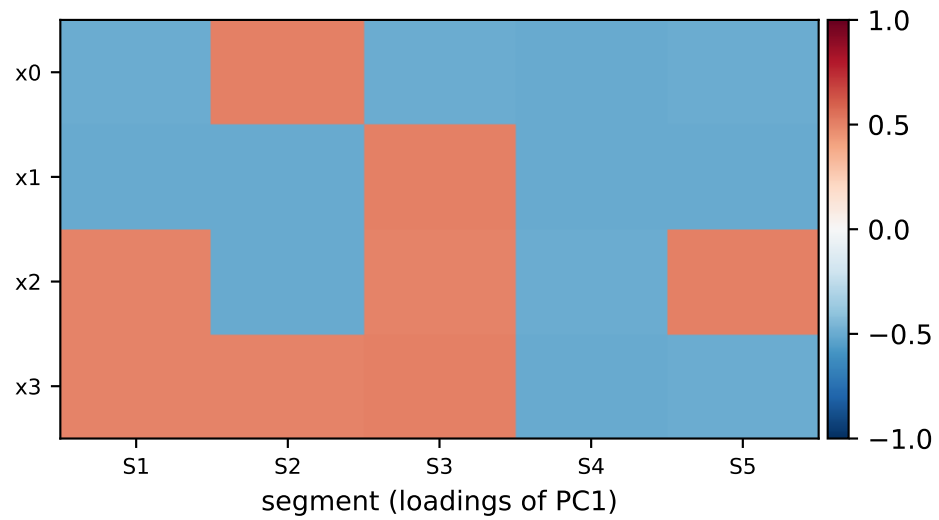


Figure 5.2. First principal direction of the PCA model of every segment found by the Q cost. The segments differ in how the four variables load on the common latent factor. Produced by `examples/03_latent_segmentation/plot_01_pca_t2_q.py`.

5.2 Dynamic PCA segmentation

When segments differ in their *dynamics* – a time constant, a damping, a dead time – while their static correlations stay similar, a static PCA model sees little. Dynamic PCA [46] fits the PCA model to the lag matrix (2.5): an exact dynamic relation such as $y_k = a_1 y_{k-1} + a_2 y_{k-2} + b\, u_{k-1}$ becomes a static relation among the columns of $\mathbf{X}_L$, i.e. a direction of (near-)zero variance, and a change of the parameters rotates this direction. Segmenting with the PCA costs of the lag matrix detects changes of the dynamics [6, 13]. Two details matter in practice. First, the initial segments must be long enough for a model of $n(L+1)$ variables; the package enforces at least $5n(L+1)$ samples. Second, the number of lags must be chosen. Ku, Storer and Georgakis count the near-zero eigenvalues of the lag-matrix covariance and increase L until no new linear relation appears; slow dynamics and step-like inputs create approximate relations to which this count is sensitive. The package therefore also offers a criterion based on the smallest eigenvalue: the ratio $\lambda_{\min}/\lambda_{\max}$ drops by orders of magnitude at the lag that completes a dynamic relation and only slightly at superfluous lags, and the selected lag is the last drop larger than a factor (`select_dpca_lag`; default factor 8).

```
static = sg.PCASegmenter(n_segments=3, n_components=1).segment(Y)
dynamic = sg.DPCASegmenter(n_segments=3, lags=2, n_components=5).segment(Y)
for name, s in [("PCA", static), ("DPCA", dynamic)]:
    print(
        f"{name:5s}",
        s.change_points,
        evaluate_segmentation(cps, s, len(Y), margin=15)["f1"],
```

```
    )
```

```
Y1, _, _ = mv.datasets.make_dynamic_switching(
    order=1, n_segments=3, length=300, random_state=0
)
print("selected lag, first-order system: ", select_dpca_lag(Y1))
print("selected lag, second-order system:", select_dpca_lag(Y))
print(
    "Ku et al. (1995) count of relations:",
    select_dpca_lag(Y1, method="ku"),
    select_dpca_lag(Y, method="ku"),
)
```

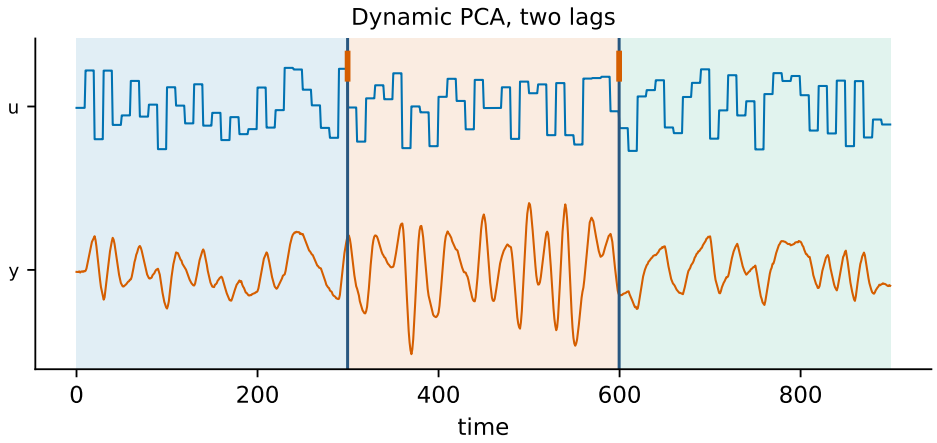


Figure 5.3. A second-order system whose damping changes twice while its static gain stays one. Static PCA misses the changes; dynamic PCA with two lags finds both exactly. Produced by `examples/03_latent_segmentation/plot_02_dynamic_pca.py`.

The pH neutralization process [80] is the classical test of this idea, used by Bankó, Dobos and Abonyi [6]: the titration curve makes the process gain change by orders of magnitude around neutrality, so the local dynamics differ between operating ranges (fig. 5.4). Section 13.2 uses the same segmentation to identify local models.

5.3 On-line segmentation with recursive PCA

Batch segmentation needs the whole record. Dobos and Abonyi developed an on-line version for monitoring [13]: a recursive (dynamic) PCA model is updated with every sample, and the Krzanowski similarity between the running model and the model of the current segment signals a change. The recursive update of the mean and the covariance with a forgetting factor λ [81] is

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1} + (1-\lambda_t)\,\mathbf{d}_t, \qquad \boldsymbol{\Sigma}_t = \lambda_t\big(\boldsymbol{\Sigma}_{t-1} + (1-\lambda_t)\,\mathbf{d}_t\mathbf{d}_t^{\mathsf{T}}\big), \qquad \mathbf{d}_t = \mathbf{x}_t - \boldsymbol{\mu}_{t-1}. \tag{5.2}$$

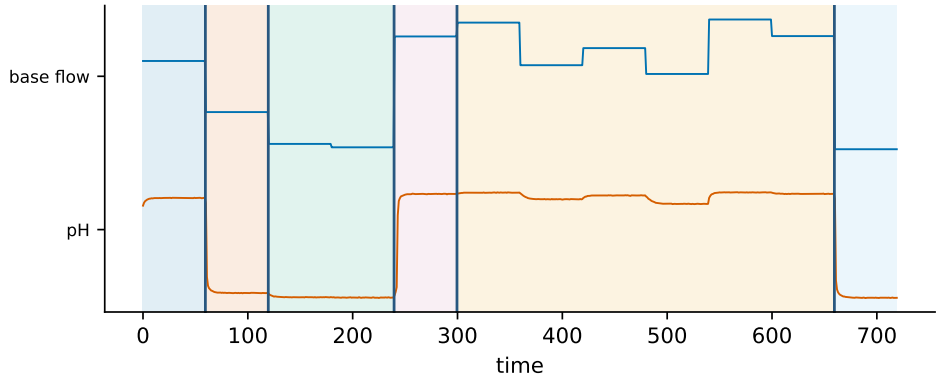


Figure 5.4. Dynamic PCA segmentation of the simulated pH neutralization process (base flow and pH). The segments follow the operating ranges of the titration curve. Produced by `examples/03_latent_segmentation/plot_02_dynamic_pca.py`.

A constant λ trades tracking speed against noise. A variable forgetting factor in the spirit of Fortescue, Kershenbaum and Ydstie [82] shortens the memory when a sample is poorly explained: with m_t the Mahalanobis distance of $\mathbf{d}_t$ under Σ_{t-1},

$$\lambda_t = \operatorname{clip}\!\left(1 - (1-\lambda_0)\, m_t / n,\ \lambda_{\min},\ \lambda_{\max}\right). \tag{5.3}$$

Algorithm 5.1 states the detector. Because the recursive model needs some samples to follow a change, the detection comes with a delay; the package optionally removes it by a retrospective search for the best two-segment split of the samples of the closing segment, which places the reported change point where the change happened rather than where it was noticed.

```
stream = sg.RecursivePCASegmenter(
    threshold=0.7,
    n_components=1,
    forgetting=0.97,
    warmup=60,
    patience=8,
    scale=(X[:200].mean(0), X[:200].std(0)),
)
for t, x in enumerate(X):
    if stream.partial_fit(x):
        print(f"sample {t}: change declared at
    {stream.change_points_[-1]}")
```

Algorithm 5.1 On-line segmentation with recursive PCA

Input: stream $\mathbf{x}_1, \mathbf{x}_2, \ldots$; threshold $s_{\min}$; components p; forgetting λ_0; warm-up w; patience h
Output: change points, declared on line

start a segment; collect w samples; fit the reference model $\mathbf{U}_{\text{ref}}$ and initialize the recursion;
for *every new sample* $\mathbf{x}_t$ **do**
  update $\boldsymbol{\mu}_t, \Sigma_t$ by (5.2) with λ_t from (5.3); $\mathbf{U}_t \leftarrow$ leading p eigenvectors of Σ_t;
  $s_t \leftarrow s_{\text{PCA}}(\mathbf{U}_{\text{ref}}, \mathbf{U}_t)$;
  if $s_t < s_{\min}$ *for* h *consecutive samples* **then**
    declare a change at the first of them (or at the best retrospective split);
    start a new segment with the samples after the change; collect up to w samples; refit $\mathbf{U}_{\text{ref}}$;

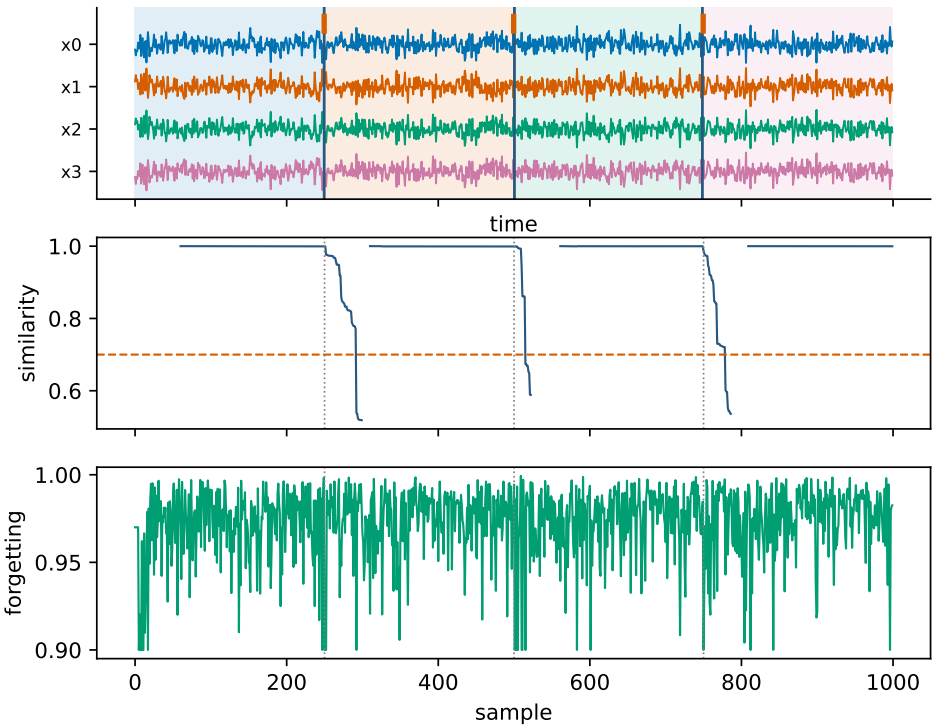


Figure 5.5. On-line segmentation with recursive PCA. Top: detected (lines) and true (marks) change points; middle: Krzanowski similarity of the running model to the segment model with the threshold; bottom: the variable forgetting factor, which drops after every change. Produced by `examples/03_latent_segmentation/plot_03_recursive_pca.py`.

Algorithm 5.2 Modified Gath–Geva fuzzy segmentation

Input: series $\mathbf{X}$; initial number of clusters c; latent dimension p; fuzzy exponent m; merge threshold κ
Output: fuzzy memberships $\mathbf{U} \in [0,1]^{T \times c'}$ and a monotone crisp segmentation

initialize the memberships with c equal, overlapping segments;
repeat
  update α_i, the time centres and spreads v_i^t, σ_i, and the PPCA models $\boldsymbol{\mu}_i, \mathbf{W}_i, \sigma_{x,i}$ from the weights u_{it}^m;
  $D_{it}^2 \leftarrow 1/(\alpha_i p(\mathbf{z}_t \mid \eta_i))$; $\quad u_{it} \leftarrow 1 \big/ \sum_j (D_{it}/D_{jt})^{2/(m-1)}$;
until *the memberships converge*;
prune clusters with small total membership;
while *two adjacent clusters have compatibility* $s_{\mathrm{PCA}}(\mathbf{W}_i, \mathbf{W}_j)\, e^{-\Delta_{ij}^2/2} \geq \kappa$ **do**
  merge them and re-estimate
crisp labels $\leftarrow$ monotone labelling of $\mathbf{U}$ by dynamic programming;

5.4 Fuzzy segmentation

Transitions between operating regimes are rarely abrupt. The modified Gath–Geva algorithm of Abonyi, Feil, Németh and Árva treats segmentation as fuzzy clustering of the time-extended samples $\mathbf{z}_t = [t, \mathbf{x}_t^\mathsf{T}]^\mathsf{T}$ [1, 2, 83]. Every cluster i is the product of a Gaussian in time, which keeps the cluster contiguous and defines a fuzzy set of the segment on the time axis, and a probabilistic PCA model of the variables [84, 85], which measures the homogeneity of the segment by its latent structure:

$$p(\mathbf{z}_t \mid \eta_i) = \mathcal{N}\big(t \mid v_i^t, \sigma_i^2\big)\; \mathcal{N}\big(\mathbf{x}_t \mid \boldsymbol{\mu}_i,\; \mathbf{W}_i\mathbf{W}_i^\mathsf{T} + \sigma_{x,i}^2 \mathbf{I}\big). \tag{5.4}$$

The memberships follow the Gath–Geva (fuzzy maximum likelihood) iteration with distances $D_{it}^2 = 1/(\alpha_i\, p(\mathbf{z}_t \mid \eta_i))$, where α_i are the cluster priors. The number of segments is chosen by compatible cluster merging: adjacent clusters with similar PPCA subspaces and overlapping time fuzzy sets are merged (algorithm 5.2). The package adds two safeguards learned from practice: clusters with negligible total membership are pruned before merging, and the crisp segmentation is read off the memberships by a dynamic program that enforces a monotone sequence of labels, which removes the flicker of the raw maximum-membership labels at the boundaries. With `lags > 0` the PPCA models use lag-extended samples, which combines the fuzzy segmentation with dynamic PCA.

```
gg = sg.GGSegmenter(
    n_segments=6, n_components=1
)  # start with more clusters than needed
seg = gg.segment(X)
print(
    "clusters after merging:",
    seg.n_segments,
    "| change points:",
    seg.change_points,
    "| true:",
    cps,
)
print(evaluate_segmentation(cps, seg, len(X), margin=10)["f1"])
```

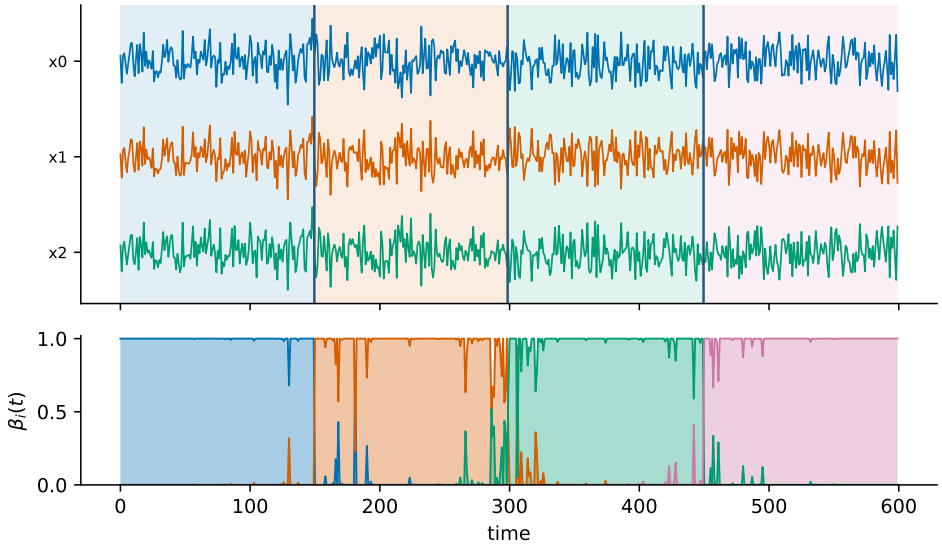


Figure 5.6. Fuzzy segmentation with the modified Gath–Geva algorithm, started with six clusters. Top: the series with the crisp segments; bottom: the fuzzy memberships, which overlap at the transitions. Compatible cluster merging reduces the six initial clusters to the four true segments. Produced by `examples/03_latent_segmentation/plot_04_fuzzy_segmentation.py`.

5.5 Segmentation by Fisher information

When the purpose of an analysis is to identify a process model, the useful parts of a historical record are those that carry information on the model parameters. Optimal experiment design measures this information by the Fisher information matrix [86]; Dobos, Bankó and Abonyi turned it into a segmentation criterion [11, 12, 14, 87, 88]. With the output sensitivities $\mathbf{S}_t = \partial \mathbf{y}_t / \partial \boldsymbol{\theta}$ computed along the

measured inputs, the information of a segment is

$$\mathbf{F}(a,b) = \frac{1}{b-a}\sum_{t=a}^{b-1} \mathbf{S}_t^{\mathsf{T}}\mathbf{Q}\,\mathbf{S}_t, \tag{5.5}$$

where $\mathbf{Q}$ is the inverse covariance of the measurement errors. Scalar OED criteria summarize it: $\log\det\mathbf{F}$ (D-criterion), the trace of $\mathbf{F}^{-1}$ (A), the smallest eigenvalue, and the condition number $\lambda_{\max}/\lambda_{\min}$, the modified E-criterion of Bernaerts et al. [89]. Two segmentation principles are implemented (`FisherSegmenter`). The *criterion* method computes a criterion in a moving window – the "evaluating time series" of Dobos et al. – and segments it with a univariate method (bottom-up L2 or PAA); the segments are then ranked by their information content. The *eigen* method compares the *directions* of the information: neighbouring segments are merged in the order of the Krzanowski similarity of the leading eigenvectors of their Fisher matrices, so that the final segments are informative about different parameter combinations [14]. Sensitivities come from the model by finite differences (`finite_difference_sensitivities`) or, for ARX models, analytically (`arx_sensitivities`).

```
names = ["kp", "kI", "kfm"]
p0 = np.array([MMA_PARAMETERS[k] for k in names])


def simulate(p):
    return simulate_mma_cstr(FI, params=dict(zip(names, p, strict=True)))


S = sg.finite_difference_sensitivities(
    simulate, p0, rel_step=1e-4
)  # (T, 1, 3), scaled
fs = sg.FisherSegmenter(
    n_segments=5, criterion="logD", window=20, initial_size=10
).fit(S)
print(fs.segmentation_.change_points)
print(fs.ranking_[["segment", "start", "stop", "logD", "E"]].round(2))
```

5.6 Model-based segmentation of energy indicators

The last latent-variable cost is a plain regression: a segment is homogeneous if one linear model predicts the target from the other variables. Abonyi, Kulcsár, Balaton and Nagy used it to segment energy monitoring data, so that every segment yields a targeting model of the key energy indicator for one operating range [16] (`CostRegression`). On data whose coefficients change while every variable moves smoothly, the segments coincide with the operating ranges, and the local models are the targets (fig. 5.8). Kalman filtering of noisy measurements before segmentation [3] combines with any of the costs of this chapter through `preprocessing.kalman_smoother`.

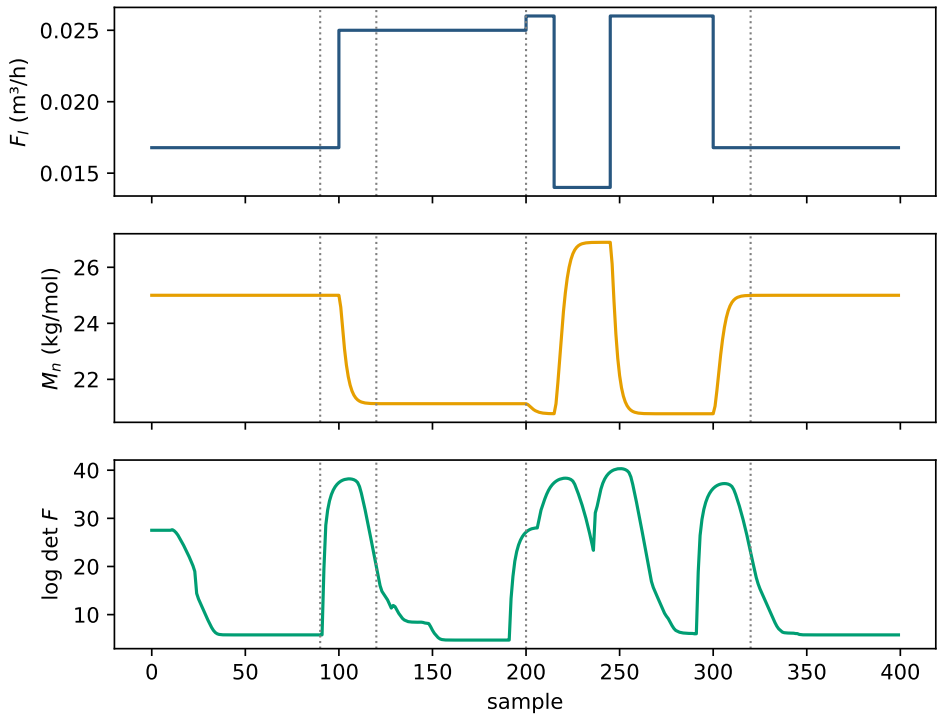


Figure 5.7. Segmentation of an operating record of the methyl methacrylate polymerization reactor [90] by Fisher information. From top: initiator flow, number-average molecular weight, and the windowed log det **F** of the kinetic parameters k_p, k_I, k_{fm}. The excitation period and the grade transition carry the information; constant operation adds almost nothing. Produced by `examples/03_latent_segmentation/plot_05_fisher_mma.py`.

```
for a, b in reg.segments:
    A = np.c_[np.ones(b - a), X[a:b, 1:]]
    coef = np.linalg.lstsq(A, X[a:b, 0], rcond=None)[0]
    print(
        f"[{a:4d},{b:4d})  KEI = {coef[0]:6.2f} {coef[1]:+.3f}*load
    {coef[2]:+.3f}*feed"
    )
```

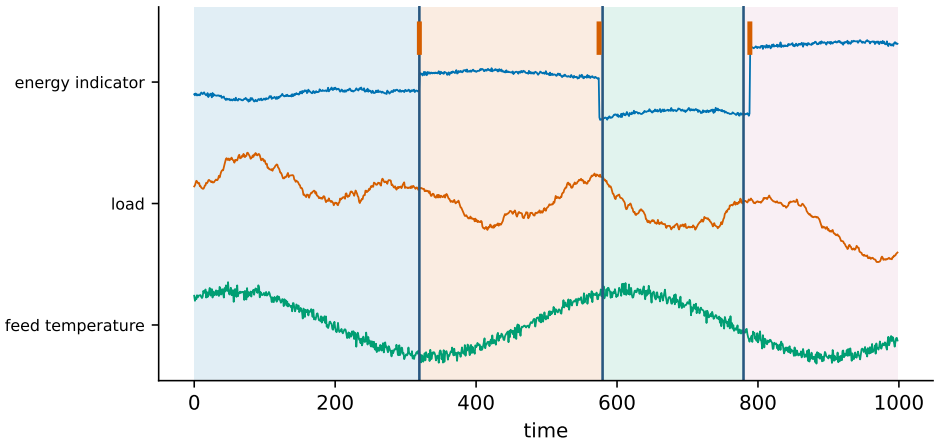


Figure 5.8. Model-based segmentation of energy monitoring data: the key energy indicator (top) depends linearly on the load and the feed temperature with coefficients that change between operating ranges; the regression cost recovers the ranges. Produced by `examples/03_latent_segmentation/plot_06_energy_regression.py`.

In a nutshell **Latent-variable segmentation**

A segment is homogeneous if one latent-variable model describes it. The PCA costs sum the eigenvalues outside (Q, correlation structure) or inside (T^2, drift) a p-component model; on the lag matrix they detect changes of the dynamics, with the lag chosen from the smallest eigenvalue. Recursive PCA with a variable forgetting factor and the Krzanowski similarity segments a stream on line, and a retrospective split removes the detection delay. The modified Gath–Geva algorithm makes segments fuzzy by combining Gaussian memberships in time with PPCA models and merges compatible clusters. Fisher information segments historical data by what they say about model parameters; a regression cost by the validity of one input–output model. In `mvtsa`: `PCASegmenter`, `DPCASegmenter`, `select_dpca_lag`, `RecursivePCASegmenter`, `GGSegmenter`, `FisherSegmenter`, `CostRegression`, `KrzanowskiSegmenter`.

CHAPTER

6



Probabilistic, profile-based and state segmentation

Not every segmentation problem fits the cost-and-search form of chapter 4. This chapter collects the other families: Bayesian on-line detection, which tracks the posterior of the time since the last change; hidden Markov and sparse Gaussian state models, which give recurring segments the same label; segment-then-cluster state detection; profile methods, which find changes of *shape* from nearest-neighbour structure; predictive homogeneity, which lets any scikit-learn model judge whether two segments differ; consensus, which combines segmenters; and stream processing. Benchmarks find no segmenter that dominates on all data [31, 32], so knowing when each family works is as important as the algorithms themselves.

6.1 Bayesian on-line change point detection

Adams and MacKay track the posterior distribution of the *run length* r_t, the number of samples since the last change, with an exact recursion [25]. Within a run the samples are independent draws from a model with a conjugate prior; with a constant hazard $H = 1/\lambda$ (the prior expected run length is λ),

$$P(r_t = r + 1, \mathbf{x}_{1:t}) = P(r_{t-1} = r, \mathbf{x}_{1:t-1})\ \pi_t^{(r)}\ (1 - H), \tag{6.1}$$

$$P(r_t = 0, \mathbf{x}_{1:t}) = \textstyle\sum_r P(r_{t-1} = r, \mathbf{x}_{1:t-1})\ \pi_t^{(r)}\ H, \tag{6.2}$$

where $\pi_t^{(r)} = p(\mathbf{x}_t \mid \mathbf{x}_{t-r:t-1})$ is the posterior predictive density given the last r samples – a multivariate Student t for the Normal–inverse-Wishart model of the package (`model="full"`), or a product of univariate Student densities for independent Normal–Gamma models (`"diag"`). The run-length posterior is truncated at `max_run`. With a constant hazard the probability of $r_t = 0$ equals H at

every step, so the evidence for a change at t is read a few samples later: `change_probability_[t]` is the posterior probability, after `delay` more samples, that a new run started at t, and its peaks above a threshold are the change points. The prior covariance is set from a robust estimate of the noise level (the median absolute first difference), which makes the detector usable without tuning on standardized data.

```
bo = sg.BOCPD(hazard=150, model="full")
seg = bo.segment(X)
print("detected:", seg.change_points, "\ntrue:    ", cps)
print(evaluate_segmentation(cps, seg, len(X), margin=10)["f1"])
```

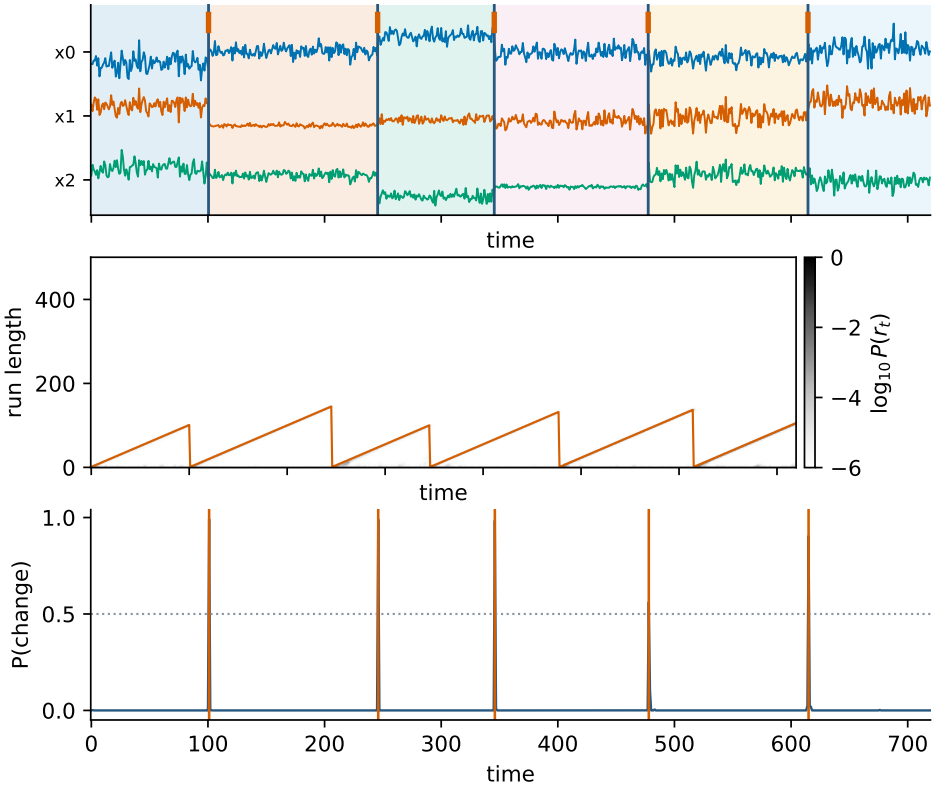


Figure 6.1. Bayesian on-line change point detection with a Normal–inverse-Wishart model. From top: the segmented series; the run-length posterior, whose mass falls back to zero at every change; the change probability with its threshold. Produced by `examples/04_modern_segmentation/plot_01_bocpd.py`.

6.2 States: segments that recur

A process that switches among a few operating regimes produces segments that *recur*. State detection gives recurring segments the same label, which is what an operator wants to know: not only that the process changed, but into which known regime. Three detectors are implemented.

Sticky hidden Markov model. A Gaussian HMM [91] with full or diagonal covariances is fitted by expectation–maximization with a scaled forward–backward recursion, restarted from several k-means initializations. A pseudo-count added to the self-transitions in every M-step (the *sticky* prior, after [92]) discourages implausibly short visits; after Viterbi decoding, runs shorter than `min_duration` are absorbed by their neighbours. The number of states can be selected by BIC (`n_states="bic"`).

Segment, then cluster. Any segmenter cuts the series; every segment is described by a local model – a Gaussian $(\boldsymbol{\mu}, \Sigma)$, a PCA model or its mean – and the segment models are clustered hierarchically (`StateSegmenter`). This is the clustering of multivariate windows by the similarity of their PCA models of Singhal and Seborg [93], generalized to any segmenter, descriptor and distance. The default distance is the 2-Wasserstein distance of the Gaussians,

$$W_2^2\big(\mathcal{N}(\boldsymbol{\mu}_1,\Sigma_1),\mathcal{N}(\boldsymbol{\mu}_2,\Sigma_2)\big) = \|\boldsymbol{\mu}_1-\boldsymbol{\mu}_2\|^2 + \operatorname{tr}\Big(\Sigma_1+\Sigma_2-2\big(\Sigma_2^{1/2}\Sigma_1\Sigma_2^{1/2}\big)^{1/2}\Big), \quad (6.3)$$

which compares level and covariance in one number [94]. Short segments – typically transitions – give unreliable models; they are left out of the clustering and assigned afterwards to the nearest state.

Sparse Gaussian states. Every state is a Gaussian on short stacked windows with a sparse precision matrix, estimated by the graphical lasso, and the state sequence is decoded by dynamic programming with a penalty β per switch (`SparseGaussianStateSegmenter`). This is the Toeplitz inverse covariance-based clustering of Hallac et al. [95] without its block-Toeplitz constraint: the sparse precision matrix shows which variables interact in each state.

```
models = {
    "sticky HMM": sg.GaussianHMMSegmenter(
        n_states=3, sticky=10, min_duration=10
    ),
    "segment + cluster": sg.StateSegmenter(n_states=3),
    "sparse Gaussian": sg.SparseGaussianStateSegmenter(
        n_states=3, window=2, beta=40
    ),
}
rows = []
for name, m in models.items():
    s = m.segment(R)
    rows.append(
```

```
        {
            "method": name,
            **evaluate_segmentation(
                cps, s, len(R), margin=10, true_states=states
            ),
        }
    )
print(pd.DataFrame(rows)[["method", "f1", "covering",
    "state_ari"]].round(3))
```

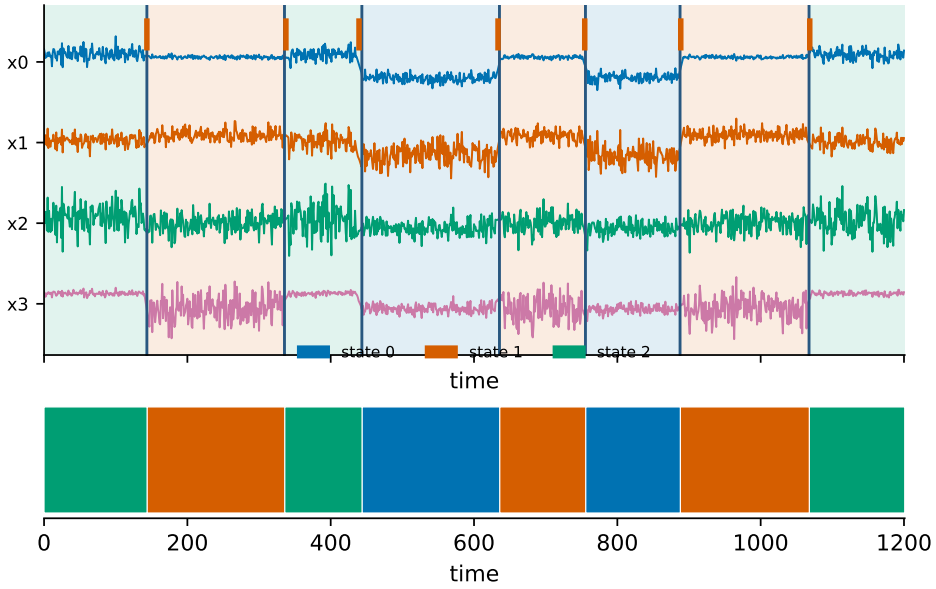


Figure 6.2. Recurring operating states of a process with three regimes, found by segment-then-cluster: the segmentation (top) and the state timeline (bottom). The three detectors of section 6.2 reach state ARI above 0.96. Produced by `examples/04_modern_segmentation/plot_02_states.py`.

The BIC of the HMM on these data asks for four states rather than three: the fourth state collects the few samples of the smooth transitions, which none of the regime models explains. Information criteria count such systematic deviations as structure – a reminder that a selected number of states is a property of the model as much as of the process.

6.3 Changes of shape: ClaSP and FLUSS

When segments differ in the *shape* of repeating patterns rather than in their statistics – a gait changing from walking to running, a machine cycle changing its profile – subsequence methods work best. FLUSS [27] draws an arc from every subsequence to its nearest neighbour (the matrix-profile index, section 3.6) and counts the arcs crossing every position; few crossings, relative to the count expected without regimes, mark a boundary. ClaSP [28] labels the subsequences left of a candidate split 0 and those right of it 1, and scores the split by the cross-validated accuracy of a k-nearest-neighbour classifier; the split with the highest score is accepted if the score is high enough, and the halves are segmented recursively. The package computes the classification profile of all splits in $O(N \log N)$ from a single k-NN profile, by counting for every subsequence the largest split at which the majority of its neighbours lies to its right.

Both profiles score splits between *subsequences*: a subsequence that starts at index i covers $[i, i + w)$, so the index of the best split lies about half a window before the change in sample time. `mvtsa` moves the reported change point and the profile by $w/2$ by default (`center=True`), which removes this systematic bias; the original convention is available with `center=False`. The z-normalized profiles see only shape; `normalize=False` makes them sensitive to level and amplitude as well.

```
clasp = sg.ClaSPSegmenter(width=40, n_segments=3)
seg = clasp.segment(x)
print("ClaSP:", seg.change_points, "| width", clasp.width_)
```

6.4 Predictive homogeneity

Two adjacent segments belong together if no learning machine can tell their samples apart. This is the classifier two-sample test of Lopez-Paz and Oquab [96]: train a classifier to predict from which segment a sample comes; if its cross-validated accuracy is at chance level, the segments come from the same distribution. `mvtsa` turns the test into a merging criterion: the dissimilarity of two adjacent segments is the balanced accuracy rescaled to $[0, 1]$, $d = \max\{0,\ 2(\mathrm{acc} - \frac{1}{2})\}$, and bottom-up merging (`NeighborMergeSegmenter`) joins the least distinguishable pair first. The classifier is any scikit-learn estimator: a linear model sees level shifts, quadratic discriminant analysis also covariance changes, a random forest nonlinear differences; stacking short windows instead of single samples (`window`) lets it see dynamics. A regression variant measures cross-prediction: a model fitted on one segment is applied to the other, and the loss relative to its cross-validated loss on its own segment is the dissimilarity – for chosen targets from chosen inputs, or for every variable from all the others. The batch, merging counterpart of the

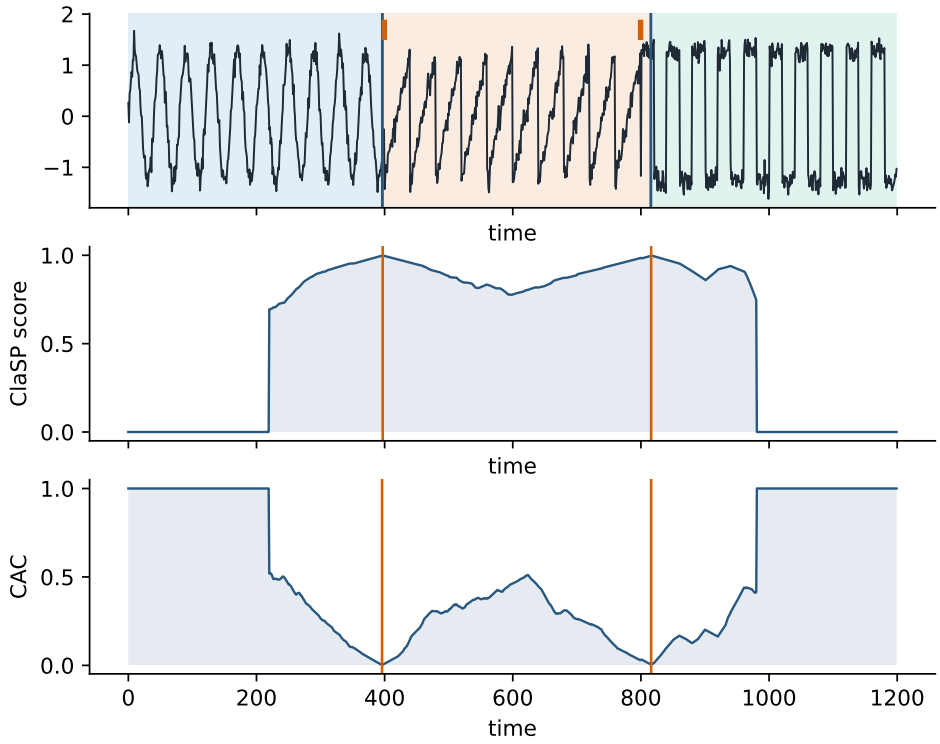


Figure 6.3. A signal that changes from a sine to a sawtooth to a square wave of the same period. From top: the ClaSP segmentation; the classification score profile; the corrected arc curve of FLUSS. Both find the shape changes. Produced by `examples/04_modern_segmentation/plot_03_clasp_fluss.py`.

self-supervised idea of ClaSP, predictive homogeneity is the most flexible criterion of the package: whatever a model can learn, it can segment.

```
rows = []
for name, est in [
    ("logistic", LogisticRegression(max_iter=500)),
    ("QDA", QuadraticDiscriminantAnalysis()),
    ("random forest", RandomForestClassifier(n_estimators=60,
    random_state=0)),
]:
    s = sg.PredictiveSegmenter(
        "classifier", estimator=est, n_segments=4, initial_size=30
    ).segment(X)
    rows.append(
        {"estimator": name, **evaluate_segmentation(cps, s, len(X),
    margin=15)}
    )
print(pd.DataFrame(rows)[["estimator", "f1", "covering"]].round(3))
```

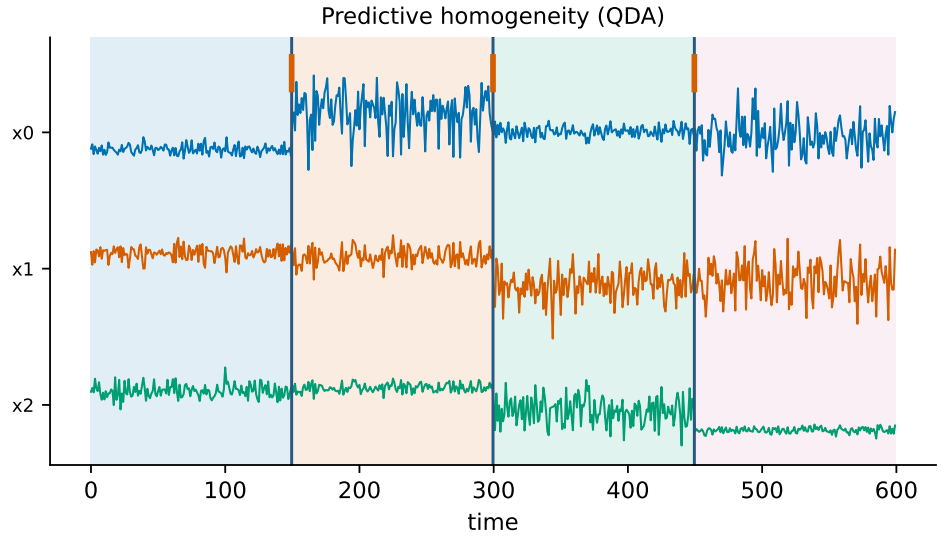


Figure 6.4. Segmentation by predictive homogeneity with quadratic discriminant analysis on a series with changes of the level, the variance and the correlation. Produced by `examples/04_modern_segmentation/plot_04_predictive.py`.

6.5 Consensus segmentation

Different costs, searches and parameters make different mistakes. A consensus of several segmenters is more robust than any of them, and it quantifies the uncertainty of every boundary. Every member m with change points C_m and weight w_m votes with a Gaussian kernel of bandwidth h; the *boundary agreement*

$$a(t) = \frac{1}{\sum_m w_m} \sum_m w_m \max_{c \in C_m} \exp\left(-\frac{(t-c)^2}{2h^2}\right) \tag{6.4}$$

is the weighted fraction of members that place a change point near t, and its peaks above a threshold (one half by default) are the consensus change points (`ConsensusSegmenter`, fig. 6.5). The same profile, computed over perturbed copies of the data instead of over different methods, measures stability (section 10.4).

```
members = [
    sg.BottomUp("normal", n_segments=5, initial_size=10),
    sg.PELT("normal", penalty=200, jump=5, min_size=20),
    sg.TopDown("kernel", n_segments=5, jump=5),
    sg.BOCPD(hazard=150),
    sg.WindowDiscrepancy("normal", width=40, n_segments=5),
]
cons = sg.ConsensusSegmenter(members, threshold=0.5)
seg = cons.segment(X)
for m, s in zip(
    ["bottom-up", "PELT", "top-down", "BOCPD", "window"],
```

```
    cons.members_,
    strict=True,
):
    print(f"{m:9s} {s.change_points}")
print("consensus", seg.change_points, "| true", cps)
print(evaluate_segmentation(cps, seg, len(X), margin=10)["f1"])
```

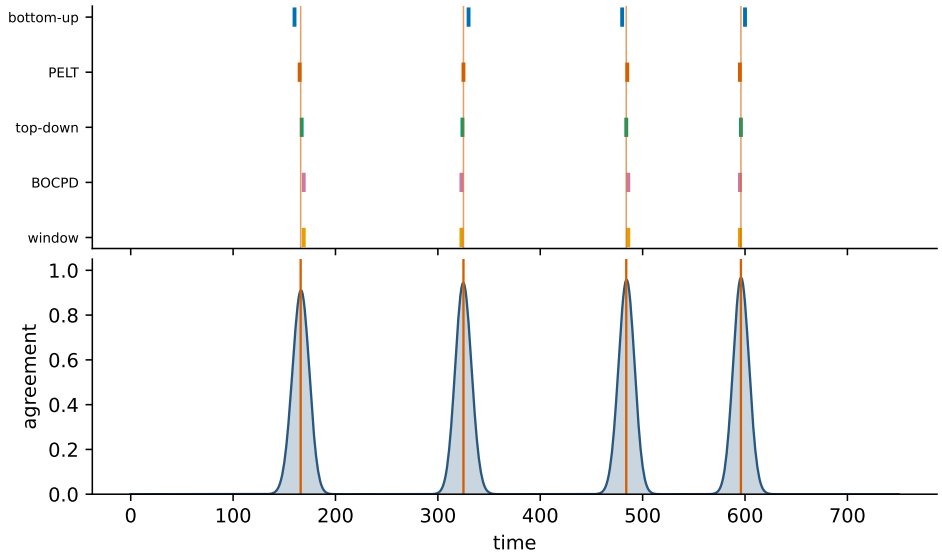


Figure 6.5. Consensus of five segmenters: the change points of every member (rows), the boundary agreement profile and the consensus change points. Produced by `examples/04_modern_segmentation/plot_05_consensus.py`.

6.6 Streams

Three detectors process data sample by sample. Recursive PCA (section 5.3) and BOCPD have their own on-line recursions. A multivariate CUSUM chart [97] accumulates the excess of the squared Mahalanobis distance of new samples from a reference distribution, $S_t = \max(0,\ S_{t-1} + m_t - n - k)$, signals when S_t exceeds h, and re-estimates the reference from the next samples (`CUSUMDetector`). Finally, `StreamSegmenter` applies *any* batch segmenter to a sliding buffer and reports a change point once it lies far enough from the end of the buffer to be stable; the reporting delay is the price of using a batch method on line.

```
stream = sg.StreamSegmenter(
    sg.PELT("l2", jump=5), window=300, step=25, guard=20
)
reported = []
for t, x in enumerate(X):
    for c in stream.partial_fit(x):
        reported.append((t, c))
```

In a nutshell **Probabilistic, profile-based and state segmentation**

BOCPD tracks the run-length posterior exactly with conjugate Gaussian models and reads the change evidence with a short delay. Sticky HMMs, segment-then-cluster with Wasserstein distances and sparse Gaussian states label recurring regimes. ClaSP and FLUSS find changes of shape from nearest-neighbour structure; centring their profiles removes a half-window bias. Predictive homogeneity turns any scikit-learn classifier or regressor into a bottom-up merging criterion. Consensus combines segmenters by boundary agreement. CUSUM, recursive PCA, BOCPD and a buffered wrapper of batch methods handle streams. In `mvtsa`: `BOCPD`, `GaussianHMMSegmenter`, `StateSegmenter`, `SparseGaussianStateSegmenter`, `ClaSPSegmenter`, `FLUSSSegmenter`, `PredictiveSegmenter`, `ConsensusSegmenter`, `CUSUMDetector`, `StreamSegmenter`.

```
for t, c in reported:
    print(f"at sample {t:5d}: change point {c} (delay {t - c})")
```

CHAPTER

7



Elastic similarity

Two series that follow the same pattern at different speeds are far apart in the Euclidean sense and close in any sensible one. This chapter develops the family of elastic measures that compensate time distortions: dynamic time warping with its constraints, step patterns and lower bounds; its variants for derivatives, phase penalties, local shape and multivariate data; the edit-type measures; the soft, differentiable relatives with their expected alignments; and the averaging, diagnosis and learning of warping. Chapter 8 builds on these tools to warp sequences of segments instead of samples.

7.1 Dynamic time warping

The recursion (2.2) defines DTW [40]; three choices refine it. The *step pattern* fixes the allowed steps and their weights: in the symmetric pattern of type I every step costs c_{ij}, in the symmetric type II a diagonal step costs $2c_{ij}$ so that all paths have the same total weight $T_x + T_y$ and the distance can be normalized by it, and the asymmetric pattern lets only one of the series be stretched [98, 99]. A *penalty* added to every non-diagonal step discourages pathological warping. A *global constraint* restricts the path to the Sakoe–Chiba band $|i - j| \leq w$ [40] or to the Itakura parallelogram [41]; besides speed, a band of a few per cent of the length often improves nearest-neighbour accuracy [100, 101]. `dtw` returns the distance, `dtw_path` the full result with the path and the matrices; `window_mask` builds the constraints (integer half-width, fraction of the length, or `("itakura", s)` with maximal slope s).

```
res = wp.dtw_path(x, y)
print(
    f"Euclidean {np.linalg.norm(x - y) ** 2:.2f} | DTW {res.distance:.2f} "
    f"| path length {len(res.path)}"
)
```

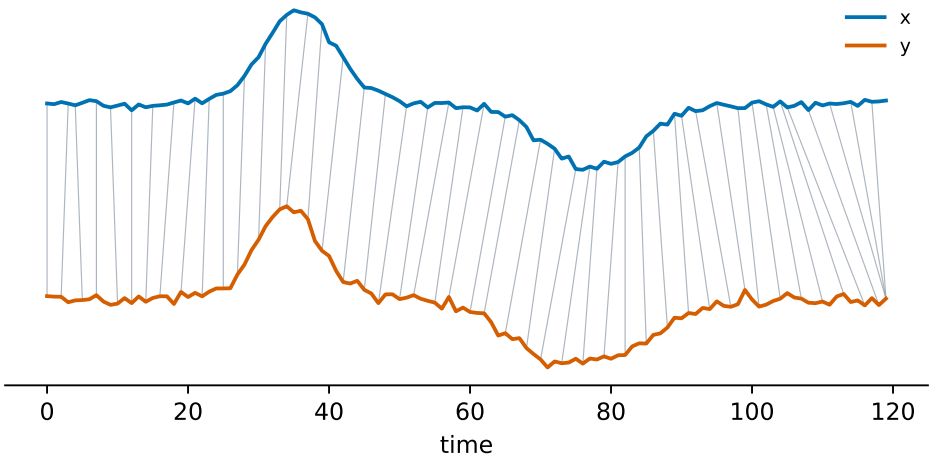


Figure 7.1. A DTW alignment of two warped copies of a two-bump signal; every third pair of matched samples is connected. Produced by `examples/05_warping/plot_01_dtw_basics.py`.

Lower bounds. Nearest-neighbour search under DTW is accelerated by lower bounds that are cheaper than DTW and discard most candidates. LB_Kim compares the first, last, largest and smallest values [102]; LB_Keogh compares the query with the envelope of the candidate under a band of half-width r and lower-bounds the constrained DTW [103]. `subsequence_dtw` finds the best match of a short query anywhere in a long series by an open-begin, open-end recursion; `fast_dtw` approximates DTW in linear time by refining a coarse path [104].

```
print(
    f"LB_Kim {wp.lb_kim(x, y):.3f} <= LB_Keogh {wp.lb_keogh(x, y,
    radius=12):.3f} "
    f"<= DTW(window=12) {wp.dtw(x, y, window=12):.3f}"
)
```

7.2 Variants

Derivative DTW aligns the estimated derivatives instead of the values, which matches shapes rather than levels and avoids the singularities of DTW on series with different offsets [105] (`ddtw`).

Weighted DTW multiplies the local cost by a logistic function of the phase difference, $w(|i-j|) = w_{\max} / \big(1 + \exp(-g(|i-j| - m/2))\big)$, a soft band [106] (`wdtw`).

Amerced DTW adds a constant penalty ω to every non-diagonal step; small ω gives DTW, large ω the Euclidean distance, and the penalty is easier to tune than a window [107] (`adtw`).

shapeDTW aligns local shape descriptors (raw subsequences, slopes, PAA) of a window around every sample [108] (`shape_dtw`).

Multivariate DTW has two strategies: the dependent one uses the squared Euclidean distance of the sample vectors as local cost (one path for all variables, `dtw`), the independent one warps every variable separately and sums the distances (`dtw_independent`); neither dominates [109], and both extend to the other elastic measures [110].

7.3 Edit-type measures

Edit distances count the cost of transforming one series into the other. LCSS matches samples within a tolerance ε and a time window δ and counts the length of the longest common subsequence, ignoring outliers [111]; EDR counts the edits needed under the same tolerance [112]. ERP charges gaps against a constant reference value g and is a metric [113]. TWED combines edit operations with a stiffness parameter ν that penalizes time differences and a constant λ for deletions, also a metric [114]. MSM transforms one series into the other by move, split and merge operations with a cost c for splits and merges [115]. All are dynamic programs with the same $O(T_x T_y)$ structure as DTW and run in the same compiled kernels. Figure 7.2 compares the measures on the Cylinder–Bell–Funnel problem [116], the classic test of elastic similarity: the three classes differ in shape, but the onsets and lengths of the events vary.

```
measures = {
    "euclidean": {},
    "dtw": {},
    "dtw (10% band)": {"metric": "dtw", "window": 0.1},
    "ddtw": {},
    "wdtw": {"g": 0.05},
    "adtw": {"penalty": 0.5},
    "shape_dtw": {"width": 9},
    "lcss": {"epsilon": 0.5, "delta": 10},
    "edr": {"epsilon": 0.25},
```

```
    "erp": {},
    "twed": {},
    "msm": {"c": 0.5},
    "sbd": {},
}
rows = []
for name, kw in measures.items():
    kw = dict(kw)
    metric = kw.pop("metric", name)
    t0 = time.perf_counter()
    D = pairwise_distances(C, metric=metric, **kw)
    rows.append(
        {
            "measure": name,
            "1-NN accuracy": knn_accuracy(D, y),
            "seconds": time.perf_counter() - t0,
        }
    )
table = pd.DataFrame(rows).sort_values("1-NN accuracy", ascending=False)
print(table.round(3).to_string(index=False))
```

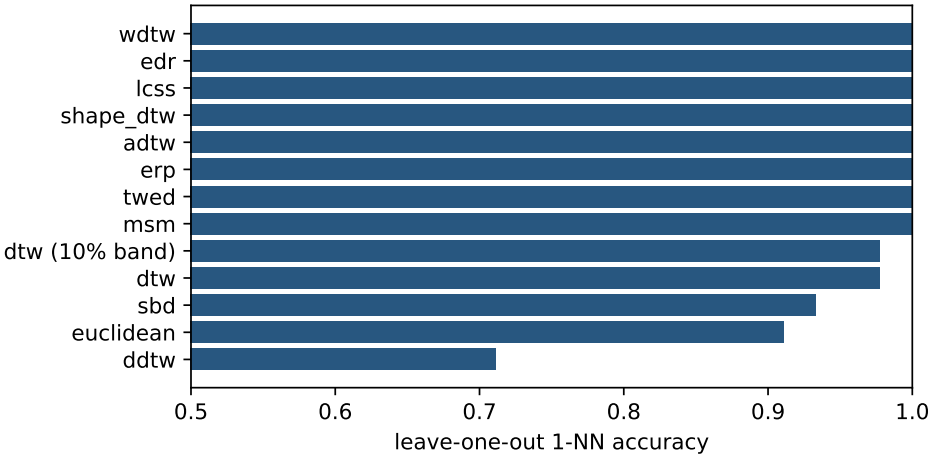


Figure 7.2. Leave-one-out 1-NN accuracy of thirteen measures on 45 Cylinder–Bell–Funnel series. All elastic measures beat the Euclidean distance; derivative DTW suffers from the noise of the derivatives. Produced by `examples/05_warping/plot_02_dtw_variants.py`.

7.4 Soft alignment

DTW is not differentiable: its minimum switches between paths. Soft-DTW replaces the minimum of the recursion by the soft minimum

$$\min^{\gamma}(a_1, \dots, a_k) = -\gamma \log \sum_{i=1}^{k} e^{-a_i/\gamma}, \tag{7.1}$$

which tends to the minimum as $\gamma \to 0$ [29]. The resulting value $\mathrm{sdtw}_\gamma(\mathbf{X}, \mathbf{Y})$ is differentiable, and its gradient with respect to the cost matrix is the *expected alignment* $\mathbf{E} = \partial\, \mathrm{sdtw}_\gamma / \partial \mathbf{C}$: the probability, under a Gibbs distribution over all paths, that (i, j) are aligned. A backward recursion computes it at the cost of the forward pass. The expected alignment shows how certain an alignment is: a sharp ridge where the series match unambiguously, a diffuse band where they do not (fig. 7.3). Because $\mathrm{sdtw}_\gamma(\mathbf{X}, \mathbf{X}) < 0$, soft-DTW is not a proper dissimilarity; the soft-DTW divergence

$$D_\gamma(\mathbf{X}, \mathbf{Y}) = \mathrm{sdtw}_\gamma(\mathbf{X}, \mathbf{Y}) - \tfrac{1}{2}\mathrm{sdtw}_\gamma(\mathbf{X}, \mathbf{X}) - \tfrac{1}{2}\mathrm{sdtw}_\gamma(\mathbf{Y}, \mathbf{Y}) \tag{7.2}$$

is non-negative and zero only for identical series [117]. The global alignment kernel sums the exponentiated similarities of all paths and is positive definite, which makes it usable in kernel methods such as spectral clustering [118, 119] (`gak`). The shape-based distance SBD is one minus the maximal normalized cross-correlation over all shifts, computed with the fast Fourier transform [30] (`sbd`).

```
fig, axes = plt.subplots(1, 3, figsize=(9, 3.2))
for ax, g in zip(axes, (0.01, 0.1, 1.0), strict=True):
    value, E = wp.soft_dtw_alignment(x, y, gamma=g)
    viz.plot_alignment_uncertainty(E, path=wp.dtw_path(x, y).path, ax=ax)
    ax.set_title(f"$\\gamma$={g}, soft-DTW={value:.2f}")
```

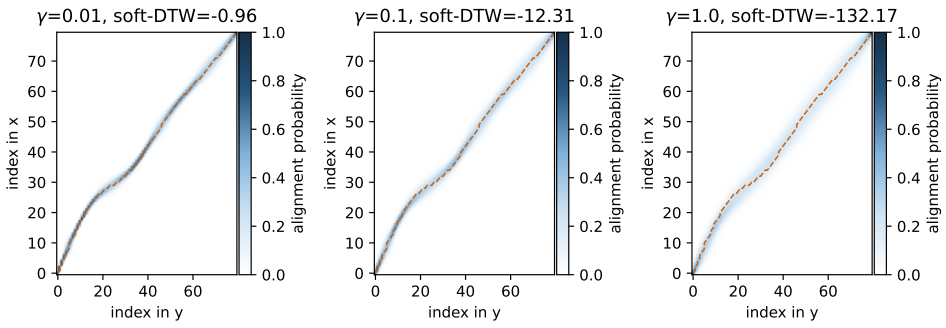


Figure 7.3. Expected alignments of soft-DTW for three temperatures, with the hard DTW path. At $\gamma = 0.01$ the expectation concentrates on the DTW path; at $\gamma = 1$ it spreads over many plausible paths. Produced by `examples/05_warping/plot_03_soft_dtw.py`.

Algorithm 7.1 DTW barycenter averaging (DBA)

Input: collection $\mathbf{X}^{(1)}, \dots, \mathbf{X}^{(N)}$; initial average $\mathbf{A}$ (e.g. the medoid); iterations I
Output: average $\mathbf{A}$

for I *iterations* **do**
  for every sample $\mathbf{a}_k$ of $\mathbf{A}$ start an empty list L_k;
  for *every series* $\mathbf{X}^{(m)}$ **do**
    compute the DTW path between $\mathbf{A}$ and $\mathbf{X}^{(m)}$; append every $\mathbf{x}_j^{(m)}$ aligned to $\mathbf{a}_k$ to L_k;
  $\mathbf{a}_k \leftarrow$ mean of L_k for every k;

7.5 Averaging

The arithmetic mean of warped series blurs their features. DTW barycenter averaging (DBA) iterates two steps: align every series to the current average with DTW, then replace every sample of the average by the mean of the samples aligned to it [33] (algorithm 7.1). The soft-DTW barycenter minimizes the sum of soft-DTW values to the series by gradient descent [29]. The medoid – the member with the smallest total distance – is the robust alternative. Averages are the centres of k-means clustering under DTW (chapter 9).

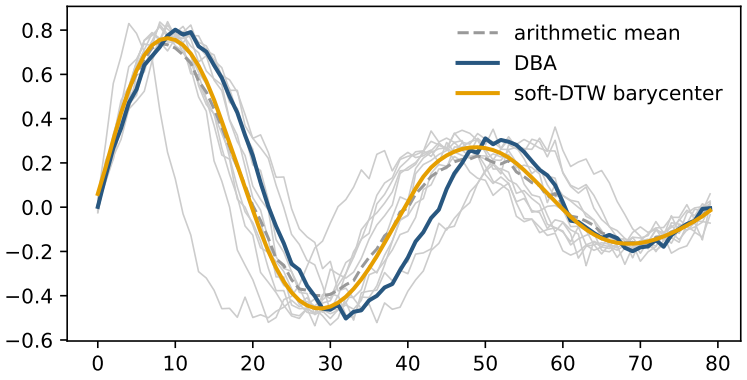


Figure 7.4. Twelve warped noisy copies of a damped oscillation (grey) and three averages: the arithmetic mean blurs the oscillation, DBA and the soft-DTW barycenter keep its shape. Produced by `examples/05_warping/plot_03_soft_dtw.py`.

In a nutshell Elastic similarity

DTW aligns two series by a monotone path of minimal cost; step patterns, penalties and Sakoe–Chiba or Itakura windows shape the path, and LB_Kim and LB_Keogh accelerate searches. Derivative, weighted, amerced and shape DTW change what is aligned or how warping is penalized; the dependent and independent strategies extend DTW to many variables. LCSS, EDR, ERP, TWED and MSM count edits; soft-DTW makes alignment differentiable and yields expected alignments, its divergence a proper dissimilarity; GAK is a positive definite kernel, SBD a fast shape distance. DBA and the soft-DTW barycenter average warped series. Path statistics, the tightness of warping, window learning and the elastic ensemble diagnose and tune warping. In `mvtsa`: `dtw`, `dtw_path`, `window_mask`, `lb_keogh`, `ddtw`, `wdtw`, `adtw`, `shape_dtw`, `lcss`, `edr`, `erp`, `twed`, `msm`, `soft_dtw`, `soft_dtw_alignment`, `gak`, `sbd`, `dba`, `pairwise_distances`.

7.6 Diagnosing and learning warping

How much warping does a data set need? A warping path is summarized by its length relative to the series, the share of non-diagonal steps, the longest horizontal and vertical runs and the area between the path and the diagonal (`path_statistics`); the warping function $\phi(i)$, the average aligned index of $\mathbf{Y}$ for every index of $\mathbf{X}$, shows the distortion directly. The tightness of warping (ToW) of Bankó and Abonyi, the average path length relative to the number of aligned elements over many pairs, compares how much different measures warp [7] (`tightness_of_warping`, section 8.4). A window can be learned by maximizing the leave-one-out 1-NN accuracy over candidate widths [100] (`learn_window`), and the elastic ensemble lets several measures vote, each with a weight equal to its leave-one-out accuracy on the training collection [120] (`ElasticEnsemble`).

```
best, scores = wp.learn_window(
    C, y, candidates=(0.0, 0.02, 0.05, 0.1, 0.2, None)
)
print(
    "accuracy by window:",
    {str(k): round(v, 3) for k, v in scores.items()},
    "-> best",
    best,
)
```

CHAPTER

8



Correlation-based and segment-level warping

Sample-level DTW compares multivariate series sample by sample with the Euclidean distance, so it sees levels and ignores how the variables move together. This chapter warps *sequences of segments* instead. Correlation-based DTW segments every series into pieces of homogeneous correlation structure and aligns the segment sequences with a PCA similarity as local cost; dynamic PCA makes it aware of process dynamics; Riemannian and mixed local distances extend it to segment models that differ in level, scale or trend. For univariate series, the MPLA, SPLA and MDPLA measures warp piecewise linear approximations, and episode alignment compares qualitative trends. Warping a few dozen segments instead of hundreds of samples is also fast.

8.1 Correlation-based dynamic time warping

Bankó and Abonyi combined the two lines of the group's work, PCA segmentation and DTW, into correlation-based dynamic time warping (CBDTW) [4, 5, 121]. The idea is that in many industrial data sets the correlation of the variables carries the real information, so two series should be compared by the *sequence of their correlation structures*. PCA similarity factors compare whole series by one correlation structure each [21, 45, 93] and cannot follow changes of the structure within a series; DTW follows the changes but ignores the structure. CBDTW does both (fig. 8.1, algorithm 8.1): every series of a database is segmented bottom-up with the PCA cost of section 5.1; a PCA model is fitted to every segment; and DTW aligns the segment sequences of two series with the local cost $1 - s_{\mathrm{PCA}}$ between their segment models. A query is segmented "according to the database", with the

Algorithm 8.1 Correlation-based dynamic time warping (CBDTW)

Input: collection $\mathcal{C}$; number of segments c (or a fraction of the length); components p; lags L; local distance d
Output: segment models of every series; distances between series

standardize all series with the statistics of the whole collection;
for *every series* $\mathbf{X} \in \mathcal{C}$ **do**
    segment $\mathbf{X}$ bottom-up into c segments with the PCA cost (on the lag matrix if $L > 0$); fit a PCA model $(\boldsymbol{\mu}_k, \mathbf{U}_k, \boldsymbol{\Lambda}_k)$ to every segment;
for *a pair of series with segment models* $A_1..A_{c_x}$ *and* $B_1..B_{c_y}$ **do**
    local cost matrix $C_{kl} \leftarrow d(A_k, B_l)$, by default $1 - s_{\mathrm{PCA}}(\mathbf{U}_k, \mathbf{V}_l)$;
    CBDTW $\leftarrow$ DTW of C (symmetric type I steps, optional window over segment indices);

same settings and scaling. Without warping, comparing segment k with segment k gives the SEGPCA baseline [5] (`segpca_distance`, `CBDTW(warping=False)`).

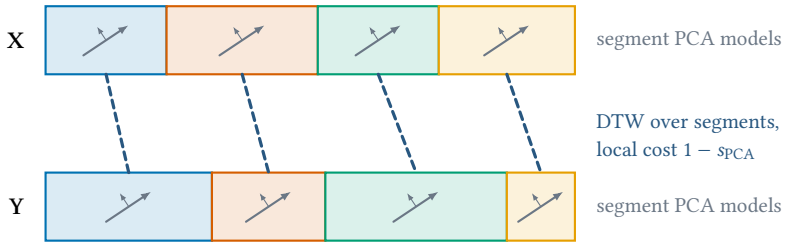


Figure 8.1. Correlation-based dynamic time warping. Both series are segmented by their correlation structure (the arrows sketch the principal directions of the segment models); DTW then aligns the segment sequences with the dissimilarity $1 - s_{\mathrm{PCA}}$ of the segment models as local cost. Segments of different lengths are matched.

The number of segments is a parameter of the method. Bankó selected it with the relative reduction rate of the cost curve (section 4.4) on a sample of the database [8]; the precision–recall analysis of retrieval (section 10.3) is the external check. On synthetic on-line signatures – pen position, pressure and angles of several writers, warped and perturbed – CBDTW retrieves the signatures of the same writer as reliably as DTW at a small fraction of the cost (fig. 8.2).

```
cb = wp.CBDTW(n_segments=10, n_components=2).fit(data)
path = cb.path(0, 1)
print(
    "segments:",
    len(cb.models_[0]),
    len(cb.models_[1]),
    "| CBDTW distance:",
    round(cb.distance(0, 1), 3),
```

```
)
```

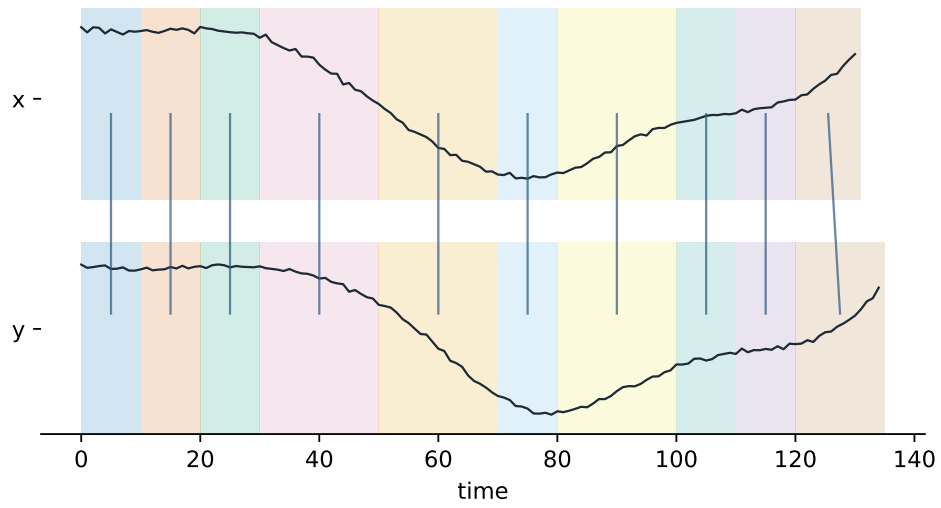


Figure 8.2. Segment-level alignment of two signatures of the same writer by CBDTW: the coloured bands are the segments of the two series, the lines connect aligned segments. Produced by `examples/06_correlation_warping/plot_01_cbdtw_signatures.py`.

When the information is in the correlation. The advantage of CBDTW is largest when classes differ in the *order* of correlation structures rather than in levels or shapes. In the collection of fig. 8.3, four classes share one pool of correlation structures of eight variables and differ only in their sequence, and strong noise covers the variables; sample-level DTW reaches a 1-NN accuracy of 0.80 and a mean average precision of 0.68, CBDTW 1.0 and 0.999.

```
measures = {
    "CBDTW": wp.CBDTW(n_segments=8, n_components=2).fit(C),
    "CBDTW, Grassmann": wp.CBDTW(
        n_segments=8, n_components=2, local="grassmann"
    ).fit(C),
    "DTW": "dtw",
    "DTW_I": "dtw_i",
}
rows, curves = [], {}
for name, m in measures.items():
    D = pairwise_distances(C, metric=m)
    curves[name] = precision_recall_curve_knn(D, y, r_items=9)
    rows.append(
        {
            "measure": name,
            "1-NN": knn_accuracy(D, y),
            "mAP": mean_average_precision(D, y),
        }
    )
print(pd.DataFrame(rows).round(3).to_string(index=False))
```

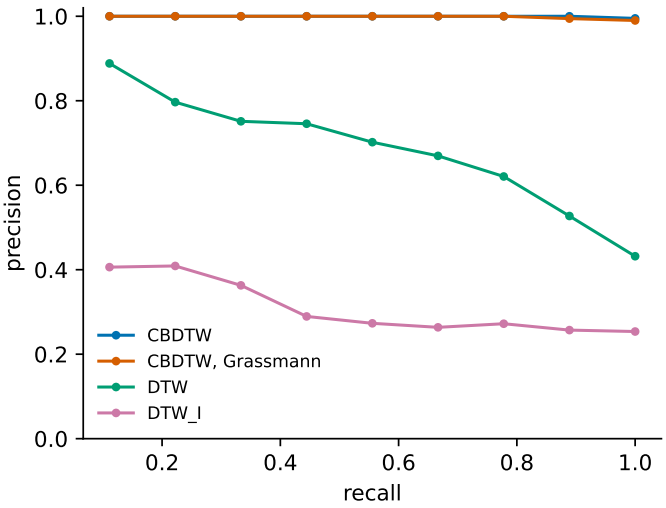


Figure 8.3. Precision–recall curves of retrieval (Bankó's Algorithm 3.1, section 10.3) on a collection whose classes differ only in the order of their correlation structures. Produced by `examples/06_correlation_warping/plot_02_correlation_sequences.py`.

8.2 Process-dynamics-aware warping

In the second thesis of his dissertation, Bankó extended CBDTW with dynamic PCA [6, 8]: the segmentation cost and the segment models are computed on the lag matrix (2.5), so that two series are compared by the sequence of their input–output *dynamics* (`CBDTW(lags=L)`). In the collection of fig. 8.4, the classes differ only in the order in which the time constant of a first-order system changes. The static correlation of input and output already reflects the time constant – a slow system follows its input less closely – so static CBDTW is a strong baseline (mAP 0.88); the dynamic segment models capture the input–output relation itself and separate the classes perfectly, while sample-level DTW reaches 0.59. The number of retained components matters: the dynamic relations live in the smallest directions of the lag-matrix covariance, so the best models keep all but the last one to three components.

```
rows = []
for name, m in [
    (
        "DPCA-CBDTW (2 lags, 4 PCs)",
        wp.CBDTW(n_segments=3, n_components=4, lags=2).fit(C),
    ),
    ("CBDTW (1 PC)", wp.CBDTW(n_segments=3, n_components=1).fit(C)),
    ("DTW", "dtw"),
]:
    D = pairwise_distances(C, metric=m)
    rows.append(
```

```
        {
            "measure": name,
            "1-NN": knn_accuracy(D, y),
            "mAP": mean_average_precision(D, y),
        }
    )
print(pd.DataFrame(rows).round(3).to_string(index=False))
```

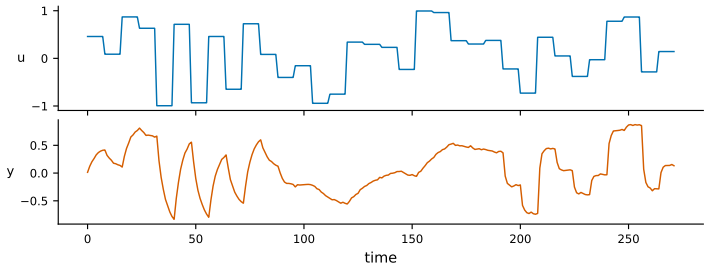


Figure 8.4. A series of the collection used in section 8.2: input u and output y of a first-order system whose time constant changes twice. Produced by `examples/06_correlation_warping/plot_03_dynamic_cbdtw.py`.

8.3 Riemannian and mixed segment distances

The Krzanowski similarity compares the principal subspaces of two segments and nothing else. When segments also differ in level, scale or covariance shape, other local distances are better. A segment model is a Gaussian $(\boldsymbol{\mu}, \Sigma)$, and symmetric positive definite matrices form a Riemannian manifold with several natural distances (`LOCAL_DISTANCES`): the log-Euclidean distance $\|\log \Sigma_1 - \log \Sigma_2\|_F$ [122], the affine-invariant distance $\|\log(\Sigma_1^{-1/2}\Sigma_2\Sigma_1^{-1/2})\|_F$ [123], the Bures–Wasserstein distance $\left(\operatorname{tr}\Sigma_1 + \operatorname{tr}\Sigma_2 - 2\operatorname{tr}(\Sigma_1^{1/2}\Sigma_2\Sigma_1^{1/2})^{1/2}\right)^{1/2}$ [94], the 2-Wasserstein distance of the Gaussians (6.3), which adds the means, and the Jeffreys divergence (symmetric Kullback–Leibler). Principal angles give the Grassmann and chordal distances of subspaces, and the λ and Eros similarities weight the angles by the eigenvalues [43, 45].

On real speech data the difference is large. The JapaneseVowels collection [124] holds twelve LPC cepstrum coefficients of nine speakers; the utterances are short (7 to 29 frames), so a segment carries few samples. The correlation structure alone separates the speakers poorly (1-NN accuracy 0.74), while the Gaussian Wasserstein distance between whole-utterance models matches the independent multivariate DTW (0.97) at a fraction of its cost (fig. 8.5).

```
rows = []
for name, m in [
    ("DTW", "dtw"),
    ("DTW_I", "dtw_i"),
    ("CBDTW, Krzanowski", wp.CBDTW(n_segments=1, n_components=2)),
    (
        "CBDTW, log-Euclidean",
        wp.CBDTW(n_segments=1, n_components=3, local="log_euclidean"),
    ),
    (
        "CBDTW, Jeffreys",
        wp.CBDTW(n_segments=1, n_components=3, local="jeffreys"),
    ),
    (
        "CBDTW, Gaussian W2",
        wp.CBDTW(n_segments=1, n_components=3, local="gaussian_w2"),
    ),
    (
        "CBDTW, Gaussian W2, 2 segments",
        wp.CBDTW(
            n_segments=2, n_components=3, local="gaussian_w2",
    initial_size=6
        ),
    ),
]:
    t0 = time.perf_counter()
    D = pairwise_distances(C, metric=m if isinstance(m, str) else m.fit(C))
    rows.append(
        {
            "measure": name,
            "1-NN": knn_accuracy(D, y),
            "mAP": mean_average_precision(D, y),
            "seconds": time.perf_counter() - t0,
        }
    )
print(pd.DataFrame(rows).round(3).to_string(index=False))
```

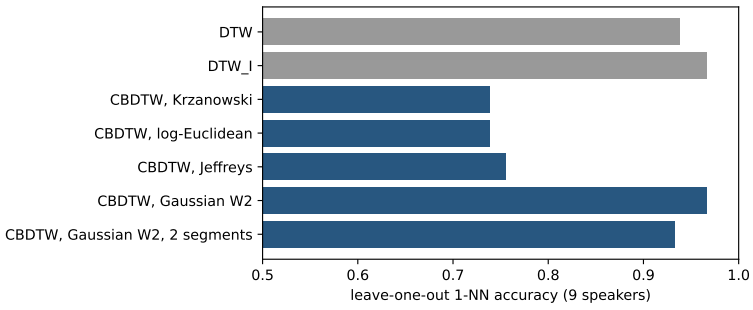


Figure 8.5. Leave-one-out 1-NN accuracy of sample-level DTW (grey) and of CBDTW with different local distances (blue) on 180 utterances of JapaneseVowels. Produced by `examples/06_correlation_warping/plot_04_japanese_vowels.py`.

A mixed segment distance. `MixedSegmentDistance` combines several aspects of two segment models in a weighted sum: the difference of their levels (in units of the collection standard deviation), of their slope angles, of their principal subspaces ($1 - s_{\mathrm{PCA}}$), of their scales ($|\log(\operatorname{tr}\Sigma_a / \operatorname{tr}\Sigma_b)|$) and of their durations. It generalizes the MDPLA measure of the next section to multivariate segment models. The weights are learned by leave-one-out accuracy on a simplex grid (`learn_segment_weights`); the segmentations are computed once, so the search is cheap. On the BasicMotions recordings [125], the learned weighting puts most weight on the correlation structure and the scale and classifies every recording correctly.

```
model = wp.CBDTW(
    n_segments=6,
    n_components=2,
    local=wp.MixedSegmentDistance(
        {"level": 1, "trend": 1, "structure": 1, "scale": 1}
    ),
)
best, scores = wp.learn_segment_weights(model, C, y)
print("learned weights:", best)
top = sorted(scores.items(), key=lambda kv: -kv[1])[:5]
print(
    pd.DataFrame(
        [{**dict(zip(best, w, strict=True)), "accuracy": a} for w, a in
    top]
    ).round(3)
)
```

8.4 Warping piecewise linear approximations

For univariate series, Bankó and Abonyi defined segment-level dissimilarities of piecewise linear approximations [7]. Every series is approximated by c linear segments (bottom-up PLA, section 3.1); a segment is described by its mean $\bar{x}$, scaled to $[-1, 1]$ by the largest absolute segment mean of the collection, and by the angle of its slope s, $\arctan(s)$ scaled by $2/\pi$. The local dissimilarities of two segments are

$$d_{\mathrm{MPLA}} = |\bar{x}_i - \bar{y}_j|, \qquad d_{\mathrm{SPLA}} = \tfrac{2}{\pi}|\arctan s_i - \arctan s_j|, \tag{8.1}$$

$$d_{\mathrm{MDPLA}} = w_M\, d_{\mathrm{MPLA}} + w_S\, d_{\mathrm{SPLA}}, \tag{8.2}$$

and DTW aligns the segment sequences with one of them (`PLADistance`). MPLA compares levels, SPLA trends, and the mixed MDPLA both. Bankó and Abonyi also introduced the *tightness of warping* to measure how much a measure warps: the average length of the warping paths relative to the number of segments [7]. A measure that warps less produces more natural alignments, and mixing the level and the trend reduces warping. The right weighting depends on the data, which is why the package learns it (fig. 8.6).

```
rows = []
for dname, (C, y) in datasets.items():
    for kind in ("mpla", "spla", "mdpla"):
        p = wp.PLADistance(n_segments=12, kind=kind).fit(C)
        tow = wp.tightness_of_warping(
            list(range(len(C))), p.path, max_pairs=200, random_state=0
        )
        rows.append(
            {
                "data": dname,
                "measure": kind.upper(),
                "1-NN": knn_accuracy(p.pairwise(), y),
                "ToW": tow,
            }
        )
    rows.append(
        {
            "data": dname,
            "measure": "DTW",
            "1-NN": knn_accuracy(pairwise_distances(C, metric="dtw"), y),
            "ToW": np.nan,
        }
    )
print(pd.DataFrame(rows).round(3).to_string(index=False))
```

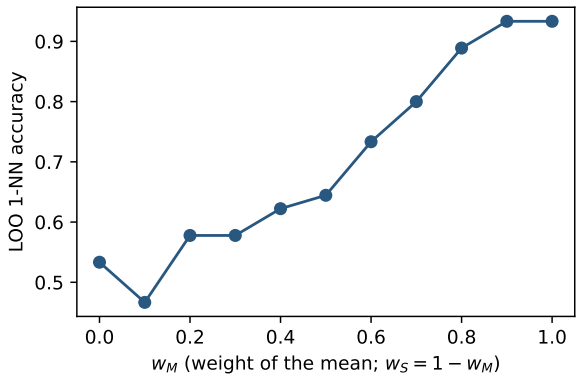


Figure 8.6. Leave-one-out 1-NN accuracy of MDPLA on Cylinder–Bell–Funnel as a function of the weight of the mean ($w_S = 1 - w_M$). On these data the level carries most of the information. Produced by `examples/06_correlation_warping/plot_05_pla_mdpla.py`.

8.5 Aligning qualitative episodes

Balaskó, Bankó and Abonyi compared process trends by their qualitative episodes (section 3.3) [9, 10]. The episode strings of two signals are aligned with the Needleman–Wunsch algorithm [126] (or locally with Smith–Waterman [127]): the score of aligning two episodes comes from a substitution table of the seven primitives – identical primitives score highest, similar ones (a concave and a convex increase) less, opposite ones are penalized – and gaps cost a constant. The score is normalized so that a perfect alignment gives 10. Weighting the substitution scores by the size of the episodes – one third for small, two thirds for medium episodes, by the terciles of their durations and magnitudes – makes small wiggles count less (`align_episodes`). Because the alignment works on a handful of symbols, it is fast and insensitive to the time scale: a slower and rescaled copy of a signal scores almost 10, a noisy one less, an inverted one much less (fig. 8.7). As an extension, `episode_dtw` warps episode sequences with a local cost that combines the primitive dissimilarity with the log-ratio of the durations and the relative difference of the magnitudes.

```
for k in ("slower and scaled", "noisy", "different"):
    score, pairs = wp.align_episodes(eps["reference"], eps[k])
    print(
        f"reference vs {k:18s}: score {score:5.2f} / 10, "
        f"episode DTW {wp.episode_dtw(eps['reference'], eps[k]):.3f}"
    )
```

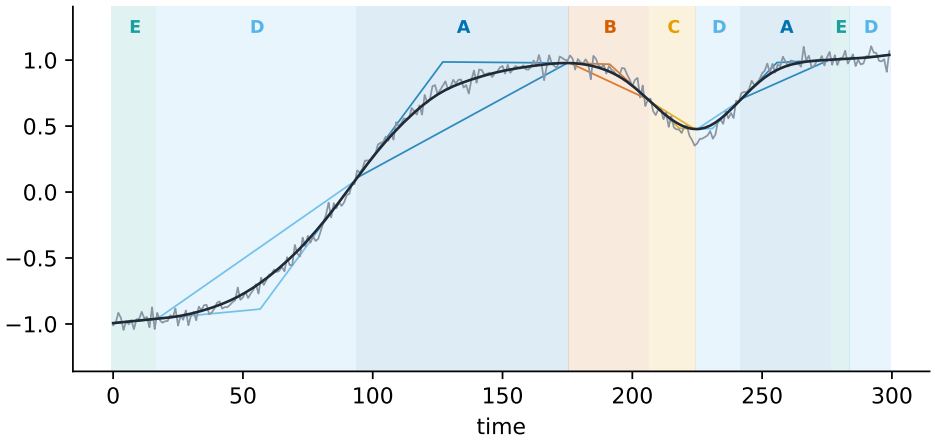


Figure 8.7. Qualitative episodes of the noisy copy of the reference signal used in section 8.5. Produced by `examples/06_correlation_warping/plot_07_episodes.py`.

In a nutshell **Correlation-based and segment-level warping**

CBDTW segments every series by its correlation structure and warps the segment sequences with $1 - s_{\mathrm{PCA}}$ as local cost; without warping it is SEG-PCA. Dynamic PCA makes it compare input–output dynamics; Riemannian distances (log-Euclidean, affine-invariant, Bures–Wasserstein, Gaussian W_2, Jeffreys) and a learned mixed distance compare richer segment models. MPLA, SPLA and MDPLA warp piecewise linear approximations by level, trend or both, and the tightness of warping measures how much a measure warps. Episode alignment compares qualitative trends with a substitution table and size weights. In `mvtsa`: `CBDTW`, `segpca_distance`, `LOCAL_DISTANCES`, `MixedSegmentDistance`, `learn_segment_weights`, `PLADistance`, `tightness_of_warping`, `align_episodes`, `episode_dtw`.

CHAPTER

9



Clustering

Clustering groups series, segments or moving objects. This chapter starts with partitional methods, which need a notion of a cluster centre – the arithmetic mean, the DBA or soft-DTW barycenter, the shape centroid of k-Shape, or a medoid – and continues with relational methods, which need only a distance matrix and therefore work with every measure of chapters 7 and 8. Maps of distance matrices, including the fuzzy Sammon mapping, show the structure of a collection. The last sections cluster features, segments and trajectories.

9.1 Partitional clustering

k-means alternates between assigning every series to the nearest centre and recomputing the centres. For time series the two steps need an elastic distance and a matching average: the arithmetic mean with the Euclidean distance, DBA with DTW [33], or the soft-DTW barycenter with soft-DTW [29] (`TimeSeriesKMeans`, `metric="euclidean"`, `"dtw"`, `"softdtw"`). k-Shape uses the shape-based distance SBD and a centroid that maximizes the sum of squared normalized cross-correlations with the aligned members, the leading eigenvector of a matrix built from them [30] (`KShape`). The Odyssey benchmark found that none of 80 clustering methods outperformed k-Shape with a statistically significant difference [128], and a later study extended the comparison to 84 methods including deep learning and foundation models [129]; k-Shape is thus the natural baseline. When the centre is a member of the collection, any distance can be used: k-medoids by partitioning around medoids [130] (`KMedoids`) and the relational fuzzy c-medoids [131] (`FuzzyCMedoids`), whose memberships $u_{ik} \propto d_{ik}^{-2/(m-1)}$ quantify the ambiguity of every assignment and feed the fuzzy Sammon mapping of section 9.3.

```
models = {
    "k-means (Euclidean)": cl.TimeSeriesKMeans(
        6, metric="euclidean", random_state=0
    ),
    "k-means (DBA)": cl.TimeSeriesKMeans(
        6, metric="dtw", window=0.1, max_iter=10, random_state=0
    ),
    "k-Shape": cl.KShape(6, random_state=0),
    "k-medoids (DTW)": cl.KMedoids(
        6, metric="dtw", metric_params={"window": 0.1}
    ),
    "fuzzy c-medoids (DTW)": cl.FuzzyCMedoids(
        6, metric="dtw", metric_params={"window": 0.1}, random_state=0
    ),
}
rows = []
for name, m in models.items():
    m.fit(C)
    rows.append({"algorithm": name, **clustering_scores(y, m.labels_)})
print(pd.DataFrame(rows).round(3).to_string(index=False))
```

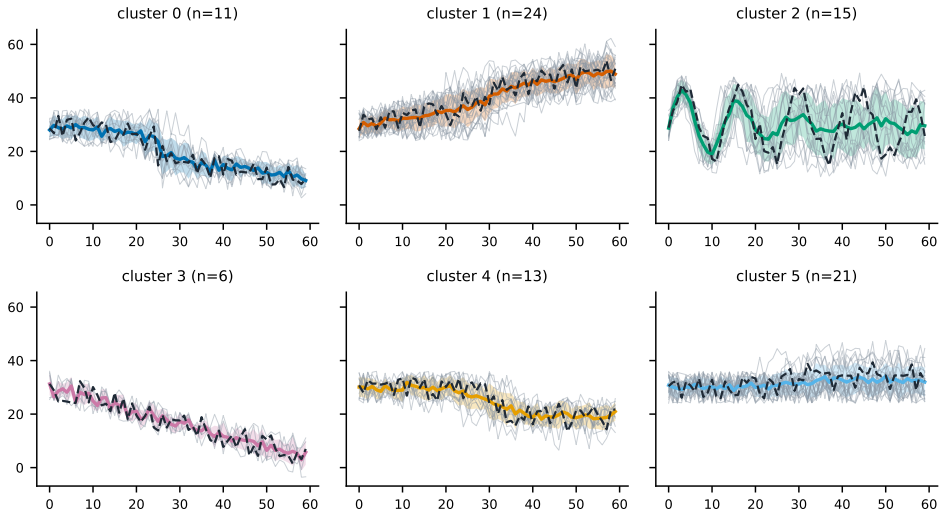


Figure 9.1. Clusters of the synthetic control charts [132] found by k-means with DBA centres: members (grey), mean and standard deviation band, and the DBA centre (dashed). Produced by `examples/07_clustering/plot_01_partitional.py`.

9.2 Relational clustering

Relational methods take a distance matrix $\mathbf{D}$. Agglomerative hierarchical clustering merges the closest clusters under single, complete, average or Ward linkage and returns a dendrogram that can be cut at any level (`HierarchicalClustering`). Spectral clustering embeds the objects by the leading eigenvectors of a normalized affinity matrix, $\exp(-d_{ij}^2/2\sigma^2)$ or a kernel such as GAK, and clusters the embedding [133] (`SpectralClustering`). DBSCAN and HDBSCAN find dense groups and leave outliers unassigned [134, 135] (`DensityClustering`). Consensus clustering accumulates the evidence of several clusterings in a co-association matrix – the fraction of clusterings that put two objects together – and clusters it hierarchically [136] (`ConsensusClustering`).

The choice of linkage matters more than one might expect. Average linkage tends to isolate a few outlying series in clusters of their own and leave one huge cluster when the number of clusters is fixed; Ward's criterion and medoid-based methods give balanced partitions (fig. 9.2). On BasicMotions, recordings of four activities with a wrist-worn accelerometer and gyroscope [125], k-medoids on CBDTW distances with the Gaussian Wasserstein local cost reaches an adjusted Rand index of 0.88, ahead of every other combination of the example.

```
D_dtw = pairwise_distances(C, metric="dtw_i", window=0.1)
D_cb = (
    wp.CBDTW(n_segments=6, n_components=2, local="gaussian_w2")
    .fit(C)
    .pairwise()
)
models = {
    "Ward, DTW_I": cl.HierarchicalClustering(
        4, linkage="ward", metric="precomputed"
    ).fit(D_dtw),
    "Ward, CBDTW (W2)": cl.HierarchicalClustering(
        4, linkage="ward", metric="precomputed"
    ).fit(D_cb),
    "k-medoids, CBDTW (W2)": cl.KMedoids(4,
    metric="precomputed").fit(D_cb),
    "spectral, CBDTW (W2)": cl.SpectralClustering(4,
    metric="precomputed").fit(
        D_cb
    ),
    "features + k-means": cl.FeatureClustering(features="catch").fit(C),
    "signatures + k-means": cl.FeatureClustering(
        features="signature", signature_depth=2
    ).fit(C),
}
models["consensus"] = cl.ConsensusClustering(n_clusters=4).fit_partitions(
    [m.labels_ for m in models.values()]
)
rows = [
    {"model": k, **clustering_scores(y, m.labels_)} for k, m in
    models.items()
```

```
]
print(pd.DataFrame(rows).round(3).to_string(index=False))
```

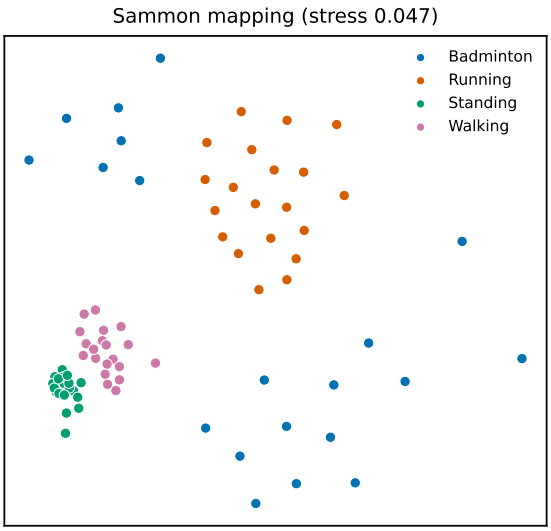


Figure 9.2. Sammon map of the CBDTW (Gaussian W_2) distances of the forty BasicMotions recordings, coloured by activity. Produced by `examples/07_clustering/plot_03_basic_motions.py`.

9.3 Maps of distance matrices

A distance matrix is easier to understand as a picture. Sammon mapping places the objects in the plane so that their distances match the original ones, with small distances weighted more [137] (`sammon`); metric MDS and t-SNE are the alternatives of `plot_embedding`. For a fuzzy clustering, the fuzzy Sammon mapping of Feil, Balaskó and Abonyi maps the objects relative to the cluster prototypes, preserving the distances to the prototypes weighted by the memberships [18] (`fuzzy_sammon`). It needs only Nc distances instead of $N^2/2$, so it scales to large collections, and it shows the clusters with the prototypes at their centres (fig. 9.3). Ordering a distance matrix by the clusters (`plot_distance_matrix`) and the dendrogram complete the views.

```
fcm = cl.FuzzyCMedoids(3, metric="precomputed", random_state=0).fit(D)
Y_fuzzy, Z_proto = viz.fuzzy_sammon(D, fcm.memberships_,
    fcm.medoid_indices_)
```

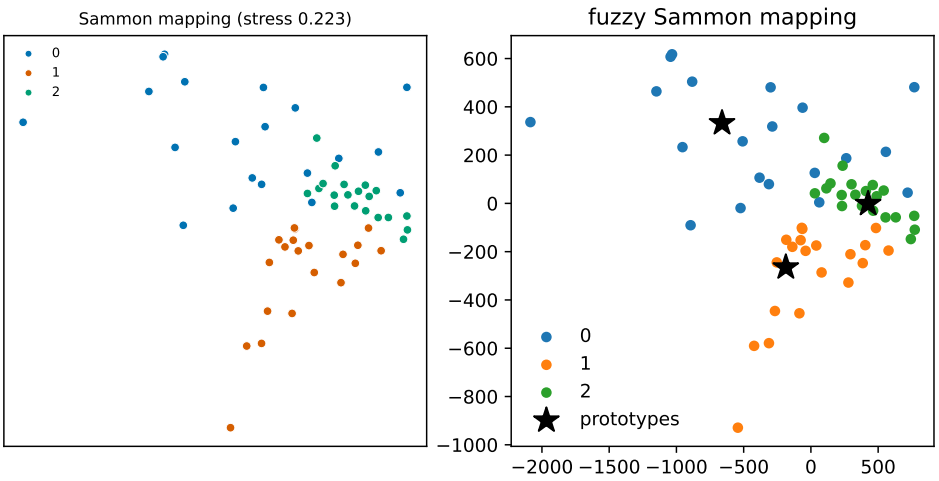


Figure 9.3. Sammon map (left) and fuzzy Sammon map (right, with the medoid prototypes as stars) of DTW distances of Cylinder–Bell–Funnel series, coloured by class. Produced by `examples/07_clustering/plot_02_hierarchical_maps.py`.

9.4 Features, segments and trajectories

Features. Clustering the feature vectors or path signatures of section 3.4 with any vector clusterer is fast, handles series of different lengths and gives interpretable clusters (`FeatureClustering`).

Segments. Clustering the *segments* of many series instead of whole series yields a vocabulary of operating patterns: every series becomes a sequence of pattern labels, and the patterns can be compared across batches, shifts or units (`SegmentClustering`). The segment models are the Gaussian or PCA descriptors of section 6.2, compared by the 2-Wasserstein distance or by PCA similarity in the spirit of Singhal and Seborg [93]. In fig. 9.4, six batches cut from one process with three recurring regimes are segmented and their segments clustered; the sample-level agreement of the pattern labels with the true regimes reaches an adjusted Rand index of 0.90.

```
sc = cl.SegmentClustering(
    segmenter=sg.BottomUp("normal", penalty=60, initial_size=20),
    n_clusters=3
).fit(C)
for i, seg in enumerate(sc.segmentations_):
    print(
        f"series {i}: {seg.n_segments} segments, patterns
    {seg.labels.tolist()}"
    )
pred = np.concatenate(
    [seg.to_labels(states=True) for seg in sc.segmentations_]
)
true = np.concatenate(
```

```
    [
        mv.Segmentation(c, len(x), labels=st).to_labels(states=True)
        for x, (c, st) in zip(C, series, strict=True)
    ]
)
print(
    "sample-level agreement with the true regimes:",
    clustering_scores(true, pred),
)
```

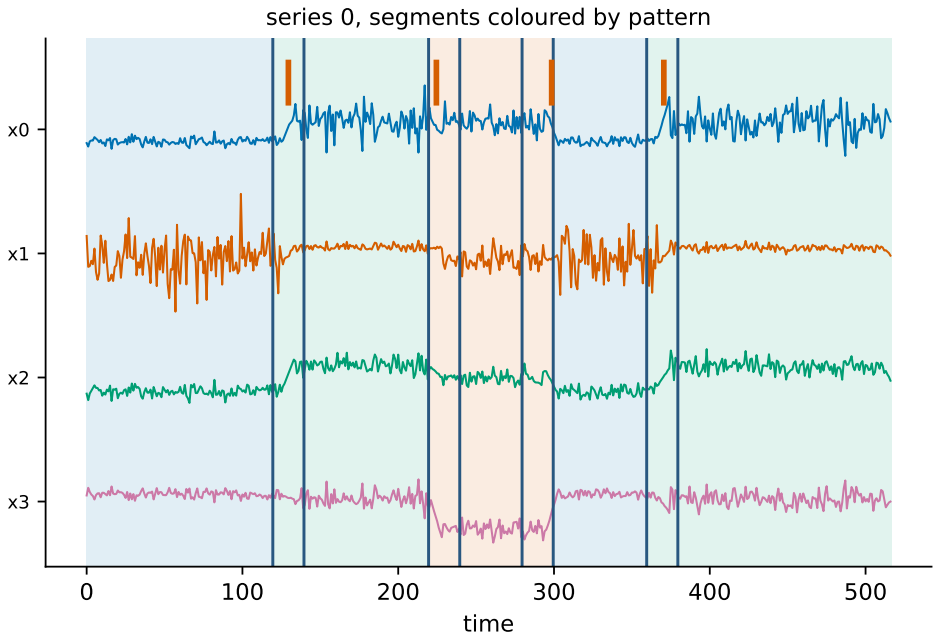


Figure 9.4. The segments of one batch, coloured by the pattern they were assigned to when the segments of all six batches were clustered together. Produced by `examples/07_clustering/plot_04_segment_clustering.py`.

Trajectories. When many objects – units, sensors, customers, countries – move in a common phase space, two questions arise: which objects follow similar paths, and which stay neighbours over time. Tarcsay, Abonyi and Németh answer the second with the neighbourhood preservation of the objects: the share of time steps at which two objects are among each other's k nearest neighbours [19]. DBSCAN on one minus this similarity yields groups of co-moving objects, and repeating the analysis in consecutive windows shows how the groups evolve (`TrajectoryClustering`; `method="trajectory"` clusters by the mean distance of whole trajectories instead).

In a nutshell Clustering

Partitional methods need a centre: the mean, the DBA or soft-DTW barycenter, the k-Shape centroid, or a medoid; k-Shape is the strongest simple baseline. Relational methods – hierarchical, spectral, density-based, consensus, *k*-medoids and fuzzy *c*-medoids – work with any distance matrix; Ward's criterion and medoids give balanced partitions where average linkage isolates outliers. Sammon and fuzzy Sammon maps show distance matrices and fuzzy clusterings. Features and signatures, segments and trajectories are clustered with the same tools. In `mvtsa`: `TimeSeriesKMeans`, `KShape`, `KMedoids`, `FuzzyCMedoids`, `HierarchicalClustering`, `SpectralClustering`, `DensityClustering`, `ConsensusClustering`, `FeatureClustering`, `SegmentClustering`, `TrajectoryClustering`, `sammon`, `fuzzy_sammon`.

```
tc = cl.TrajectoryClustering(method="neighborhood", k=6, window=20).fit(P)
print("labels:", tc.labels_)
print(
    "labels per 20-step window:",
    [lab.tolist()[:6] for lab in tc.labels_window_],
)
```

CHAPTER

10



Evaluation and validation

A segmentation, an alignment or a clustering is a hypothesis about the data. This chapter shows how to test it: with metrics that compare it with a ground truth when one exists, with retrieval curves for distance measures, with stability under perturbations when no truth is available, and with statistical tests when many methods are compared on many data sets. It closes with the systematic tuning of parameters. Every metric has blind spots, so the chapter also shows how the metrics react to typical mistakes.

10.1 Segmentation metrics

Let $\mathcal{T}$ be the true and $\mathcal{P}$ the predicted change points of a series of length T.

Detection. A predicted change point is a true positive if it lies within a margin M of a true one that has not been matched yet; greedy matching in the order of the distances makes the matching one-to-one. Precision, recall and their harmonic mean, the F1 score, follow [31] (`change_point_scores`).

Overlap. The covering of the true segmentation G by the predicted one G',

$$C(G, G') = \frac{1}{T} \sum_{A \in G} |A| \max_{A' \in G'} \frac{|A \cap A'|}{|A \cup A'|}, \tag{10.1}$$

weights the best Jaccard overlap of every true segment by its length [31, 138] (`covering`). It needs no margin and degrades gracefully with small shifts.

Boundary error. The Hausdorff distance

$$H(\mathcal{T}, \mathcal{P}) = \max\left\{ \max_{t \in \mathcal{T}} \min_{p \in \mathcal{P}} |t - p|,\ \max_{p \in \mathcal{P}} \min_{t \in \mathcal{T}} |t - p| \right\} \tag{10.2}$$

is the worst boundary error; the annotation error $||\mathcal{P}| - |\mathcal{T}||$ compares the numbers of change points; the mean boundary error averages the distances of matched pairs.

States. When segments carry state labels, the adjusted Rand index and the adjusted mutual information of the per-sample labels measure how well the states are recovered, independently of the label names (`state_scores`). `evaluate_segmentation` returns all of these at once.

Figure 10.1 and the table printed by the example show how the metrics react to typical mistakes. A shift of 15 samples is a complete failure for the F1 score with a margin of 10 but costs little covering; a missed and an extra change point cost similar F1 but very different Hausdorff distances; a flood of spurious change points keeps the recall at one while precision and covering collapse. No single number describes a segmentation; F1 with a sensible margin and the covering together are a good default, and recent work proposes interpretable measures that also classify the kind of error [139].

```
rows = [
    {"case": k, **evaluate_segmentation(truth, v, T, margin=10)}
    for k, v in cases.items()
]
table = pd.DataFrame(rows)[
    [
        "case",
        "precision",
        "recall",
        "f1",
        "covering",
        "hausdorff",
        "annotation_error",
    ]
]
print(table.round(3).to_string(index=False))
```

10.2 Clustering metrics

With true labels, a clustering is scored by the adjusted Rand index, the adjusted and normalized mutual information, the purity and the accuracy after the best matching of clusters to classes (`clustering_scores`). Without labels, internal indices compare the compactness and the separation of the clusters on the distance matrix: the silhouette and the Dunn index (`silhouette`, `dunn_index`); for fuzzy partitions the partition coefficient and the partition entropy measure how crisp the memberships are (`partition_coefficient`, `partition_entropy`).

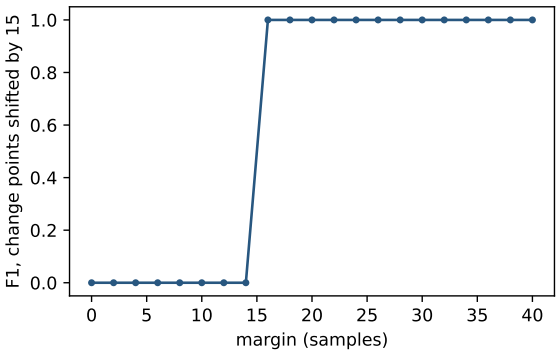


Figure 10.1. F1 score of change points shifted by 15 samples as a function of the margin: the score jumps from zero to one when the margin reaches the shift. Produced by `examples/08_evaluation/plot_01_metrics.py`.

10.3 Retrieval

A distance measure is evaluated by how well it retrieves objects of the same class. The leave-one-out 1-NN accuracy is the standard number (`knn_accuracy`). Bankó used a finer view [8]: for every query the other objects are ranked by distance, and when the r-th object of the query's class is reached after k retrieved objects, the precision at recall level r is r/k; averaging over the queries gives a precision–recall curve (`precision_recall_curve_knn`, fig. 8.3). The mean average precision summarizes the curve in one number (`mean_average_precision`); it rewards measures that rank *all* objects of a class high, not only the nearest one.

10.4 Stability

Without ground truth, a segmentation can still be validated by its stability. The series is perturbed many times – Gaussian noise of a relative level is added and a random block of a given fraction of the series is segmented – and the change points of every run are mapped back to global time. The boundary frequency, the agreement profile (6.4) over the runs, measures how often a change point is found near every sample; the frequency at a change point of the clean segmentation is its *support*, and the mean support is the stability (`segmentation_stability`). The same idea, with resampled objects instead of perturbed series, validates clusterings [140] (`clustering_stability`).

Stability is informative but not a substitute for a selection rule. With too few segments the boundaries wander between runs; once all true changes are found the support saturates. The extra change points of an over-segmentation can be stable too, if they sit at consistent features of the data – in fig. 10.2 right next to the true changes.

```
good = segmentation_stability(
    sg.BottomUp("normal", n_segments=5, initial_size=10),
    X,
    n_runs=20,
    random_state=0,
)
over = segmentation_stability(
    sg.BottomUp("normal", n_segments=9, initial_size=10),
    X,
    n_runs=20,
    random_state=0,
)
print(
    "5 segments: support",
    good["support"].round(2),
    "stability",
    round(good["stability"], 2),
)
print(
    "9 segments: support",
    over["support"].round(2),
    "stability",
    round(over["stability"], 2),
)
print(
    "change points with 9 segments:",
    over["reference"].change_points,
    "| true:",
    cps,
)
```

10.5 Comparing methods

A claim that one method is better than another needs many data sets and a statistical test. The `Benchmark` class runs every method on every data set, records the metrics and the run times, and keeps going when a method fails. The Friedman test checks whether the average ranks of k methods on N data sets differ at all; if they do, the Nemenyi test declares two methods different when their average ranks differ by more than the critical difference

$$\mathrm{CD} = q_\alpha \sqrt{\frac{k(k+1)}{6N}}, \tag{10.3}$$

where q_α is the studentized range statistic divided by $\sqrt{2}$ [141] (`friedman_nemenyi`). The critical-difference diagram draws the ranks with bars joining methods that do not differ significantly (fig. 10.3). Because post-hoc tests on mean ranks depend on the other methods of the comparison [142], pairwise Wilcoxon signed-rank tests with Holm's correction are provided as well (`wilcoxon_holm`). Ten data sets are too few to separate eight methods: the example finds the

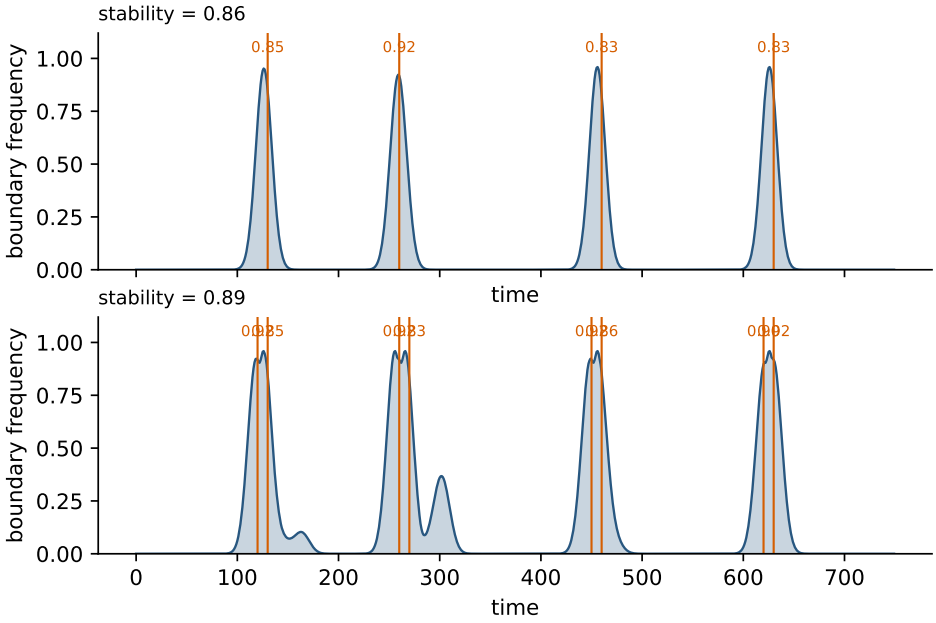


Figure 10.2. Boundary frequency over twenty perturbed runs for five (top) and nine segments (bottom), with the support of every change point of the clean segmentation. Produced by `examples/08_evaluation/plot_02_stability.py`.

Friedman test highly significant, but only the largest rank differences exceed the critical difference.

```
stat = friedman_nemenyi(bench.pivot("covering"))
print(
    f"Friedman p = {stat['p_value']:.2g}, CD =
    {stat['critical_difference']:.2f}"
)
print(
    wilcoxon_holm(bench.pivot("covering"))
    .head(5)
    .round(4)
    .to_string(index=False)
)
```

10.6 Tuning

Parameters are tuned against labelled series by a grid search on a segmentation metric averaged over the series (`grid_search_segmenter`), or without labels by maximizing stability (`criterion="stability"`). A one-parameter sensitivity study shows how robust a method is: `sensitivity` runs a factory over a range of values and records the number of segments, the change points, the run time and

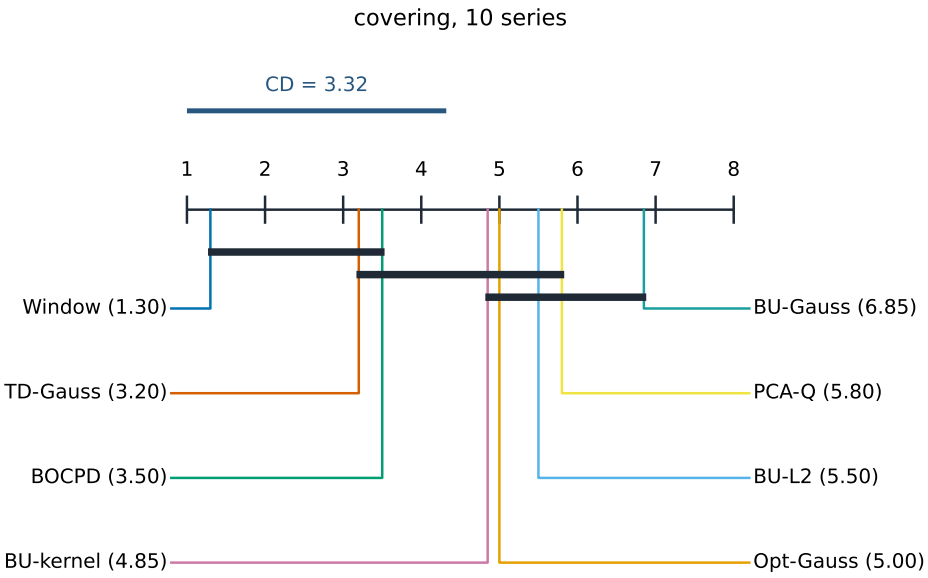


Figure 10.3. Critical difference diagram of eight segmenters on ten series (covering). Methods joined by a bar do not differ significantly by the Nemenyi test. Produced by `examples/08_evaluation/plot_03_benchmark.py`.

an optional score. The example scripts of the package end with such studies, and the parameters of the book's figures come from them.

```
best, table = grid_search_segmenter(
    lambda **p: sg.PCASegmenter(n_segments=4, **p),
    {
        "n_components": [1, 2, 3],
        "statistic": ["Q", "T2", "both"],
        "initial_size": [5, 10, 20],
    },
    train,
    criterion="covering",
)
print("best:", best)
print(table.head(6).round(3).to_string(index=False))
```

In a nutshell **Evaluation and validation**

Change points are scored by F1 within a margin, by the covering of the true segments, by the Hausdorff distance and by the annotation error; states and clusterings by ARI and AMI; distance measures by 1-NN accuracy, precision–recall curves and mean average precision. Stability under noise and block resampling validates segmentations without ground truth. Methods are compared over many data sets with the Friedman test, the Nemenyi critical difference and Wilcoxon–Holm tests; parameters are tuned by grid search or stability. In `mvtsa`: `evaluate_segmentation`, `clustering_scores`, `knn_accuracy`, `precision_recall_curve_knn`, `mean_average_precision`, `segmentation_stability`, `Benchmark`, `friedman_nemenyi`, `wilcoxon_holm`, `grid_search_segmenter`, `sensitivity`.

CHAPTER

11



Visualization and explanation

> A change point is useful only if the analyst understands what changed; a regime only if the operator can recognize it. This chapter collects the tools that make results readable: a one-page segmentation report, the contributions of the variables to every change, tables of segment statistics, decision-tree rules that describe the states, regime transition networks, zone-labelled visibility graphs that connect time series to statistical process control, and views of alignments and collections.

11.1 A segmentation report

`segmentation_dashboard` puts on one page what an analyst wants to see about a segmentation: the segmented series with the true change points if known, the timeline of the states, the cost curve of the merge hierarchy with the relative reduction rate (or the segment lengths), and the contributions of the variables to every change (fig. 11.1). The other views of the package – `plot_segmentation` with a piecewise constant or linear fit, `plot_fuzzy_segmentation` with the memberships, `plot_segmentogram`, `plot_pca_statistics`, `plot_segment_loadings`, `plot_run_length`, `plot_profile`, `plot_boundary_agreement` – appear throughout the book.

```
st = sg.StateSegmenter(
    segmenter=sg.BottomUp("normal", n_segments=6, initial_size=15),
    n_states=3
)
seg = st.segment(X)
tree = sg.BottomUp("normal", initial_size=15).full_tree(X)
mv.viz.segmentation_dashboard(X, seg, names=names, truth=cps, tree=tree)
```

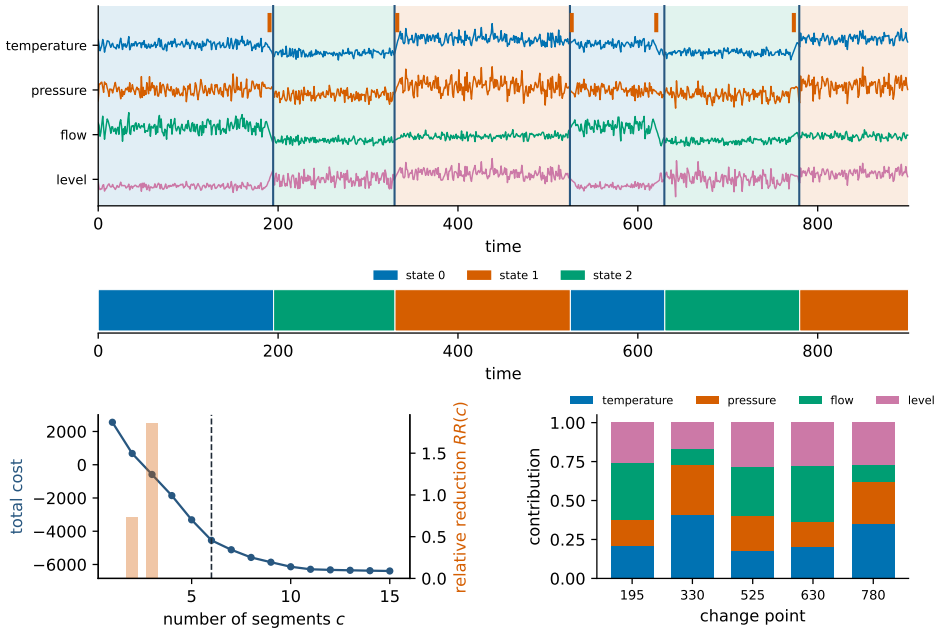


Figure 11.1. A one-page segmentation report of a process with three recurring regimes: segmented series and true change points (top), state timeline, cost curve with the relative reduction rate, and the contributions of the four variables to every change. Produced by `examples/09_visualization/plot_01_dashboard.py`.

11.2 What changed?

Contribution plots of multivariate statistical process control attribute an alarm to variables [23]; `boundary_contributions` does the same for every change point of a segmentation. For each boundary it compares the two adjacent segments (or windows around the boundary) by the standardized shift of every variable's mean, $|\bar{x}_{\text{right}} - \bar{x}_{\text{left}}|/s$, the change of its variance, $|\log(s^2_{\text{right}}/s^2_{\text{left}})|$, and the change of its correlations with the others (the row norm of the difference of the correlation matrices); `kind="all"` normalizes the three to unit sum and adds them. `describe_segments` tabulates the statistics of every segment – start, stop, means, standard deviations, slopes and the explained variance of the local PCA model – so that segments can be compared side by side.

```
print(boundary_contributions(X, seg, kind="all", names=names).round(2))
print(describe_segments(X, seg, names=names).round(2).iloc[:, :8])
```

11.3 Rules for states and transition networks

Varga, Szeifert, Réti and Abonyi described the operating regimes of an industrial reactor by decision trees: a shallow tree trained to predict the regime from the process variables turns hidden knowledge into a readable rule base [17]. `explain_states` fits such a tree to the state labels of a segmentation and returns the rules, their training accuracy and the importances of the variables; two levels are usually enough to describe a handful of regimes in terms an operator can check (section 13.3). `transition_graph` builds the regime transition network: states are nodes with their number of visits and total and mean durations, observed transitions are edges with their counts and probabilities (`plot_transition_graph`).

11.4 Six-sigma visibility graphs

Visibility graphs turn a series into a network (section 3.5), but the meaning of their network measures is often unclear. Ferenczi, Kummer and Abonyi connect them to statistical process control [20]: the nodes of the horizontal visibility graph are labelled with the zones of a Shewhart chart – C within one standard deviation of the centre line, B within two, A within three, and out-of-control beyond – and with the violations of the Western Electric rules [143] (one point beyond three sigma; two of three beyond two sigma on one side; four of five beyond one sigma on one side; eight in a row on one side). The edges carry the zone pairs they connect, so the zone-transition statistics of the graph summarize the behaviour of the process in SPC terms (`six_sigma_visibility_graph`, `western_electric_rules`, `spc_zones`). In fig. 11.2 the global features of the graphs of an in-control and a drifting process are nearly identical – the mean degree of a horizontal visibility graph of an uncorrelated series is close to four whatever its distribution [65] – while the zone labels, the edge statistics and the rule flags reveal the drift at once.

```
G_ok = rp.six_sigma_visibility_graph(in_control, center=0, sigma=1)
G_drift = rp.six_sigma_visibility_graph(drifting, center=0, sigma=1)
rows = []
for name, G in [("in control", G_ok), ("drifting", G_drift)]:
    f = rp.graph_features(G)
    rules = {
        k: sum(G.nodes[i][k] for i in G.nodes)
        for k in ("rule1", "rule2", "rule3", "rule4")
    }
    rows.append(
        {
            "process": name,
            **{
                k: f[k]
                for k in ("mean_degree", "degree_entropy", "clustering")
            },
            **rules,
```

```
        }
    )
print(pd.DataFrame(rows).round(3).to_string(index=False))
```

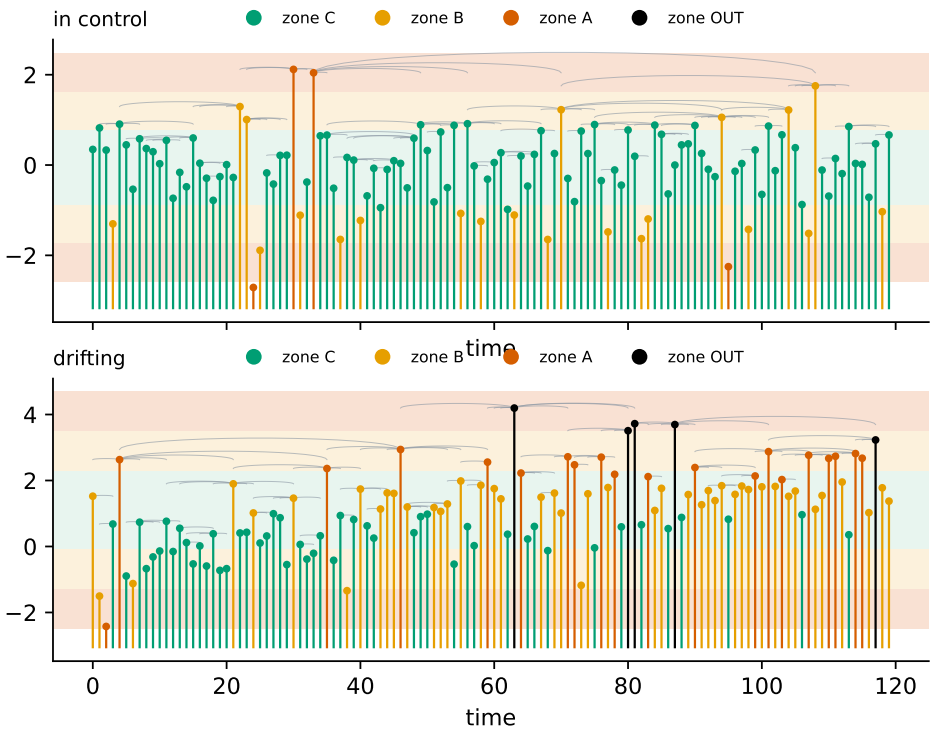


Figure 11.2. Six-sigma visibility graphs of an in-control (top) and a drifting process (bottom): samples coloured by their control-chart zone, visibility arcs above them, zones shaded. Produced by `examples/09_visualization/plot_02_visibility_graphs.py`.

11.5 Alignments and collections

For warping, the views of the package show an alignment as connected samples (`plot_alignment`), the cost matrix with the path and the window (`plot_cost_matrix`), the expected alignment of soft-DTW (`plot_alignment_uncertainty`), the warping functions of several pairs (`plot_warping_function`), segment-level alignments (`plot_segment_alignment`) and averages (`plot_average`). For collections, they show distance matrices ordered by cluster, Sammon, MDS and t-SNE maps, dendrograms and cluster prototypes (fig. 11.3).

In a nutshell **Visualization and explanation**

A one-page report shows a segmentation, its states, its cost curve and the contributions of the variables to every change. Contributions split each change into shifts of means, variances and correlations; segment tables, decision-tree rules and transition networks describe segments and regimes in operators' terms. Six-sigma visibility graphs label the nodes of a visibility graph with control-chart zones and Western Electric rules. Alignment, cost-matrix, uncertainty and map views make warping and clustering transparent. In `mvtsa`: `segmentation_dashboard`, `boundary_contributions`, `describe_segments`, `explain_states`, `transition_graph`, `six_sigma_visibility_graph`, and the `plot_*` functions of `mvtsa.viz`.

```
a, b = G[0], G[int(np.flatnonzero(y == y[0])[1])]
res = wp.dtw_path(a, b, window=0.1)
_, E = wp.soft_dtw_alignment(a, b, gamma=0.05)
```

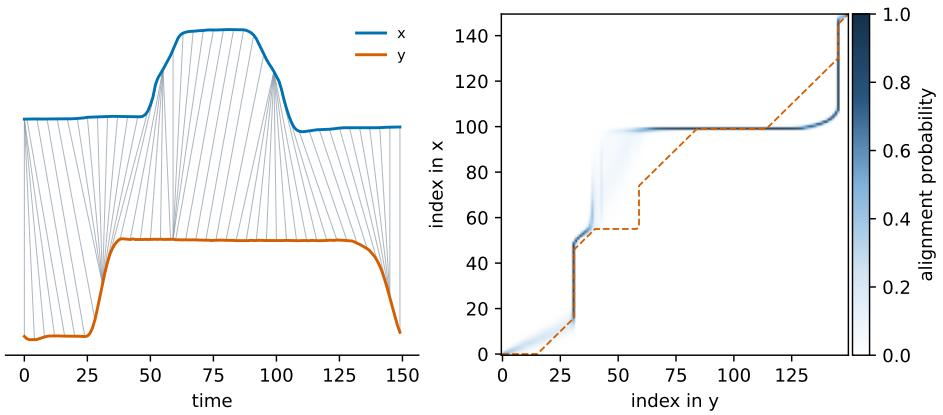


Figure 11.3. Two GunPoint series of the same class [144]: the DTW alignment with a 10 % window (left) and the expected alignment of soft-DTW with $\gamma = 0.05$ (right). Produced by `examples/09_visualization/plot_04_warping_maps.py`.

CHAPTER

12



Model operations

> An analysis that runs once in a notebook is an experiment; an analysis that runs every day on new data is a model in operation, and it has to be reproducible, versioned, monitored and documented. This chapter shows how `mvtsa` supports the life cycle of time-series models: configurations as code, an analysis pipeline that learns regimes and labels new data, model files with integrity checks, a registry with stages, experiment tracking, data contracts, drift monitoring and model cards. The layer is deliberately light – files and JSON – and mirrors to MLflow when it is available [145].

12.1 The life cycle

Figure 12.1 shows the life cycle the package supports. An analysis is designed and tuned on historical data (chapter 10); its configuration is written to a file; the fitted model is saved with its metadata and registered; the production version labels new data; a monitor compares every new batch with the reference data and raises an alert when the data drift; the alert triggers a new round of analysis. Every arrow of the figure is a function of the package.

12.2 Configurations and pipelines

Every object of the package – segmenters, costs, warping models, clusterers, pipelines, nested ensembles – is written to a plain dictionary by `to_config` and rebuilt by `from_config`; `save_config` and `load_config` store the dictionary as JSON or YAML. A configuration file under version control is the reproducible description of an analysis. `AnalysisPipeline` is the deployable unit: it standardizes the data, segments them, learns recurring regimes by clustering the segment

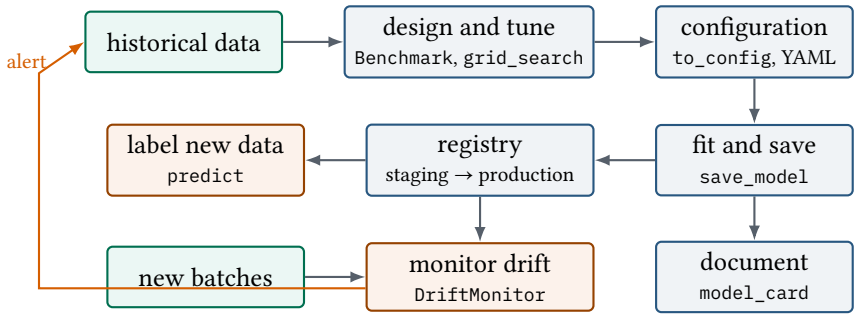


Figure 12.1. Life cycle of a time-series model in `mvtsa`: design on historical data, configuration as code, fitted and registered models, production use, drift monitoring that triggers a new analysis, and documentation.

models of the training data (Gaussian descriptors and the 2-Wasserstein distance by default) and labels the segments of new data with the nearest regime model.

```
pipe = mlops.AnalysisPipeline(
    sg.BottomUp("normal", n_segments=8, initial_size=15), n_regimes=3
)
cfg = mlops.to_config(pipe)
print(mlops.save_config(pipe, workdir / "pipeline.yaml").read_text())
same = mlops.from_config(cfg)  # an identical, unfitted pipeline
```

12.3 Model files, registry and tracking

`save_model` writes a model directory with the pickled object, its configuration and a metadata file: the versions of the package and its libraries, the creation time, a fingerprint of the training data, metrics and tags, and a SHA-256 hash of the model file that `load_model` verifies, so that a corrupted or replaced file is detected before it is used. `ModelRegistry` keeps the versions of named models in a folder with stages – staging, production, archived – and loads the production version by name. `ExperimentTracker` records runs with their parameters, metrics, figures and models in local JSON files and mirrors them to MLflow when it is installed.

```
pipe.fit(X_train)
seg = pipe.predict(X_new)
print(
    "regimes of the new data:",
    seg.labels.tolist(),
    "| true regimes:",
    states[8:].tolist(),
)
reg = mlops.ModelRegistry(workdir / "registry")
v1 = reg.register(
    "regimes",
    pipe,
    data=X_train,
```

```
    metrics={"covering": 0.97},
    tags={"plant": "demo"},
)
reg.promote("regimes", v1, "production")
print(reg.list()[["name", "version", "stage", "created"]])
production = reg.load("regimes", stage="production")
```

12.4 Contracts and drift

A model is valid only while the data look like the training data. A `DataContract` inferred from the reference data checks the schema (number and names of the variables), missing values and ranges of new data before anything else. The `DriftMonitor` checks every new batch against the reference in several ways: the fractions of samples beyond the T^2 and Q limits of the reference PCA model (expected: the significance level), the Krzanowski similarity of the batch PCA model to the reference model, the largest per-variable Kolmogorov–Smirnov statistic, the population stability index of every variable,

$$\mathrm{PSI} = \sum_b \left(q_b - p_b\right) \log \frac{q_b}{p_b}, \tag{12.1}$$

with p_b and q_b the reference and current shares of bin b, and optionally the distribution of segment durations under a segmenter. An alert is raised when an exceedance rate is several times the significance level, the similarity falls below a threshold or a PSI exceeds 0.25. In fig. 12.2 the correlation structure of the process changes after the reference period; the Q exceedance and the similarity react in the first batch after the change.

```
mon = mlops.DriftMonitor(n_components=1, similarity_threshold=0.8).fit(
    reference
)
for a in range(300, len(X), 100):
    r = mon.update(X[a : a + 100])
    print(
        f"batch {a:4d}-{a + 100:4d}: Q exceed {r['q_exceedance']:.2f}  "
        f"similarity {r['similarity']:.2f}  "
        f"alerts {r['alerts']}"
    )
```

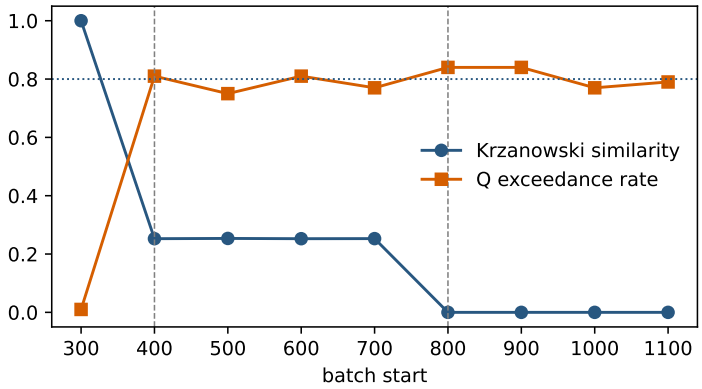


Figure 12.2. Batch-wise drift monitoring: Krzanowski similarity of every batch model to the reference model (circles) and the rate of Q exceedances (squares); the true changes of the correlation structure are dashed. Produced by `examples/10_mlops/plot_02_drift_monitoring.py`.

12.5 Model cards and the command line

A model card documents what a model is for, on which data it was evaluated, how well it performed and where its limits are [146]; `model_card` writes one in Markdown from the model's configuration, the metrics and free-text fields. Finally, the `mvtsa` command line segments a CSV file with any registered method and writes the segmentation and a plot, computes the distance matrix of a folder of series, monitors a file against a saved reference, and reports the installation – the entry points for scheduled jobs.

```
card = mlops.model_card(
    sg.PCASegmenter(n_segments=5, n_components=1),
    name="PCA segmentation of unit 7",
    purpose="Detect changes of the correlation structure of four process
     variables.",
    data_description="Synthetic data with four correlation switches.",
    metrics={"f1": 1.0, "covering": 1.0},
    limitations="Blind to pure mean shifts; assumes one latent variable.",
    path=workdir / "MODEL_CARD.md",
)
print(card)
```

In a nutshell **Model operations**

Configurations as code make analyses reproducible; `AnalysisPipeline` learns regimes and labels new data; model files carry versions, data fingerprints and integrity hashes; the registry manages versions and stages; the tracker records experiments locally and in MLflow; data contracts and the drift monitor – T^2/Q exceedances, PCA similarity, KS, PSI, segment durations – tell when a model is no longer valid; model cards document it. In `mvtsa`: `to_config`, `from_config`, `AnalysisPipeline`, `save_model`, `load_model`, `ModelRegistry`, `ExperimentTracker`, `DataContract`, `DriftMonitor`, `model_card`, the `mvtsa` command.

CHAPTER

13



Case studies

Three case studies combine the methods of the book on process data. The first analyses a campaign of grade changes of a polymerization reactor: where the steady grades and the transitions are, which grade the reactor is in, and which periods identify its kinetics. The second identifies the operating regions of a pH neutralization process by their local dynamics. The third turns the recurring regimes of a process into rules, a transition network and explanations of every switch. Together they show the typical workflow: segment, label, explain, and select data for modelling.

13.1 Grade transitions of a polymerization reactor

The isothermal free-radical polymerization of methyl methacrylate in a continuous stirred tank [90] is the case study of Dobos, Bankó and Abonyi [11, 12]. The reactor has four states – the monomer and initiator concentrations and the zeroth and first moments of the dead polymer chains – one input, the initiator flow F_I, and one output, the number-average molecular weight $M_n = D_1/D_0$; the nominal flow $F_I = 0.016783\,\mathrm{m^3/h}$ gives $M_n \approx 25\,000\,\mathrm{kg/kmol}$. `simulate_mma_cstr` integrates the model with the parameters of `MMA_PARAMETERS`. The campaign of fig. 13.1 produces three grades in the order A, B, C, B, A, C, A, with ramped transitions and small moves of the operators.

Grades and transitions. A penalized bottom-up segmentation with the Gaussian cost followed by the clustering of the segment models into three states labels the record (`StateSegmenter`). The state sequence reads A, B, C, B, A, (B), C, (B), A: the two direct transitions between A and C pass through the molecular weight of grade B and are labelled B, which is correct from the point of view of the product and a reminder that a transition is not a grade. The transition network gives the hours spent in every grade.

```
seg = sg.StateSegmenter(
    segmenter=sg.BottomUp("normal", penalty=40, initial_size=10),
    n_states=3
).segment(X)
print("segments:", seg.n_segments, "| grade sequence:",
    seg.labels.tolist())
G = transition_graph(seg, sampling_period=0.05)
print(
    {v: round(G.nodes[v]["total_duration"], 1) for v in G.nodes},
    "hours per state",
)
```

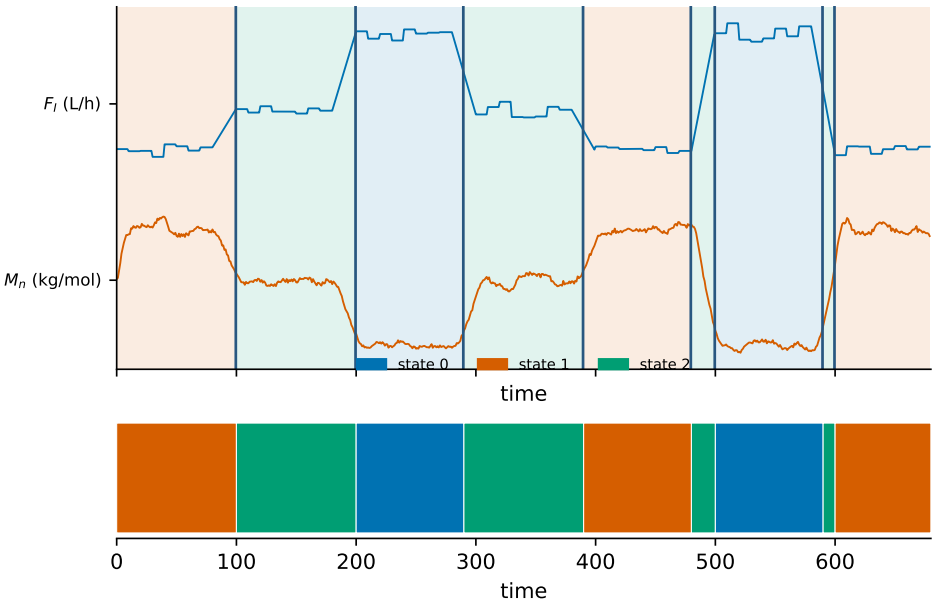


Figure 13.1. A campaign of grade changes of the MMA reactor: initiator flow and molecular weight with the segments coloured by grade (top) and the grade timeline (bottom). Produced by `examples/11_applications/plot_01_polymer_grades.py`.

Informative periods. Which parts of the record identify the propagation and initiator decomposition rate constants? The Fisher-information segmentation of section 5.5, with sensitivities computed by finite differences of the simulator, ranks the transitions far above the steady grades (fig. 13.2). An identification data set built from the transitions is short and informative; long steady operation adds little – the conclusion of the segmentation-based experiment design of Dobos et al.

```
names = ["kp", "kI"]
p0 = np.array([MMA_PARAMETERS[k] for k in names])
S = sg.finite_difference_sensitivities(
    lambda p: simulate_mma_cstr(FI, params=dict(zip(names, p,
    strict=True))),
    p0,
```

```
    rel_step=1e-4,
)
fs = sg.FisherSegmenter(
    n_segments=seg.n_segments, criterion="logD", window=20, initial_size=10
).fit(S)
print(
    fs.ranking_[["start", "stop", "logD"]]
    .head(5)
    .round(2)
    .to_string(index=False)
)
```

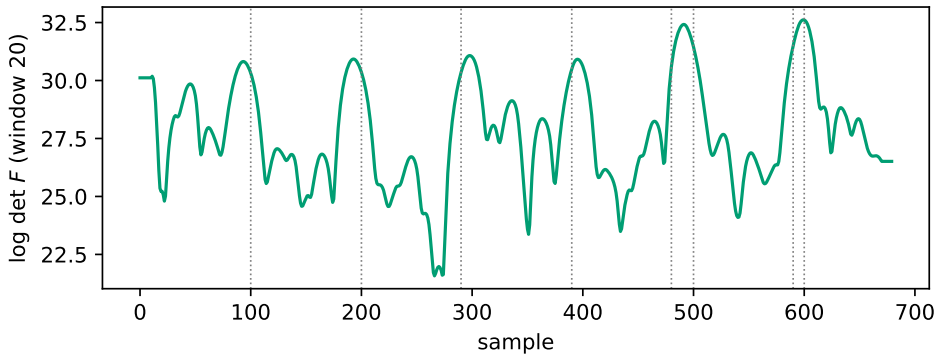


Figure 13.2. Windowed log det **F** of the kinetic parameters k_p, k_I along the campaign of fig. 13.1, with the grade boundaries. Produced by `examples/11_applications/plot_01_polymer_grades.py`.

13.2 Operating regions of a pH process

The neutralization of a strong acid by a strong base in a stirred tank [80] is the standard example of a process whose gain depends on the operating point: around neutrality a small change of the base flow moves the pH by several units, far from it hardly at all. `simulate_ph_cstr` solves the component balances and the charge balance. Segmenting the record by its local dynamics with dynamic PCA (section 5.2, lag selected automatically) and fitting a first-order ARX model $y_{k+1} = a\,y_k + b\,u_k + c$ to every segment gives the local static gain $b/(1-a)$ and the pole a of each operating region (fig. 13.3). The gains near pH 6.6 are an order of magnitude larger than at the extremes – the information a gain-scheduled controller needs.

```
seg = sg.DPCASegmenter(n_segments=8, lags="auto",
    n_components=0.99).segment(P)
rows = []
for a, b in seg.segments:
    u, y = P[a:b, 0], P[a:b, 1]
    A = np.column_stack([y[:-1], u[:-1], np.ones(b - a - 1)])
```

```
    (a1, b1, _), *_ = np.linalg.lstsq(A, y[1:], rcond=None)
    rows.append(
        {
            "start": a,
            "stop": b,
            "mean pH": y.mean(),
            "local gain": b1 / (1 - a1) if a1 < 1 else np.nan,
            "pole": a1,
        }
    )
print(pd.DataFrame(rows).round(3).to_string(index=False))
```

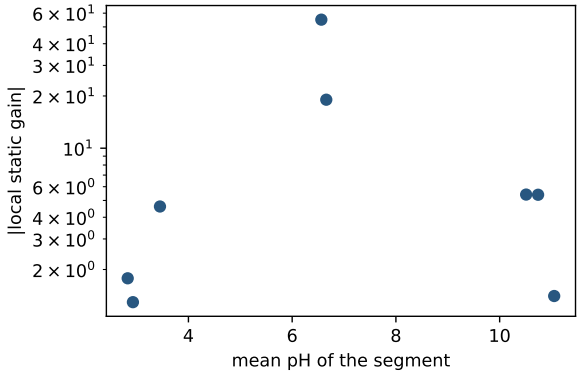


Figure 13.3. Absolute local static gain of the first-order models of the segments against the mean pH of the segment: the gain peaks around neutrality. Produced by `examples/11_applications/plot_02_ph_process.py`.

13.3 Explaining operating regimes

The last case study starts from a record of four process variables that switches among three regimes. A sticky HMM labels the regimes; a two-level decision tree describes them by the variables with a training accuracy of 0.98 (section 11.3); the transition network shows that every regime can follow every other; and the boundary contributions tell which variables moved at each switch (fig. 13.4). The rules are the part an operator keeps: they turn an unsupervised segmentation into a vocabulary of regimes that can be checked against experience and used in an operator support system, the purpose of the decision-tree analysis of Varga et al. [17].

```
ex = explain_states(X, seg, names=names, max_depth=2)
print(f"training accuracy of the rules: {ex['accuracy']:.3f}")
print(ex["rules"])
print(ex["importances"].round(3))
```

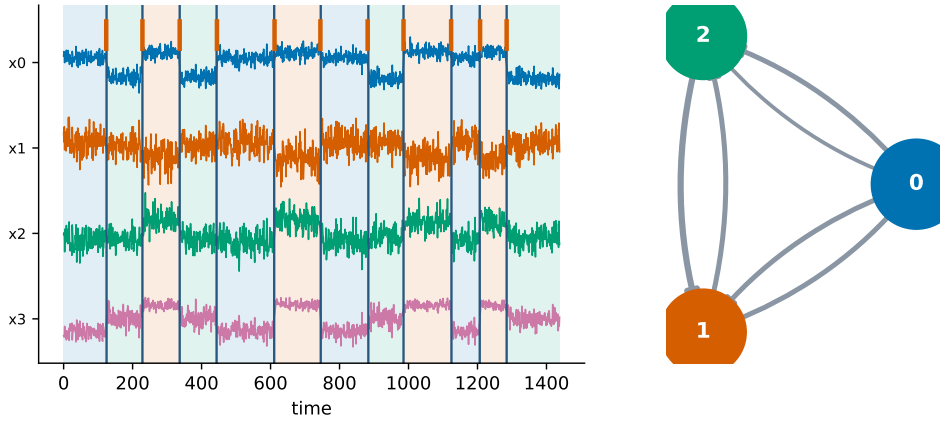


Figure 13.4. Regimes of a process found by a sticky HMM (left) and their transition network (right; node size: time spent in the regime, edge width: transition probability). Produced by `examples/11_applications/plot_03_regime_explanation.py`.

In a nutshell Case studies

In a polymer grade campaign, state segmentation labels grades and transitions and Fisher-information segmentation shows that the transitions identify the kinetics. In a pH process, dynamic-PCA segmentation and local ARX models reveal the operating-point dependence of the gain. In a regime record, a sticky HMM, decision-tree rules, a transition network and boundary contributions make the regimes understandable. The common workflow: look, segment, label and compare, evaluate, explain, operate.

13.4 A workflow

The case studies follow the same workflow, which is also a summary of the book. *Look* at the data and choose what homogeneity means: levels, correlation structure, dynamics, relations, shapes. *Segment* with a matching cost or model, inspect the merge hierarchy, and choose the number of segments with several rules and a stability check. *Label* recurring segments as states and *compare* series by warping their segment sequences or samples. *Evaluate* against whatever ground truth exists, and compare methods statistically. *Explain* the result by contributions, rules and transition networks. *Operate* it: write the configuration, save and register the model, monitor drift, document the model. Each step has a section of this book and a function of the package.

APPENDIX

A



API reference

This appendix lists every public class and function of `mvtsa`, module by module, with the first sentence of its docstring. The lists are generated from the `__all__` attribute of each module by `book/build_api_appendix.py`, so they are complete by construction; the full signatures, parameters, and attributes are in the online documentation and in the docstrings (`help(mvtsa.segmentation.PCASegmenter)`). The conventions of the API are described in section 2.6, the notation of the docstrings in table 2.1.

A.1 `mvtsa`: Core objects and shortcuts

Name	Kind	Description
`Segmentation`	class	A segmentation of one series
`as_series`	function	Return X as a 2-D float array (T, n)
`as_collection`	function	Return a collection of series as a list of (T_i, n) arrays
`segment`	function	One-line segmentation: segment(X, "pca", n_segments=4)
`pairwise_distances`	function	Distance matrix of a collection (or between two collections)
`segmentation`	module	subpackage, see section mvtsa.segmentation
`warping`	module	subpackage, see section mvtsa.warping
`clustering`	module	subpackage, see section mvtsa.clustering
`representation`	module	subpackage, see section mvtsa.representation
`evaluation`	module	subpackage, see section mvtsa.evaluation
`viz`	module	subpackage, see section mvtsa.viz
`mlops`	module	subpackage, see section mvtsa.mlops
`datasets`	module	subpackage, see section mvtsa.datasets

Name	Kind	Description
`preprocessing`	module	subpackage, see section mvtsa.preprocessing
`distances`	module	subpackage, see section mvtsa.distances
`simplify`	module	subpackage, see section mvtsa.simplify
`explain`	module	subpackage, see section mvtsa.explain
`io`	module	subpackage, see section mvtsa.io
`__version__`	constant	str constant

A.2 `mvtsa.preprocessing`: Preprocessing

Name	Kind	Description
`zscore`	function	Z-normalize every variable (zero mean, unit standard deviation)
`minmax`	function	Scale every variable to feature_range
`robust_scale`	function	Center by the median and scale by the interquartile range
`Standardizer`	class	Fit-once, apply-many standardization (for pipelines and deployment)
`resample`	function	Resample a series to length samples by interpolation over [0, 1]
`resample_collection`	function	Resample every series of a collection to a common length
`fill_missing`	function	Fill NaN values column-wise by "linear" interpolation, "ffill" (last observation carried forward) or "mean"
`moving_average`	function	Centered moving average (edges use the available samples)
`smooth`	function	Smooth every variable: "savgol" (Savitzky–Golay), "gaussian" or "moving" average
`derivative`	function	First or second derivative per sample ("gradient" = central differences, "savgol" = Savitzky–Golay derivative filter)
`keogh_derivative`	function	Derivative estimate of Keogh and Pazzani (2001) used by derivative DTW: d_t = ((x_t - x_{t-1}) + (x_{t+1} - x_{t-1}) / 2) / 2; the end values copy their neighbo
`dynamize`	function	Lag-extended data matrix of dynamic PCA (Ku, Storer and Georgakis, 1995)
`delay_embedding`	function	Delay-coordinate embedding of a univariate series (Takens): rows [x(t), x(t+tau), ..., x(t+(dim-1)tau)]
`ami_lag`	function	Embedding delay at the first minimum of the average mutual information (Fraser and Swinney, 1986)
`false_nearest_neighbors`	function	Embedding dimension by the false nearest neighbour method of Kennel, Brown and Abarbanel (1992)

Name	Kind	Description
`kalman_smoother`	function	Linear Kalman filter and Rauch–Tung–Striebel smoother
`detrend`	function	Remove a polynomial trend of the given order from every variable

A.3 `mvtsa.representation`: Representations

Name	Kind	Description
`PCAModel`	class	Principal component model with monitoring statistics
`PPCAModel`	class	Probabilistic PCA (Tipping and Bishop, 1999)
`RecursivePCA`	class	Recursive (exponentially weighted) PCA with a variable forgetting factor
`select_dpca_lag`	function	Number of lags of dynamic PCA
`resolve_n_components`	function	Number of retained components
`krzanowski_similarity`	function	Krzanowski's PCA similarity factor (1979): $s_{PCA} = \frac{1}{p}\operatorname{tr}(U_1^{\mathsf{T}} U_2 U_2^{\mathsf{T}} U_1) = \frac{1}{p}\sum_{i,j}\cos^2\Theta_{ij}$
`lambda_similarity`	function	Eigenvalue-weighted PCA similarity factor (Johannesmeyer, 1999; Singhal and Seborg, 2002)
`eros_similarity`	function	Eros, the extended Frobenius norm similarity of Yang and Shahabi (2004): $\sum_i w_i \lvert\cos\Theta_i\rvert$ over corresponding eigenvectors
`eros_weights`	function	Eros weights (Yang and Shahabi, 2004): the normalized eigenvalue profiles averaged over a learning set, renormalized to sum to one
`principal_angles`	function	Principal angles (radians, increasing) between the column spaces of U1 and U2 (orthonormalized internally)
`grassmann_distance`	function	Geodesic (arc-length) distance on the Grassmann manifold: $\sqrt{\sum_i \theta_i^2}$
`chordal_distance`	function	Projection (chordal) distance $\sqrt{\sum_i \sin^2\theta_i}$
`mean_distance_similarity`	function	Distance similarity factor of Singhal and Seborg (2005): the probability that the Mahalanobis distance between the means is exceeded under a standard normal ref
`log_euclidean_distance`	function	$\lVert\log A - \log B\rVert_F$ (Arsigny et al., 2007)
`affine_invariant_distance`	function	$\sqrt{\sum_i \log^2\lambda_i(A^{-1}B)}$ (Pennec et al., 2006)
`bures_wasserstein_distance`	function	$\left(\operatorname{tr}A + \operatorname{tr}B - 2\operatorname{tr}(A^{1/2}BA^{1/2})^{1/2}\right)^{1/2}$, the 2-Wasserstein distance of two centered Gaussi
`gaussian_w2_distance`	function	2-Wasserstein distance between $N(\mu_1, A)$ and $N(\mu_2, B)$

Name	Kind	Description
`gaussian_jeffreys_divergence`	function	Symmetric Kullback–Leibler (Jeffreys) divergence of two Gaussians
`spd_mean`	function	Mean of SPD matrices: "log_euclidean" (closed form) or "euclidean"
`paa`	function	PAA of every variable: (n_segments, n) frame means (frames of nearly equal length when T is not divisible)
`inverse_paa`	function	Step-function reconstruction of a PAA representation
`apca`	function	APCA: bottom-up segmentation with the L2 cost (segment means are Segmentation.to_frame(X))
`PLA`	class	Piecewise linear approximation of a (multivariate) series
`pla`	function	Piecewise linear approximation
`pip`	function	Indices of the k perceptually important points of a univariate series: start with the end points and repeatedly add the point farthest from the line joining its
`SAX`	class	SAX transform with n_segments PAA frames and alphabet_size symbols
`sax`	function	Shorthand for SAX(n_segments, alphabet_size).transform(x)
`abba`	function	ABBA symbolic representation of a univariate series
`sfa`	function	SFA word of a univariate series: the first n_coefficients real/imaginary DFT values discretized by per-coefficient breakpoints (equi-depth bins learned from the
`Episode`	class	One episode: primitive code, half-open sample range and magnitude (signed change of the signal over the episode)
`triangular_episodes`	function	Episode sequence of a univariate signal
`episode_string`	function	The primitive codes of an episode sequence as a string
`size_classes`	function	Size class ("small", "medium", "large") of every episode from its duration and absolute magnitude
`multivariate_episodes`	function	Episodes of a multivariate signal: of the first principal component (method="pca", as in Balaskó et al., 2007) or of every variable ("each", returns a list of s
`PRIMITIVES`	constant	Letters of the seven qualitative episode primitives
`DIRECTION`	constant	dict constant
`QTA_SUBSTITUTION`	constant	Substitution scores of the episode primitives
`extract_features`	function	Feature dictionary of one (multivariate) series: every feature of FEATURES for every channel ("x0:mean"...) and, if cross, the correlations of the channel pairs

Name	Kind	Description
feature_matrix	function	Feature matrix of a collection as a pandas DataFrame (one row per series)
FEATURES	constant	tuple constant
permutation_entropy	function	Permutation entropy of the ordinal patterns of length order (Bandt and Pompe, 2002)
signature_features	function	Truncated path signature (levels 1..depth, depth <= 3) of a multivariate series viewed as a piecewise linear path
visibility_graph	function	Natural visibility graph (divide and conquer, $O(T \log T)$ on average)
horizontal_visibility_graph	function	Horizontal visibility graph: i < j are linked if every sample between them is lower than both
six_sigma_visibility_graph	function	SPC-zone labelled horizontal visibility graph
spc_zones	function	Control-chart zone of every sample: "C", "B", "A" within 1, 2, 3 standard deviations, "OUT" beyond the 3σ control limits (center and σ default to the sample mea
western_electric_rules	function	Boolean flags of the four Western Electric rules: beyond 3σ; 2 of 3 beyond 2σ on one side; 4 of 5 beyond 1σ on one side; 8 in a row on one side of the center
ordinal_partition_network	function	Directed, weighted transition network of the ordinal patterns of length order (nodes are patterns, e.g. "012")
recurrence_network	function	ε-recurrence network: nodes are (delay-embedded) states, linked if closer than epsilon (default: the 10 % quantile of the distances)
graph_features	function	Summary features: mean degree, degree entropy, clustering, assortativity, density; for directed graphs the mean in/out degree asymmetry; for zone-labelled graph
recurrence_plot	function	Recurrence plot
gramian_angular_field	function	Gramian angular field of a univariate series rescaled to [-1, 1] (optionally PAA-reduced to n_bins samples)
markov_transition_field	function	Markov transition field: entry (i, j) is the probability of moving from the quantile state of sample i to that of sample j
dft_coefficients	function	Magnitudes of the first DFT coefficients of every variable (rows)
haar_dwt	function	Haar wavelet decomposition (approximation, details per level) of every variable; series are zero-padded to a power of two
chebyshev_coefficients	function	Chebyshev polynomial coefficients of every variable (Cai and Ng, 2004)

Name	Kind	Description
`knn_profile`	function	Indices and distances of the k nearest subsequences of every subsequence of width w
`matrix_profile`	function	Matrix profile (nearest-neighbour distance) and its index
`auto_window`	function	Subsequence width from the first dominant autocorrelation peak of every channel (median over channels), bounded to [lo, hi] (default hi = T/10)

A.4 `mvtsa.segmentation`: Segmentation

Name	Kind	Description
`Segmentation`	class	A segmentation of one series
`BaseSegmenter`	class	Common interface of all segmentation methods
`BaseCost`	class	Interface: fit(X) then error(a, b) for the samples X[a:b]
`CostL2`	class	Sum of squared deviations from the segment mean (piecewise constant model)
`CostL1`	class	Sum of absolute deviations from the segment median (robust to outliers)
`CostNormal`	class	Gaussian negative log-likelihood with segment-wise mean and full covariance, $(b-a)\log\det\hat{\Sigma}_{ab}$ (up to constants)
`CostLinear`	class	Residual sum of squares of a straight line fitted to every variable against time (piecewise linear approximation, PLA; Keogh and Pazzani, 1998)
`CostAR`	class	Residual sum of squares of autoregressive models of the given order fitted to every variable inside the segment (detects changes of the dynamics of univariate s
`CostPCA`	class	Principal component analysis based homogeneity cost
`CostKernel`	class	Kernel cost (Harchaoui and Cappé, 2007; Arlot, Celisse and Harchaoui, 2019): dispersion of the segment in the feature space of a kernel, @@M@@\{}sum_{i\{}in S}k_{ii
`CostRegression`	class	Model-based cost: residual sum of squares of a linear regression of the targets on the inputs inside the segment
`COSTS`	constant	Registry of segment costs by name
`get_cost`	function	Return a cost object from a name (see COSTS) or pass an instance through
`BottomUp`	class	Bottom-up segmentation by greedy merging of adjacent segments

Name	Kind	Description
TopDown	class	Binary segmentation: repeatedly split the segment whose best split reduces the total cost most
SlidingWindow	class	Grow each segment from its start until its cost exceeds max_error (Keogh et al., 2001); suitable for streaming data
SWAB	class	Sliding Window And Bottom-up (Keogh et al., 2001): bottom-up segmentation inside a moving buffer, emitting the leftmost segment
OptimalPartition	class	Exact minimization of the total cost for a fixed number of segments by dynamic programming over a grid of candidate change points
PELT	class	Pruned Exact Linear Time search (Killick et al., 2012): minimizes $\sum_i cost(S_i) + \beta\,(c-1)$ exactly, pruning candidates that can never be optimal
WindowDiscrepancy	class	Sliding two-window discrepancy $d(t) = cost(t-w, t+w) - cost(t-w, t) - cost(t, t+w)$ and peak picking (Truong, Oudre and Vayatis, 2020)
MergeTree	class	Hierarchy of segmentations produced by bottom-up merging
bic_penalty	function	BIC-type penalty for PELT: n_params *log(T)* sigma2
PCASegmenter	class	Segmentation by the homogeneity of local PCA models
DPCASegmenter	class	Process-dynamics-aware segmentation with dynamic PCA
KrzanowskiSegmenter	class	Off-line bottom-up segmentation by the similarity of neighbouring (dynamic) PCA models
RecursivePCASegmenter	class	On-line segmentation with recursive (dynamic) PCA
pca_segment_models	function	Fit a (dynamic) PCA model to every segment of a segmentation
GGSegmenter	class	Fuzzy segmentation with time-constrained PPCA clusters
monotone_labels	function	Contiguous crisp segmentation that agrees best with fuzzy memberships
FisherSegmenter	class	Segmentation of historical data by the information they carry on the parameters of a process model
fisher_information	function	Fisher information matrix of the samples [a, b) from the sensitivities S (T, p) or (T, n_outputs, p); Q is the inverse measurement-error covariance of the outpu
oed_criteria	function	OED criteria of a Fisher information matrix

Name	Kind	Description
`criterion_series`	function	Moving-window OED criterion: the "evaluating time series" of Dobos, Bankó and Abonyi (2010)
`finite_difference_sensitivities`	function	Output sensitivities $\partial y/\partial p$ by forward finite differences
`arx_sensitivities`	function	Sensitivities of the one-step-ahead prediction of an ARX model, $\hat{y}(k) = \sum_i a_i y(k-i) + \sum_j b_j u(k-d-j+1)$, with respect to its parameters: th
`BOCPD`	class	Bayesian on-line change point detection
`GaussianHMMSegmenter`	class	State detection with a Gaussian HMM
`ClaSPSegmenter`	class	Recursive segmentation by classification score profiles
`FLUSSSegmenter`	class	Regime extraction with the corrected arc curve (FLUSS)
`clasp_profile`	function	Classification score profile of one series (or of X restricted to a range): entry i scores the split before subsequence i
`arc_curve`	function	Corrected arc curve (CAC) of FLUSS from a matrix profile index
`StateSegmenter`	class	Segment-then-cluster state detection
`SparseGaussianStateSegmenter`	class	TICC-style state detection with sparse Gaussian window models
`segment_descriptors`	function	Local model of every segment
`segment_distance_matrix`	function	Pairwise distances of segment descriptors
`NeighborMergeSegmenter`	class	Bottom-up merging of adjacent segments by a pairwise dissimilarity
`GraphBottomUp`	class	Bottom-up merging of connected regions of a graph
`ClassifierHomogeneity`	class	Dissimilarity of two segments by a classifier two-sample test
`RegressionHomogeneity`	class	Dissimilarity of two segments by cross-prediction
`PredictiveSegmenter`	class	Bottom-up segmentation by predictive homogeneity
`ConsensusSegmenter`	class	Ensemble segmentation by boundary agreement
`boundary_agreement`	function	Boundary agreement profile of a list of segmentations (or change point arrays) of the same series
`cost_curve`	function	Total cost of the bottom-up segmentation for 1..max_segments segments (one merge run)
`relative_reduction`	function	Relative reduction rate $RR(c)$ of a cost curve
`knee_point`	function	Knee of a convex decreasing (or concave increasing) curve: the point farthest from the chord between the end points of the normalized curve
`permutation_test_segments`	function	Permutation test for the number of segments

Name	Kind	Description
`select_n_segments`	function	Choose the number of segments of a cost-based segmentation
`penalty_path`	function	Number of change points found by PELT for a range of penalties
`StreamSegmenter`	class	Windowed application of a batch segmenter to a stream
`CUSUMDetector`	class	Multivariate CUSUM change detector
`segment`	function	One-line segmentation: segment(X, "pca", n_segments=4)
`SEGMENTERS`	constant	Registry of segmenter names used by `segment`

A.5 `mvtsa.warping`: Warping and elastic similarity

Name	Kind	Description
`dtw`	function	DTW distance of two (multivariate) series
`dtw_path`	function	Full DTW result: distance, optimal warping path (L, 2), local and accumulated cost matrices
`DTWResult`	class	Distance, warping path and accumulated cost matrix of one alignment
`dtw_independent`	function	Independent multivariate DTW: the sum of the DTW distances of the variables, each warped separately (DTW_I)
`ddtw`	function	Derivative DTW: DTW of the Keogh–Pazzani derivative estimates
`wdtw`	function	Weighted DTW: the local cost of cell (i, j) is multiplied by $w_{max}/(1+\exp(-g(\lvert i-j\rvert - m_c)))$, where m_c is half of the length; large phase
`adtw`	function	Amerced DTW: every non-diagonal (warping) step costs an additional penalty (Herrmann and Webb, 2023)
`shape_dtw`	function	shapeDTW: DTW on local shape descriptors of every sample; the optimal path is used to accumulate the *raw* sample distances (Zhao and Itti, 2018)
`fast_dtw`	function	FastDTW (Salvador and Chan, 2007): project the path found at a coarser resolution, expand it by radius cells and refine
`lb_kim`	function	LB_Kim lower bound of DTW (squared Euclidean local cost): first and last samples
`lb_keogh`	function	LB_Keogh lower bound of band-constrained DTW (squared Euclidean cost, equal lengths): the distance of the candidate from the envelope of the query

Name	Kind	Description
`subsequence_dtw`	function	Best warped match of query inside a longer series
`local_cost_matrix`	function	Local cost matrix C[i, j] = d(x_i, y_j)
`window_mask`	function	Admissible cells of the warping path
`STEP_PATTERNS`	constant	Names of the DTW step patterns
`lcss`	function	LCSS distance $1 - LCSS/\min(T_x, T_y)$
`edr`	function	Edit distance on real sequences; with normalize divided by the longer length
`erp`	function	Edit distance with real penalty; gaps cost the distance to the reference value g (scalar or vector)
`twed`	function	Time warp edit distance with stiffness nu and edit penalty lam; tx, ty are time stamps (default: sample indices)
`msm`	function	Move–split–merge distance with split/merge cost c; multivariate series are compared channel by channel (sum, the independent strategy of Shifaz et al., 2023)
`soft_dtw`	function	Soft-DTW value $\mathrm{dtw}_\gamma = -\gamma \log \sum_A e^{-\langle A,C\rangle/\gamma}$; tends to DTW as $\gamma \to 0$
`soft_dtw_alignment`	function	Soft-DTW value and the expected alignment matrix $E = \nabla_C \mathrm{dtw}_\gamma(C)$: a fuzzy warping path whose entries are the probabilities that tw
`soft_dtw_divergence`	function	Soft-DTW divergence $\mathrm{dtw}_\gamma(x,y) - \frac{1}{2}(\mathrm{dtw}_\gamma(x,x) + \mathrm{dtw}_\gamma(y,y)) \geq 0$, zero iff $x = y$ (Blondel et al
`gak`	function	Global alignment kernel with a Gaussian local kernel
`ncc`	function	Coefficient-normalized cross-correlation sequence of two series (summed over the channels), computed with the FFT; length T_x+T_y-1
`sbd`	function	Shape-based distance $1 - \max_w NCC_w(x,y)$ of k-Shape (in [0, 2]); with return_shift also the optimal shift of y
`CBDTW`	class	Correlation-based dynamic time warping (Bankó and Abonyi, 2012)
`PLADistance`	class	Dissimilarities of piecewise linear approximations: MPLA, SPLA and the mixed MDPLA measure (Bankó and Abonyi, 2015)
`MixedSegmentDistance`	class	Weighted multivariate segment dissimilarity (a generalization of MDPLA)
`SegmentModel`	class	Descriptor of one segment: location, level, trend and correlation structure
`segment_models`	function	Descriptors of the segments of X
`segment_local_cost`	function	Local cost matrix between two sequences of segment models

Name	Kind	Description
segment_dtw	function	DTW over two sequences of segment models
segpca_distance	function	SEGPCA baseline: equal numbers of PCA-homogeneous segments compared index by index with $1 - s_{PCA}$ (no warping)
LOCAL_DISTANCES	constant	Names of the local dissimilarities of segment models
align	function	Pairwise alignment of two symbol sequences
align_episodes	function	Alignment of two episode sequences (Balaskó, Bankó and Abonyi, 2007)
episode_distance	function	Dissimilarity $1 - \text{score}/10$ of two episode sequences
episode_dtw	function	DTW over episode sequences (an mvtsa extension)
dba	function	DTW barycenter averaging of a (multivariate) collection
soft_dtw_barycenter	function	Soft-DTW barycenter by L-BFGS on the average's samples
medoid	function	Medoid of a collection under any distance function
path_statistics	function	Statistics of one warping path (L, 2)
tightness_of_warping	function	Tightness of warping (ToW) of a data set under one local dissimilarity
warping_function	function	Warping function $\varphi(i)$: the mean index of the second series aligned to sample i of the first
loo_1nn_accuracy	function	Leave-one-out 1-NN accuracy from a distance matrix
learn_window	function	Band width maximizing the LOO 1-NN accuracy (ties: the narrowest)
learn_segment_weights	function	Grid search of the weights of a segment-level measure
ElasticEnsemble	class	1-NN ensemble of elastic distances (after Lines and Bagnall, 2015)

A.6 mvtsa.distances: Distances

Name	Kind	Description
euclidean	function	
manhattan	function	
chebyshev	function	
correlation_distance	function	$1 - r$ of the flattened (multivariate) series
cosine_distance	function	
pca_distance	function	$1 - s$ with a PCA similarity factor of the two whole series (no time alignment; series may have different lengths)
DISTANCES	constant	Registry of distance functions by name

Name	Kind	Description
`get_distance`	function	Distance function from a name (see DISTANCES) or a callable
`pairwise_distances`	function	Distance matrix of a collection (or between two collections)
`is_elastic`	function	Whether a named metric accepts series of different lengths

A.7 `mvtsa.clustering`: Clustering

Name	Kind	Description
`TimeSeriesKMeans`	class	k-means for time series
`KShape`	class	k-Shape clustering (Paparrizos and Gravano, 2015)
`KMedoids`	class	k-medoids by PAM (BUILD + SWAP) on a distance matrix
`FuzzyCMedoids`	class	Relational fuzzy c-medoids (Krishnapuram, Joshi, Nasraoui and Yi, 2001)
`HierarchicalClustering`	class	Agglomerative clustering on a distance matrix
`SpectralClustering`	class	Spectral clustering of time series
`DensityClustering`	class	DBSCAN or HDBSCAN on a time series distance matrix (label -1 = noise)
`ConsensusClustering`	class	Evidence-accumulation consensus of several clusterings
`coassociation_matrix`	function	Fraction of partitions in which two objects share a cluster (noise labels -1 never co-occur)
`FeatureClustering`	class	Cluster series by feature vectors
`SegmentClustering`	class	Cluster the segments of a collection into recurring regimes
`TrajectoryClustering`	class	Clustering of objects moving in a common phase space
`neighborhood_preservation`	function	Mean Jaccard similarity of the k-nearest-neighbour sets of every pair of objects along time

A.8 `mvtsa.evaluation`: Evaluation and validation

Name	Kind	Description
`change_point_scores`	function	Precision, recall and F1 of change points with a tolerance margin
`f1_score_cp`	function	F1 of change points (see change_point_scores)

Name	Kind	Description
covering	function	Covering of the true segmentation by the predicted one: $C(G, G') = \frac{1}{T}\sum_{A\in G}\lvert A\rvert \max_{A'\in G'} J(A, A')$ with the Jaccard index @@M@@
hausdorff	function	Hausdorff distance of the two change point sets (inf if exactly one is empty)
annotation_error	function	Absolute difference of the numbers of change points
mean_boundary_error	function	Mean distance of every true change point to the nearest predicted one
state_scores	function	Adjusted Rand index, adjusted mutual information and normalized mutual information of per-sample state (or segment) labels
evaluate_segmentation	function	All segmentation scores at once
purity	function	Fraction of objects in the majority class of their cluster
matched_accuracy	function	Accuracy after the optimal one-to-one matching of clusters and classes (Hungarian algorithm)
clustering_scores	function	ARI, AMI, NMI, purity and matched accuracy
silhouette	function	Mean silhouette width from a distance matrix (noise label -1 ignored)
dunn_index	function	Smallest between-cluster distance divided by the largest cluster diameter
partition_coefficient	function	Bezdek's partition coefficient of a membership matrix (N, c)
partition_entropy	function	Bezdek's partition entropy of a membership matrix (N, c)
knn_accuracy	function	Leave-one-out k-NN classification accuracy from a distance matrix (1 - the 1-NN error rate of the UCR experiments)
precision_recall_curve_knn	function	Precision at recall levels r = 1..r_items (Bankó, 2015, Alg
mean_average_precision	function	Mean average precision of leave-one-out retrieval
segmentation_stability	function	Stability of a segmentation under perturbations
clustering_stability	function	Mean pairwise ARI of clusterings of random subsamples, compared on the objects they share
sensitivity	function	Run factory(**{param: v}) for every value and record the number of segments, the change points and an optional score(segmentation)
friedman_nemenyi	function	Friedman test and Nemenyi critical difference
wilcoxon_holm	function	Pairwise Wilcoxon signed-rank tests with Holm's correction; returns a pandas DataFrame (method_a, method_b, p_value, p_holm, significant)

Name	Kind	Description
`Benchmark`	class	Systematic comparison of methods on data sets
`grid_search_segmenter`	function	Grid search of segmenter parameters

A.9 `mvtsa.simplify`: Simplification

Name	Kind	Description
`smooth_states`	function	Absorb runs shorter than min_duration samples into a neighbouring run (the longer neighbour), repeatedly, until all runs are long enough
`merge_short_segments`	function	Remove change points so that every segment has at least min_length samples
`prune_change_points`	function	Remove change points whose removal increases the total cost by less than min_gain (default: relative times the cost of the unsegmented series), weakest first
`reduce_segments`	function	Coarsest level of a bottom-up merge hierarchy whose *explained* cost reduction is at least keep of the finest level's: @@M@@(cost_1-cost_c)/(cost_1-cos
`simplify_pla`	function	Piecewise linear approximation with the fewest segments whose mean squared error per sample stays below max_error
`compression_report`	function	Compression ratio and reconstruction error of a piecewise constant ("mean") or linear ("linear") representation
`reduce_components`	function	Return a copy of a fitted PCAModel keeping the fewest components that explain explained of the variance
`select_prototypes`	function	Indices of representative series: the n_per_class medoids of every class (or of the whole set) under the distance matrix D, selected greedily (k-medoids BUILD s

A.10 `mvtsa.explain`: Explanation

Name	Kind	Description
`boundary_contributions`	function	Contribution of the variables to every change

Name	Kind	Description
describe_segments	function	Per-segment table: start, stop, length, state, means, standard deviations, slopes and the variance explained by the first principal components of the segment
explain_states	function	Decision-tree explanation of state labels
transition_graph	function	Regime transition network of a state-labelled segmentation

A.11 mvtsa.mlops: Model operations

Name	Kind	Description
to_config	function	Dictionary {"class": "module.Class", "params": {...}} of an estimator
from_config	function	Rebuild an estimator from to_config output
save_config	function	Write a configuration (or the configuration of an object) to JSON or YAML
load_config	function	Read a JSON or YAML configuration file
AnalysisPipeline	class	Preprocess -> segment -> learn regimes -> label new data
save_model	function	Save a model directory (model.joblib, config.json, metadata.json)
load_model	function	Load a model saved by save_model; with verify the file hash is checked and a warning is raised if the package version differs
ModelRegistry	class	File-based model registry
ExperimentTracker	class	Experiment tracking with a local backend and optional MLflow mirroring
data_fingerprint	function	SHA-256 fingerprint of an array or collection (shape and values)
DataContract	class	Expected properties of input data
DriftMonitor	class	Batch-wise drift monitoring against reference data
population_stability_index	function	Population stability index of a univariate feature (bins: reference quantiles)
model_card	function	Markdown model card of an mvtsa model (written to path if given)

A.12 `mvtsa.io`: Import and export

Name	Kind	Description
`from_dataframe`	function	Series (T, n) and variable names from a wide pandas DataFrame (sorted by time_column if given)
`to_dataframe`	function	Wide DataFrame of a series, optionally with segment and state columns
`collection_from_long`	function	Collection from a long table (one row per sample, id_column per series); returns (collection, labels or None, ids)
`to_numpy3d`	function	3-D array of a collection
`from_numpy3d`	function	Collection from a 3-D array; trailing all-NaN samples are removed
`save_segmentation`	function	Write a segmentation to JSON
`load_segmentation`	function	
`from_ruptures`	function	Segmentation from ruptures breakpoints (last element = T)
`to_ruptures`	function	
`ruptures_segment`	function	Run a ruptures detector (pip install ruptures) and return an mvtsa Segmentation
`to_spmviz`	function	Symbol or episode sequences as an spmviz.SequenceDatabase (pip install spmviz); episodes are replaced by their primitive codes
`matrix_profile_stumpy`	function	Matrix profile of a univariate series with STUMPY (pip install stumpy): (profile, index)

A.13 `mvtsa.viz`: Visualization

Name	Kind	Description
`plot_series`	function	Small multiples: one panel per variable
`plot_segmentation`	function	Series with the segments shaded (state colours if the segmentation has state labels), change points as vertical lines, optional true change points (dashed, at t
`plot_fuzzy_segmentation`	function	Two panels: the series with the crisp segments, and the fuzzy memberships of the segments along time
`plot_states`	function	State timeline: one coloured bar per segment
`plot_profile`	function	A change profile (ClaSP score, FLUSS arc curve, BOCPD change probability, boundary agreement, similarity) with the detected change points

Name	Kind	Description
plot_run_length	function	Run-length posterior of BOCPD (log scale) with the MAP run length
plot_cost_curve	function	Total cost against the number of segments, with the relative reduction rate $RR(c)$ as bars on a second axis (Bankó, 2015, Figures 3.3 and 3.6)
plot_segmentogram	function	Segmentogram: every level of a bottom-up merge hierarchy as one row (top: one segment, bottom: max_segments), segments drawn as alternating bands, change points
plot_boundary_agreement	function	Consensus segmentation: the change points of every member (one row each) above the boundary agreement profile
plot_pca_statistics	function	Hotelling T^2 and Q of a PCA model along the series with their control limits
plot_segment_loadings	function	Heatmap of the loadings of one principal component of every segment (rows: variables, columns: segments): how the correlation structure changes from segment to
plot_contributions	function	Stacked bars of the variable contributions to every change point (output of boundary_contributions)
plot_transition_graph	function	Regime transition network (transition_graph): node size = total duration, edge width = transition probability
plot_alignment	function	The two series (y shifted down by offset) with grey lines joining the aligned samples (Bankó, 2015, Figure 2.6)
plot_cost_matrix	function	Local (or accumulated) cost matrix with the optimal warping path and the two series along the axes
plot_alignment_uncertainty	function	Expected alignment matrix of soft-DTW (probability that two samples are aligned) with the hard DTW path on top
plot_warping_function	function	Warping functions of one or several paths, normalized to [0, 1], against the diagonal
plot_segment_alignment	function	Segment-level alignment of correlation-based DTW: both series with their segments as coloured bands and the aligned segments connected at their centres
plot_average	function	Members of a cluster (grey) and their (DBA or soft-DTW) average
sammon	function	Sammon mapping (Sammon, 1969) of a distance matrix by Newton-type iterations; returns (Y, stress)

Name	Kind	Description
`fuzzy_sammon`	function	Fuzzy Sammon mapping (Feil, Balaskó and Abonyi, 2007) for relational fuzzy clusterings: the data are mapped so that their distances to the cluster prototypes (m
`plot_dendrogram`	function	Dendrogram of a hierarchical clustering
`plot_cluster_prototypes`	function	One panel per cluster: members (thin, grey), their mean +- standard deviation band (resampled to a common length) and the centre if given (centers is a dict lab
`plot_distance_matrix`	function	Distance matrix heatmap, ordered by labels ("label"), by the leaves of an average-linkage dendrogram ("cluster") or as given
`plot_embedding`	function	2-D map of a distance matrix by Sammon mapping, metric MDS or t-SNE, coloured by labels
`plot_pla`	function	Series and its piecewise linear approximation; each segment is annotated with its slope angle
`plot_sax`	function	z-normalized series, its PAA and the SAX symbols with the Gaussian breakpoints
`plot_episodes`	function	Series with its qualitative episodes: coloured bands labelled by the primitive codes and, above them, the triangular glyphs of the representation of Cheung and
`plot_recurrence`	function	Recurrence plot (thresholded or distance matrix)
`plot_visibility_graph`	function	Series as bars with its visibility arcs drawn above; if the graph is zone labelled (six_sigma_visibility_graph) nodes are coloured by their SPC zone and the con
`plot_critical_difference`	function	Critical difference diagram (Demšar, 2006): methods on the average rank axis, with bars joining methods whose ranks differ by less than the critical difference
`plot_precision_recall`	function	Precision against recall levels for several measures (Bankó, 2015, Figures 3.4–3.9); curves maps names to the output of precision_recall_curve_knn
`plot_metric_heatmap`	function	Heatmap of a data set x method table (e.g. Benchmark.pivot) with the values printed
`plot_stability`	function	Boundary frequency under perturbations (segmentation_stability) with the reference change points and their support
`segmentation_dashboard`	function	One-page summary of a segmentation: the segmented series, the state timeline, the cost curve (if a merge tree is given) and the variable contributions to the ch
`BRAND`	constant	str constant
`OKABE_ITO`	constant	list constant

CATEGORICAL	constant	list constant

A.14 `mvtsa.datasets`: Data sets

Name	Kind	Description
make_piecewise_stationary	function	Piecewise stationary Gaussian series
make_correlation_switching	function	Segments with equal means and variances but different correlation structure: in every state the variables are driven by n_latent hidden factors through a differ
make_dynamic_switching	function	Linear time-variant SISO systems of Bankó (2015, Ch
make_regime_process	function	Recurring operating regimes of a process: every regime has its own operating point and latent correlation structure; the regimes recur in random order with smoo
make_energy_process	function	Energy monitoring data in the spirit of Abonyi et al. (2013): the specific energy consumption (column 0, the key energy indicator) depends linearly on the load
simulate_ph_cstr	function	pH neutralization of a strong acid by a strong base in a continuous stirred tank (the pH case study of Bankó, 2015, Ch
simulate_mma_cstr	function	Isothermal MMA polymerization CSTR (four states, one input)
MMA_PARAMETERS	constant	Kinetic and operating parameters of the MMA polymerization reactor
make_cbf	function	Cylinder–Bell–Funnel data (Saito, 1994): three classes of univariate patterns with random onset, duration and amplitude
make_synthetic_control	function	Synthetic control chart patterns (Alcock and Manolopoulos, 1999): normal, cyclic, increasing trend, decreasing trend, upward shift, downward shift
make_signatures	function	Synthetic on-line signatures: pen trajectories with five channels [x, y, pressure, azimuth, altitude]
make_motion_collection	function	Multivariate motion-like series (accelerometer + gyroscope): every class is an activity with its own frequency content and inter-channel phase pattern, with ran
make_correlation_sequences	function	Collection whose classes differ only in the *sequence of correlation structures*
make_dynamic_sequences	function	Collection of input–output series [u, y] of a first-order system whose time constant follows a class-specific sequence of three values (fast, medium, slow)
random_time_warp	function	Apply a smooth, monotone random time warping to a series (and resample it to length)

Name	Kind	Description
`load_basic_motions`	function	BasicMotions: 6-channel activity recordings (see module doc)
`load_japanese_vowels`	function	JapaneseVowels: 12-channel, variable-length speech data
`load_gunpoint`	function	GunPoint: univariate hand trajectories
`load_bundled`	function	Load a bundled collection: (list_of_series, labels)
`data_home`	function	
`fetch_ucr_uea`	function	Download (once) and load a UCR/UEA archive data set
`fetch_tssb`	function	Download (once) and load TSSB series
`read_ts_file`	function	Read a .ts file of the archives: (list_of_series, labels)

APPENDIX

B



The example gallery

> Every method of the toolbox has at least one runnable example script. The scripts generate or load their own data, explain the method in their comments, and end with a study of the main parameters. The same scripts produce the gallery of the online documentation, the Jupyter notebooks, and every computed figure and code listing of this book. The tables below list them by folder of `examples/`, with the first figure of the book that each script produced.

B.1 First steps: series, segmentations, representations, PCA models

Folder `examples/01_basics/`. The methods are described in chapter 1.

Script	Topic	Figure
`plot_01_first_steps.py`	First steps: segment, look, evaluate	1.4
`plot_02_representations.py`	Representations: PAA, PLA, SAX, episodes	3.1
`plot_03_pca_models.py`	PCA models: T^2, Q and similarity factors	2.4

B.2 Cost-based segmentation: costs, searches, number of segments

Folder `examples/02_cost_segmentation/`. The methods are described in chapter 4.

Script	Topic	Figure
`plot_01_costs_and_search.py`	Costs and search strategies	4.1
`plot_02_segmentogram.py`	The merge hierarchy: segmentogram and persistence	4.2
`plot_03_number_of_segments.py`	How many segments?	—

B.3 Latent-variable segmentation: PCA, dynamic PCA, recursive PCA, fuzzy and Fisher segmentation

Folder `examples/03_latent_segmentation/`. The methods are described in chapter 5.

Script	Topic	Figure
`plot_01_pca_t2_q.py`	PCA segmentation with the T^2 and Q statistics	5.2
`plot_02_dynamic_pca.py`	Dynamic PCA segmentation	5.3
`plot_03_recursive_pca.py`	On-line segmentation with recursive PCA	5.5
`plot_04_fuzzy_segmentation.py`	Fuzzy segmentation with the modified Gath–Geva algorithm	5.6
`plot_05_fisher_mma.py`	Informative segments for parameter estimation (MMA polymerization)	5.7
`plot_06_energy_regression.py`	Operating ranges of an energy indicator	5.8

B.4 Probabilistic, profile-based and state segmentation

Folder `examples/04_modern_segmentation/`. The methods are described in chapter 6.

Script	Topic	Figure
`plot_01_bocpd.py`	Bayesian on-line change point detection	6.1
`plot_02_states.py`	Recurring operating states	6.2
`plot_03_clasp_fluss.py`	Shape changes: ClaSP and FLUSS	6.3
`plot_04_predictive.py`	Predictive homogeneity with any scikit-learn model	6.4
`plot_05_consensus.py`	Consensus of segmentations	6.5
`plot_06_streaming.py`	Segmentation of a data stream	—

B.5 Elastic similarity: DTW and its relatives

Folder `examples/05_warping/`. The methods are described in chapter 7.

Script	Topic	Figure
`plot_01_dtw_basics.py`	Dynamic time warping and global constraints	7.1
`plot_02_dtw_variants.py`	A family of elastic measures on Cylinder–Bell–Funnel	7.2
`plot_03_soft_dtw.py`	Soft-DTW: a differentiable alignment and its uncertainty	7.3
`plot_04_diagnostics_learning.py`	Diagnosing and tuning warping	—

B.6 Correlation-based and segment-level warping

Folder `examples/06_correlation_warping/`. The methods are described in chapter 8.

Script	Topic	Figure
`plot_01_cbdtw_signatures.py`	Correlation-based dynamic time warping	8.2
`plot_02_correlation_sequences.py`	When the information is in the correlation structure	8.3
`plot_03_dynamic_cbdtw.py`	Process-dynamics-aware warping: DPCA-CBDTW	8.4
`plot_04_japanese_vowels.py`	Real data: Japanese vowels and Riemannian local distances	8.5
`plot_05_pla_mdpla.py`	Piecewise linear warping: MPLA, SPLA and MDPLA	8.6
`plot_06_mixed_segments.py`	A multivariate mixed segment distance	—

Script	Topic	Figure
`plot_07_episodes.py`	Comparing qualitative trends: episode alignment	8.7

B.7 Clustering series, segments and trajectories

Folder `examples/07_clustering/`. The methods are described in chapter 9.

Script	Topic	Figure
`plot_01_partitional.py`	Partitional clustering: k-means, k-Shape, k-medoids	9.1
`plot_02_hierarchical_maps.py`	Hierarchical clustering and distance maps	9.3
`plot_03_basic_motions.py`	Clustering multivariate recordings: BasicMotions	9.2
`plot_04_segment_clustering.py`	Clustering segments: recurring patterns across series	9.4
`plot_05_trajectories.py`	Objects that move together: trajectory clustering	—

B.8 Evaluation, validation and benchmarking

Folder `examples/08_evaluation/`. The methods are described in chapter 10.

Script	Topic	Figure
`plot_01_metrics.py`	Measuring segmentation quality	10.1
`plot_02_stability.py`	Is the segmentation stable?	10.2
`plot_03_benchmark.py`	Benchmarking segmenters with statistical tests	10.3
`plot_04_tuning.py`	Tuning segmenter parameters	—

B.9 Visualization and explanation

Folder `examples/09_visualization/`. The methods are described in chapter 11.

Script	Topic	Figure
`plot_01_dashboard.py`	A one-page segmentation report	11.1
`plot_02_visibility_graphs.py`	Six-sigma visibility graphs	11.2

Script	Topic	Figure
`plot_03_recurrence_images.py`	Series as images and networks	3.4
`plot_04_warping_maps.py`	Looking inside alignments and collections	11.3

B.10 Model operations

Folder `examples/10_mlops/`. The methods are described in chapter 12.

Script	Topic	Figure
`plot_01_pipeline_registry.py`	From analysis to a deployable model	—
`plot_02_drift_monitoring.py`	Monitoring a deployed model for drift	12.2
`plot_03_tracking_cards.py`	Experiment tracking and model cards	—

B.11 Case studies

Folder `examples/11_applications/`. The methods are described in chapter 13.

Script	Topic	Figure
`plot_01_polymer_grades.py`	Grade transitions of a polymerization reactor	13.1
`plot_02_ph_process.py`	Operating regions of a pH neutralization process	13.3
`plot_03_regime_explanation.py`	Explaining operating regimes	13.4

APPENDIX

C



Method index and migration

This appendix lists the methods of the book with their classes or functions in `mvtsa` and their origins, and explains how to move from the group's earlier MATLAB implementations to the package. *Group* marks methods developed at the University of Pannonia, *new* methods introduced in this book; the others implement methods of the literature.

Method	API	Origin
Segmentation		
PCA segmentation, T^2 and Q costs (group)	`PCASegmenter`, `CostPCA`	[1, 2, 8]
Modified Gath–Geva fuzzy segmentation (group)	`GGSegmenter`	[2]
Kalman filtering and segmentation (group)	`kalman_smoother` + any segmenter	[3]
Dynamic-PCA segmentation (group)	`DPCASegmenter`, `select_dpca_lag`	[6, 46]
Recursive-PCA on-line segmentation (group)	`RecursivePCA-Segmenter`, `RecursivePCA`	[13, 81, 82]
Fisher-information segmentation (group)	`FisherSegmenter`	[11, 12, 14, 88]
Model-based energy segmentation (group)	`CostRegression`	[16]
Krzanowski neighbour merging	`KrzanowskiSegmenter`	[14, 21]
Costs L2, L1, Gaussian, linear, AR, kernel	`get_cost`, `COSTS`	[36, 72, 73]
Bottom-up, top-down, sliding window, SWAB	`BottomUp`, `TopDown`, `SlidingWindow`, `SWAB`	[76, 77, 78]
Optimal partitioning, PELT, penalty path	`OptimalPartition`, `PELT`, `penalty_path`	[24, 74, 75]

Method	API	Origin
Merge tree, segmentogram, persistence (new)	`MergeTree`, `plot_segmentogram`	this book
Relative reduction rate, permutation test	`select_n_segments`	[8, 79]
Bayesian on-line change point detection	`BOCPD`	[25]
Sticky Gaussian HMM	`GaussianHMMSegmenter`	[91, 92]
Sparse Gaussian states	`SparseGaussian-StateSegmenter`	[95]
Segment-then-cluster states (new)	`StateSegmenter`	after [93]
ClaSP, FLUSS (centred)	`ClaSPSegmenter`, `FLUSSSegmenter`	[27, 28]
Predictive homogeneity (new)	`PredictiveSegmenter`	after [96]
Consensus segmentation (new)	`ConsensusSegmenter`	this book
CUSUM, buffered streaming	`CUSUMDetector`, `StreamSegmenter`	[97]
Warping and similarity		
Correlation-based DTW, SEGPCA (group)	`CBDTW`, `segpca_distance`	[4, 5]
Dynamic-PCA CBDTW (group)	`CBDTW(lags=...)`	[8]
MPLA, SPLA, MDPLA, tightness of warping (group)	`PLADistance`, `tightness_of_warping`	[7]
Episode alignment (group)	`triangular_episodes`, `align_episodes`	[9, 10, 58]
Riemannian and mixed segment distances, episode DTW (new)	`LOCAL_DISTANCES`, `MixedSegmentDistance`, `episode_dtw`	this book, with [94, 122, 123]
DTW, windows, step patterns, lower bounds	`dtw`, `window_mask`, `lb_keogh`, `lb_kim`	[40, 41, 102, 103]
DDTW, WDTW, ADTW, shapeDTW, FastDTW	`ddtw`, `wdtw`, `adtw`, `shape_dtw`, `fast_dtw`	[104, 105, 106, 107, 108]
Dependent and independent multivariate DTW	`dtw`, `dtw_independent`	[109, 110]
LCSS, EDR, ERP, TWED, MSM	`lcss`, `edr`, `erp`, `twed`, `msm`	[111, 112, 113, 114, 115]
Soft-DTW, divergence, GAK, SBD	`soft_dtw`, `soft_dtw_divergence`, `gak`, `sbd`	[29, 30, 117, 119]
DBA, soft-DTW barycenter, elastic ensemble	`dba`, `soft_dtw_barycenter`, `ElasticEnsemble`	[29, 33, 120]
Clustering, representation, visualization		
Fuzzy Sammon mapping (group)	`fuzzy_sammon`	[18]

Method	API	Origin
Neighbourhood-preservation trajectory clustering (group)	`TrajectoryClustering`	[19]
Six-sigma visibility graphs (group)	`six_sigma_visibility_graph`	[20]
Decision-tree description of regimes (group)	`explain_states`	[17]
k-Shape, DBA k-means, k-medoids, fuzzy c-medoids	`KShape`, `TimeSeriesKMeans`, `KMedoids`, `FuzzyCMedoids`	[30, 33, 130, 131]
Spectral, density-based, consensus clustering	`SpectralClustering`, `DensityClustering`, `ConsensusClustering`	[133, 134, 135, 136]
PAA, APCA, PLA, PIP, SAX, ABBA, SFA	`paa`, `apca`, `pla`, `pip`, `sax`, `abba`, `sfa`	[49, 50, 51, 52, 55, 56, 57]
Features, path signatures	`extract_features`, `signature_features`	after [60, 62, 63]
Visibility, recurrence, ordinal networks; images	`visibility_graph`, `recurrence_network`, `ordinal_partition_network`, `gramian_angular_field`	[64, 65, 66, 68, 70]
Matrix profile, k-NN profile	`matrix_profile`, `knn_profile`	[26, 71]

Migration from the MATLAB toolboxes

Two MATLAB implementations of the group's methods are available on the MathWorks File Exchange: *Correlation based dynamic time warping of multivariate time series* (CBDTW) and *Fuzzy clustering based time-series segmentation* (the modified Gath–Geva algorithm). Table C.2 maps their main steps to the package. Three differences matter in practice. First, `mvtsa` standardizes explicitly and records the statistics in the fitted object, so that queries are segmented "according to the database". Second, change points are the first samples of new segments (0-based), and the boundaries 0 and T are not listed. Third, the number of principal components can be a fraction of explained variance, resolved once on the whole series and then kept fixed for every segment, which keeps segment models comparable.

Table C.2. From the MATLAB toolboxes to `mvtsa`.

MATLAB step	`mvtsa`
bottom-up PCA segmentation of a series (T^2 or Q cost)	`PCASegmenter(n_segments, n_components, statistic="Q")` or `BottomUp("pca", ...)`
segment PCA models and the $1 - s_{\mathrm{PCA}}$ cost matrix	`segment_models`, `segment_local_cost`
DTW of the segment sequences (CBDTW)	`CBDTW(...).fit(collection).pairwise()`
SEGPCA baseline	`CBDTW(..., warping=False)`, `segpca_distance`
precision–recall evaluation of retrieval	`precision_recall_curve_knn`, `plot_precision_recall`
modified Gath–Geva fuzzy segmentation with compatible merging	`GGSegmenter(n_segments, n_components, merge=True)`
fuzzy memberships and segment models	`memberships_`, `models_`, `centers_t_`, `sigmas_t_`

Bibliography

[1] János Abonyi, Balázs Feil, Sándor Németh, and Péter Árva. “Fuzzy clustering based segmentation of time-series”. In: *Advances in Intelligent Data Analysis V (IDA 2003)*. Vol. 2810. Lecture Notes in Computer Science. Springer, 2003, pp. 275–285. DOI: 10.1007/978-3-540-45231-7_26.

[2] János Abonyi, Balázs Feil, Sándor Németh, and Péter Árva. “Modified Gath–Geva clustering for fuzzy segmentation of multivariate time-series”. In: *Fuzzy Sets and Systems* 149.1 (2005), pp. 39–56.

[3] Balázs Feil, János Abonyi, Sándor Németh, and Péter Árva. “Monitoring process transitions by Kalman filtering and time-series segmentation”. In: *Computers & Chemical Engineering* 29.6 (2005), pp. 1423–1431.

[4] Zoltán Bankó and János Abonyi. “Correlation-based dynamic time warping”. In: *Proceedings of the 8th International Symposium of Hungarian Researchers on Computational Intelligence and Informatics*. Budapest, 2007, pp. 295–306.

[5] Zoltán Bankó and János Abonyi. “Correlation based dynamic time warping of multivariate time series”. In: *Expert Systems with Applications* 39.17 (2012), pp. 12814–12823. DOI: 10.1016/j.eswa.2012.05.012.

[6] Zoltán Bankó, László Dobos, and János Abonyi. “Dynamic principal component analysis in multivariate time-series segmentation”. In: *Conservation, Information, Evolution* 1.1 (2011), pp. 11–24.

[7] Zoltán Bankó and János Abonyi. “Mixed dissimilarity measure for piecewise linear approximation based time series applications”. In: *Expert Systems with Applications* 42.21 (2015), pp. 7664–7675.

[8] Zoltán Bankó. “Correlation-based elastic similarity measures for data mining of technological time series”. PhD thesis. Veszprém: University of Pannonia, Doctoral School in Chemical Engineering and Material Sciences, 2015. DOI: 10.18136/PE.2015.601.

[9] Balázs Balaskó, Zoltán Bankó, and János Abonyi. “Analyzing trends by symbolic episode representation and sequence alignment”. In: *Proceedings of the Mediterranean Conference on Control and Automation (MED 2007)*. Athens, 2007, pp. 1–6.

[10] Balázs Balaskó, Sándor Németh, and János Abonyi. “Time-series similarity – application to qualitative process trend analysis”. In: *Acta Agraria Kaposváriensis* 11.2 (2007), pp. 197–209.

[11] László Dobos, Zoltán Bankó, and János Abonyi. "Segmentation-based optimal experiment design". In: *Proceedings of the 10th International Symposium of Hungarian Researchers on Computational Intelligence and Informatics*. Budapest, 2009, pp. 279–289.

[12] László Dobos, Zoltán Bankó, and János Abonyi. "Qualifying and segmentation of historical process data using optimal experiment design techniques for supporting parameter estimation". In: *Acta Electrotechnica et Informatica* 10.2 (2010), pp. 28–32.

[13] László Dobos and János Abonyi. "On-line detection of homogeneous operation ranges by dynamic principal component analysis based time-series segmentation". In: *Chemical Engineering Science* 75 (2012), pp. 96–105.

[14] László Dobos and János Abonyi. "Fisher information matrix based time-series segmentation of multivariate streaming data for supporting model identification". In: *Chemical Engineering Science* 101 (2013), pp. 99–108.

[15] László Csaba Dobos. "Kísérlettervezéses technikák a technológiák elemzésére és optimálására (Experiment design techniques for the analysis and optimization of technologies)". PhD thesis. Veszprém: University of Pannonia, 2015.

[16] János Abonyi, Tibor Kulcsár, Miklós Balaton, and László Nagy. "Historical process data based energy monitoring – model based time-series segmentation to determine target values". In: *Chemical Engineering Transactions* 35 (2013), pp. 931–936. DOI: 10.3303/CET1335155.

[17] Tamás Varga, Ferenc Szeifert, József Réti, and János Abonyi. "Decision tree based qualitative analysis of operating regimes in industrial production processes". In: *Hungarian Journal of Industrial Chemistry* 35 (2007), pp. 95–99.

[18] Balázs Feil, Balázs Balaskó, and János Abonyi. "Visualization of fuzzy clusters by fuzzy Sammon mapping projection: application to the analysis of phase space trajectories". In: *Soft Computing* 11 (2007), pp. 479–488. DOI: 10.1007/s00500-006-0111-5.

[19] Bálint Levente Tarcsay, János Abonyi, and Sándor Németh. "Neighborhood preservation-based trajectory clustering for analyzing temporal behavior of dynamic systems". In: *Digital Chemical Engineering* (2025).

[20] Martin Ferenczi, Alex Kummer, and János Abonyi. "Six-sigma approach-based visibility graphs". In: *Scientific Reports* 16.1 (2026).

[21] W. J. Krzanowski. "Between-groups comparison of principal components". In: *Journal of the American Statistical Association* 74.367 (1979), pp. 703–707.

[22] J. Edward Jackson. *A User's Guide to Principal Components*. New York: Wiley, 1991.

[23] Theodora Kourti. "Application of latent variable methods to process control and multivariate statistical process control in industry". In: *International Journal of Adaptive Control and Signal Processing* 19.4 (2005), pp. 213–246.

[24] Rebecca Killick, Paul Fearnhead, and Idris A. Eckley. "Optimal detection of changepoints with a linear computational cost". In: *Journal of the American Statistical Association* 107.500 (2012), pp. 1590–1598.

[25] Ryan Prescott Adams and David J. C. MacKay. *Bayesian online changepoint detection*. arXiv:0710.3742. 2007.

[26] Chin-Chia Michael Yeh et al. “Matrix profile I: all pairs similarity joins for time series”. In: *Proceedings of the 2016 IEEE International Conference on Data Mining (ICDM)*. 2016, pp. 1317–1322.

[27] Shaghayegh Gharghabi, Yifei Ding, Chin-Chia Michael Yeh, Kaveh Kamgar, Liudmila Ulanova, and Eamonn Keogh. “Matrix profile VIII: domain agnostic online semantic segmentation at superhuman performance levels”. In: *Proceedings of the 2017 IEEE International Conference on Data Mining (ICDM)*. 2017, pp. 117–126.

[28] Arik Ermshaus, Patrick Schäfer, and Ulf Leser. “ClaSP: parameter-free time series segmentation”. In: *Data Mining and Knowledge Discovery* 37 (2023), pp. 1262–1300. DOI: 10.1007/s10618-023-00923-x.

[29] Marco Cuturi and Mathieu Blondel. “Soft-DTW: a differentiable loss function for time-series”. In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. PMLR. 2017, pp. 894–903.

[30] John Paparrizos and Luis Gravano. “k-Shape: efficient and accurate clustering of time series”. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 2015, pp. 1855–1870.

[31] Gerrit J. J. van den Burg and Christopher K. I. Williams. *An evaluation of change point detection algorithms*. arXiv:2003.06222. 2020.

[32] Félix Chavelli, Arik Ermshaus, Patrick Schäfer, Fan Yang, John Paparrizos, and Paul Boniol. “TSB-SEG: a systematic time-series segmentation benchmark”. In: *Advances in Neural Information Processing Systems (NeurIPS 2026)*. 2026.

[33] François Petitjean, Alain Ketterlin, and Pierre Gançarski. “A global averaging method for dynamic time warping, with applications to clustering”. In: *Pattern Recognition* 44.3 (2011), pp. 678–693.

[34] Fabian Pedregosa et al. “Scikit-learn: machine learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[35] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. “Numba: a LLVM-based Python JIT compiler”. In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. 2015, pp. 1–6.

[36] Charles Truong, Laurent Oudre, and Nicolas Vayatis. “Selective review of offline change point detection methods”. In: *Signal Processing* 167 (2020), p. 107299.

[37] Romain Tavenard et al. “Tslearn, a machine learning toolkit for time series data”. In: *Journal of Machine Learning Research* 21.118 (2020), pp. 1–6.

[38] Matthew Middlehurst et al. “aeon: a Python toolkit for learning from time series”. In: *Journal of Machine Learning Research* 25.289 (2024), pp. 1–10.

[39] Sean M. Law. “STUMPY: a powerful and scalable Python library for time series data mining”. In: *Journal of Open Source Software* 4.39 (2019), p. 1504.

[40] Hiroaki Sakoe and Seibi Chiba. “Dynamic programming algorithm optimization for spoken word recognition”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26.1 (1978), pp. 43–49.

[41] Fumitada Itakura. “Minimum prediction residual principle applied to speech recognition”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 23.1 (1975), pp. 67–72.

[42] J. Edward Jackson and Govind S. Mudholkar. "Control procedures for residuals associated with principal component analysis". In: *Technometrics* 21.3 (1979), pp. 341–349.

[43] Ashish Singhal and Dale E. Seborg. "Pattern matching in historical batch data using PCA". In: *IEEE Control Systems Magazine* 22.5 (2002), pp. 53–63.

[44] Matthew C. Johannesmeyer, Ashish Singhal, and Dale E. Seborg. "Pattern matching in historical data". In: *AIChE Journal* 48.9 (2002), pp. 2022–2038.

[45] Kiyoung Yang and Cyrus Shahabi. "A PCA-based similarity measure for multivariate time series". In: *Proceedings of the 2nd ACM International Workshop on Multimedia Databases*. 2004, pp. 65–74.

[46] Wenfu Ku, Robert H. Storer, and Christos Georgakis. "Disturbance detection and isolation by dynamic principal component analysis". In: *Chemometrics and Intelligent Laboratory Systems* 30.1 (1995), pp. 179–196.

[47] Andrew M. Fraser and Harry L. Swinney. "Independent coordinates for strange attractors from mutual information". In: *Physical Review A* 33.2 (1986), pp. 1134–1140.

[48] Matthew B. Kennel, Reggie Brown, and Henry D. I. Abarbanel. "Determining embedding dimension for phase-space reconstruction using a geometrical construction". In: *Physical Review A* 45.6 (1992), pp. 3403–3411.

[49] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. "Dimensionality reduction for fast similarity search in large time series databases". In: *Knowledge and Information Systems* 3.3 (2001), pp. 263–286.

[50] Byoung-Kee Yi and Christos Faloutsos. "Fast time sequence indexing for arbitrary Lp norms". In: *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB)*. 2000, pp. 385–394.

[51] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. "Locally adaptive dimensionality reduction for indexing large time series databases". In: *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*. 2001, pp. 151–162.

[52] Fu-Lai Chung, Tak-Chung Fu, Robert Luk, and Vincent Ng. "Flexible time series pattern matching based on perceptually important points". In: *IJCAI Workshop on Learning from Temporal and Spatial Data*. 2001, pp. 1–7.

[53] Tak-Chung Fu, Fu-Lai Chung, Robert Luk, and Chak-Man Ng. "Representing financial time series based on data point importance". In: *Engineering Applications of Artificial Intelligence* 21.2 (2008), pp. 277–300.

[54] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. "A symbolic representation of time series, with implications for streaming algorithms". In: *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*. 2003, pp. 2–11.

[55] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. "Experiencing SAX: a novel symbolic representation of time series". In: *Data Mining and Knowledge Discovery* 15.2 (2007), pp. 107–144.

[56] Steven Elsworth and Stefan Güttel. "ABBA: adaptive Brownian bridge-based symbolic aggregation of time series". In: *Data Mining and Knowledge Discovery* 34.4 (2020), pp. 1175–1200.

[57] Patrick Schäfer and Mikael Högqvist. "SFA: a symbolic Fourier approximation and index for similarity search in high dimensional datasets". In: *Proceedings of the 15th International Conference on Extending Database Technology (EDBT)*. 2012, pp. 516–527.

[58] J. T.-Y. Cheung and George Stephanopoulos. "Representation of process trends – Part I. A formal representation framework". In: *Computers & Chemical Engineering* 14.4–5 (1990), pp. 495–510.

[59] Ben D. Fulcher and Nick S. Jones. "hctsa: a computational framework for automated time-series phenotyping using massive feature extraction". In: *Cell Systems* 5.5 (2017), pp. 527–531.

[60] Carl H. Lubba, Sarab S. Sethi, Philip Knaute, Simon R. Schultz, Ben D. Fulcher, and Nick S. Jones. "catch22: CAnonical time-series CHaracteristics". In: *Data Mining and Knowledge Discovery* 33.6 (2019), pp. 1821–1852.

[61] Christoph Bandt and Bernd Pompe. "Permutation entropy: a natural complexity measure for time series". In: *Physical Review Letters* 88.17 (2002), p. 174102.

[62] Kuo-Tsai Chen. "Integration of paths – a faithful representation of paths by noncommutative formal power series". In: *Transactions of the American Mathematical Society* 89.2 (1958), pp. 395–407.

[63] Terry Lyons. "Rough paths, signatures and the modelling of functions on streams". In: *Proceedings of the International Congress of Mathematicians (ICM 2014)*. Vol. IV. 2014, pp. 163–184.

[64] Lucas Lacasa, Bartolo Luque, Fernando Ballesteros, Jordi Luque, and Juan Carlos Nuño. "From time series to complex networks: the visibility graph". In: *Proceedings of the National Academy of Sciences* 105.13 (2008), pp. 4972–4975.

[65] B. Luque, L. Lacasa, F. Ballesteros, and J. Luque. "Horizontal visibility graphs: exact results for random time series". In: *Physical Review E* 80.4 (2009), p. 046103.

[66] Norbert Marwan, Jonathan F. Donges, Yong Zou, Reik V. Donner, and Jürgen Kurths. "Complex network approach for recurrence analysis of time series". In: *Physics Letters A* 373.46 (2009), pp. 4246–4254.

[67] J.-P. Eckmann, S. Oliffson Kamphorst, and D. Ruelle. "Recurrence plots of dynamical systems". In: *Europhysics Letters* 4.9 (1987), pp. 973–977.

[68] Michael Small. "Complex networks from time series: capturing dynamics". In: *IEEE International Symposium on Circuits and Systems (ISCAS 2013)*. 2013, pp. 2509–2512.

[69] Michael McCullough, Michael Small, Thomas Stemler, and Herbert Ho-Ching Iu. "Time lagged ordinal partition networks for capturing dynamics of continuous dynamical systems". In: *Chaos* 25.5 (2015), p. 053101.

[70] Zhiguang Wang and Tim Oates. "Imaging time-series to improve classification and imputation". In: *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*. 2015, pp. 3939–3945.

[71] Yan Zhu et al. "Matrix profile II: exploiting a novel algorithm and GPUs to break the one hundred million barrier for time series motifs and joins". In: *Proceedings of the 2016 IEEE International Conference on Data Mining (ICDM)*. 2016, pp. 739–748.

[72] Zaïd Harchaoui and Olivier Cappé. "Retrospective multiple change-point estimation with kernels". In: *IEEE/SP 14th Workshop on Statistical Signal Processing*. 2007, pp. 768–772.

[73] Sylvain Arlot, Alain Celisse, and Zaïd Harchaoui. "A kernel multiple change-point algorithm via model selection". In: *Journal of Machine Learning Research* 20.162 (2019), pp. 1–56.

[74] Ivan E. Auger and Charles E. Lawrence. "Algorithms for the optimal identification of segment neighborhoods". In: *Bulletin of Mathematical Biology* 51.1 (1989), pp. 39–54.

[75] Kaylea Haynes, Idris A. Eckley, and Paul Fearnhead. "Computationally efficient changepoint detection for a range of penalties". In: *Journal of Computational and Graphical Statistics* 26.1 (2017), pp. 134–143.

[76] A. J. Scott and M. Knott. "A cluster analysis method for grouping means in the analysis of variance". In: *Biometrics* 30.3 (1974), pp. 507–512.

[77] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. "An online algorithm for segmenting time series". In: *Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM)*. 2001, pp. 289–296.

[78] Johan Himberg, Kalle Korpiaho, Heikki Mannila, Johanna Tikanmäki, and Hannu Toivonen. "Time series segmentation for context recognition in mobile devices". In: *Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM)*. 2001, pp. 203–210.

[79] Kari T. Vasko and Hannu T. T. Toivonen. "Estimating the number of segments in time series data using permutation tests". In: *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM)*. 2002, pp. 466–473.

[80] T. J. McAvoy, E. Hsu, and S. Lowenthal. "Dynamics of pH in controlled stirred tank reactor". In: *Industrial & Engineering Chemistry Process Design and Development* 11.1 (1972), pp. 68–70.

[81] Weihua Li, H. Henry Yue, Sergio Valle-Cervantes, and S. Joe Qin. "Recursive PCA for adaptive process monitoring". In: *Journal of Process Control* 10.5 (2000), pp. 471–486.

[82] T. R. Fortescue, L. S. Kershenbaum, and B. E. Ydstie. "Implementation of self-tuning regulators with variable forgetting factors". In: *Automatica* 17.6 (1981), pp. 831–835.

[83] I. Gath and A. B. Geva. "Unsupervised optimal fuzzy clustering". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 11.7 (1989), pp. 773–780.

[84] Michael E. Tipping and Christopher M. Bishop. "Probabilistic principal component analysis". In: *Journal of the Royal Statistical Society: Series B* 61.3 (1999), pp. 611–622.

[85] Michael E. Tipping and Christopher M. Bishop. "Mixtures of probabilistic principal component analyzers". In: *Neural Computation* 11.2 (1999), pp. 443–482.

[86] Gaia Franceschini and Sandro Macchietto. "Model-based design of experiments for parameter precision: state of the art". In: *Chemical Engineering Science* 63.19 (2008), pp. 4846–4872.

[87] László Dobos, Zoltán Bankó, and János Abonyi. "Optimal experiment design integrated with time-series segmentation". In: *Proceedings of the 8th International Symposium on Applied Machine Intelligence and Informatics (SAMI 2010)*. Herl'any, 2010, pp. 206–210.

[88] László Dobos and János Abonyi. "Fisher information based time-series segmentation of streaming process data for monitoring and supporting on-line parameter estimation in energy systems". In: *21st European Symposium on Computer Aided Process Engineering (ESCAPE 21)*. Chalkidiki, 2011, pp. 1844–1849.

[89] K. Bernaerts, R. D. Servaes, S. Kooyman, K. J. Versyck, and J. F. Van Impe. "Optimal temperature input design for estimation of the square root model parameters: parameter accuracy and model validity restrictions". In: *International Journal of Food Microbiology* 73 (2002), pp. 145–157.

[90] Francis J. Doyle, Babatunde A. Ogunnaike, and Ronald K. Pearson. "Nonlinear model-based control using second-order Volterra models". In: *Automatica* 31.5 (1995), pp. 697–714.

[91] Lawrence R. Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition". In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286.

[92] Emily B. Fox, Erik B. Sudderth, Michael I. Jordan, and Alan S. Willsky. "A sticky HDP-HMM with application to speaker diarization". In: *The Annals of Applied Statistics* 5.2A (2011), pp. 1020–1056.

[93] Ashish Singhal and Dale E. Seborg. "Clustering multivariate time-series data". In: *Journal of Chemometrics* 19.8 (2005), pp. 427–438.

[94] Rajendra Bhatia, Tanvi Jain, and Yongdo Lim. "On the Bures–Wasserstein distance between positive definite matrices". In: *Expositiones Mathematicae* 37.2 (2019), pp. 165–191.

[95] David Hallac, Sagar Vare, Stephen Boyd, and Jure Leskovec. "Toeplitz inverse covariance-based clustering of multivariate time series data". In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2017, pp. 215–223.

[96] David Lopez-Paz and Maxime Oquab. "Revisiting classifier two-sample tests". In: *International Conference on Learning Representations (ICLR)*. 2017.

[97] E. S. Page. "Continuous inspection schemes". In: *Biometrika* 41.1/2 (1954), pp. 100–115.

[98] Lawrence Rabiner and Biing-Hwang Juang. *Fundamentals of Speech Recognition*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[99] Toni Giorgino. "Computing and visualizing dynamic time warping alignments in R: the dtw package". In: *Journal of Statistical Software* 31.7 (2009), pp. 1–24.

[100] Chotirat Ann Ratanamahatana and Eamonn J. Keogh. "Making time-series classification more accurate using learned constraints". In: *Proceedings of the 2004 SIAM International Conference on Data Mining*. 2004, pp. 11–22.

[101] Chotirat Ann Ratanamahatana and Eamonn J. Keogh. "Three myths about dynamic time warping data mining". In: *Proceedings of the 2005 SIAM International Conference on Data Mining*. 2005, pp. 506–510.

[102] Sang-Wook Kim, Sanghyun Park, and Wesley W. Chu. "An index-based approach for similarity search supporting time warping in large sequence databases". In: *Proceedings of the 17th International Conference on Data Engineering (ICDE)*. 2001, pp. 607–614.

[103] Eamonn Keogh and Chotirat Ann Ratanamahatana. "Exact indexing of dynamic time warping". In: *Knowledge and Information Systems* 7.3 (2005), pp. 358–386.

[104] Stan Salvador and Philip Chan. "Toward accurate dynamic time warping in linear time and space". In: *Intelligent Data Analysis* 11.5 (2007), pp. 561–580.

[105] Eamonn J. Keogh and Michael J. Pazzani. "Derivative dynamic time warping". In: *Proceedings of the 2001 SIAM International Conference on Data Mining*. 2001, pp. 1–11.

[106] Young-Seon Jeong, Myong K. Jeong, and Olufemi A. Omitaomu. "Weighted dynamic time warping for time series classification". In: *Pattern Recognition* 44.9 (2011), pp. 2231–2240.

[107] Matthieu Herrmann and Geoffrey I. Webb. "Amercing: an intuitive and effective constraint for dynamic time warping". In: *Pattern Recognition* 137 (2023), p. 109333.

[108] Jiaping Zhao and Laurent Itti. "shapeDTW: shape dynamic time warping". In: *Pattern Recognition* 74 (2018), pp. 171–184.

[109] Mohammad Shokoohi-Yekta, Bing Hu, Hongxia Jin, Jun Wang, and Eamonn Keogh. "Generalizing DTW to the multi-dimensional case requires an adaptive approach". In: *Data Mining and Knowledge Discovery* 31.1 (2017), pp. 1–31.

[110] Ahmed Shifaz, Charlotte Pelletier, François Petitjean, and Geoffrey I. Webb. "Elastic similarity and distance measures for multivariate time series". In: *Knowledge and Information Systems* 65 (2023), pp. 2665–2698. DOI: 10.1007/s10115-023-01835-4.

[111] Michail Vlachos, George Kollios, and Dimitrios Gunopulos. "Discovering similar multidimensional trajectories". In: *Proceedings of the 18th International Conference on Data Engineering (ICDE)*. 2002, pp. 673–684.

[112] Lei Chen, M. Tamer Özsu, and Vincent Oria. "Robust and fast similarity search for moving object trajectories". In: *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*. 2005, pp. 491–502.

[113] Lei Chen and Raymond Ng. "On the marriage of Lp-norms and edit distance". In: *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*. 2004, pp. 792–803.

[114] Pierre-François Marteau. "Time warp edit distance with stiffness adjustment for time series matching". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.2 (2009), pp. 306–318.

[115] Alexandra Stefan, Vassilis Athitsos, and Gautam Das. "The move-split-merge metric for time series". In: *IEEE Transactions on Knowledge and Data Engineering* 25.6 (2013), pp. 1425–1438.

[116] Naoki Saito. "Local feature extraction and its applications using a library of bases". PhD thesis. New Haven, CT: Yale University, 1994.

[117] Mathieu Blondel, Arthur Mensch, and Jean-Philippe Vert. "Differentiable divergences between time series". In: *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics*. Vol. 130. PMLR. 2021, pp. 3853–3861.

[118] Marco Cuturi, Jean-Philippe Vert, Øystein Birkenes, and Tomoko Matsui. "A kernel for time series based on global alignments". In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2007, pp. II-413–II-416.

[119] Marco Cuturi. "Fast global alignment kernels". In: *Proceedings of the 28th International Conference on Machine Learning*. 2011, pp. 929–936.

[120] Jason Lines and Anthony Bagnall. "Time series classification with ensembles of elastic distance measures". In: *Data Mining and Knowledge Discovery* 29.3 (2015), pp. 565–592.

[121] Zoltán Bankó and János Abonyi. "Dynamic time warping of segmented time series". In: *Advances in Intelligent and Soft Computing* 75 (2010), pp. 117–125.

[122] Vincent Arsigny, Pierre Fillard, Xavier Pennec, and Nicholas Ayache. "Geometric means in a novel vector space structure on symmetric positive-definite matrices". In: *SIAM Journal on Matrix Analysis and Applications* 29.1 (2007), pp. 328–347.

[123] Xavier Pennec, Pierre Fillard, and Nicholas Ayache. "A Riemannian framework for tensor computing". In: *International Journal of Computer Vision* 66.1 (2006), pp. 41–66.

[124] Mineichi Kudo, Jun Toyama, and Masaru Shimbo. "Multidimensional curve classification using passing-through regions". In: *Pattern Recognition Letters* 20.11–13 (1999), pp. 1103–1111.

[125] Anthony Bagnall et al. *The UEA multivariate time series classification archive, 2018*. arXiv:1811.00075. 2018.

[126] Saul B. Needleman and Christian D. Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins". In: *Journal of Molecular Biology* 48.3 (1970), pp. 443–453.

[127] Temple F. Smith and Michael S. Waterman. "Identification of common molecular subsequences". In: *Journal of Molecular Biology* 147.1 (1981), pp. 195–197.

[128] John Paparrizos and Sai Prasanna Teja Reddy. "Odyssey: an engine enabling the time-series clustering journey". In: *Proceedings of the VLDB Endowment* 16.12 (2023), pp. 4066–4069.

[129] John Paparrizos and Sai Prasanna Teja Reddy Bogireddy. "Time-series clustering: a comprehensive study of data mining, machine learning, and deep learning methods". In: *Proceedings of the VLDB Endowment* 18.11 (2025), pp. 4380–4395.

[130] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. New York: Wiley, 1990.

[131] Raghu Krishnapuram, Anupam Joshi, Olfa Nasraoui, and Liyu Yi. "Low-complexity fuzzy relational clustering algorithms for web mining". In: *IEEE Transactions on Fuzzy Systems* 9.4 (2001), pp. 595–607.

[132] Robert J. Alcock and Yannis Manolopoulos. "Time-series similarity queries employing a feature-based approach". In: *Proceedings of the 7th Hellenic Conference on Informatics*. Ioannina, 1999.

[133] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. "On spectral clustering: analysis and an algorithm". In: *Advances in Neural Information Processing Systems 14*. 2002, pp. 849–856.

[134] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. "A density-based algorithm for discovering clusters in large spatial databases with noise". In: *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD)*. 1996, pp. 226–231.

[135] Ricardo J. G. B. Campello, Davoud Moulavi, and Jörg Sander. "Density-based clustering based on hierarchical density estimates". In: *Advances in Knowledge Discovery and Data Mining (PAKDD 2013)*. Vol. 7819. Lecture Notes in Computer Science. 2013, pp. 160–172.

[136] Ana L. N. Fred and Anil K. Jain. "Combining multiple clusterings using evidence accumulation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27.6 (2005), pp. 835–850.

[137] John W. Sammon. "A nonlinear mapping for data structure analysis". In: *IEEE Transactions on Computers* C-18.5 (1969), pp. 401–409.

[138] Pablo Arbeláez, Michael Maire, Charless Fowlkes, and Jitendra Malik. "Contour detection and hierarchical image segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.5 (2011), pp. 898–916.

[139] Félix Chavelli, Paul Boniol, and Michaël Thomazo. "Toward interpretable evaluation measures for time series segmentation". In: *Advances in Neural Information Processing Systems 38 (NeurIPS 2025)*. 2025.

[140] Asa Ben-Hur, André Elisseeff, and Isabelle Guyon. "A stability based method for discovering structure in clustered data". In: *Pacific Symposium on Biocomputing*. 2002, pp. 6–17.

[141] Janez Demšar. "Statistical comparisons of classifiers over multiple data sets". In: *Journal of Machine Learning Research* 7 (2006), pp. 1–30.

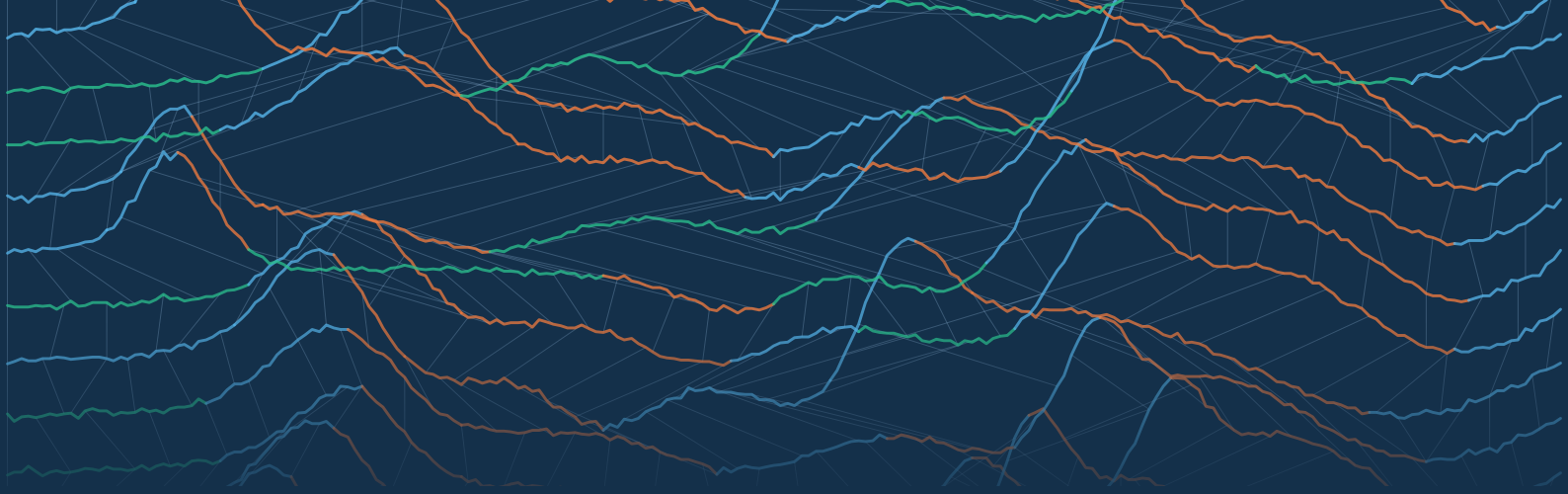
[142] Alessio Benavoli, Giorgio Corani, and Francesca Mangili. "Should we really use post-hoc tests based on mean-ranks?" In: *Journal of Machine Learning Research* 17.5 (2016), pp. 1–10.

[143] Western Electric Company. *Statistical Quality Control Handbook*. Indianapolis: Western Electric Co., 1956.

[144] Hoang Anh Dau et al. "The UCR time series archive". In: *IEEE/CAA Journal of Automatica Sinica* 6.6 (2019), pp. 1293–1305.

[145] Matei Zaharia et al. "Accelerating the machine learning lifecycle with MLflow". In: *IEEE Data Engineering Bulletin* 41.4 (2018), pp. 39–45.

[146] Margaret Mitchell et al. "Model cards for model reporting". In: *Proceedings of the Conference on Fairness, Accountability, and Transparency (FAT* '19)*. 2019, pp. 220–229.



How to cut, align and group multivariate time series

Process records, sensor streams, signatures and motion recordings are multivariate time series whose meaning lies in how the variables move together. This book shows how to cut them into homogeneous segments, how to compare them when their time axes do not agree, and how to group them — and how to check, explain and operate the resulting models.

It starts from the segmentation and warping methods developed at the University of Pannonia over two decades — PCA- and dynamic-PCA-based segmentation, fuzzy segmentation with the modified Gath–Geva algorithm, recursive PCA for on-line segmentation, Fisher-information segmentation for experiment design, correlation-based dynamic time warping, mixed piecewise linear measures and episode alignment — and places them among the best methods of the field: PELT and optimal partitioning, Bayesian on-line change point detection, hidden Markov and sparse Gaussian state models, ClaSP and FLUSS, soft-DTW, k-Shape and DBA. New methods include predictive homogeneity with any scikit-learn model, Riemannian segment distances, merge persistence and consensus segmentation.

Every method is implemented in `mvtsa`, an open-source Python package with evaluation, validation, visualization and model-operations tools. Each figure in the book is produced by an example script of the package; the same scripts run as Jupyter notebooks.

```
pip install mvtsa
```

github.com/ComplexSystemsIntelligence/mvtsa