

On the Efficiency and Interpretability of Permutation Entropy in Quantitative Electroencephalography

Sebastian Johann Georg Berger

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor-Ingenieurs (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender: Prof. Dr. Oliver Hayden

Prüfer der Dissertation: 1. Prof. Dr.-Ing. Werner Hemmert
2. Prof. Dr. Gerhard Schneider


Die Dissertation wurde am 15.06.2020 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 05.03.2021 angenommen.

Sebastian Berger. *On the Efficiency and Interpretability of Permutation Entropy in Quantitative Electroencephalography*. Dissertation. Technische Universität München, Munich, Germany, 2021.

© 2021 Sebastian Berger.

Department of Electrical and Computer Engineering, Technische Universität München, 80333 Munich, Germany.

Department of Anaesthesiology and Intensive Care, Klinikum rechts der Isar, Technische Universität München, 81675 Munich, Germany.

 This work is licensed under the Creative Commons BY-NC-ND 3.0 DE Licence. Visit <https://creativecommons.org/licenses/by-nc-nd/3.0/de/> to obtain a copy of this licence, or write to Creative Commons, PO Box 1866, Mountain View, CA 94042.

To Hedwig, Jakob, and Kristine.

Abstract

This dissertation is concerned with *permutation entropy* in its application as a signal parameter for quantitative analysis of the electroencephalogram (EEG). Permutation entropy is a complexity measure from the field of non-linear systems theory. It is based on the idea of quantising the phase space of an embedded time series by means of so-called *ordinal patterns*. The present work focusses on two aspects: the algorithmically efficient extraction of ordinal patterns from arbitrary time series, as well as the interpretability of permutation entropy in the context of EEG analysis.

The fundamental problem of computing ordinal patterns stems from the fact that their number increases with the factorial of the embedding dimension used. The algorithms presented here significantly reduce the runtime of ordinal analysis methods, and also extend the maximum embedding dimension practically available by an order of magnitude.

The question on the meaning of permutation entropy in EEG is answered empirically by the identification of characteristic regularities in the probability distributions of ordinal patterns in EEG. Those results bridge the gap between permutation entropy and the Fourier transform, and thus establish an immediate relation with the classical methodology of spectral EEG analysis.

The discussion of those topics is prepended with a description of the mathematical foundations, as well as an outline on the relations between ordinal time series analysis and non-linear systems theory as whole.

Zusammenfassung

Diese Dissertation befasst sich mit der *Permutationsentropie* in ihrer Anwendung als Signalparameter für die quantitative Analyse des Elektroenzephalogramms (EEG). Die Permutationsentropie ist ein Komplexitätsmaß aus der nichtlinearen Systemtheorie. Sie beruht auf der Idee, den Phasenraum einer eingebetteten Zeitreihe mit Hilfe so genannter *ordinaler Muster* zu diskretisieren. In vorliegender Arbeit stehen diesbezüglich zwei Aspekte im Vordergrund: die algorithmisch effiziente Gewinnung von ordinalen Mustern aus allgemeinen Zeitreihen, sowie die Interpretierbarkeit der Permutationsentropie im Kontext der EEG-Analyse.

Das grundsätzliche Problem bei der Berechnung ordinaler Muster besteht darin, dass deren Anzahl mit der Fakultät der verwendeten Einbettungsdimension anwächst. Die hier vorgestellten Algorithmen verkürzen die Laufzeit ordinaler Analyseverfahren deutlich und erhöhen zudem die in der Praxis maximal nutzbare Einbettungsdimension um eine Größenordnung.

Die Frage nach der Bedeutung der Permutationsentropie des EEG wird durch den Nachweis charakteristischer Regelmäßigkeiten in den Wahrscheinlichkeitsverteilungen der ordinalen Muster des EEG empirisch beantwortet. Diese Ergebnisse erlauben einen Brückenschlag zwischen Permutationsentropie und Fouriertransformation und stellen somit einen unmittelbaren Bezug zur klassischen Methodik der spektralen EEG-Analyse her.

Der Behandlung dieser Themen sind eine Beschreibung der mathematischen Grundlagen, sowie eine Einordnung der ordinalen Zeitreihenanalyse in den Gesamtzusammenhang der nichtlinearen Systemtheorie vorangestellt.

Acknowledgement

I want to convey my sincere gratitude to Prof. Dr.-Ing. Werner Hemmert, whose support and careful supervision guided me through all phases of my doctoral candidacy.

I also want to express my profound gratitude to Prof. Dr. med. Gerhard Schneider, who enabled me to work full-time in an exceptional research environment, granted me an extraordinary degree of creative latitude, constantly encouraged me to think outside the box, and provided hands-on support when the going got rough.

Moreover, I want to thank Prof. Dr. med. Eberhard F. Kochs, who set up my initial position as a research assistant, considered my opinion on technical matters from day one, and allowed me to take project responsibility at an early stage of my scientific career.

Furthermore, I want to thank my long-time mentor, Prof. Dr. rer. nat. Denis Jordan, who invited me to become a doctoral candidate, introduced me to quantitative electroencephalography, and got me excited about ordinal pattern analysis. You always found the patience and time to answer my questions, support my ideas, and embrace my scepticism with open ears, open heart, and open mind. It was a privilege to be working with you.

I also want to thank Dr. med. Andreas Ranft for all of his support. Thank you for your interest in my work, your broad expertise, your honest feedback and constant encouragement. I tremendously enjoyed our interdisciplinary discourses.

Another big thank you goes to Andrii Kravtsiv, who not only shared the office, but also all the highs and lows of everyday research with me. Thank you for brainstorming, developing, publishing and travelling together (I will never forget Kyiv).

Over the years, I was fortunate enough to collaborate with many other amazing people, and therefore want to express my deep appreciation for everybody who contributed in one way or another to the success of this endeavour. Thank you all.

A final thank you goes to my friends and family for providing constant moral support. Above all, I will be forever grateful to my beloved Hedwig, Jakob, and Kristine, who went through a lot of zigzags with me, but never bothered to count them. This work is dedicated to the three of you.

Contents

1	Introduction	1
2	Mathematical Preliminaries	5
2.1	Iversonian Brackets	5
2.2	Combinatorics	6
2.2.1	Totally Ordered Sets	6
2.2.2	Permutations	6
2.2.3	The Factorial Number System	7
2.3	Probability Theory	8
2.3.1	Power Sets	8
2.3.2	Sigma-Algebras	9
2.3.3	Probability Spaces	9
2.3.4	Discrete Random Variables	10
2.3.5	Discrete Stochastic Processes	12
2.3.6	Other Forms of Notation	13
2.3.7	Markov Chains	14
2.4	Information Theory	17
2.4.1	Shannon Entropy	17
2.4.2	Joint Entropy	19
2.4.3	Conditional Entropy	19
2.4.4	Transfer Entropy	20
3	Chaos and Determinism	23
3.1	Dynamical Systems	23
3.1.1	Fixed-Point Attractors	25
3.1.2	Periodic Attractors	26
3.1.3	Strange Attractors	28
3.2	Delay Embeddings	29
3.3	Entropy as a Complexity Measure	31
3.3.1	Entropy in Dynamical Systems	31

3.3.2	Ordinal Complexity	32
4	Ordinal Patterns	35
4.1	Definition and Notation	35
4.1.1	Permutation Representation	36
4.1.2	Rank Representation	37
4.1.3	Definition of Ordinal Patterns	39
4.2	Sequences of Ordinal Patterns	40
4.2.1	Partitioning the Euclidean Space	40
4.2.2	Beyond Delay Embeddings and Phase Spaces	42
4.2.3	Ordinal Processes	43
4.2.4	Ordinal Markov Chains	44
4.3	Permutation Entropy	44
4.3.1	Empirical Permutation Entropy	46
4.3.2	Properties of Permutation Entropy	47
4.4	Symbolic Transfer Entropy	48
4.4.1	Neuroscientific Purpose	48
4.4.2	Mathematical Formalism	48
5	Encoding Ordinal Patterns	51
5.1	Requirements and Pitfalls	52
5.2	The Lehmer Code	53
5.3	Lehmer-Encoding Ordinal Patterns	55
5.4	The Keller-Sinn-Emonds Code	58
5.5	Algorithms for Encoding Time Series	59
5.5.1	Plain Algorithm	60
5.5.2	Overlap Algorithm	61
5.5.3	Lookup Algorithm	64
5.6	Implementation and Run-Time Performance	66
5.6.1	Asymptotic Computational Complexities	67
5.6.2	Memory Alignment	68
5.6.3	Compilation versus Interpretation	69
5.6.4	Run-Time Test Environment	70
5.6.5	Test Signal Generation	70
5.6.6	Plain Algorithm	72
5.6.7	Overlap Algorithm	76

5.6.8	High Pattern Orders	77
5.6.9	Lookup Algorithm	87
5.6.10	Sequence Length and Time Lag	89
5.7	Summary	91
6	Permutation Entropy of the Electroencephalogram	93
6.1	Electroencephalography	93
6.1.1	Quantitative EEG Analysis	95
6.2	Permutation Entropy as an EEG Parameter	97
6.2.1	Estimating Permutation Entropy from EEG	98
6.3	The Dynamics of Permutation Entropy	98
6.3.1	Probability Reallocation	99
6.3.2	Probability Balance Coefficients	100
6.4	Complexity and Pseudo-Complexity	102
6.4.1	A New Class of Ordinal Patterns?	103
6.4.2	Pseudo-Complexity in Ordinal Pattern Analysis	104
6.5	The Entropy of Peaks	105
6.5.1	An Open-Source, Open-Data Approach	106
6.5.2	EEG Segmentation and Processing	106
6.5.3	Principle Components of the Probability Balances	107
6.5.4	Eliminating Pseudo-Complexity	110
6.6	Linearising Permutation Entropy	113
6.6.1	Signal Peaks and Spectral Bandwidth	113
6.6.2	Counting Zigzags in EEG	115
6.7	Permutation Entropy as a Spectral Estimator	117
6.7.1	Zero Crossings and the Dominant Frequency Principle	117
6.7.2	Higher Order Crossings	118
6.7.3	The Spectral Estimation Hypothesis	119
6.8	Higher Pattern Orders and Time Lags	120
6.8.1	Prospects for Patterns of Higher Order	120
6.8.2	Sampling, Resampling and Aliasing	120
6.8.3	The Time Lag as a Downsampling Factor	122
6.8.4	Frequency Aliasing in Permutation Entropy	123
6.8.5	Anti-Aliased Permutation Entropy	125
6.9	Summary	125

7	Conclusions	129
7.1	Efficiency	129
7.2	Interpretability	130

List of Figures

1.1	Permutation Entropy During Loss Of Responsiveness	2
2.1	State Diagram of a Markov Chain	16
3.1	Series RLC Resonator (Circuit Diagram)	25
3.2	Series RLC Resonator (Phase Portrait)	26
3.3	Van der Pol Oscillator (Phase Portrait)	27
3.4	Lorenz System (Phase Portrait and Chaotic Bifurcation)	28
3.5	Lorenz System (Attractor Reconstruction)	30
4.1	Ordinal Patterns of Order $m = 4$	38
4.2	Ordinal Partitioning in Three Dimensions	41
4.3	Extracting Ordinal Patterns from a Time Series	42
4.4	Ordinal Process as a Markov Chain	45
5.1	Overlapping Patterns of Order $m = 4$	64
5.3	Performance of Vectorised Plain vs. Overlap Algorithm (MATLAB) . . .	76
5.4	Performance of Plain vs. Overlap Algorithm	77
5.5	Memory Alignment for High Pattern Orders	78
5.6	Performance of Standard vs. Arbitrary-Precision Overlap Algorithm . . .	85
5.7	Performance of Arbitrary-Precision Overlap Algorithm	86
5.8	Performance of Lookup Algorithm for Varying Signal Bandwidth	88
5.9	Performance of Lookup vs. Overlap Algorithm	89
5.10	Influence of Sequence Length on Performance	90
5.11	Influence of Time Lag Parameter on Performance	91
6.1	International 10–20 System (EEG Electrode Positions)	93
6.2	Exemplary Multi-Channel EEG Recording	94
6.3	Binary Entropy Function	101
6.4	Extra-Ordinal Patterns	103
6.5	Kernel Density Estimate for Permutation Entropy in Sleep EEG	107

List of Figures

6.6	Kernel Density Estimate for Principle Components	109
6.7	Entropy-of-Peaks Function	112
6.8	Power Transfer Function of the ∇ -Operator	119
6.9	Permutation Entropy vs. Peak Probability for Orders $m \in \{3, 4, 5\}$. . .	121
6.10	Frequency Aliasing Function	122
6.11	Frequency Aliasing in Permutation Entropy	124

List of Tables

5.1	Mapping Permutations onto Inversion Counts	54
5.2	Lehmer-Encoded Patterns of Order $m = 4$	56
5.3	Operation Counts of Encoding Algorithms	67
5.4	Memory Requirements of Ordinal Patterns	69
5.5	Performance of the Plain Algorithm on Various Platforms	72
5.6	Performance of the Vectorised Plain Algorithm on Various Platforms	75
6.1	Principle Components of Balance Coefficients	108
6.2	First Principle Component of Balance Coefficients	111
6.3	Correlation between Permutation Entropy and Peak Probability	116

List of Publications

- M. Kreuzer, M. A. Stern, D. Hight, S. Berger, G. Schneider, J. W. Sleight, and P. S. García. “Spectral and Entropic Features Are Altered by Age in the Electroencephalogram in Patients under Sevoflurane Anesthesia”, *Anesthesiology*, vol. 132, pp. 1003–1016, 2020. DOI: 10.1097/ALN.0000000000003182.
- S. Berger, A. Kravtsiv, G. Schneider, and D. Jordan. “Teaching Ordinal Patterns to a Computer: Efficient Encoding Algorithms Based on the Lehmer Code”, *Entropy*, vol. 21, no. 10, 1023, 2019. DOI: 10.3390/e21101023.
- H. Lee, D. Golkowski, D. Jordan, S. Berger, R. Ilg, J. Lee, G. A. Mashour, and U. Lee. “Relationship of critical dynamics, functional connectivity, and states of consciousness in large-scale human brain networks”, *NeuroImage*, vol. 188, pp. 228–238, 2019. DOI: 10.1016/j.neuroimage.2018.12.011.
- C. Heiser, P. Fthenakis, A. Hapfelmeier, S. Berger, B. Hofauer, W. Hohenhorst, E. F. Kochs, K. J. Wagner, and G. M. Edenharter. “Drug-induced sleep endoscopy with target-controlled infusion using propofol and monitored depth of sedation to determine treatment strategies in obstructive sleep apnea”, *Sleep and Breathing*, vol. 21, no. 3, pp. 737–744, 2017. DOI: 10.1007/s11325-017-1491-8.
- D. Golkowski, A. Ranft, T. Kiel, V. Riedl, P. Kohl, G. Rohrer, J. Pientka, S. Berger, C. Preibisch, C. Zimmer, G. A. Mashour, G. Schneider, E. F. Kochs, R. Ilg, and D. Jordan. “Coherence of BOLD signal and electrical activity in the human brain during deep sevoflurane anesthesia”, *Brain and Behavior*, vol. 7, no. 7, e00679, 2017. DOI: 10.1002/brb3.679.
- S. Berger, G. Schneider, E. F. Kochs, and D. Jordan. “Permutation Entropy: Too Complex a Measure for EEG Time Series?”, *Entropy*, vol. 19, no. 12, 692, 2017. DOI: 10.3390/E19120692.
- A. Ranft, D. Golkowski, T. Kiel, V. Riedl, P. Kohl, G. Rohrer, J. Pientka, S. Berger, A. B. Thul, M. Maurer, C. Preibisch, C. Zimmer, G. A. Mashour, E. F. Kochs, D. Jordan, and R. Ilg. “Neural Correlates of Sevoflurane-induced Un-

consciousness Identified by Simultaneous Functional Magnetic Resonance Imaging and Electroencephalography”, *Anesthesiology*, vol. 125, no. 5 , pp. 861–872, 2016. DOI: 10.1097/ALN.0000000000001322.

- D. Jordan, R. Ilg, V. Riedl, A. Schorer, S. Grimberg, S. Neufang, A. Omerovic, S. Berger, G. Untergehrer, C. Preibisch, E. Schulz, T. Schuster, M. Schröter, V. Spormaker, C. Zimmer, B. Hemmer, A. Wohlschläger, E. F. Kochs, and G. Schneider. “Simultaneous Electroencephalographic and Functional Magnetic Resonance Imaging Indicate Impaired Cortical Top–Down Processing in Association with Anesthetic-induced Unconsciousness”, *Anesthesiology*, vol. 119, no. 5, pp. 1031–1042, 2013. DOI: 10.1097/ALN.0b013e3182a7ca92.

1 Introduction

When the article *Permutation Entropy: A Natural Complexity Measure for Time Series* [1] by Christoph Bandt and Bernd Pompe appeared in 2002, it established a novel perspective on time series: instead of considering absolute amplitudes, one focusses exclusively on the rank associations between adjacent values. Those are called the *ordinal patterns* of the time series. The approach is conceptually similar to the rank correlation coefficients used in non-parametric statistics—where, for instance, the Spearman correlation between a pair of observables is the Pearson correlation between their respective ordinal ranks [2].

In their original publication, Bandt and Pompe anticipated that permutation entropy would become particularly relevant for the analysis of “heart and brain data” [1]. About two years later, Yinhe Cao and colleagues reported on having successfully detected epileptic seizures in the electroencephalogram (EEG) by means of permutation entropy [3], which renders those authors the first to propose an EEG-related application of this complexity measure. More articles on using permutation entropy for the analysis of seizure EEG followed over the years [4–13].

Sleep research is another field that took strong interest in permutation entropy as an EEG parameter: Gu Li, Yingle Fan, and Quan Pang were apparently the first to use it for automatic sleep scoring [14], and subsequent publications on sleep-related changes of permutation entropy in EEG include [10, 12, 15–19]. Apart from that, permutation entropy has also been applied to EEG in the contexts of coma [20–23], Alzheimer’s disease [24, 25], aging and cognitive decline [26, 27], stroke [28–30], stress [31–33], and in various other clinical and experimental settings [34–41].

Judging by the number of articles published, however, the area where permutation entropy made the biggest impact as an EEG parameter is: monitoring the “depth” of general anaesthesia. Almost simultaneously, three articles on this specific application were published in 2008 [42–44], and have likely inspired some of the studies that followed [45–75]. The following example may help substantiate the relevance of permutation entropy for EEG analysis, and for the monitoring of general anaesthesia, in particular.

Example 1. *General anaesthesia is commonly induced by intravenous administration of an hypnotic drug (like propofol, for instance). The patient to be anaesthetised usually stops responding to verbal command within seconds after the injection, and this loss of responsiveness (LoR) is a clinical sign that anaesthesia has successfully been induced [76].*

For the purpose of clinical trials, the LoR can be assessed systematically by repeatedly instructing the subject to squeeze the investigator’s hand [58]. Data recorded during the induction can then be put in temporal relation with the LoR. An exemplary epoch of EEG, as well as its corresponding permutation entropy values, are depicted in Figure 1.1.

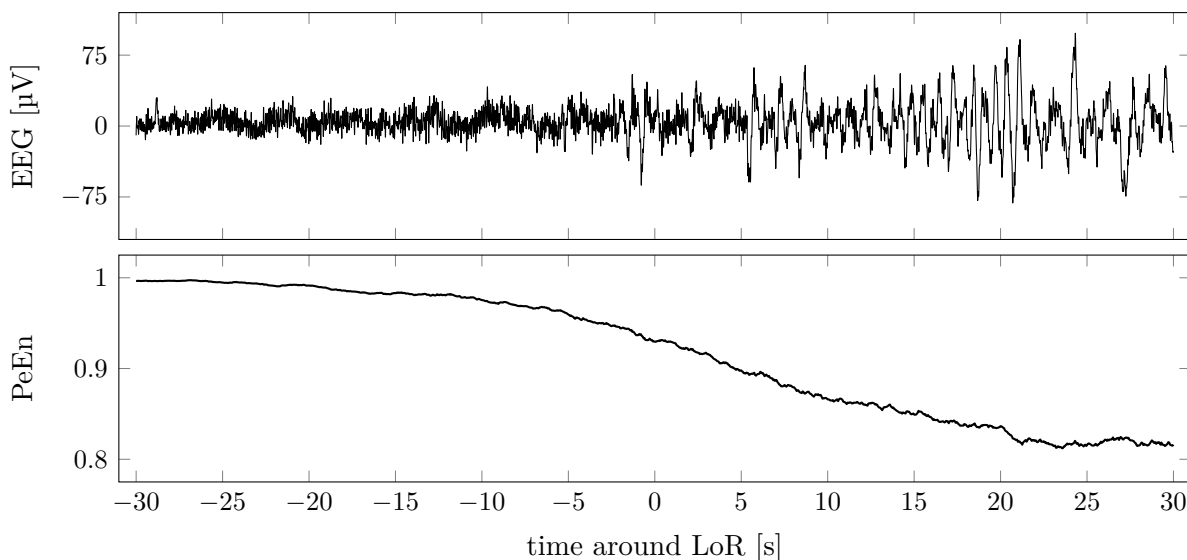


Figure 1.1 An epoch of EEG as recorded from a patient undergoing induction of general anaesthesia via propofol injection. Electrode locations were Fp1–Cz (see Figure 6.1), and data were sampled at 200 Hz. Permutation entropy (PeEn, order $m = 3$, time lag $\tau = 1$, normalised, see Chapter 4) was computed on maximally-overlapping signal windows of 10 s length. Its values suggest a monotonous transition from the wakeful to the anaesthetised state, and perfect discrimination between the two.

Despite the fact that permutation entropy demonstrates high performance in various EEG classification tasks, the question *why* this method actually yields such favourable results has apparently not been raised. But does a method proposed as a complexity measure for time series necessarily measure complexity in all kinds of time series? And if so, can we stop at that point, or do we have to ask further questions about the nature of this complexity?

From this author's perspective, the sheer number of reports on permutation entropy in quantitative EEG analysis provided sufficient reason to set out and study the internals of this measure: after all, anything learned about a single EEG parameter can be something learned about EEG as a whole, and possibly about the underlying neural processes.

This endeavour yielded new insights into the relations between ordinal patterns and the EEG [77], as well as efficient algorithms for ordinal time series analysis in general [78]. Those results are described in the dissertation at hand, and the author here seizes the opportunity to discuss some aspects in more depth, and to provide a careful introduction to the theoretical underpinnings. The work is organised in the following way:

- Chapter 2 introduces the mathematical preliminaries used throughout the thesis. Those include ideas from the fields of combinatorics, probability theory, as well as information theory.
- Chapter 3 provides an overview of dynamical systems theory, delay embeddings, and entropic complexity measures, which constitute the theoretical basis of permutation entropy and related parameters.
- Chapter 4 explains the notion of *ordinal patterns* in detail, and establishes a useful means of notation. The chapter continues with a description of how ordinal patterns relate to stochastic processes and time series, and how probability distributions of ordinal patterns can be characterised by means of *permutation entropy*, as well as *symbolic transfer entropy*.
- Chapter 5 focusses on the computational problem of turning time series into ordinal pattern sequences. A *numerical encoding scheme* for ordinal patterns is proposed, and three different algorithms for their efficient extraction from time series are discussed. Strengths and weaknesses of those algorithms are derived, and further substantiated by benchmark results.
- Chapter 6 gives a short introduction to quantitative electroencephalography, and demonstrates that permutation entropy can be simplified substantially without compromising its suitability as an EEG signal parameter. Implications are discussed in detail, and yield a direct relation between permutation entropy and the power spectrum of the EEG.
- Chapter 7 summarises the ideas and results presented, draws some conclusions on using ordinal patterns for EEG analysis, and provides an outlook on possible future investigations.

2 Mathematical Preliminaries

The study of real-world time series is often an interdisciplinary endeavour, and electroencephalography is a particularly good example: physicians, biologists, neuroscientists, and psychologists, but also mathematicians, physicists, engineers, and computer scientists work together in this field. Any such collaboration depends on communication, and communication requires a language that is both concise and mutually understood. In this vein, a detailed overview of the mathematical concepts that enable the present communication is given in the following.

2.1 Iversonian Brackets

Throughout this work, we shall utilise a highly convenient, if slightly uncommon notational convention called the *Iversonian bracket*. It originated from a book by Kenneth Iverson [79], wherein the author stipulated that relational expressions take on numerical values: 1 if the expression holds true, and 0 otherwise. For instance, $(23 < 42) = 1$, whereas $(255 < -1) = 0$. Donald Knuth extended the scope of this notation from order relations to logical expressions in general. He further suggested to use square brackets for reasons of clarity, and coined the term Iversonian bracket notation [80]. In terms of computer science, Iversonian bracketing represents a data type conversion from Boolean to integer. For a given logical expression L , it holds that

$$[L] = \begin{cases} 0, & \text{if } L \text{ is false,} \\ 1, & \text{if } L \text{ is true.} \end{cases} \quad (2.1)$$

This notational convention is very useful when counting elements to which a certain logical condition applies. For example, the number of positive elements in a finite time series $\{x_1, x_2, \dots, x_N\}$ can compactly be written as $\sum_{t=1}^N [x_t > 0]$.

2.2 Combinatorics

As its name immediately implies, permutation entropy builds upon basic principles from the field of combinatorics. Especially the algorithmic work presented in Chapter 5 depends on some less commonly known combinatorial concepts. All of those aspects are therefore summarised in the following.

2.2.1 Totally Ordered Sets

A *totally ordered set* X is a collection of pairwise distinct elements, endowed with a binary relation \prec , such that the following properties [81] hold for any elements $x_1, x_2, x_3 \in X$:

- If $x_1 \prec x_2$, then $x_1 \neq x_2$ (strictness).
- If $x_1 \prec x_2$ and $x_2 \prec x_3$, then $x_1 \prec x_3$ (transitivity).
- If $x_1 \neq x_2$, then $x_1 \prec x_2$ or $x_2 \prec x_1$ (completeness).

For instance, the natural numbers \mathbb{N} , the integers \mathbb{Z} , as well as the real numbers \mathbb{R} are totally ordered by the binary relation $<$. Conversely, the relation \leq does not establish a total order in any of those sets because it violates the strictness requirement.

2.2.2 Permutations

A *permutation* is a bijective function $\sigma: S \rightarrow S$ on an arbitrary set S . Permutations thus reflect the process of rearranging a collection of pairwise distinct objects. A total of $m!$ different permutation functions exist for any set of $|S| = m$ elements, whereby

$$m! = \prod_{k=1}^m k = m \cdot (m-1) \cdot (m-2) \cdot \dots \cdot 2 \cdot 1 \quad (2.2)$$

is the factorial of m , and represents the number of possible sorting orders that exist for a set of m elements. A permutation function on a set $S = \{s_1, s_2, \dots, s_m\}$ can be written explicitly using the two-row matrix notation in terms of

$$\sigma = \begin{pmatrix} s_1 & s_2 & \dots & s_m \\ \sigma(s_1) & \sigma(s_2) & \dots & \sigma(s_m) \end{pmatrix}. \quad (2.3)$$

Therein, each column represents a particular relation of the form $s_i \mapsto \sigma(s_i)$, so intuitively, the overall matrix serves the purpose of a lookup table. However, if the set S is endowed with a natural order, the upper row of the matrix does not provide any new information, and can therefore be omitted. For instance, the permutation

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 1 & 3 & 2 & 5 \end{pmatrix} = (4, 1, 3, 2, 5)$$

remains unambiguous when not mentioning the order of the first five natural numbers. Depending on context, both forms of notation will be used for the scope of this writing, that is,

$$\sigma = \begin{pmatrix} s_1 & \cdots & s_m \\ \sigma(s_1) & \cdots & \sigma(s_m) \end{pmatrix} = (\sigma(s_1), \dots, \sigma(s_m)). \quad (2.4)$$

2.2.3 The Factorial Number System

The *factorial number system* is a specific realisation of a *mixed-radix positional numeral system*, which in turn is a class of numeral systems studied by Georg Cantor [82]. Recall that in a standard positional numeral system to some base $b \in \mathbb{N}$ (like the ubiquitous decimal system to the base $b = 10$), any non-negative integer $n \in \mathbb{N}_0$ is representable by a distinct numeral, that is, by a string of digits of the form $d_N \dots d_2 d_1 d_0$, for which it holds that

$$n = \sum_{i=0}^N d_i \cdot b^i, \quad \text{where } d_i \in \mathbb{N}_0 \text{ and } d_i < b. \quad (2.5)$$

In comparison with standard numeral systems, the base/radix of a mixed-radix numeral system is not a constant factor, but a function $i \mapsto b(i)$ of the digit position i . Specifically, for base functions $b: \mathbb{N}_0 \rightarrow \mathbb{N}_0$, any non-negative integer $n \in \mathbb{N}_0$ can be written as a unique mixed-radix numeral $d_N \dots d_2 d_1 d_0$, which represents the value

$$n = \sum_{i=0}^N \left(d_i \prod_{j=0}^i b(j) \right), \quad \text{where } d_i \in \mathbb{N}_0 \text{ and } d_i < b(i+1). \quad (2.6)$$

Within this framework, the factorial number system is the mixed-radix numeral system to the base function $b(i) = i$. Its corresponding strings of digits of the form $d_N \dots d_2 d_1 d_0$ are called *factoradic numerals*, and are to be interpreted as in

$$n = \sum_{i=0}^N d_i (i!), \quad \text{where } d_i \in \mathbb{N}_0 \text{ and } d_i < i + 1. \quad (2.7)$$

Frequently, those numerals are subscripted with an “!” where ambiguity would otherwise arise.

Example 2. *The five-digit factoradic numeral $23110_!$ represents the natural number $23110_! = 2 \cdot 4! + 3 \cdot 3! + 1 \cdot 2! + 1 \cdot 1! + 0 \cdot 0! = 69$. By way of further illustration, the factoradic numerals corresponding to the non-negative integers $n \in \{0, 1, \dots, 23\}$ are*

$0_! = 0,$	$1000_! = 6,$	$2000_! = 12,$	$3000_! = 18,$
$10_! = 1,$	$1010_! = 7,$	$2010_! = 13,$	$3010_! = 19,$
$100_! = 2,$	$1100_! = 8,$	$2100_! = 14,$	$3100_! = 20,$
$110_! = 3,$	$1110_! = 9,$	$2110_! = 15,$	$3110_! = 21,$
$200_! = 4,$	$1200_! = 10,$	$2200_! = 16,$	$3200_! = 22,$
$210_! = 5,$	$1210_! = 11,$	$2210_! = 17,$	$3210_! = 23.$

2.3 Probability Theory

Besides being based on combinatorics, permutation entropy also depends on probability theory. This is partly due to the fact that time series are often modelled as realisations of stochastic processes, so time series analysis as a whole is deeply grounded in probability theory, and permutation entropy is no exception in this regard. Moreover, any variant of entropy is of course a probabilistic concept, so the overview of information theory given in Section 2.4 builds on those foundations as well. A basic recapitulation of probability theory is therefore provided in the following.

2.3.1 Power Sets

The *power set* $\mathcal{P}(S)$ of a set S is the set of all possible subsets of S , including the empty set \emptyset , as well as S itself. For example, let $S = \{1, 2, 3\}$, then

$$\mathcal{P}(S) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}.$$

For a set of $|S| = m$ elements, its power set $\mathcal{P}(S)$ contains a total of

$$|\mathcal{P}(S)| = \sum_{k=0}^m \binom{m}{k} = 2^m \tag{2.8}$$

elements (= subsets), which follows directly from the *binomial theorem* [83].

2.3.2 Sigma-Algebras

A σ -algebra Σ on a set S is a family of sets over S (that is, a set of subsets of S) for which certain conditions hold. In particular,

$$S \in \Sigma, \quad (2.9a)$$

$$E \in \Sigma \implies (S \setminus E) \in \Sigma, \quad (2.9b)$$

$$(E_1 \in \Sigma) \wedge (E_2 \in \Sigma) \wedge \cdots \wedge (E_n \in \Sigma) \implies (E_1 \cup E_2 \cup \cdots \cup E_n) \in \Sigma. \quad (2.9c)$$

We therefore say that Σ is closed under complements, and closed under countable unions. Moreover, it holds that $\emptyset \in \Sigma$, because $\emptyset = S \setminus S$.

It is easily confirmed that the power set $\mathcal{P}(S)$ is a possible σ -algebra on the set S . Other examples for σ -algebras, and some reasons for their relevance in probability theory, will be given in the next subsection.

2.3.3 Probability Spaces

A *probability space* is a triple (Ω, Σ, \Pr) , and is used to mathematically model a random experiment. In particular, Ω is the *sample set*, and contains all possible outcomes of the random experiment. For instance, if rolling a six-faced die is modelled, the sample set will usually be defined as $\Omega = \{1, 2, 3, 4, 5, 6\}$.

The set Σ is a σ -algebra on the sample set Ω , and contains the events of the experiment. One could, for example, use $\Sigma = \{\emptyset, \{1, 3, 5\}, \{2, 4, 6\}, \Omega\}$ to define the events “*an even number of pips appeared*” and “*an odd number of pips appeared*” in a die-roll experiment. By definition, Σ will always contain the sets \emptyset and Ω . Those are called the *impossible event* and the *certain event*. Also note that any single outcome from Ω can be an individual event in Σ , and is called an *elementary event*.

Finally, the symbol $\Pr: \Sigma \rightarrow [0, 1] \subset \mathbb{R}$ denotes a function that assigns to each event in Σ a probability between zero and one, and is thus called a *probability measure*. In accordance with the definition of σ -algebras, the probability measure \Pr has the fundamental property that, for any event $E \in \Sigma$,

$$\Pr(E) + \Pr(\Omega \setminus E) = 1. \quad (2.10)$$

2 Mathematical Preliminaries

Consequently, it holds that $\Pr(\Omega) = 1$, whereas $\Pr(\emptyset) = 0$. Moreover, the same property also implies that

$$\Pr(E_1 \cup E_2) = \Pr(E_1) + \Pr(E_2) \quad \text{if} \quad E_1 \cap E_2 = \emptyset. \quad (2.11)$$

Intuitively, if the events E_1 and E_2 are mutually exclusive, then the probability that either E_1 or E_2 will happen is the sum of their individual probabilities.

2.3.4 Discrete Random Variables

A *discrete random variable* is a function $X: \Omega \rightarrow \mathcal{X}$ that maps the set of possible outcomes Ω of a random experiment onto a finite set of values \mathcal{X} . The latter is called the *support set* of the random variable. In case of a die-roll experiment, for instance, one can define a random variable X on the support set $\mathcal{X} = \{1, 2, \dots, 6\}$. For each roll of the die, X will then take on a distinct value $x \in \mathcal{X}$ that reflects the number of pips shown. In this context, x is called a *realisation* of the random variable X .

Depending on the nature of the underlying experiment, X will take on each of its possible values $x \in \mathcal{X}$ at a certain probability. These probabilities are given by the *probability mass function*

$$\begin{aligned} f_X: \mathcal{X} &\rightarrow [0, 1] \subset \mathbb{R}, \\ x &\mapsto \Pr(X = x). \end{aligned} \quad (2.12)$$

Keeping with the die-roll example, and further assuming that a fair die is used, it holds that $f_X(x) = 1/6$ for all $x \in \mathcal{X}$.

A random experiment may involve not only one, but a collection of random variables $\{X_1, \dots, X_m\}$ with their respective support sets $\{\mathcal{X}_1, \dots, \mathcal{X}_m\}$, and the results of those random variables may be interdependent. Such interrelations can be modelled by a *joint probability mass function*

$$\begin{aligned} f_{X_1, \dots, X_m}: \mathcal{X}_1 \times \dots \times \mathcal{X}_m &\rightarrow [0, 1] \subset \mathbb{R}, \\ (x_1, \dots, x_m) &\mapsto \Pr(X_1 = x_1, \dots, X_m = x_m). \end{aligned} \quad (2.13)$$

This function provides the probability that, for the same instance of a random experiment, the variable X_1 takes on the value x_1 , the variable X_2 takes on the value x_2 , and so forth

for all of the m random variables involved, and all of the outcomes those variables may yield.

If the joint probability mass of a set of m random variables is known, the joint probability mass of any *subset* of those random variables can be obtained by summation. For instance, if probabilities in terms of $f_{X,Y,Z}$ are given, it holds that

$$f_{Y,Z}(y, z) = \sum_{x \in \mathcal{X}} f_{X,Y,Z}(x, y, z). \quad (2.14)$$

This also includes the *marginal probability mass* f_{X_i} of any single random variable X_i , which can be derived by successively summing up the m -dimensional joint probability mass function f_{X_1, \dots, X_m} along $m - 1$ of its dimensions. Notice that a pair of random variables X and Y are called *identically distributed* if $f_X = f_Y$ holds for their marginal probability masses.

As stated earlier, joint probabilities are used to model dependencies between random variables. In this regard, consider that a pair of random variables X and Y are called *independent* if $f_{X,Y} = f_X \cdot f_Y$ holds, that is, if the probability of simultaneously observing X and Y in some particular states is fully determined by their respective marginal probability masses. In general, though, the outcome of Y may influence the outcome of X , such that $f_{X,Y} \neq f_X \cdot f_Y$, but rather $f_{X,Y} = f_{X|Y} \cdot f_Y$. In this constellation, the function

$$\begin{aligned} f_{X|Y}: \mathcal{X} \times \mathcal{Y} &\rightarrow [0, 1] \subset \mathbb{R}, \\ (x, y) &\mapsto \Pr(X = x \mid Y = y), \end{aligned} \quad (2.15)$$

is called the *conditional probability mass function* of X given Y , which represents the probability of receiving the outcome $X = x$, provided that the outcome $Y = y$ is already known. The inverse relation $f_{Y|X}$, that is, the dependence of Y on X , can be modelled analogously because, quite intuitively, it holds that

$$f_{X|Y} \cdot f_Y = f_{X,Y} = f_{Y|X} \cdot f_X. \quad (2.16)$$

This interrelation is widely known as *Bayes' Theorem*. In the general case, that is, for a set $\{X_1, X_2, \dots, X_m\}$ of m random variables, conditional probability masses for more than two variables, like $f_{X_1, X_2, X_3|X_4}$ or $f_{X_1, X_2|X_3, X_4}$, can be derived in the same manner. Any such function stems from an m -dimensional joint probability mass f_{X_1, \dots, X_m} , which fully defines the interrelations between the m discrete random variables.

2.3.5 Discrete Stochastic Processes

A *stochastic process* is a set of random variables $\{X_t\}_{t \in T}$ endowed with an index set T that establishes a notion of sequentiality among those variables. We will here be looking at *discrete stochastic processes* exclusively, that is, the random variables will have finite support sets, and the indices $t \in T$ will relate to equidistantly spaced discrete points in time. Without loss of generality, let us assume $T = \mathbb{N}$ for the rest of this section, such that

$$\{X_t\}_{t \in \mathbb{N}} = \{X_1, X_2, \dots, X_N\} \quad \text{with} \quad N \rightarrow \infty. \quad (2.17)$$

What makes a set of time-aligned random variables a discrete stochastic process is that all random variables draw their respective outcomes from the same finite support set \mathcal{X} , and are governed by a joint probability mass function

$$\begin{aligned} f_{X_1, \dots, X_N} : \mathcal{X}^N &\rightarrow [0, 1], \\ (x_1, \dots, x_N) &\mapsto \Pr(X_1 = x_1, \dots, X_N = x_N). \end{aligned} \quad (2.18)$$

As described in Section 2.3.4, this function fully determines all probabilistic relations among the N random variables of the stochastic process.

A particularly important class of stochastic processes are the so-called *stationary processes*. In a nutshell, a discrete stochastic process is called (strictly) stationary if and only if its joint probability mass function is invariant to time shifts, that is, if

$$f_{X_{1+\tau}, \dots, X_{N+\tau}} = f_{X_1, \dots, X_N}, \quad \text{for} \quad \tau \in \mathbb{Z}, \quad N \rightarrow \infty. \quad (2.19)$$

Stationarity is a very powerful property, and has important implications for time series analysis. In general, when sampling from some stochastic process $\{X_t\}$, each random variable can be observed exactly once, because X_t is unknown for all times preceding t , takes on a value $x_t \in \mathcal{X}$ at time t , and then maintains this value forever. In other words, a random variable X_t has exactly one realisation x_t throughout all of space and time. Therefore, after having sampled the realisation $X_t = x_t$, we cannot draw any conclusions on the probability mass function f_{X_t} of that random variable—apart from the humble observation that $f_{X_t}(x_t) \neq 0$, of course.

Conversely, if the process $\{X_t\}$ is known to be stationary in the sense of Equation (2.19), it holds that $f_{X_i} = f_{X_j}$ for any $i \in \mathbb{N}$ and $j \in \mathbb{N}$, and all random variables in $\{X_t\}$ are

thus identically distributed. Their marginal probability mass function,

$$f_X = f_{X_1} = f_{X_2} = \cdots = f_{X_N} \quad \text{for} \quad N \rightarrow \infty,$$

can then be estimated from a sequence of realisations sampled over time, that is, from a time series $\{x_{t_1}, x_{t_2}, \dots, x_{t_N}\}$, where $t_i \in \mathbb{N}$ for all $i \in \{1, 2, \dots, N\}$. More specifically, the limit

$$\forall x \in \mathcal{X}: \quad f_X(x) = \lim_{N \rightarrow \infty} \frac{1}{N} \sum_{i=1}^N [x_{t_i} = x] \quad (2.20)$$

represents the probability $\Pr(X_t = x)$ of finding the stationary process $\{X_t\}$ in a particular state x of its support set \mathcal{X} , for any arbitrary observation time t .

Joint probability mass functions, as in $f_{X_t, X_{t+\tau}, X_{t+2\tau}}$, can be estimated analogously, that is, by counting tuples of occurrences. Notice that the absolute time index t is insignificant here: for a stationary process, any joint probability mass function is fully determined by the temporal distance between the random variables involved.

2.3.6 Other Forms of Notation

While it is formally correct to write probability mass functions as in Equation (2.18), this notation becomes quite cumbersome for more involved relations. Where no ambiguity can arise, it is therefore common practice to use the notation $p(x_{t_1}, \dots, x_{t_N})$ instead, whereby

$$p(x_{t_1}, \dots, x_{t_N}) = f_{X_{t_1}, \dots, X_{t_N}}(x_{t_1}, \dots, x_{t_N}) = \Pr(X_{t_1} = x_{t_1}, \dots, X_{t_N} = x_{t_N}). \quad (2.21)$$

This shorthand is especially useful in the context of stationary processes, which are invariant to time shifts anyway (see Section 2.3.5).

A similar notational convention can be used for joint and conditional probabilities. Given two discrete stochastic processes $\{X_t\}$ and $\{Y_t\}$, we will thus write

$$p(x_{t_1} \mid y_{t_2}) = \frac{p(x_{t_1}, y_{t_2})}{p(y_{t_2})} \quad (2.22)$$

for the probability $f_{X_{t_1} \mid Y_{t_2}}(x_{t_1}, y_{t_2}) = \Pr(X_{t_1} = x_{t_1} \mid Y_{t_2} = y_{t_2})$ of finding the variable X_{t_1} in state $x_{t_1} \in \mathcal{X}$ at time t_1 , if it is known that Y_{t_2} takes on $y_{t_2} \in \mathcal{Y}$ at time t_2 .

For marginal probabilities, there is another useful means of notation. Given the probability mass $f_{X_t}(x)$ of a random variable X_t on the support set $\mathcal{X} = \{x_1, x_2, \dots, x_N\}$, a time-dependent vector of $|\mathcal{X}| = N$ elements,

$$\mathbf{p}(t) = \left(f_{X_t}(x_1) \quad f_{X_t}(x_2) \quad \cdots \quad f_{X_t}(x_N) \right)^\top, \quad (2.23)$$

can be constructed, and is referred to as a *stochastic vector*, or *probability vector*. Due to the properties of probability mass functions, it holds that

- $\mathbf{p}(t) \geq \mathbf{0}$, that is, the vector is non-negative, which in turn means that each of its elements is non-negative [84], and
- $\|\mathbf{p}(t)\|_1 = 1$, that is, the sum over all elements of $\mathbf{p}(t)$ equals 1.

Stationary processes are characterised by a single, time-independent probability vector, which can be written as a constant \mathbf{p} instead.

2.3.7 Markov Chains

A discrete stochastic process $\{X_t\}$, with $t \in \mathbb{Z}$, is called a *Markov chain* if it holds that

$$p(x_t \mid x_{t-1}) = p(x_t \mid x_{t-1}, x_{t-2}, x_{t-3}, \dots), \quad (2.24)$$

which is hence known as the *Markov property* [84]. In other words, the state x_t of a Markov chain is no less predictable from the preceding state x_{t-1} alone than it is predictable from the entire past of the process. Notice that the Markov property does in no way imply that the state x_t be independent from the earlier states $\{x_{t-2}, x_{t-3}, \dots\}$. Much rather, x_t depends on x_{t-1} , which depends on x_{t-2} , which in turn depends on x_{t-3} , and so forth, giving rise to the notion of a chain indeed. By contrast with other discrete stochastic processes, though, the decisive history of a Markov chain can be accumulated in a single state. For this reason, Markov chains may be called *memoryless*.

The probabilistic properties of a Markov chain with the state space $\mathcal{X} = \{x_1, x_2, \dots, x_m\}$ can be described by an $m \times m$ transition matrix

$$\mathbf{P} = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,m} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m,1} & p_{m,2} & \cdots & p_{m,m} \end{pmatrix}. \quad (2.25)$$

If the transition matrix \mathbf{P} is time-invariant, which we assume throughout this writing, the Markov chain is called *homogeneous*. The matrix element $p_{i,j} = \Pr(X_t = x_j \mid X_{t-1} = x_i)$ represents the probability that the Markov chain will transition from state x_i to state x_j in a single step. The rows of the matrix \mathbf{P} are transposed probability vectors in the sense of Equation (2.23), so each row sums up to 1. This is consistent with intuition, because at each time step, we know with absolute certainty that one out of the m possible transitions from x_i to x_j will happen. Given that probability vectors do not contain negative elements, it also follows that $\mathbf{P} \geq \mathbf{0}$, which means that transition matrices are non-negative [84].

Using its transition matrix \mathbf{P} , the time-dependent evolution of a Markov chain can be expressed by the recursion

$$\mathbf{p}(t+1) = \mathbf{P}^\top \mathbf{p}(t), \quad \text{for all } t \in \mathbb{Z}, \quad (2.26)$$

whereby $\mathbf{p}(t)$ represents the marginal probability mass of the process at time t . For each iteration, the transition matrix \mathbf{P} thus redistributes the probabilities $\mathbf{p}(t)$ of finding the Markov chain in a particular state. Just like other stochastic processes, Markov chains can be stationary in the sense of Equation (2.19). As has been described in Section 2.3.5, stationarity implies that the probability vector \mathbf{p} is time-invariant. It therefore holds for any *stationary Markov chain* that $\mathbf{p} = \mathbf{P}^\top \mathbf{p}$, such that the probability vector \mathbf{p} is an eigenvector with eigenvalue $\lambda = 1$ of the transposed transition matrix \mathbf{P}^\top .

Example 3. Let $\{X_t\}$, with $t \in \mathbb{N}$, be a Markov chain on the state space $\mathcal{X} = \{x_1, x_2, x_3\}$, and governed by the transition matrix

$$\mathbf{P} = \begin{pmatrix} 0.6 & 0.2 & 0.2 \\ 0.1 & 0.5 & 0.4 \\ 0.3 & 0.3 & 0.4 \end{pmatrix}.$$

Also assume that the Markov chain is known to be in state $X_1 = x_2$ at time $t = 1$, such that its initial probability vector is $\mathbf{p}(1) = (0 \ 1 \ 0)^\top$. By iterating Equation (2.26), we can obtain

$$\mathbf{p}(2) = \mathbf{P}^\top \mathbf{p}(1) = \begin{pmatrix} 0.6 & 0.1 & 0.3 \\ 0.2 & 0.5 & 0.3 \\ 0.2 & 0.4 & 0.4 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = (0.1 \ 0.5 \ 0.4)^\top,$$

2 Mathematical Preliminaries

and subsequently,

$$\begin{aligned}
 \mathbf{p}(3) &= (0.2300 \quad 0.3900 \quad 0.3800)^\top, \\
 \mathbf{p}(4) &= (0.2910 \quad 0.3550 \quad 0.3540)^\top, \\
 \mathbf{p}(5) &= (0.3163 \quad 0.3419 \quad 0.3418)^\top, \\
 &\vdots \\
 \mathbf{p}(9) &= (0.3332 \quad 0.3334 \quad 0.3334)^\top, \\
 \mathbf{p}(10) &= (0.3333 \quad 0.3334 \quad 0.3334)^\top, \\
 \mathbf{p}(11) &= (0.3333 \quad 0.3333 \quad 0.3333)^\top, \\
 \mathbf{p}(12) &= (0.3333 \quad 0.3333 \quad 0.3333)^\top, \\
 &\vdots
 \end{aligned}$$

for any time step $t \in \mathbb{N}$.

Consistent with the above, the matrix \mathbf{P}^\top has three eigenvalues $\lambda_1 = 1$, $\lambda_2 = 0.4$, $\lambda_3 = 0.1$, and it holds for the eigenvector \mathbf{x}_1 associated with λ_1 that

$$\frac{\mathbf{x}_1}{\|\mathbf{x}_1\|_1} = \mathbf{p} = (1/3 \quad 1/3 \quad 1/3)^\top,$$

which is the stationary distribution the stochastic process converges to. Figure 2.1 depicts a state diagram for the Markov chain considered in this example.

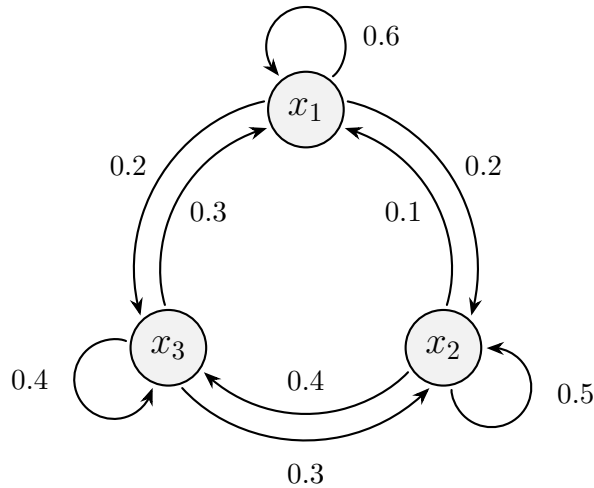


Figure 2.1 State diagram for a Markov chain with support set $\mathcal{X} = \{x_1, x_2, x_3\}$, and transition probabilities as given in Example 3.

2.4 Information Theory

Claude Shannon is considered the creator of *information theory*. To this day, his famous article series *A Mathematical Theory of Communication* [85, 86] constitutes the mathematical basis of virtually any communication technology our professional and private lives have grown to rely upon. The recurring theme in information theory is *entropy*, a statistic on the irregularity of a probability distribution. Under certain model assumptions, irregularity is equivalent to information content, which explains the pivotal role of entropy in information theory. While independently derived, the measure is mathematically identical with the concept of entropy as used in statistical mechanics. Legend has it that John von Neumann first spotted this similarity, and thus suggested to Shannon [87]:

You should call it entropy, for two reasons. In the first place your uncertainty function has been used in statistical mechanics under that name, so it already has a name. In the second place, and more important, no one knows what entropy really is, so in a debate you will always have the advantage.

Nowadays also referred to as *Shannon entropy*, the measure has a multitude of applications in research, among them, quantitative data analysis. In particular, various methods in EEG analysis depend on entropy-based parameters. The present section provides an overview of aspects in information theory that are relevant for this scope of application.

2.4.1 Shannon Entropy

Let X be a random variable, drawing from the finite support set $\mathcal{X} = \{x_1, x_2, \dots, x_m\}$ as governed by the probability mass function f_X . Then, its Shannon entropy [85] is defined as

$$H(X) = \sum_{i=1}^m f_X(x_i) \log \frac{1}{f_X(x_i)} = - \sum_{i=1}^m p(x_i) \log p(x_i). \quad (2.27)$$

The rationale is that each possible outcome $x_i \in \mathcal{X}$ of the random variable X carries an *information content* of $-\log p(x_i)$, so the more unlikely a particular outcome is, the more information it contains. Shannon entropy thus quantifies the average uncertainty about the value that X will take on once its underlying random experiment is carried out.

2 Mathematical Preliminaries

Theoretically, any impossible outcome of X contains an infinite amount of information, as is reflected by the limit

$$\lim_{p_i \rightarrow 0} -\log(p_i) = +\infty.$$

Still, impossible outcomes do never actually appear, so their average contribution to entropy is

$$\lim_{p_i \rightarrow 0} -p_i \log(p_i) = 0. \quad (2.28)$$

By convention, this limit is taken for any $p(x_i) = 0$ to enable steady continuation of singularities. Thus, the measure is well-defined for any probability mass function.

For instance, if the given experiment has only one possible outcome, such that $p(x_i) = 1$, while $p(x_j) = 0$ for all $x_j \in \mathcal{X} \setminus \{x_i\}$, there is no uncertainty, and Shannon entropy $H(X) = 0$. By contrast, the uncertainty about the outcome of X is maximal if all possible results are equally likely, that is, if $p(x_i) = |\mathcal{X}|^{-1}$ for all $x_i \in \mathcal{X}$. This yields the result $H(X) = \log |\mathcal{X}|$, which is the maximum entropy a discrete random variable with $|\mathcal{X}|$ states can have. In the range between those two extremes, any pairwise change that renders a given probability distribution “more uniform” will be accompanied by an increase in entropy [85]. This essential property will be exploited in Chapter 6.

Shannon entropy is not restricted to random variables, but can be used as a statistic on *any* probability mass function—like the marginal probability distribution of a discrete stationary process, for instance. Therefore, we will frequently use the notation

$$H(\mathbf{p}) = -\sum_{i=1}^m p_i \log(p_i), \quad \text{with} \quad \mathbf{p} = (p_1 \ p_2 \ \cdots \ p_m)^\top, \quad (2.29)$$

where \mathbf{p} is a probability vector in terms of Equation 2.23.

Notice that the base of the logarithm can be selected arbitrarily, as it merely determines the unit of measure. Usually, either the binary or the decadic logarithm is chosen, and their corresponding unit names are *shannon* and *hartley*, respectively. In particular, $\log_2 2 = 1$ Sh, while $\log_{10} 10 = 1$ Hart. Hartleys are also named *bans*, while shannons are nowadays ubiquitously referred to as *bits*. Throughout this work, we will be using the binary logarithm function exclusively.

2.4.2 Joint Entropy

Following the same principles as for the univariate case, the *joint entropy* of a pair of random variables X and Y , with images \mathcal{X} and \mathcal{Y} and joint probability mass function $f_{X,Y}$, is defined as

$$H(X, Y) = - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(x, y), \quad (2.30)$$

and for a set of N random variables $\{X_1, \dots, X_N\}$, this generalises into

$$H(X_1, \dots, X_N) = - \sum_{x_1 \in \mathcal{X}_1} \cdots \sum_{x_N \in \mathcal{X}_N} p(x_1, \dots, x_N) \log p(x_1, \dots, x_N). \quad (2.31)$$

Joint entropy thus provides the average uncertainty about the result of a random experiment involving more than one random variable.

It is easy to see that joint entropy is symmetrical, that is, $H(X, Y) = H(Y, X)$. Moreover, it holds that [85]

$$H(X, Y) \leq H(X) + H(Y), \quad (2.32)$$

whereby equality applies if and only if the random variables X and Y are independent. This is fully in line with intuition. Firstly, the joint uncertainty about a pair of outcomes cannot exceed the uncertainties about the individual outcomes. Moreover, if two outcomes are correlated, then knowing one of them will necessarily resolve some of the uncertainty about the other.

2.4.3 Conditional Entropy

In a similar manner, the remaining uncertainty about a random variable X , provided that the outcome of another variable Y is already known, can be modelled as the *conditional entropy*

$$H(X | Y) = - \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log p(x | y) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x, y) \log \frac{p(x, y)}{p(y)}. \quad (2.33)$$

Notice that the averaging has to be performed by weighting the per-symbol information content with $p(x, y)$, although the automatisms of human pattern-matching may at first glance suggest to multiply with $p(x | y)$ instead.

Conditional entropy has a set of interesting properties [85]. First and foremost, it holds that

$$H(X | Y) = H(X, Y) - H(Y). \quad (2.34)$$

which further implies that $H(X | Y) = H(X)$ if and only if the random variables X and Y are independent. Conversely, it holds that $H(X | Y) = 0$ in cases where knowing the outcome of Y fully determines X . Moreover, it follows that conditional entropy is asymmetric in general, because

$$H(X, Y) = H(X | Y) + H(Y) = H(Y | X) + H(X). \quad (2.35)$$

2.4.4 Transfer Entropy

Transfer entropy is a relatively recent addition to the framework of information theory. Introduced by Thomas Schreiber in 2000 [88], it is a measure of *directional* information exchange among a pair of random processes. We here limit our considerations to transfer entropy between discrete-time processes on finite state spaces. Given two such processes $\{X_t\}_{t \in \mathbb{Z}}$ and $\{Y_t\}_{t \in \mathbb{Z}}$ on the respective state spaces \mathcal{X} and \mathcal{Y} , transfer entropy from $\{X_t\}$ to $\{Y_t\}$ is defined as [88, 89]

$$T_{X \rightarrow Y} = \sum p(y_{t+1}, \mathbf{y}_t^{(r)}, \mathbf{x}_t^{(s)}) \log \frac{p(y_{t+1} | \mathbf{y}_t^{(r)}, \mathbf{x}_t^{(s)})}{p(y_{t+1} | \mathbf{y}_t^{(r)})}. \quad (2.36)$$

Therein, the (implicitly threefold) summation iterates all $y_{t+1} \in \mathcal{Y}$, all $\mathbf{y}_t^{(r)} \in \mathcal{Y}^r$, and all $\mathbf{x}_t^{(s)} \in \mathcal{X}^s$. The vectors $\mathbf{y}_t^{(r)}$ and $\mathbf{x}_t^{(s)}$ are realisations of the delay vectors of random variables $\mathbf{X}_t^{(s)} = (X_t, X_{t-1}, \dots, X_{t-s+1})$ and $\mathbf{Y}_t^{(r)} = (Y_t, Y_{t-1}, \dots, Y_{t-r+1})$.

In accordance with Wiener’s causality principle, transfer entropy builds upon the following reasoning: if the process $\{X_t\}$ does not transfer any information to the process $\{Y_t\}$, then the future of $\{Y_t\}$ is no more uncertain in the light of its own past than it would be if also the past of $\{X_t\}$ were known. Expressed in terms of entropies, this means that

$$H(Y_{t+1} | \mathbf{Y}_t^{(r)}) = H(Y_{t+1} | \mathbf{Y}_t^{(r)}, \mathbf{X}_t^{(s)}), \quad (2.37)$$

which Schreiber called a “generalized Markov property” [88]. From this perspective, the dimensions r and s model the amount of memory of the process $\{Y_t\}$.

Transfer entropy is designed to quantify the deviation from this Markovian property, that is,

$$\mathbb{T}_{X \rightarrow Y} = \mathbb{H}(Y_{t+1} | \mathbf{Y}_t^{(r)}) - \mathbb{H}(Y_{t+1} | \mathbf{Y}_t^{(r)}, \mathbf{X}_t^{(s)}), \quad (2.38)$$

which is arithmetically identical with the explicit formulation in terms of Equation (2.36), but arguably more helpful in understanding Schreiber's connectivity measure. Clearly, transfer entropy takes on its minimum value $\mathbb{T}_{X \rightarrow Y} = 0$ if

$$\mathbb{H}(Y_{t+1} | \mathbf{Y}_t^{(r)}, \mathbf{X}_t^{(s)}) = \mathbb{H}(Y_{t+1} | \mathbf{Y}_t^{(r)}),$$

that is, if $\mathbf{X}_t^{(s)}$ does not provide any additional information on the outcome of Y_{t+1} at all. In turn, transfer entropy takes on its maximum value $\mathbb{T}_{X \rightarrow Y} = \mathbb{H}(Y_{t+1} | \mathbf{Y}_t^{(r)})$ for

$$\mathbb{H}(Y_{t+1} | \mathbf{Y}_t^{(r)}, \mathbf{X}_t^{(s)}) = 0,$$

that is, if $\mathbf{X}_t^{(s)}$ completely disambiguates the outcome of Y_{t+1} . Therefore, transfer entropy can be normalised to the value range $[0, 1]$ by means of dividing by $\mathbb{H}(Y_{t+1} | \mathbf{Y}_t^{(r)})$.

Finally, transfer entropy is asymmetric under an exchange of its arguments, that is, it generally holds that $\mathbb{T}_{X \rightarrow Y} \neq \mathbb{T}_{Y \rightarrow X}$. Hence, there is a source process, a target process, and a notion of directionality.

3 Chaos and Determinism

Permutation entropy originated from *dynamical systems theory*, the mathematical and physical discipline that—very generally speaking—studies system behaviour over time. Dynamical systems theory is a broad subject, as well as a deep one, so even a half-decent overview would be well beyond the scope of the present work. That being said, the next chapter merely outlines those basic concepts of the theory that constitute the foundations of permutation entropy, and thus, of ordinal time series analysis in general.

For a more profound introduction to this field, the reader is recommended the classic journal article *Ergodic theory of chaos and strange attractors* by Jean-Pierre Eckmann and David Ruelle [90], as well as the textbook *Nonlinear Time Series Analysis* by Holger Kantz and Thomas Schreiber [91]. Readers with a neuroscientific background may also find the two-part article series *Is there chaos in the brain?* by Philippe Faure and Henri Korn [92, 93] insightful.

3.1 Dynamical Systems

Virtually any abstract or physical entity that has a time-dependent state can be interpreted as a dynamical system. This state is the collection of parameters that unambiguously describe the system at any fixed (though otherwise arbitrary) instant of time. In terms of mathematics, those state parameters provide the basis vectors that span the *phase space* of the system: for a dynamical system with m state variables, its state at time t can be modelled as a point in m -dimensional Euclidean space, that is, as a vector $\mathbf{x}(t) \in \mathbb{R}^m$.

Besides the dimensionality of its phase space, a dynamical system is characterised by the value progression of its state over time. This is governed by a so-called *evolution rule*. In particular, for a system with deterministic dynamics, its state $\mathbf{x}(t)$ at time t is a function of the preceding states $\{\mathbf{x}(\tau) \mid \tau < t\}$ exclusively. Figuratively speaking, the future of a deterministic dynamical system is unambiguous in the light of its own past. By contrast, systems that lack this property are called *random* dynamical systems [94].

In its most general form, a deterministic dynamical system can have an infinite amount of memory, that is, its state $\mathbf{x}(t)$ at time t may possibly be a function of *all* preceding states [95]. However, for systems commonly regarded as deterministic, a single state vector $\mathbf{x}(t_0)$ at a particular instant t_0 fully determines the system's behaviour at each and every other point in time. Under this condition, the evolution rule is effectively a function of the form

$$\begin{aligned} f: \mathbb{R}^{m+2} &\rightarrow \mathbb{R}^m \\ \mathbf{x}(t_0), t_0, t &\mapsto \mathbf{x}(t). \end{aligned} \tag{3.1}$$

Thus, if a state $\mathbf{x}(t_0)$ and its absolute observation time t_0 are known, the system's state $\mathbf{x}(t)$ at any other time t is predetermined. Another simplification over the general case is the following: while the state of a dynamical system is of course expected to change over time, the underlying evolution rule can often be assumed as time-invariant, which means that

$$\forall (t_0, t_1) \in \mathbb{R}^2: \quad \mathbf{x}(t_0) = \mathbf{x}(t_1) \implies \mathbf{x}(t_0 + t) = \mathbf{x}(t_1 + t). \tag{3.2}$$

The behaviour of the dynamical system is then fully determined by an initial state $\mathbf{x}(0) \in \mathbb{R}^m$ obtained at an *arbitrary* moment in time—which is commonly referred to as $t = 0$ for reasons of simplicity. In this important special case, the structure of the evolution rule simplifies to

$$\begin{aligned} f: \mathbb{R}^{m+1} &\rightarrow \mathbb{R}^m \\ \mathbf{x}(0), t &\mapsto \mathbf{x}(t), \end{aligned} \tag{3.3}$$

and the dynamical system is called time-invariant.

The series RLC resonator described by the circuit diagram in Figure 3.1 is a basic example of a deterministic time-invariant system. Its state is comprised of two variables: the voltage across the capacitor u_C , and the current through the inductor i_L . According to Kirchhoff's circuit laws, the evolution rule of this system is given by a pair of linear homogeneous differential equations with constant coefficients, namely

$$u'_C(t) = \frac{1}{C}i_L(t), \tag{3.4a}$$

$$i'_L(t) = -\frac{1}{L}u_C(t) - \frac{R}{L}i_L(t). \tag{3.4b}$$

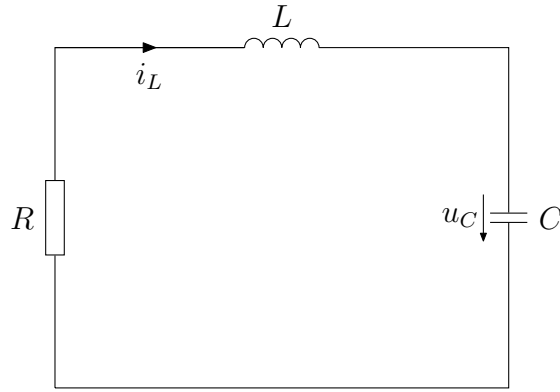


Figure 3.1 Series RLC circuit.

Consistently, the circuit is called a second-order linear time-invariant (LTI) system. Such systems are well-understood and possess closed-form solutions [96]. Specifically, the above evolution rule can be rewritten as a second-order differential equation,

$$u_C''(t) + 2\alpha u_C'(t) + \omega_0^2 u_C(t) = 0,$$

which describes a damped harmonic oscillator with attenuation $\alpha = R/2L$ and resonance frequency $\omega_0 = 1/\sqrt{LC}$. Let us limit this example to an underdamped configuration, where $\alpha < \omega_0$. In this case, the evolution rule has the closed-form solution

$$u_C(t) = u_0 e^{-\alpha t} \cos(\sqrt{\alpha^2 + \omega_0^2} t + \varphi_0), \quad (3.5a)$$

$$i_L(t) = C u_C'(t). \quad (3.5b)$$

The system hence oscillates at the angular frequency $\omega_d = \sqrt{\alpha^2 + \omega_0^2}$. Note that the nature of the dynamics is virtually invariant to the initial state vector $\mathbf{x}(0)$, which merely determines the constant coefficients u_0 and φ_0 in the above solution.

3.1.1 Fixed-Point Attractors

Given a dynamical system with up to three state variables, its *phase portrait* often proves insightful. It depicts a set of trajectories for different initial state vectors, and each such trajectory is a graph of the time-dependent relations between the system's state variables. Figure 3.2 displays an exemplary phase portrait for the underdamped series RLC circuit discussed in the previous section.

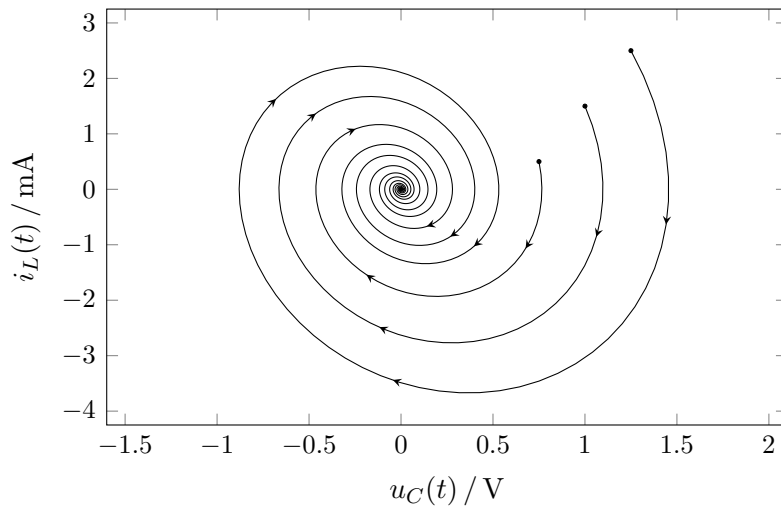


Figure 3.2 Phase portrait of an underdamped series RLC resonator, simulated for the circuit parameters $R = 100 \Omega$, $L = 100 \text{ mH}$, $C = 1 \mu\text{F}$, and three different initial states. Notice that the general nature of the dynamics is unaffected by the choice of the initial state.

This phase portrait is easily interpreted physically. In an RLC circuit, both the capacitor C and the inductor L store electrical energy, which figuratively swings back and forth between the two, giving rise to the oscillations shown in Figure 3.2. The energy is transported by an alternating electrical current flowing through all three devices R , L and C . With each oscillation, a fraction of the overall energy is dissipated as heat in the resistor R , and hence eliminated from the system. The damped harmonic oscillator therefore approaches the stable state

$$\lim_{t \rightarrow \infty} \mathbf{x}(t) = (0 \text{ V} \quad 0 \text{ A})^T,$$

where all electrical energy has been dissipated, and the dynamics come to a halt. This behaviour is invariant with respect to the initial state $\mathbf{x}(0)$, which merely scales and rotates the respective trajectory, but neither affects its shape, nor its point of convergence. In terms of dynamical systems theory, such a point of convergence is called a fixed-point attractor.

3.1.2 Periodic Attractors

For dynamical systems that follow non-linear evolution rules, attractors are not limited to fixed points, but may express various other geometrical shapes [90]. As an example,

consider the *Van der Pol Oscillator* [97], which is a system governed by the second-order non-linear differential equation

$$x''(t) + \mu(x^2(t) - 1)x'(t) + x(t) = 0, \quad (3.6a)$$

or equivalently, by the two-dimensional differential equation system

$$x'(t) = y(t), \quad (3.6b)$$

$$y'(t) = \mu(1 - x^2(t))y(t) - x(t). \quad (3.6c)$$

These equations describe a damped oscillator with a non-linear and state-dependent attenuation $\mu(x^2(t) - 1)$. Most decisively, the attenuation takes on negative values for $|x(t)| < 1$, thus actively repelling the trajectory from the origin of the phase space. While no closed-form solution exists for the Van der Pol Oscillator, a phase portrait as the one depicted in Figure 3.3 can be obtained by numerical integration of Equation System (3.6).

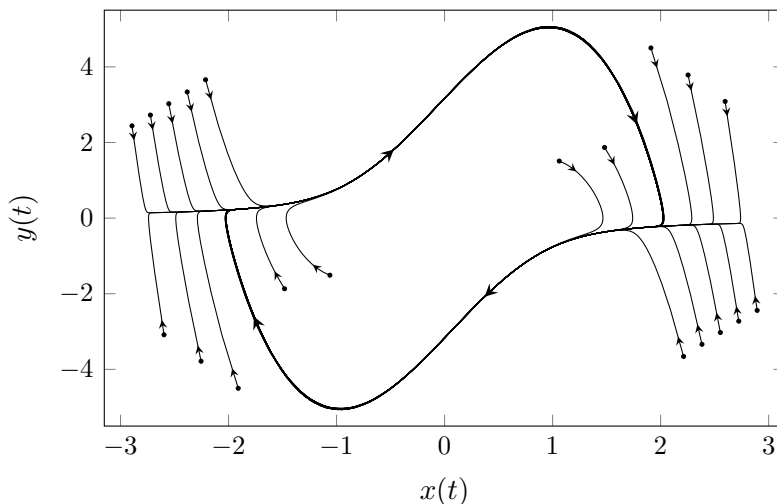


Figure 3.3 Phase portrait of the Van der Pol Oscillator for $\mu = 3$.

Its phase portrait reveals the characteristic attractor of the system: for any $\mu > 0$, and for any initial state other than the zero vector, the dynamics of the Van der Pol Oscillator converge towards a closed curve called a *limit cycle*. By contrast with fixed-point attractors, which are self-bijective under the evolution rule, limit cycles are periodic, that is, their dynamics do not stall in one discrete state, but follow the shape of the attractor in a repetitive manner.

3.1.3 Strange Attractors

Some dynamical systems, while still being perfectly deterministic, are extremely sensitive to deviations of their initial conditions, rendering long-term predictions practically impossible. The Lorenz system is a coursebook example of this phenomenon. It was named after meteorologist Edward Lorenz, who in his famous 1963 article [98] discussed a non-linear model of atmospheric convection given by the differential equation system

$$x'(t) = \sigma y(t) - \sigma x(t), \quad (3.7a)$$

$$y'(t) = \rho x(t) - x(t)z(t) - y(t), \quad (3.7b)$$

$$z'(t) = x(t)y(t) - \beta z(t). \quad (3.7c)$$

Using $\sigma = 10$, $\beta = 8/3$, and $\rho = 28$ as per the original publication, numerical integration yields the system's phase portrait as shown in Figure 3.4. It reveals an attractor so unusual in shape that David Ruelle and Floris Takens later coined the term *strange attractor* for any such non-trivial structure [99].

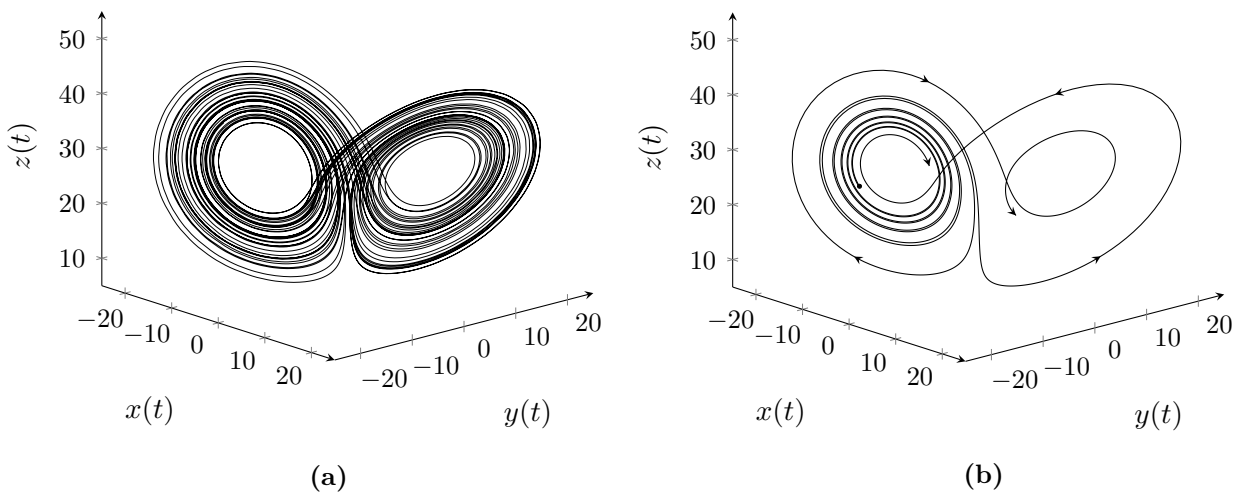


Figure 3.4 (a) The phase portrait of the Lorenz system ($\sigma = 10$, $\beta = 8/3$, $\rho = 28$) is dominated by a strange attractor. (b) This attractor expresses chaotic behaviour, that is, minor changes to the initial conditions result in fundamentally differing trajectories.

The Lorenz attractor depicted in Figure 3.4 is highly sensitive to state deviations. Even state vectors located infinitesimally close to one another will diverge exponentially, eventually resulting in vastly differing trajectories. Such unpredictable, yet fully deterministic dynamics are called chaotic, and the overall phenomenon is referred to as chaos [100].

3.2 Delay Embeddings

So far, we have outlined that a time-invariant deterministic dynamical system is fully defined by its initial state $\mathbf{x}(0) \in \mathbb{R}^m$ and by its evolution rule, which is a function of the form

$$\begin{aligned} f: \mathbb{R}^{m+1} &\rightarrow \mathbb{R}^m \\ \mathbf{x}(0), t &\mapsto \mathbf{x}(t). \end{aligned} \tag{3.8}$$

We implicitly assumed that the system's evolution rule was available as a set of differential equations, and also took for granted that all state variables be known and individually observable, such that the system can be solved for any instant in time—either analytically, or by numerical integration.

Hardly any of these assumptions hold true when studying real-world dynamical systems. Much rather, the number of state variables will often be unknown, and even if not, the overall state will in general not be fully observable. In turn, a predefined evolution rule will neither be available, of course. Notwithstanding, any dynamical system will likely express *some* kind of observable dynamic properties—be it spacial location, temperature, emission of sound, radiation of light, electrical voltage fluctuation, or some other parameter repeatedly observable as time goes by.

Using merely one such measurable signal $s(t)$, it is theoretically possible to reconstruct the m -dimensional attractor of an unknown dynamical system—provided that such an attractor exists. In all brevity, this can be done in the following way: assuming that the dynamics of an unknown system are governed by an m -dimensional attractor, one obtains a so-called *delay embedding* of this attractor by constructing a delay vector

$$\mathbf{s}(t) = \left(s(t) \quad s(t - \tau) \quad s(t - 2\tau) \quad \cdots \quad s(t - 2m\tau) \right)^\top \in \mathbb{R}^{2m+1} \tag{3.9}$$

of (at least) $2m + 1$ dimensions from the measured signal $s(t)$. For reasons yet to be discussed, it then holds that each point visited by $\mathbf{s}(t) \in \mathbb{R}^{2m+1}$ corresponds to exactly one point on the m -dimensional attractor, and vice versa. Due to this one-to-one relationship, any state change of the system is reflected by a change in $\mathbf{s}(t)$. For example, a sequence of recurring values in $\mathbf{s}(t)$ necessarily implies a recurring progression of system states. Thus, an appropriate delay embedding provides a hook into the internal structure of a dynamical system.

Apparently, the delay embedding technique was first described in a publication by Packard and colleagues [101]. However, the authors also reference their private communication with David Ruelle on this matter. In any case, the foundations for the approach were provided by the *Whitney Embedding Theorem* [102], which states that “a generic smooth map F from a d -dimensional smooth compact manifold M to \mathbb{R}^{2d+1} is actually a diffeomorphism on M ”—as has been summarised by Tim Sauer and colleagues [103]. In terms of mathematical topology, a diffeomorphism is a map $F : A \rightarrow B$, such that distinct points in A cannot coincide in B . In dynamical systems theory, such a map F is called an *embedding*.

The other important theorem for the delay embedding technique is known as the *Takens Embedding Theorem* [104]. In simple words, the theorem assures that instead of observing $2m + 1$ different signals in parallel, it is sufficient to use $2m + 1$ delayed versions of the exact same univariate signal to obtain an embedding of an m -dimensional attractor. This directly leads to the delay vectors described by Equation (3.9).

To give an example, the topology of the Lorenz attractor discussed in Section 3.1.3 can be reconstructed in three-dimensional Euclidean space by using delayed versions of the y -component of its system state. The result of doing so is visualised in Figure 3.5.

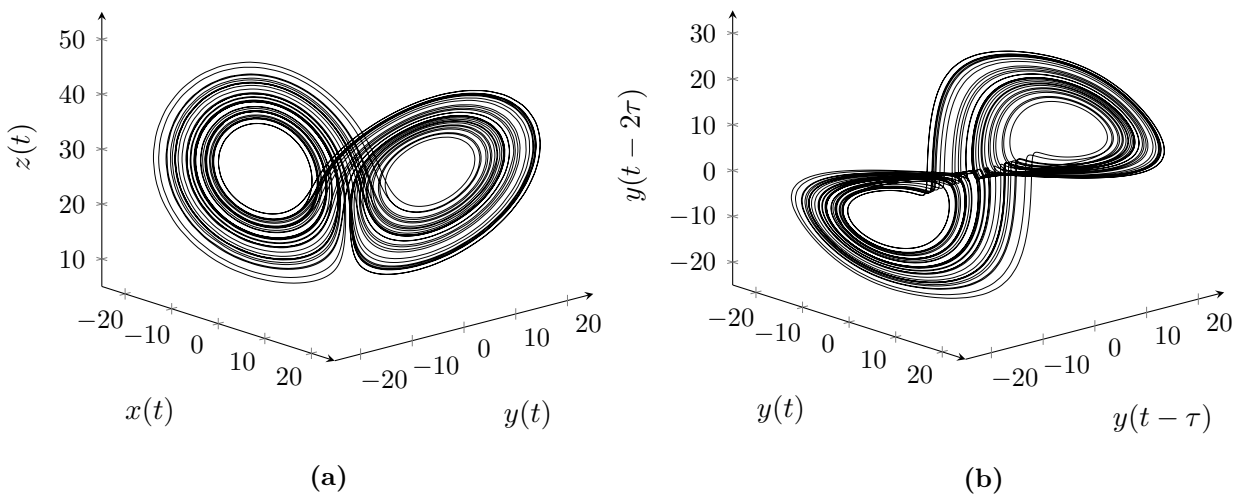


Figure 3.5 (a) The phase portrait of the Lorenz system in terms of Equation System (3.7), simulated for $\sigma = 10$, $\beta = 8/3$, $\rho = 28$. (b) Reconstruction of the phase portrait, using three delayed versions of the y -component, and the time lag $\tau = 0.1$.

3.3 Entropy as a Complexity Measure

As opposed to its meaning in everyday language, the term “chaos” in the sense of dynamical systems theory does not imply randomness. Quite the contrary, chaos strictly excludes any dependence on chance, and chaotic systems rigorously follow deterministic rules. For a long time, however, observations of what eventually turned out to be complex determinism had been misinterpreted as being of stochastic origin [92]. This is hardly surprising, of course: when studying a chaotic system, one and the same experiment, repeated under allegedly identical conditions, may yield fundamentally differing results for each repetition. To be able to distinguish deterministic chaos from randomness, a particular set of analysis tools has therefore been developed over time. By design, such measures are invariant to the chaotic bifurcations induced by varying initial conditions, but rather assess the *extent* to which a given system behaves in a chaotic manner. The most common among those so-called *complexity measures* are: Lyapunov exponents, entropies, and fractal dimensions [90, 92]. These three types of parameter are of great importance to dynamical systems theory, and have remarkable properties and interrelations [105]. For the purpose of this writing, we will focus on entropy-based complexity measures exclusively.

3.3.1 Entropy in Dynamical Systems

Entropy can be understood as a statistic quantifying the degree of uniformity of a discrete probability distribution \mathbf{p} . In particular, $H(\mathbf{p})$ takes on its maximum for an actual uniform distribution, whereas $H(\mathbf{p}) = 0$ if all but one of the probabilities in \mathbf{p} are zero. See Section 2.4 for a more thorough description.

In dynamical systems theory, entropy is used to quantify the irregularity of dynamical processes, including (but not limited to) deterministic dynamics. In short, the approach is as follows: given a dynamical system with state vector $\mathbf{x}(t) \in M$, its phase space M is partitioned into m disjoint subspaces, such that

$$M = P_1 \cup P_2 \cup \dots \cup P_m. \quad (3.10)$$

For this partition, an m -dimensional probability vector \mathbf{p} can then be defined, wherein for each $i \in \{1, 2, \dots, m\}$, its value $p_i = \Pr(\mathbf{x}(t) \in P_i)$ represents the probability that the system takes on a state $\mathbf{x}(t)$ residing in the subspace P_i . The entropy $H(\mathbf{p})$ then

quantifies the time-averaged degree of uniformity at which the dynamical system revisits the different partitions P_i of its phase space M .

Needless to say, the nature of the partition $P_1 \cup P_2 \cup \dots \cup P_m$ plays a crucial role. For $m \rightarrow \infty$, that is, when using an infinitely fine quantisation of the phase space, the above approach eventually leads to the *Kolmogorov-Sinai-Entropy*, which is of special interest for abstract dynamical systems theory [90]. For finite m , on the other hand, the set of subspaces $\{P_1, P_2, \dots, P_m\}$ itself can be understood as the discrete phase space of another dynamical system, which then gives rise to the notion of *symbolic dynamics*.

3.3.2 Ordinal Complexity

Permutation entropy, as well as ordinal time series analysis in general, is closely related to symbolic dynamics, that is, it is based on quantising the phase space of a dynamical system. Recall that under practical conditions, we usually have to rely on delay-embedded sequences of measurement values as a proxy into the unknown dynamics of the system under study (see Section 3.2). In this empirical setting, careful data preprocessing and parameter tweaking are inevitable if one expects to receive meaningful results [106].

Against the same backdrop, Bandt and Pompe argued that finding an adequate partition for a reconstructed phase space poses intricate problems, and may be bound to fail under real-world conditions [1]. In the same article, the authors therefore demanded that “the symbol sequence must come naturally from the [time series], without further model assumptions”, and proposed a novel partitioning technique that would meet this requirement. In essence, they suggested to map each delay vector of the embedded time series onto a discrete symbol representing the order relations among the elements of that vector. Those symbols are called *ordinal patterns*, and they will be discussed in a lot more detail in the following chapters.

For the time being, suffice it to say that a total of $m!$ different ordinal patterns exist for m -dimensional embedding vectors, because a tuple of m elements can be rearranged in no more than $m!$ different ways. The so-called *ordinal transformation* [107] thus partitions an m -dimensional phase-space reconstruction into $m!$ disjoint subspaces. It does not require any model assumptions apart from the dimension m and the time lag τ used for embedding the time series.

As outlined in Section 3.3.1, the resulting subspaces can be understood as the discrete states of a symbolic dynamics, which can then be further analysed. For this purpose, Bandt and Pompe suggested various entropy-based measures that can be obtained from a finite sequence of ordinal patterns. The simplest (and arguably, the most popular) among those propositions is to compute the entropy of a marginal probability distribution of ordinal patterns, estimated from the pattern sequence by counting occurrences. The authors coined the term *permutation entropy* for this complexity measure.

Following the initial publication of 2002, numerous extensions of permutation entropy have been devised, for instance, the methods proposed in [47, 108–110]. Other information-theoretic measures have also been applied to ordinal pattern distributions, among them: conditional entropy [111], mutual information [112], and transfer entropy [88, 113]. Moreover, recurrence plots [114] and various correlation functions [115] were transferred to the ordinal pattern space [107]. On a more abstract level, ordinal patterns have been tightly integrated into the general theory of symbolic dynamics. A thorough introduction to such matters is provided in the book *Permutation Complexity in Dynamical Systems* by José Amigó [116].

4 Ordinal Patterns

This chapter is based on previously published work by the same author [77, 78].

The previous chapter summarised those aspects and intricacies of dynamical systems theory that eventually led to the invention of permutation entropy, and its fundamental building blocks: the ordinal patterns. We shall proceed by describing precisely what those ordinal patterns are, how sequences of ordinal patterns relate to one another, and how they can be utilised for the study of real-word data—after all, this is what ordinal patterns have been invented for.

Being comparatively new, both the terminology and mathematical formulation used in ordinal time series analysis may vary between authors. The following section thus provides an overview of the key concepts underlying this framework, and introduces the definitions, notations and nomenclature used throughout the present work.

4.1 Definition and Notation

Let $(x_1, x_2, \dots, x_m) \in S^m$ be an m -tuple of pairwise distinct elements from a totally ordered set S , such that $x_i = x_j$ if and only if $i = j$. The *ordinal pattern* of this tuple is an abstract entity that describes how the tuple's elements relate to one another in terms of position and rank order. By way of illustration, the ordinal pattern of the tuple $(17, 7, 8) \in \mathbb{N}^3$ is fully specified by the verbal description:

“There are three elements, the first is the greatest, the second is the least.” (4.1)

This same ordinal pattern also applies to any other tuple $(x_1, x_2, x_3) \in S^3$ for which the order relations $x_2 < x_3 < x_1$ hold. By contrast, each of the six possible permutations of the elements $\{x_1, x_2, x_3\}$ yields a different ordinal pattern. Consequently, any given

m -tuple of pairwise distinct elements $(x_1, x_2, \dots, x_m) \in S^m$ has exactly one out of $m!$ different ordinal patterns. This interrelation can be described by a surjective map

$$\text{op}: S^m \rightarrow \Omega_m = \{\pi_1, \pi_2, \dots, \pi_{m!}\}, \quad (4.2)$$

wherein the tuple length $m \in \{2, 3, \dots\}$ is called the *embedding dimension* or *order* of the set of ordinal patterns Ω_m .

Literature knows various means of notating the order relations reflected by a particular ordinal pattern $\pi_i \in \Omega_m$. Compare, for instance, the conventions used in [1, 107, 115, 116]. As their common denominator, those representations all stem from the duality between ordinal patterns and permutations, and their majority are variations of merely two complementary forms of notation. Both will be discussed in the following.

4.1.1 The Permutation Representation

Given an m -tuple $(x_1, x_2, \dots, x_m) \in S^m$ of pairwise distinct elements from a totally ordered set S , its ordinal pattern $\text{op}(x_1, x_2, \dots, x_m) \in \Omega_m$ can be represented by a unique permutation function $\sigma: \mathbb{N} \rightarrow \mathbb{N}$ on the tuple indices, such that

$$\text{op}(x_{\sigma(1)}, x_{\sigma(2)}, \dots, x_{\sigma(m)}) = \pi_{\text{ref}} \in \Omega_m,$$

where π_{ref} is some predefined reference pattern. Although this reference pattern could in principle be selected arbitrarily, the ordinal pattern described by “*there are m elements, all sorted in ascending order*” constitutes a natural choice because it reflects the total order of the set S . Using this particular π_{ref} , the ordinal pattern $\text{op}(x_1, x_2, \dots, x_m)$ can be expressed by the permutation function

$$\sigma: \mathbb{N} \rightarrow \mathbb{N}, \quad \text{such that} \quad x_{\sigma(1)} < x_{\sigma(2)} < \dots < x_{\sigma(m)}. \quad (4.3)$$

For instance, the ordinal pattern of the tuple $(21, 55, 89, 34, 13) \in \mathbb{N}^5$ is representable by the permutation

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 1 & 4 & 2 & 3 \end{pmatrix} = (5, 1, 4, 2, 3).$$

For sake of simplicity, permutations describing ordinal patterns are sometimes written as plain numerals, yielding the compact notation 51423 for the above example. Although

this shorthand notation becomes ambiguous for orders $m > 10$, it is still useful because such high-dimensional patterns are rarely written out in practice.

Apart from the minor detail that Bandt and Pompe used zero-based indexing (and would have written 40312 instead of 51423), the permutation representation is the notation proposed in their original article [1]. Conceptually similar conventions can be found in publications by other authors, for instance in [107, 116]. Nevertheless, a different formal representation of ordinal patterns shall be used throughout the present work.

4.1.2 The Rank Representation

As described in the previous section, the ordinal pattern of an m -tuple of distinct elements $(x_1, x_2, \dots, x_m) \in S^m$ can be represented by means of the specific permutation function σ for which it holds that $x_{\sigma(1)} < x_{\sigma(2)} < \dots < x_{\sigma(m)}$. Due to the fact that permutation functions are self-bijective (and thus invertible), the same ordinal pattern can also be denoted by the inverse permutation function

$$\sigma^{-1}: \mathbb{N} \rightarrow \mathbb{N}, \quad \text{such that} \quad \sigma^{-1}(i) < \sigma^{-1}(j) \iff x_i < x_j. \quad (4.4)$$

Thus, the ordinal pattern of the aforementioned tuple $(21, 55, 89, 34, 13) \in \mathbb{N}^5$ can alternatively be referred to by either

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 1 & 4 & 2 & 3 \end{pmatrix}, \quad \text{or} \quad \sigma^{-1} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 5 & 3 & 1 \end{pmatrix}.$$

From the condition given in Equation (4.4), it follows immediately that

$$\rho_i = \sigma^{-1}(i) = 1 + \sum_{j=1}^m [x_i > x_j] \quad \forall i \in \{1, 2, \dots, m\} \quad (4.5)$$

denotes the *ordinal rank* of the element x_i within the tuple (x_1, x_2, \dots, x_m) . Thus, any given pattern $\text{op}(x_1, x_2, \dots, x_m)$ can be represented by an m -tuple of ranks $(\rho_1, \rho_2, \dots, \rho_m)$, whereby the greatest element in (x_1, x_2, \dots, x_m) is assigned the rank m , the second-greatest is given the rank $m - 1$, and so forth for all m elements. The resulting map $(x_1, x_2, \dots, x_m) \mapsto (\rho_1, \rho_2, \dots, \rho_m)$ is reminiscent of the rankings used in statistics such as Spearman's rank correlation coefficient [2].

4 Ordinal Patterns

In analogy with the permutation representation, a simplified notation suffices to denote ordinal patterns of order $m < 10$ by their ranks, yielding $(2, 4, 5, 3, 1) = \mathbf{24531}$ for example. To avoid ambiguity, numerals denoting ordinal patterns in rank representation will be set in boldface for the scope of this work. At this point, an example may prove insightful.

Example 4. For any tuple $(x_1, x_2, x_3, x_4) \in S^4$ of pairwise distinct elements from a totally ordered set S , its ordinal pattern of order $m = 4$ is one out of the $m! = 24$ patterns in $\Omega_4 = \{\mathbf{1234}, \mathbf{1243}, \mathbf{1324}, \mathbf{1342}, \mathbf{1423}, \mathbf{1432}, \dots, \mathbf{4321}\}$. Figure 4.1 displays the shapes of those ordinal patterns, and their representations in terms of permutations and ranks.

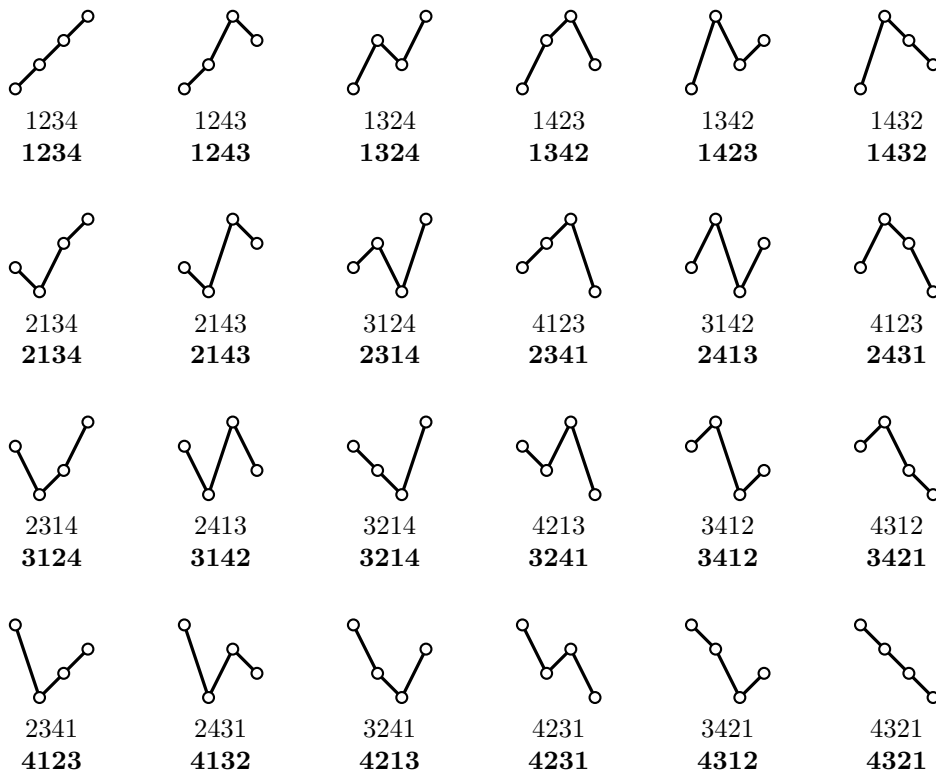


Figure 4.1 The principle shapes of the $m! = 24$ ordinal patterns of order $m = 4$, labelled by their permutation representation (regular type) and rank representation (boldface).

Bandt himself regards the rank representation as the “most intuitive way to denote these patterns” [115], and apparently prefers it over the permutation representation in his more recent publications, for example in [16, 115, 117]. With regard to being intuitive,

notice that any given tuple of ranks $(\rho_1, \rho_2, \dots, \rho_m)$ can be interpreted as a sequence of amplitudes, and that those amplitudes immediately reveal the structure of the underlying ordinal pattern (see Figure 4.1).

Apart from this subjective increase in clarity as compared to the permutation representation, the rank representation also enables a very convenient definition of ordinal patterns. This definition will be derived and discussed in the following.

4.1.3 A Practical Definition of Ordinal Patterns

So far, we required that all m elements of $(x_1, x_2, \dots, x_m) \in S^m$ be pairwise distinct, such that no ties (that is, $x_i = x_j$ for $i \neq j$) can occur. Under this prerequisite, both the permutation representation in terms of Equation (4.3), as well as the rank representation given by Equation (4.4) are unambiguous, and the ordinal pattern $\text{op}(x_1, x_2, \dots, x_m)$ is hence well-defined.

For practical applications, it is desirable to overcome the limitation on pairwise distinct elements, such that arbitrary input data can be analysed. To that end, a common approach is to stipulate $\rho_i < \rho_j$ for any pair of values $x_i = x_j$ if their order of appearance is $i < j$, and vice versa. Using this convention, any arbitrary m -tuple $(x_1, x_2, \dots, x_m) \in S^m$ from a totally ordered set S has a well-defined ordinal pattern. As a somewhat extreme example, it then holds that $\text{op}(42, 42, 42, 42, 42) = \mathbf{12345}$, for instance.

Depending on the amplitude distribution of the input data, it may in some cases be required to use a more sophisticated technique of resolving tied values [118]. However, tied ranks are very rare in electrophysiological recordings (and in many other kinds of data sampled from value-continuous sources at high amplitude resolution). Therefore, the aforementioned approach is sufficient for the scope of the present work. Adopting this simple convention, we arrive at the following definition of the ordinal pattern.

Definition 1. For any given m -tuple $(x_1, x_2, \dots, x_m) \in S^m$ from a totally ordered set S , its ordinal pattern $\pi_i \in \Omega_m = \{\pi_1, \pi_2, \dots, \pi_m\}$ of order m is the unique m -tuple of ranks $(\rho_1, \rho_2, \dots, \rho_m) \in \{1, 2, \dots, m\}^m$, such that

$$\forall i, j \in \{1, 2, \dots, m\}: \rho_i < \rho_j \iff (x_i < x_j \vee (x_i = x_j \wedge i < j)). \quad (4.6)$$

Notice that by the above definition, any given $\pi_i \in \Omega_m$ does not only *represent* an ordinal pattern, but rather *is* the ordinal pattern itself. To motivate this extension, consider the following. Let $\pi_i = (\rho_1, \rho_2, \dots, \rho_m) = \text{op}(x_1, x_2, \dots, x_m)$ be the ordinal pattern of some m -tuple $(x_1, x_2, \dots, x_m) \in S^m$. Then, under Definition 1, the rather curious expression

$$\text{op}(\pi_i) = \text{op}(\rho_1, \rho_2, \dots, \rho_m)$$

is actually well-defined: it is the ordinal pattern of the ordinal pattern π_i , which is nothing but the ordinal pattern of an m -tuple of pairwise distinct integers. It is easily confirmed that $\text{op}(\pi_i) = \pi_i$ for all $\pi_i \in \Omega_m$, which implies that the function $\text{op} = \text{op} \circ \text{op}$ is an idempotence—the ordinal pattern of another ordinal pattern is that other ordinal pattern. Thus, we can understand the ordinal transformation $\text{op}: S^m \rightarrow \mathbb{N}^m$ as a form of non-linear vector quantisation.

4.2 Sequences of Ordinal Patterns

The fundamental idea proposed by Bandt and Pompe [1] is to transform a time series $\{x_t\}$ with time indices $t \in \{1, 2, \dots\}$ into a sequence of discrete ordinal patterns $\{\pi_t\}$ prior to any further processing. As described in Section 3.3.2, this approach builds upon the delay embeddings used in dynamical systems theory: after having selected a pattern order $m \in \{2, 3, \dots\}$, and a time lag $\tau \in \{1, 2, \dots\}$, one obtains the sequence

$$\{\pi_t\} \quad \text{with} \quad \pi_t = \text{op}(x_t, x_{t+\tau}, \dots, x_{t+(m-1)\tau}), \quad (4.7)$$

that is, the ordinal patterns of the (time-reversed) delay vectors of the time series $\{x_t\}$. This transformation combines phase-space reconstruction by means of delay embedding with a non-linear form of vector quantisation. The latter partitions an m -dimensional space into $m!$ disjoint subspaces. Some authors call the pattern order m the *embedding dimension* of the ordinal pattern, thereby explicitly referencing the system-theoretic origins of the ordinal approach.

4.2.1 Partitioning the Euclidean Space

Let us consider in more detail how the ordinal transform partitions the Euclidean space \mathbb{R}^m . The most simple case is for the order $m = 2$, where there are merely $2! = 2$

different ordinal patterns. According to Definition 1, it then holds that

$$\text{op}(x_t, x_{t+\tau}) = \begin{cases} \mathbf{12}, & \text{for } x_t \leq x_{t+\tau}, \\ \mathbf{21}, & \text{for } x_t > x_{t+\tau}. \end{cases}$$

Thus, for patterns of order $m = 2$, the diagonal line $x_t = x_{t+\tau}$ partitions the Euclidean plane \mathbb{R}^2 into two disjoint subspaces.

For the order $m = 3$, a total of $3! = 6$ ordinal patterns exist, and those relate to disjoint subspaces of the \mathbb{R}^3 . More specifically, it holds that

$$\text{op}(x_t, x_{t+\tau}, x_{t+2\tau}) = \begin{cases} \mathbf{123}, & \text{for } x_t \leq x_{t+\tau} \text{ and } x_t \leq x_{t+2\tau} \text{ and } x_{t+\tau} \leq x_{t+2\tau}, \\ \mathbf{132}, & \text{for } x_t \leq x_{t+\tau} \text{ and } x_t \leq x_{t+2\tau} \text{ and } x_{t+\tau} > x_{t+2\tau}, \\ \mathbf{213}, & \text{for } x_t > x_{t+\tau} \text{ and } x_t \leq x_{t+2\tau} \text{ and } x_{t+\tau} \leq x_{t+2\tau}, \\ \mathbf{231}, & \text{for } x_t \leq x_{t+\tau} \text{ and } x_t > x_{t+2\tau} \text{ and } x_{t+\tau} > x_{t+2\tau}, \\ \mathbf{312}, & \text{for } x_t > x_{t+\tau} \text{ and } x_t > x_{t+2\tau} \text{ and } x_{t+\tau} \leq x_{t+2\tau}, \\ \mathbf{321}, & \text{for } x_t > x_{t+\tau} \text{ and } x_t > x_{t+2\tau} \text{ and } x_{t+\tau} > x_{t+2\tau}. \end{cases}$$

The three-dimensional space is hence partitioned by three planes, which are given by the equations $x_t = x_{t+\tau}$, $x_t = x_{t+2\tau}$, and $x_{t+\tau} = x_{t+2\tau}$. Those planes all intersect in a common line, which is the space diagonal $x_t = x_{t+\tau} = x_{t+2\tau}$. Consequently, the \mathbb{R}^3 is partitioned into six disjoint wedges. Each has an opening angle of $\pi/3$, and all share the aforementioned space diagonal as their edge. The resulting partition is visualised in Figure 4.2.

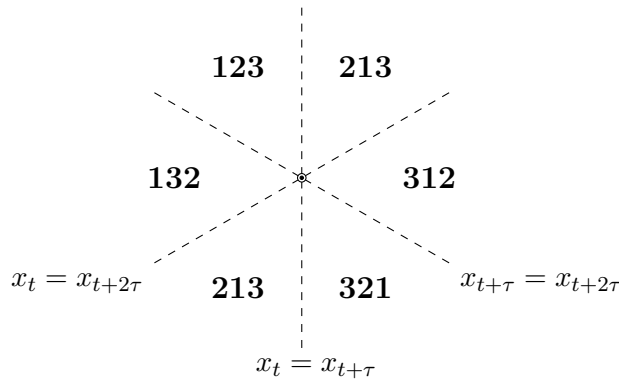


Figure 4.2 For patterns of order $m = 3$, three planes partition the \mathbb{R}^3 into six disjoint subspaces. Those planes intersect in the common line $x_t = x_{t+\tau} = x_{t+2\tau}$. The projection angle has been chosen such that this diagonal is perpendicular to the drawing canvas.

The general case is in analogy with the above: for patterns $\text{op}(x_t, x_{t+\tau}, \dots, x_{t+(m-1)\tau})$ of order m , there are $\binom{m}{2}$ distinct equalities of the form $x_{t+i\tau} = x_{t+j\tau}$, with $i \neq j$. Each of these equalities parametrises a hyperplane in $m - 1$ dimensions, and those hyperplanes partitions the \mathbb{R}^m into $m!$ disjoint subspaces, each relating to a particular ordinal pattern.

4.2.2 Beyond Delay Embeddings and Phase Spaces

For the scope of the present writing, we take the liberty to slightly deviate from the system-theoretic origins of ordinal time series analysis. Notice that according to Definition 1, not only delay vectors, but any given m -tuple $(x_1, x_2, \dots, x_m) \in S^m$ from a totally ordered set S has a distinct ordinal pattern. Of course, those m -tuples can be delay vectors in the \mathbb{R}^m , but they do not necessarily have to be. In principle, an ordinal pattern can be attributed to any tuple of sortable elements. This somewhat broader perspective will prove advantageous in the following, and especially in Chapter 6.

Against the same backdrop, notice that *positive* multiples of τ are used as time shifts in Equation (4.7), such that x_t is the *earliest* element with an influence on the ordinal pattern π_t . Although in obvious contrast with the more commonplace delay vectors of Equation (3.9), this convention makes a lot of practical sense: the original time series $\{x_t\}$, its corresponding sequence of ordinal patterns $\{\pi_t\}$, as well as the m elements $x_t, x_{t+\tau}, \dots, x_{t+(m-1)\tau}$ underlying a particular ordinal pattern π_t all start at the same time index t . Moreover, all time indices consistently increase from left to right. These interrelations are graphically summarised in Figure 4.3.

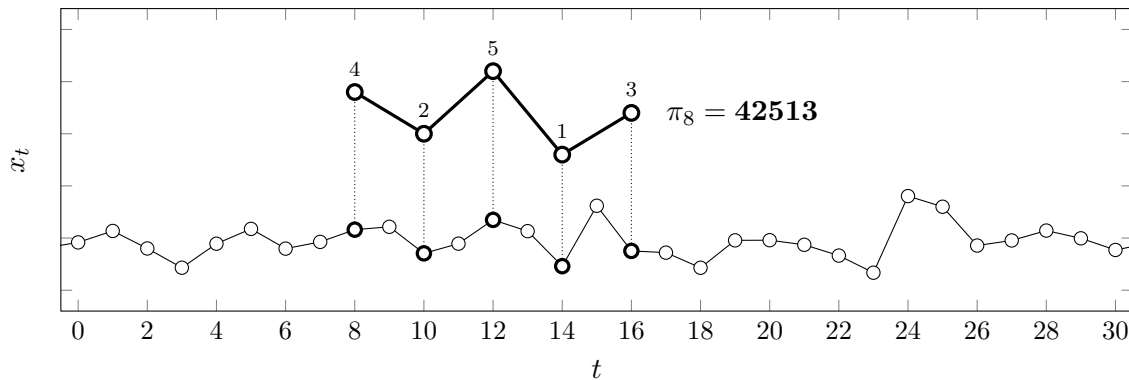


Figure 4.3 Extracting an ordinal pattern of order $m = 5$ from some random time series, using the time lag $\tau = 2$. The time index of the element x_8 (the earliest element with an influence on the ordinal pattern) determines the time index of the ordinal pattern, that is, π_8 .

A peculiarity to keep in mind is that under these conventions, the pattern π_t is not causal with regard to its time index t , but depends on the future of the time series. When analysing data, this has to be compensated for by introducing a delay somewhere in the overall processing chain. Nevertheless, this author believes that doing so is less error-prone than maintaining consistency among multiple offsets and directions of indexing.

4.2.3 Ordinal Processes

As described in Section 2.3.5, any time series $\{x_t\}$ can be understood as a particular realisation of some stochastic process $\{X_t\}$. In doing so, each value x_t is interpreted as the result of a random experiment, and this experiment is in turn modelled by a random variable X_t . Now, if any time series $\{x_t\}$ can be seen as a realisation of some stochastic process $\{X_t\}$, it makes sense to postulate that the sequence of ordinal patterns $\{\pi_t\}$, where

$$\pi_t = \text{op}(x_t, x_{t+\tau}, \dots, x_{t+(m-1)\tau}),$$

originates from an underlying stochastic process as well. In particular, this process $\{\Pi_t\}$ is time-discrete, and its state space is the set of ordinal patterns Ω_m . Keller, Sinn and Emonds coined the term *ordinal process* for this concept [107]. For the scope of this writing, the collection of random variables $\{\Pi_t\}$ will be called an ordinal process of order m with time lag τ , thereby explicitly referencing the parameters that govern the map $X_t \mapsto \Pi_t$.

The most decisive property of an ordinal process of order $m > 2$ is that its random variables can never be independent. Observe that for any time t , the m -dimensional delay vectors

$$(x_t, x_{t+\tau}, \dots, x_{t+(m-1)\tau}) \quad \text{and} \quad (x_{t+\tau}, x_{t+2\tau}, \dots, x_{t+m\tau})$$

overlap in $m - 1$ out of m values. Consequently, with $\Pi_t = \pi_t$ already fixed, an ordinal process cannot draw $\pi_{t+\tau}$ from its full state space Ω_m , but merely from a subset of cardinality $m!/(m-1)! = m$. Keller and colleagues regard this property as the very definition [107] of ordinal processes, by contrast with any other process drawing from the state space Ω_m .

As already mentioned, ordinal processes of order $m = 2$ constitute an exception in this regard, because $(x_t, x_{t+\tau})$ and $(x_{t+\tau}, x_{t+2\tau})$ overlap in merely one value, such that their

ordinal patterns have disjoint order relations. Thus, each of the patterns $\Omega_2 = \{\mathbf{12}, \mathbf{21}\}$ can be succeeded by either $\mathbf{12}$ or $\mathbf{21}$, and consistently, it holds that $2! = 2$.

4.2.4 Ordinal Markov Chains

For the time lag $\tau = 1$, any ordinal process $\{\Pi_t\}$ is a Markov chain (see Section 2.3.7). Because of the inter-pattern dependencies described in the previous section, its corresponding transition matrix \mathbf{T} is sparse, containing no more than m positive entries per row. Assuming $m = 3$, for instance, the matrix cannot be less sparse than

$$\mathbf{T} = \begin{array}{c} \mathbf{123} \quad \mathbf{132} \quad \mathbf{213} \quad \mathbf{231} \quad \mathbf{312} \quad \mathbf{321} \\ \left(\begin{array}{cccccc} \mathbf{123} & p_{1,1} & p_{1,2} & 0 & p_{1,4} & 0 & 0 \\ \mathbf{132} & 0 & 0 & p_{2,3} & 0 & p_{2,5} & p_{2,6} \\ \mathbf{213} & p_{3,1} & p_{3,2} & 0 & p_{3,4} & 0 & 0 \\ \mathbf{231} & 0 & 0 & p_{4,3} & 0 & p_{4,5} & p_{4,6} \\ \mathbf{312} & p_{5,1} & p_{5,2} & 0 & p_{5,4} & 0 & 0 \\ \mathbf{321} & 0 & 0 & p_{6,3} & 0 & p_{6,5} & p_{6,6} \end{array} \right) \end{array}.$$

In the general case of $\tau > 1$, an ordinal process behaves like τ such Markov chains interleaved. By way of further illustration, a state diagram for the order $m = 3$ (and necessarily, the time lag $\tau = 1$) is depicted in Figure 4.4.

4.3 Permutation Entropy

The current chapter so far described what ordinal patterns are, how they can be notated, and how they are related with dynamical systems, time series and stochastic processes. With these underpinnings in place, let us now shift the focus onto the actual purpose of transforming time series into ordinal patterns.

Recall that the seminal publication for the subject is called *Permutation Entropy: A Natural Complexity Measure for Time Series* [1]. In a nutshell, Bandt and Pompe therein proposed to map a given time series onto a sequence of discrete symbols, to then estimate the probability of each of those symbols by counting occurrences, and finally, to calculate the entropy of the resulting probability mass function. This value is called permutation entropy, and serves as a proxy for the complexity of the time series.

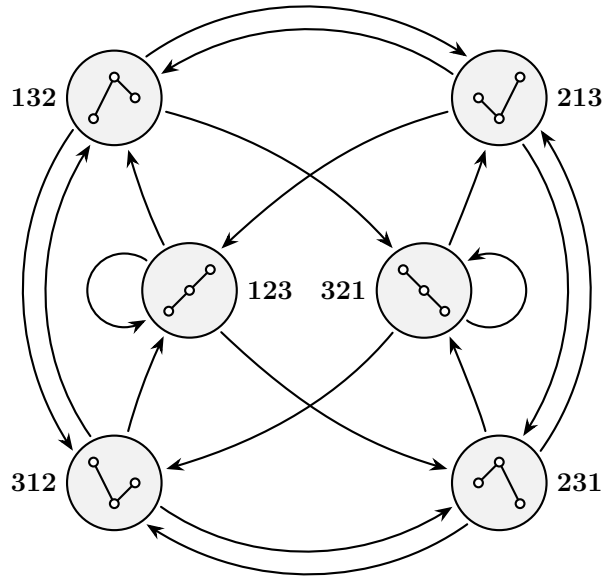


Figure 4.4 State diagram for an ordinal process of order $m = 3$ and time lag $\tau = 1$, which can be interpreted as a first-order Markov chain. Its transition probabilities depend on the underlying process, and some can be zero. However, no other transitions than the ones depicted are possible, because consecutive patterns overlap in two out of three values.

The approach just summarised is exactly the well-known method of estimating entropy by coarse-graining the reconstructed phase-space of a dynamical system (see Section 3.3.1). What sets the measure apart from other entropy-based analysis techniques is only the specific partitioning used. In particular, Bandt and Pompe proposed to turn a finite time series $\{x_1, x_2, \dots, x_N\}$ into a sequence of symbols $\{\pi_1, \pi_2, \dots, \pi_{N-m+1}\}$, where each $\pi_t = \text{op}(x_t, x_{t+1}, \dots, x_{t+m-1})$ is (now called) an ordinal pattern of order m in the sense of Definition 1. As has been discussed in Section 4.2.1, this transformation embeds the time series in an m -dimensional Euclidean space, and subsequently partitions that space into $m!$ disjoint subspaces.

While originally defined for the time lag $\tau = 1$ only, permutation entropy was soon extended to allow for arbitrary non-negative integer lags $\tau \geq 1$. See [3], for instance, which appeared about two years after the initial publication. Adopting this extension as seems to be common practice, permutation entropy of order m and time lag τ can be obtained from a given finite time series $\{x_1, x_2, \dots, x_N\}$ of N elements by computing the entropy (see Section 2.4.1)

$$H(\tilde{\mathbf{p}}) = - \sum_{i=1}^{m!} \tilde{p}_i \log \tilde{p}_i \quad (4.8)$$

of the probability vector $\tilde{\mathbf{p}} = (\tilde{p}_1 \ \tilde{p}_2 \ \cdots \ \tilde{p}_{m!})^\top$. The latter is estimated from the time series by counting ordinal pattern occurrences, that is, by obtaining

$$\tilde{p}_i = \frac{1}{N - (m-1)\tau} \sum_{t=1}^{N-(m-1)\tau} [\text{op}(x_t, x_{t+\tau}, \dots, x_{t+(m-1)\tau}) = \pi_i] \quad (4.9)$$

for all $i \in \{1, 2, \dots, m!\}$, and thus, for all ordinal patterns $\pi_i \in \Omega_m = \{\pi_1, \pi_2, \dots, \pi_{m!}\}$.

4.3.1 Empirical Permutation Entropy

While permutation entropy was initially proposed as a practical technique for time series analysis, it has subsequently been integrated more thoroughly into the theory of symbolic dynamics. Under this change of perspective, any sequence of ordinal patterns can be understood as some realisation of an abstract ordinal process (see Section 4.2.3). In turn, permutation entropy then has to be attributed to the ordinal process as a whole, instead of merely a single time series arising from it [107, 116].

As is the case for any other discrete random process, the statistical behaviour of an ordinal process $\{\Pi_t\}$ is generally governed by a joint probability mass function of the form

$$\begin{aligned} f_{\Pi_1, \Pi_2, \dots, \Pi_N} : (\Omega_m)^N &\rightarrow [0, 1], \\ (\pi_1, \pi_2, \dots, \pi_N) &\mapsto \Pr(\Pi_1 = \pi_1, \Pi_2 = \pi_2, \dots, \Pi_N = \pi_N) \end{aligned} \quad \text{for } N \rightarrow \infty.$$

Moreover, if the ordinal process is stationary, then a marginal probability mass function

$$\begin{aligned} f_{\Pi_t} : \Omega_m &\rightarrow [0, 1], \\ \pi_t &\mapsto \Pr(\Pi_t = \pi_t) \end{aligned}$$

does also exist (see Section 2.3.5), and in turn, the probability vector

$$\mathbf{p} = (p_1 \ p_2 \ \cdots \ p_{m!})^\top, \quad \text{where } p_i = f_{\Pi_t}(\pi_i) = \Pr(\Pi_t = \pi_i),$$

is well-defined. The permutation entropy of the ordinal process is then the entropy $H(\mathbf{p})$ of this marginal probability distribution. By contrast, when computing the permutation entropy of a time series $\{x_1, x_2, \dots, x_N\}$, one effectively obtains an estimate $\tilde{\mathbf{p}}$ of the marginal probability distribution \mathbf{p} of the underlying ordinal process, and the resulting

entropy $H(\tilde{\mathbf{p}})$ is therefore more adequately called *empirical permutation entropy* [107] within this framework.

Throughout the present writing, we will mostly be concerned with empirical permutation entropy, that is, with the entropy of a sequence of ordinal patterns as derived from time series data. In the following, the empirical nature of the approach will therefore only be mentioned if ambiguity may otherwise arise.

4.3.2 Properties of Permutation Entropy

Permutation entropy of order m and any time lag $\tau \geq 1$ is the entropy $H(\mathbf{p})$ of a discrete probability distribution

$$\mathbf{p} = (p_1 \ p_2 \ \cdots \ p_{m!})^\top$$

of $m!$ ordinal patterns. In accordance with the general properties of entropy described in Section 2.4, it thus holds for the value range of permutation entropy that

$$\min(H(\mathbf{p})) = 0, \quad (4.10a)$$

$$\max(H(\mathbf{p})) = \log m!. \quad (4.10b)$$

In particular, permutation entropy is zero if and only if the time series expresses merely one ordinal pattern repeatedly, as is the case for any monotonously increasing or decreasing sequence of values. Conversely, it takes on its maximum if the m ordinal patterns are uniformly distributed, such that $p_i = 1/m!$ for all $i \in \{1, 2, \dots, m!\}$. This holds true for sequences of white noise—where all values of the time series are uncorrelated with one another, and will hence express any possible sequence of order relations at equal probability [116]. Consequently, normalisation in terms of

$$h(\mathbf{p}) = \frac{1}{\log m!} H(\mathbf{p}), \quad \text{such that} \quad 0 \leq h(\mathbf{p}) \leq 1 \quad (4.11)$$

can be applied, and makes sense when qualitatively comparing entropies of different orders. Moreover, permutation entropy is invariant with regard to the absolute scaling, as well as the amplitude offset of the time series $\{x_t\}$, because it holds for the underlying ordinal patterns that

$$\text{op}(x_1, x_2, \dots, x_m) = \text{op}(cx_1 + d, cx_2 + d, \dots, cx_m + d) \quad \text{for any} \quad (c, d) \in \mathbb{R}^2.$$

4.4 Symbolic Transfer Entropy

As already mentioned in Section 3.3.2, not only Shannon entropy [85], but also other information-theoretic measures have been applied to distributions of ordinal patterns. For the scope of this writing, we limit the discussion to one that seems to be of particular relevance for the study of electroencephalographic recordings: *symbolic transfer entropy*, proposed by Matthäus Staniek and Klaus Lehnertz [113] in 2008, is the transfer entropy between two ordinal processes. It hence relates to transfer entropy in the same way that permutation entropy relates to Shannon entropy.

4.4.1 Neuroscientific Purpose

Because transfer entropy is designed to quantify directional information flow between pairs of random processes (see Section 2.4.4), it constitutes a powerful tool for detecting causal interactions within/between physiological neural networks [119]. Applying transfer entropy to measured time series is computationally challenging, though [89, 120, 121]. Quantisation by means of the ordinal transformation reduces the computational load considerably (see the beginning of Chapter 5), and may possibly improve the robustness against certain kinds of signal distortion (although one could argue that the transformation itself causes strong non-linear distortion).

Symbolic transfer entropy has been used for neuroscientific purposes from day one, that is, starting with the original publication, wherein Staniek and Lehnertz applied their measure to electroencephalographic recordings during epileptic seizures [113]. The recurring theme in the many investigations that followed is the dynamic formation and elimination of *functional connectivity* between brain regions [20, 53, 65, 122–140]. Among those publications, this doctoral candidate co-authored, and/or contributed to the studies described in [20, 53, 65, 127].

4.4.2 Mathematical Formalism

Symbolic transfer entropy was originally [113] formulated for two ordinal processes $\{\hat{X}_t\}_{t \in \mathbb{Z}}$ and $\{\hat{Y}_t\}_{t \in \mathbb{Z}}$, a common time lag τ , and a shared state space Ω_m , according to

$$T_{\hat{X} \rightarrow \hat{Y}} = \sum p(\hat{y}_{t+\delta}, \hat{y}_t, \hat{x}_t) \log \frac{p(\hat{y}_{t+\delta} | \hat{y}_t, \hat{x}_t)}{p(\hat{y}_{t+\delta} | \hat{y}_t)}. \quad (4.12)$$

In comparison to transfer entropy as given by Equation (2.36), Staniek and Lehnertz limited their measure to the causality between the state $\hat{Y}_{t+\delta}$ of the target process, and merely one pair of previous states, \hat{X}_t and \hat{Y}_t , of the source and target processes—each observed not one, but δ time steps earlier. In a subsequent publication [126], Henning Dickten and Klaus Lehnertz generalised the original formulation of symbolic transfer entropy. Now more explicitly accounting for possible delays in the interactions, they proposed the modification

$$T_{\hat{X} \rightarrow \hat{Y}} = \sum p(\hat{y}_t, \hat{y}_{t-\delta_y}, \hat{x}_{t-\delta_x}) \log \frac{p(\hat{y}_t | \hat{y}_{t-\delta_y}, \hat{x}_{t-\delta_x})}{p(\hat{y}_t | \hat{y}_{t-\delta_y})}. \quad (4.13)$$

Michael Wibral and colleagues critically reflected on causality in measures of delayed information transfer [141]. Their considerations render the separate delay parameters δ_x and δ_y particularly plausible, if not essential.

As already hinted at, a possibility of further generalisation that immediately follows from the original definition of transfer entropy [88] is

$$T_{\hat{X} \rightarrow \hat{Y}} = \sum p(\hat{y}_t, \hat{\mathbf{y}}_{t-\delta_y}^{(r)}, \hat{\mathbf{x}}_{t-\delta_x}^{(s)}) \log \frac{p(\hat{y}_t | \hat{\mathbf{y}}_{t-\delta_y}^{(r)}, \hat{\mathbf{x}}_{t-\delta_x}^{(s)})}{p(\hat{y}_t | \hat{\mathbf{y}}_{t-\delta_y}^{(r)})}, \quad (4.14)$$

that is, a version considering vectors of preceding states $\hat{\mathbf{Y}}_{t-\delta_y}^{(r)}$ and $\hat{\mathbf{X}}_{t-\delta_x}^{(s)}$ instead of single instances thereof. Staniek mentioned this possibility in his dissertation [142], but excluded it from the scope of the thesis. To the best of this author's knowledge, the extension has so far not been investigated further, so its significance for data analysis is unclear. In any case, its efficient estimation from time series constitutes an interesting computational problem. The results to be presented in the following Chapter 5 may be a first step into that direction.

5 Encoding Ordinal Patterns

This chapter is based on previously published work by the same author [78].

Besides its conceptual simplicity and its robustness against certain forms of measurement noise, computational efficiency is likely one of the most-cited advantages of ordinal time series analysis [1, 3, 47, 108–111, 113, 114, 118, 126, 143–147]. However, this well-acclaimed run-time behaviour is not a specific property of ordinal time series analysis, but constitutes a feature of discrete dynamics in general. By matter of principle, coarse-graining the phase space of a dynamical system can radically reduce the computational cost of its analysis, because quantisation turns continuous probability densities into discrete probability masses. A very concise example (intentionally unrelated to ordinal patterns) can be found in [89], wherein Andreas Kaiser and Thomas Schreiber address the intricacies of estimating transfer entropy from continuously-valued time series, as compared to the far simpler discrete case.

Before the computational benefits of symbolisation can take any effect in ordinal analysis, the (usually real-valued) input data need to be converted into sequences of discrete ordinal patterns. Somewhat paradoxically, extracting ordinal patterns from time series is computationally a lot heavier than literature commonly suggests. Determining a single ordinal pattern of order m requires a total of $(m^2 - m)/2$ pairwise comparisons, resulting in a run-time complexity of $O(m^2)$. In other words, “the computation time increases rapidly with m ”—as has been pointed out by Matthäus Staniek and Klaus Lehnertz [148], the creators of symbolic transfer entropy [113]. In a similar context, Amigó stated that “there is no substitute for substantial computational effort when [the order m] becomes sufficiently large”, and further conjectured that working with ordinal patterns beyond approximately $m = 12$ may likely be “computationally unfeasible” [116].

Apart from its run-time complexity, another closely related issue is the spatial complexity of ordinal analysis. Given that a total of $m!$ different ordinal patterns of order m exist (see Section 4.1), their memory footprint scales at a super-exponential rate of $O(m!)$. Therefore, one central question is how ordinal patterns should best be represented in the

digital domain, and another crucial aspect is the amount of extra memory required for obtaining that representation.

Against this backdrop, and motivated by the semantic gap between “high efficiency” and downright “infeasibility”, the following chapter will elaborate on the computational pitfalls and algorithmic possibilities of encoding ordinal patterns.

5.1 Requirements and Pitfalls

We defined ordinal patterns as m -tuples of ranks $(\rho_1, \rho_2, \dots, \rho_m) \in \Omega_m \subset \mathbb{N}^m$, and thereby also established an easily interpretable means of notation (see Section 4.1.3). Human interpretability is, however, not a primary concern when storing data in computer memory. Different requirements then prevail, and render the rank representation rather cumbersome. Before we present an encoding more suitable for the digital domain, let us look at these requirements.

Assume that we want to store an ordinal pattern $(\rho_1, \rho_2, \dots, \rho_m)$ of order m in the main memory of a computer. The naïve solution (for any $m < 256$) would then be to use an array of m consecutive bytes, each holding a particular rank ρ_i . However, this approach is disadvantageous in several respects, the most prominent being the following.

- While there are $m!$ distinct ordinal patterns of order m , a block of m bytes can take on 256^m different states. Due to $m! \ll 256^m$ for small m , the memory footprint of the above encoding is far from optimal in terms of memory utilisation.
- Testing a pair of ordinal patterns for equality requires up to m byte-wise comparisons, which is particularly detrimental to the run-time of operations like sorting and searching.
- Counting distinct pattern occurrences in a sequence of ordinal patterns (say, for probability estimation) requires an associative array that provides a map from each possible tuple of ranks to its respective counter variable.

Clearly, all of the shortcomings listed above can be overcome by encoding the set of ordinal patterns $\Omega_m = \{\pi_1, \pi_2, \dots, \pi_{m!}\}$ using non-negative integers $\{0, 1, \dots, m! - 1\}$, that is, by establishing a bijective map

$$\begin{aligned} \text{enc}: \Omega_m &\rightarrow \mathbb{N}_0, \\ \pi_i &\mapsto i - 1. \end{aligned} \tag{5.1}$$

Such a map could readily be implemented in software by means of a lookup table. Unfortunately, encoding a pattern would then require up to $m!$ iterations of that table, with each iteration involving up to m integer comparisons. Admittedly, replacing the lookup table by a more sophisticated data structure may mitigate performance issues to some extent. However, an arguably more elegant closed-form solution exists that directly maps any m -tuple of values $(x_1, x_2, \dots, x_m) \in X^m$ onto its ordinal pattern in numerical representation. The approach is a straightforward extension of the classical Lehmer code.

5.2 The Lehmer Code

Named in appreciation of Derrick Lehmer, the Lehmer code assigns a unique non-negative integer $i \in \{0, 1, \dots, m! - 1\}$ to each permutation of a set of m elements. The foundations of this mathematical problem had already been studied in the 19th century [149], and Lehmer incorporated them into his work on algorithms for combinatorial computing, published as *Teaching Combinatorial Tricks to a Computer* [150].

The basic idea is that a permutation is the result of successively drawing pairwise distinct elements from a set X without replacement, such that with each draw, the number of possible choices decreases by one. To take advantage thereof, the m elements of X are first enumerated according to their natural order, and starting from zero. (If X does not have a natural order, an arbitrary total order can be assigned instead.) This yields the sorted sequence

$$(a_0, a_1, \dots, a_{m-1}) = (a_i)_{i=0}^{m-1} \quad \text{with} \quad a_i \in X.$$

The first element a_{r_1} to be selected from X can then be referred to by the index r_1 . In the next step, a concordantly sorted, but independently numbered sequence

$$(b_0, b_1, \dots, b_{m-2}) = (b_i)_{i=0}^{m-2} \quad \text{with} \quad b_i \in X \setminus \{a_{r_1}\}$$

is created from the $m - 1$ remaining elements of X , and the second element b_{r_2} to be drawn is indexed by r_2 . One continues in the same manner for the remaining $m - 2$ elements. At its termination, the procedure yields an m -tuple of indices

$$(r_1, r_2, \dots, r_m) \in R_m \quad \text{with} \quad R_m = \{(r_i)_{i=1}^m \mid 0 \leq r_i \leq m - i\}. \quad (5.2)$$

Obviously, there are $|R_m| = m!$ different tuples (r_1, r_2, \dots, r_m) of length m , and those tuples correspond to the $m!$ possible permutations of X in a one-to-one manner. Before we go on, an example may prove insightful.

Example 5. Given the permutation $\sigma = (5, 2, 8, 13, 1, 3, 21)$, its representation in terms of Equation (5.2) is obtained by the seven iterations outlined in Table 5.1, which yields the tuple $(3, 1, 2, 2, 0, 0, 0) \in R_7$. Trivially, the last iteration results in $r_m = 0$ for any number of elements m , and for any permutation σ . It can therefore be omitted in practice.

Table 5.1 Mapping the permutation $\sigma = (5, 2, 8, 13, 1, 3, 21)$ onto $(3, 1, 2, 2, 0, 0, 0) \in R_7$.

Iteration	Selection	Ranks						
i	r_i	0	1	2	3	4	5	6
1	3	1	2	3	5	8	13	21
2	1	1	2	3	8	13	21	
3	2	1	3	8	13	21		
4	2	1	3	13	21			
5	0	1	3	21				
6	0	3	21					
7	0	21						

Example 5 illustrates an important property of this encoding: each r_i represents the number of elements in X that are smaller than x_{r_i} and have not yet been drawn. More formally, it holds for each iteration i that

$$r_i = \left| \left\{ x_k \in X \setminus \{x_{r_1}, \dots, x_{r_i}\} \mid x_k < x_{r_i} \right\} \right|. \quad (5.3)$$

In terms of contemporary combinatorics [81], the indices r_i are called the right inversion counts to the permutation σ . In the same diction, a pair of elements i and j is an inversion if $i < j$ and $\sigma(i) > \sigma(j)$. More specifically, this constellation is named a left inversion for j , and a right inversion for i . The right inversion counts to a permutation σ of m elements are therefore the trivial $r_m = 0$ for the rightmost position, and

$$r_i = \sum_{j=i+1}^m [\sigma(i) > \sigma(j)] \quad \text{for all} \quad 1 \leq i \leq m-1. \quad (5.4)$$

The big advantage of this encoding shows if we interpret (r_1, r_2, \dots, r_m) as the digits of a factoradic numeral (see Section 2.2.3), which is the final step of the Lehmer code.

Because of $r_i \leq m - i$ for all r_i in (r_1, r_2, \dots, r_m) , any possible string of digits $r_1 r_2 \dots r_m$ is a valid factoradic numeral in terms of Equation (2.7), and its numeric value is

$$n = \sum_{i=1}^m r_i (m - i)! = \sum_{i=1}^{m-1} r_i (m - i)!.$$

In conjunction with Equation (5.4), the Lehmer code for a permutation σ of m elements can thus be calculated as

$$n = \sum_{i=1}^{m-1} \left((m - i)! \sum_{j=i+1}^m [\sigma(i) > \sigma(j)] \right). \quad (5.5)$$

5.3 Lehmer-Encoding Ordinal Patterns

According to Definition 1, an ordinal pattern $(\rho_1, \rho_2, \dots, \rho_m) \in \Omega_m$ is nothing but an m -tuple of ranks, and thus, a distinct permutation of the natural numbers $\{1, 2, \dots, m\}$. Consequently, the Lehmer code can be used to map the ordinal patterns in Ω_m onto the non-negative integers $\{0, 1, \dots, m! - 1\}$ in a one-to-one manner. This enables a closed-form realisation

$$\begin{aligned} \text{enc}: \Omega_m &\rightarrow \mathbb{N}_0, \\ (\rho_1, \rho_2, \dots, \rho_m) &\mapsto \sum_{i=1}^{m-1} \left((m - i)! \sum_{j=i+1}^m [\rho_i > \rho_j] \right) \end{aligned} \quad (5.6)$$

of the enc-function so far only implicitly defined by Equation (5.1).

Now recall from Definition 1 that the ordinal pattern $(\rho_1, \rho_2, \dots, \rho_m) \in \Omega_m$ of an m -tuple $(x_1, x_2, \dots, x_m) \in X^m$ is defined such that

$$\rho_i < \rho_j \iff (x_i < x_j \vee (x_i = x_j \wedge i < j)).$$

Given that $i < j$ universally holds in Equation (5.6), the composition $\text{sym} = \text{enc} \circ \text{op}$ can therefore be written as

$$\begin{aligned} \text{sym}: X^m &\rightarrow \mathbb{N}_0, \\ (x_1, x_2, \dots, x_m) &\mapsto \sum_{i=1}^{m-1} \left((m - i)! \sum_{j=i+1}^m [x_i > x_j] \right). \end{aligned} \quad (5.7)$$

This function directly maps any given m -tuple $(x_1, x_2, \dots, x_m) \in X^m$ onto the numerical representation $\text{enc}(\pi_i) \in \{0, 1, \dots, m! - 1\}$ of its respective ordinal pattern. From a certain point of view, this is enabled by the idempotence of the op -function (see Section 4.1), which implies that (x_1, x_2, \dots, x_m) and $(\rho_1, \rho_2, \dots, \rho_m)$ have the same ordinal pattern, and therefore, identical inversion counts.

A property worth pointing out is that, figuratively speaking, the Lehmer code preserves the lexicographic sorting order of the permutations it encodes [150]. Transferred to the scope of the present application, this circumstance provides for a very natural enumeration of ordinal patterns: tuples of ranks, tuples of inversion counts, as well as the resulting numerical codes are all consistently sorted. The reader may find Example 6 instructive in this respect.

Example 6. Consider the $m! = 24$ ordinal patterns $(\rho_1, \rho_2, \rho_3, \rho_4) \in \Omega_4$ of order $m = 4$. Each of them has a distinct tuple of right inversion counts (r_1, r_2, r_3) . Its corresponding numerical representation n is obtained by interpreting (r_1, r_2, r_3) as the digits of a factoradic numeral, in particular, $n = 6r_1 + 2r_2 + r_3$. As shown in Table 5.2, the tuples of ranks, tuples of inversion counts, and numerical codes all obey the same lexicographic sorting order.

Table 5.2 The ranks $(\rho_1, \rho_2, \rho_3, \rho_4)$, right inversion counts (r_1, r_2, r_3) , and numerical codes n for the $m! = 24$ ordinal patterns of order $m = 4$.

Ranks				Inv.			Code	Ranks				Inv.			Code
ρ_1	ρ_2	ρ_3	ρ_4	r_1	r_2	r_3	n	ρ_1	ρ_2	ρ_3	ρ_4	r_1	r_2	r_3	n
1	2	3	4	0	0	0	0	3	1	2	4	2	0	0	12
1	2	4	3	0	0	1	1	3	1	4	2	2	0	1	13
1	3	2	4	0	1	0	2	3	2	1	4	2	1	0	14
1	3	4	2	0	1	1	3	3	2	4	1	2	1	1	15
1	4	2	3	0	2	0	4	3	4	1	2	2	2	0	16
1	4	3	2	0	2	1	5	3	4	2	1	2	2	1	17
2	1	3	4	1	0	0	6	4	1	2	3	3	0	0	18
2	1	4	3	1	0	1	7	4	1	3	2	3	0	1	19
2	3	1	4	1	1	0	8	4	2	1	3	3	1	0	20
2	3	4	1	1	1	1	9	4	2	3	1	3	1	1	21
2	4	1	3	1	2	0	10	4	3	1	2	3	2	0	22
2	4	3	1	1	2	1	11	4	3	2	1	3	2	1	23

With a view to software implementation, Equation (5.7) still provides opportunity for optimisation. In the current form, the factorial $(m - i)!$ needs to be re-evaluated for each iteration of the outer sum. In general, computing a (non-trivial) factorial requires a sequence of multiplications, but due to the specific structure of Equation (5.7), these multiplications are here avoidable in their entirety. Taking advantage of the fundamental recurrence relation $k! = k(k - 1)!$, the value of $\text{sym}(x_1, x_2, \dots, x_m)$ can be computed recursively by initialising $n_0 = 0$, and successively iterating

$$n_i = (m - i)(n_{i-1} + r_i) \quad \text{with} \quad r_i = \sum_{j=i+1}^m [x_i > x_j]. \quad (5.8)$$

The recursion terminates after iteration $i = m - 1$, and yields $n_{m-1} = \text{sym}(x_1, x_2, \dots, x_m)$ as the result. The arithmetical equivalence with Equation (5.7) can be proven by mathematical induction, and is also evident from the following example.

Example 7. Let $(r_1, r_2, \dots, r_6) \in R_6$ denote the right inversion counts to the ordinal pattern of a given tuple (x_1, x_2, \dots, x_6) . According to Equation (5.7), the numerical representation of this ordinal pattern of order $m = 6$ is obtained by computing

$$\begin{aligned} \text{sym}(x_1, x_2, \dots, x_6) &= 5 \cdot 4 \cdot 3 \cdot 2 \cdot r_1 \\ &+ 4 \cdot 3 \cdot 2 \cdot r_2 \\ &+ 3 \cdot 2 \cdot r_3 \\ &+ 2 \cdot r_4 \\ &+ r_5. \end{aligned}$$

Iterating the recurrence relation given by Equation (5.8) for the same right inversion counts (r_1, r_2, \dots, r_6) and the same pattern order $m = 6$, we obtain

$$n_5 = \text{sym}(x_1, x_2, \dots, x_6) = \left(\left((r_1 \cdot 5 + r_2) \cdot 4 + r_3 \right) \cdot 3 + r_4 \right) \cdot 2 + r_5.$$

Not only are both solutions arithmetically identical, but the recursive approach also requires considerably fewer multiplications—*quod erat illustrandum*.

The above reflections on Lehmer-encoding ordinal patterns can be summarised by means of a short piece of pseudocode. The following Algorithm 1 thus constitutes the essence of the ideas discussed so far: a remarkably simple algorithm for extracting and storing ordinal patterns in computer memory.

Algorithm 1. Given an m -tuple $(x_1, x_2, \dots, x_m) \in X^m$ of elements from a totally ordered set X , a distinct numerical representation $n \in \{0, 1, \dots, m! - 1\}$ of its ordinal pattern of order m can be obtained as outlined by the following pseudocode.

```

1 function encode_pattern
2 input
3      $(x_1, x_2, \dots, x_m) \in X^m$  with  $m \in \{2, 3, \dots\}$ 
4 output
5      $n \in \{0, 1, \dots, m! - 1\}$ 
6 begin
7      $n \leftarrow 0$ 
8     for  $i \leftarrow 1$  to  $m - 1$  do
9         for  $j \leftarrow i + 1$  to  $m$  do
10             $n \leftarrow n + [x_i > x_j]$ 
11        end
12         $n \leftarrow (m - i) n$ 
13    end
14    return  $n$ 
15 end.

```

5.4 The Keller-Sinn-Emonds Code

Keller, Sinn, and Emonds [107] had already proposed a numerical encoding scheme for ordinal patterns in 2007. Their encoding is quite similar to the approach here derived from the Lehmer code, because inversion counts are used in either case. Formal deduction and notational preferences let aside, both solutions merely differ in the way that tuples of inversion counts (r_1, r_2, \dots, r_m) are mapped onto scalar integers. In the notation of the present manuscript, Keller and colleagues suggested to use the map

$$\begin{aligned} \text{sym}^* : X^m &\rightarrow \mathbb{N}_0, \\ (x_1, x_2, \dots, x_m) &\mapsto \sum_{i=1}^{m-1} \left(\frac{m!}{(m-i+1)!} \sum_{j=i+1}^m [x_i > x_j] \right). \end{aligned} \quad (5.9)$$

As opposed to the factoradic numerals $r_1 r_2 \dots r_m$ underlying Equation (5.7), the Keller-Sinn-Emonds code is effectively based on the reversed numerals $r_m \dots r_2 r_1$, which are

interpreted as strings of digits $d_N \dots d_2 d_1 d_0$ in the mixed-radix positional numeral system (see Section 2.2.3) given by

$$n = \sum_{i=0}^N \left(d_i \prod_{j=0}^i b(j) \right) \quad \text{to the base} \quad b(j) = \begin{cases} 1 & \text{for } j = 0, \\ N - j + 2 & \text{for } 1 \leq j \leq N. \end{cases}$$

Clearly, the encoding proposed by Keller, Sinn, and Emonds serves the same purpose as the Lehmer-based approach described above, and also meets the general requirements of Section 5.1 equally well. That being said, a minor drawback of their solution is that factorial functions and integer division operations are required to evaluate Equation (5.9). Those operations are computationally expensive. As suggested in [145], the issue can be mitigated by computing the coefficients

$$w_i = \frac{m!}{(m-i+1)!} \quad \text{for } i \in \{1, 2, \dots, m-1\}$$

in advance, and looking them up during the actual encoding. The approach then requires just as many (or few) multiplications as the Lehmer encoding given by Equation (5.7). Still, the latter does not depend on lookup tables, and arguably yields a more natural enumeration of the ordinal patterns (see Table 5.2). For those reasons, the Lehmer code will be used exclusively throughout the present work.

5.5 Algorithms for Encoding Time Series

As already discussed in Chapter 4, any form of ordinal time series analysis requires that sequences of elements from a totally ordered set X (usually, the real numbers \mathbb{R}) be converted into sequences of ordinal patterns. Given a finite time series $\{x_t\}$, with $x_t \in X$ and $t \in \{1, 2, \dots, N\}$, we select a pattern order $m \geq 2$ and time lag $\tau \geq 1$, and subsequently obtain $\{\pi_t\}$, where

$$\pi_t = \text{op}(x_t, x_{t+\tau}, \dots, x_{t+(m-1)\tau}) \quad \text{for all } t \in \{1, 2, \dots, N - (m-1)\tau\}. \quad (5.10)$$

In doing so, we assign to each ordinal pattern π_t the time index t of the leftmost of its underlying tuple elements. Consequently, the last $(m-1)\tau$ indices of the finite time series $\{x_t\}$ do not reference any ordinal pattern, and the resulting pattern sequence $\{\pi_t\}$ thus comprises of $N - (m-1)\tau$ elements.

To perform this transformation in software, the encoding approach as described by Algorithm 1 can be applied. Moreover, when encoding not one, but a sequence of ordinal patterns, the intrinsic structure of ordinal processes (see Section 4.2.3) enables some optimisations that are advantageous in terms of computational efficiency. Three such algorithms for transforming time series into sequences of ordinal patterns will be discussed in the following.

5.5.1 The Plain Algorithm

Recall from Section 5.3 that Algorithm 1 maps any given m -tuple $(x_1, x_2, \dots, x_m) \in X^m$ of elements from a totally ordered set X onto a non-negative integer $n \in \{0, 1, \dots, m! - 1\}$, such that the value

$$n = \text{enc}(\text{op}(x_1, x_2, \dots, x_m)) = \text{sym}(x_1, x_2, \dots, x_m)$$

unambiguously identifies the ordinal pattern $\text{op}(x_1, x_2, \dots, x_m) = (\rho_1, \rho_2, \dots, \rho_m) \in \Omega_m$. This encoding strategy is easily expanded, so as to turn an entire time series $\{x_t\}$ into numerical representations $\{n_t\}$ of its ordinal pattern sequence $\{\pi_t\}$, whereby

$$n_t = \text{enc}(\pi_t) = \text{sym}(x_t, x_{t+\tau}, \dots, x_{t+(m-1)\tau}) \quad \forall t \in \{1, 2, \dots, N - (m-1)\tau\}. \quad (5.11)$$

This straightforward extension of Algorithm 1 merely requires an additional loop and proper indexing. It results in the following algorithm.

Algorithm 2 (Plain Algorithm). *To transform a finite time series $\{x_t\}$ of elements from a totally ordered set X into a sequence of non-negative integers $\{n_t\}$, select a pattern order $m \geq 2$ and a time lag $\tau \geq 1$. Then proceed according to the following pseudocode.*

```

1 function encode_sequence
2 input
3      $m \in \mathbb{N}$  with  $m \geq 2$ 
4      $\tau \in \mathbb{N}$  with  $\tau \geq 1$ 
5      $\{x_t\}$  with  $x_t \in X$  and  $t \in \{1, 2, \dots, N\}$ 
6 output
7      $\{n_t\}$  with  $n_t \in \{0, 1, \dots, m! - 1\}$  and  $t \in \{1, 2, \dots, N - (m-1)\tau\}$ 

```



```

8 begin
9   for  $t \leftarrow 1$  to  $N - (m - 1)\tau$  do
10      $n_t \leftarrow \text{encode\_pattern}(x_t, x_{t+\tau}, \dots, x_{t+(m-1)\tau})$ 
11   end
12 end.

```

The value $n_t \in \{0, 1, \dots, m! - 1\}$ then uniquely identifies the ordinal pattern π_t of order m , extracted from the time series $\{x_t\}$ at time index t using the time lag τ . The function `encode_pattern` is specified in Algorithm 1.

5.5.2 The Overlap Algorithm

The plain algorithm (Algorithm 2) still contains some redundant operations, and does therefore not scale too well with the pattern order m . This aspect can be targeted by further optimisation. Bandt and Pompe had already hinted at this possibility in their seminal publication on ordinal patterns, suggesting there was “an extremely fast algorithm where each pair of values need to be compared only once” [1]. Keller, Sinn, and Emonds elaborated further on this matter, demonstrating that the overlap property of ordinal processes (see Section 4.2.3) can be exploited computationally [107]. The algorithm described in the following builds upon the same basic idea, but additionally uses the recursive Lehmer encoding discussed in Section 5.3.

Written out in its entirety, the plain algorithm (Algorithm 2) converts a time series $\{x_t\}$, indexed by $t \in \{1, 2, \dots, N\}$, into a sequence of non-negative integers $\{n_t\}$, such that

$$n_t = \sum_{i=1}^{m-1} \left((m-i)! \sum_{j=i+1}^m \left[x_{t+(i-1)\tau} > x_{t+(j-1)\tau} \right] \right) \quad \forall t \in \{1, 2, \dots, N - (m-1)\tau\}. \quad (5.12)$$

As derived in Section 5.3, each evaluation of the inner sum of Equation (5.12) yields one of the $m - 1$ non-trivial right inversion counts to the ordinal pattern π_t . The encoding can hence be reformulated in terms of

$$n_t = \sum_{i=1}^{m-1} (m-i)! \cdot r_{t,i} \quad \text{where} \quad r_{t,i} = \sum_{j=i+1}^m \left[x_{t+(i-1)\tau} > x_{t+(j-1)\tau} \right]. \quad (5.13)$$

Now recall that due to the properties of ordinal processes discussed in Section 4.2.3, any two ordinal patterns $\pi_{t-\tau}$ and π_t at a distance equalling the time lag τ share all but one

of their underlying time series values. Consequently, the inversion counts to the patterns $\pi_{t-\tau}$ and π_t are strongly interrelated as well. In particular, it is easily confirmed that

$$n_t = \sum_{i=1}^{m-1} (m-i)! \cdot r_{t,i} \quad \text{where} \quad r_{t,i} = \left[x_{t+(i-1)\tau} > x_{t+(m-1)\tau} \right] + r_{t-\tau, i+1}. \quad (5.14)$$

This recurrence relation is highly advantageous in computational terms. Let us assume that in the overall process of encoding a time series $\{x_t\}$, the inversion counts to the pattern $\pi_{t-\tau}$ have just been determined. If those are kept in memory for τ more iterations, then encoding the pattern π_t merely requires $m-1$ additional comparisons. In turn, each such comparison yields one of the inversion counts to the pattern π_t , which can then be reused to efficiently encode $\pi_{t+\tau}$ another τ time steps ahead.

Merely the first τ ordinal patterns $\{\pi_1, \pi_2, \dots, \pi_\tau\}$ require additional consideration, because obviously, none of them has a neighbour located τ time steps earlier. While $\{\pi_{1-\tau}, \pi_{2-\tau}, \dots, \pi_0\}$ are hence undefined, all but the leftmost of their respective inversion counts still formally exist. More precisely,

$$\mathbf{R}_{\text{init}} = \begin{pmatrix} r_{1-\tau, 2} & r_{1-\tau, 3} & \cdots & r_{1-\tau, m-1} \\ r_{2-\tau, 2} & r_{2-\tau, 3} & \cdots & r_{2-\tau, m-1} \\ \vdots & \vdots & \ddots & \vdots \\ r_{0, 2} & r_{0, 3} & \cdots & r_{0, m-1} \end{pmatrix} \quad (5.15)$$

are all well-defined, and suffice to calculate $\{n_1, n_2, \dots, n_\tau\}$ in terms of Equation (5.14).

The procedure of encoding a time series thus comprises of two stages: first obtain the initial inversion counts \mathbf{R}_{init} , then iterate Equation (5.14) for all t , starting from $t=1$. Each such iteration yields an encoded pattern n_t , as well as a corresponding set of inversion counts. The latter are temporarily buffered in memory, reused τ iterations later, and then become obsolete. The entire process is summarised in the following Algorithm 3.

Algorithm 3 (Overlap Algorithm). *To transform a finite time series $\{x_t\}$ of elements $x_t \in X$ into a sequence of non-negative integers $\{n_t\}$ representing the ordinal patterns $\{\pi_t\}$ of the time series, select a pattern order $m \geq 2$ and time lag $\tau \geq 1$. Then proceed as follows.*

```

1 function encode_sequence
2 input
3      $\{x_t\}$    with  $x_t \in X$  and  $t \in \{1, 2, \dots, N\}$ 
4      $m \in \mathbb{N}$  with  $m \geq 2$ 
5      $\tau \in \mathbb{N}$  with  $\tau \geq 1$ 
6 output
7      $\{n_t\}$    with  $n_t \in \{0, 1, \dots, m! - 1\}$  and  $t \in \{1, 2, \dots, N - (m - 1)\tau\}$ 
8 locals
9      $\{r_{i,j}\}$  with  $i \in \{1, 2, \dots, \tau\}$  and  $j \in \{1, 2, \dots, m\}$ 
10 begin
11      $r_{i,j} \leftarrow 0$  for all  $i \in \{1, 2, \dots, \tau\}$  and all  $j \in \{1, 2, \dots, m\}$ 
12
13     /* Obtain initial right inversion counts  $\mathbf{R}_{\text{init}}$  */
14     for  $t \leftarrow 1$  to  $\tau$  do
15         for  $i \leftarrow 1$  to  $m - 2$  do
16             for  $j \leftarrow i + 1$  to  $m - 1$  do
17                  $r_{t,i+1} \leftarrow r_{t,i+1} + \left[ x_{t+(i-1)\tau} > x_{t+(j-1)\tau} \right]$ 
18             end
19         end
20     end
21
22     /* Extract and encode ordinal patterns */
23      $i \leftarrow 1$ 
24     for  $t \leftarrow 1$  to  $N - (m - 1)\tau$  do
25         for  $j \leftarrow 1$  to  $m - 1$  do
26              $r_{i,j} \leftarrow r_{i,j+1} + \left[ x_{t+(j-1)\tau} > x_{t+(m-1)\tau} \right]$ 
27              $n_t \leftarrow (m - j)(n_t + r_{i,j})$ 
28         end
29          $i \leftarrow (i \bmod \tau) + 1$  /* Increment circular buffer index */
30     end
31 end.

```

Notice that readability is favoured over efficiency in the above pseudocode description of the overlap algorithm. That is to say, a smaller buffer $\{r_{i,j}\}$ with $i \in \{1, \dots, \tau\}$ and $j \in \{2, \dots, m-1\}$ will suffice for an actual implementation, because the recurrence relation does not depend on $r_{i,1}$ at all, whereas $r_{i,m} = 0$ always. Also, the modulo operation used for circular buffer indexing may impose a considerable run-time penalty. Both of these

aspects are addressed in the reference implementation of the above Algorithm 3, which is provided in the supplements of [78].

5.5.3 The Lookup Algorithm

The overlap algorithm (Algorithm 3) is based on the fact that any two ordinal patterns $\pi_{t-\tau}$ and π_t overlap in all but one of their underlying time series values. Utilising the same interrelation, Valentina Unakafova and Karsten Keller proposed [145] a different encoding strategy that, by contrast with Algorithm 3, does not depend on buffering inversion counts. Their approach is compellingly simple: as described in Section 4.2.3, an ordinal pattern $\pi_{t-\tau}$ of order m can only have m different succeeding patterns π_t at τ time steps distance. Consequently, if the ordinal pattern $\pi_{t-\tau} = \text{op}(x_{t-\tau}, x_t, \dots, x_{t+(m-2)\tau})$ is known in advance, then the value of the expression

$$\rho_{t,m} = m - \sum_{i=1}^{m-1} \left[x_{t+(i-1)\tau} > x_{t+(m-1)\tau} \right] \quad (5.16)$$

uniquely determines the pattern $\pi_t = \text{op}(x_t, x_{t+\tau}, \dots, x_{t+(m-1)\tau})$. In connection with Definition 1, the decisive variable $\rho_{t,m} \in \{1, 2, \dots, m\}$ is easily identified as the rightmost rank of the ordinal pattern $\pi_t = (\rho_{t,1}, \rho_{t,2}, \dots, \rho_{t,m})$. As is visualised in Figure 5.1, each value that $\rho_{t,m}$ can take on represents one of the m different ordinal patterns π_t that may possibly follow after a particular pattern $\pi_{t-\tau}$.

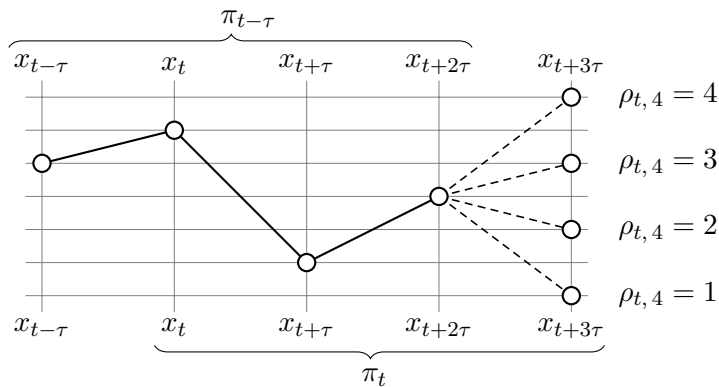


Figure 5.1 Assume pattern order $m = 4$, without loss of generality. For any fixed ordinal pattern $\pi_{t-\tau}$, its succeeding pattern $\pi_t = (\rho_{t,1}, \rho_{t,2}, \rho_{t,3}, \rho_{t,4})$ at τ time steps distance has merely one degree of freedom: its rightmost rank $\rho_{t,4}$.

These considerations essentially imply that a surjective map $(\pi_{t-\tau}, \rho_{t,m}) \mapsto \pi_t$ exists for any pattern order m and time lag τ . Likewise, there has to be a well-defined surjection

$$\begin{aligned} \mathbb{N}_0 \times \mathbb{N} &\rightarrow \mathbb{N}_0, \\ (n_{t-\tau}, \rho_{t,m}) &\mapsto n_t \end{aligned} \tag{5.17}$$

for ordinal patterns represented numerically. Notice that the particular encoding function $\text{enc}: \Omega_m \rightarrow \mathbb{N}_0$ in terms of Equation (5.1) does not make a difference here—as long as it is bijective, such that each ordinal pattern is assigned a unique numerical label. In their original publication [145], Unakafova and Keller used the encoding given by Equation (5.9).

To implement Equation (5.17), the authors rely on a lookup table holding $m! \times m$ entries, in particular

$$\mathbf{L}_m = \begin{pmatrix} L_{1,1} & \cdots & L_{1,m} \\ \vdots & \ddots & \vdots \\ L_{m!,1} & \cdots & L_{m!,m} \end{pmatrix}, \tag{5.18a}$$

such that

$$n_t = L_{i,j} \quad \text{for } i = n_{t-\tau} + 1 \quad \text{and } j = \rho_{t,n}. \tag{5.18b}$$

Encoding a time series $\{x_t\}$, with $t \in \{1, 2, \dots, N\}$, is then a matter of computing the numerical codes $\{n_1, n_2, \dots, n_\tau\}$ for the first τ patterns by direct evaluation, and subsequently iterating Equation (5.17) to obtain all remaining symbols. The following Algorithm 4 describes the process in full detail.

Algorithm 4 (Lookup Algorithm). *To transform a finite time series $\{x_t\}$ into a sequence of non-negative integers $\{n_t\}$ representing its ordinal patterns, select a pattern order $m \geq 2$ and time lag $\tau \geq 1$. Also prepare a lookup table $\{L_{i,j}\}$ according to Equation (5.18) that matches the `encode_pattern` function to be used. Then proceed as follows.*

```

1 function encode_sequence
2 input
3      $\{x_t\}$    with  $x_t \in X$  and  $t \in \{1, 2, \dots, N\}$ 
4      $m \in \mathbb{N}$  with  $m \geq 2$ 
5      $\tau \in \mathbb{N}$  with  $\tau \geq 1$ 
6 output
7      $\{n_t\}$    with  $n_t \in \{0, 1, \dots, m! - 1\}$  and  $t \in \{1, 2, \dots, N - (m - 1)\tau\}$ 
    
```

```

8 locals
9    $\{L_{i,j}\}$  with  $L_{i,j} \in \{0, 1, \dots, m! - 1\}$  and  $i \in \{1, 2, \dots, m\}$  and  $j \in \{1, 2, \dots, m\}$ 
10 begin
11    $\{L_{i,j}\} \leftarrow \text{load\_lookup\_table}(m)$ 
12
13   /* Encode first  $\tau$  ordinal patterns */
14   for  $t \leftarrow 1$  to  $\tau$  do
15      $n_t \leftarrow \text{encode\_pattern}(x_t, x_{t+\tau}, \dots, x_{t+(m-1)\tau})$ 
16   end
17
18   /* Encode all remaining patterns */
19   for  $t \leftarrow \tau + 1$  to  $N - (m - 1)\tau$  do
20      $row \leftarrow n_{t-\tau} + 1$ 
21      $col \leftarrow 1$ 
22     for  $i \leftarrow 1$  to  $m - 1$  do
23        $col \leftarrow col + [x_{t+(i-1)\tau} > x_{t+(m-1)\tau}]$ 
24     end
25      $n_t \leftarrow L_{row, col}$ 
26   end
27 end.

```

5.6 Implementation and Run-Time Performance

How an abstract algorithm is translated into actual software can make a big difference for its run-time efficiency: poor code will easily ruin the most sophisticated algorithm, whereas considerate implementation often turns a simplistic approach into a valuable piece of software. Taking into account application-dependent requirements and platform-specific peculiarities is essential in this regard. Generally speaking, there is rarely a singular solution to a multi-faceted problem.

In a similar context, Donald Knuth once wrote that “premature optimization is the root of all evil” [151]. That is to say that in software engineering, simplicity should be sacrificed for run-time performance only where the improvement is substantial to the application. Arguably, this principle applies to scientific programming in particular, where readable code may support the communication of analysis techniques and facilitate the reproducibility of research results—including one’s own results from a few years ago.

With this in mind, three encoding algorithms of varying complexity have been presented in the previous section. Each of them comes with strengths and weaknesses, and raises different implementational challenges. The following section is intended to guide the reader in selecting and implementing the most appropriate algorithm for a particular analysis task and software platform. Regarding the latter, NumPy/Python [152], GNU Octave [153], MATLAB (The Mathworks, Natick, MA, USA), and the C programming language will be considered.

5.6.1 Asymptotic Computational Complexities

The computational complexities of the plain, overlap, and lookup algorithm differ considerably in dependence of the pattern order m . This can be assessed by counting the number of basic operations the algorithms have to perform to encode a single ordinal pattern. Those operation counts, as obtained from the pseudocode listings of Section 5.5, are provided in Table 5.3.

Table 5.3 Number of operations required to encode a single ordinal pattern of order m , using either the plain algorithm (Algorithm 2), the overlap algorithm (Algorithm 3), or the lookup algorithm (Algorithm 4). Early initialisation operations have not been considered.

Algorithm	Add	Multiply	Compare	Increment	Assign	Modulo
plain	$\frac{m^2 + 3m - 2}{2}$	$m - 1$	$m^2 - 1$	$\frac{m^2 + m - 2}{2}$	$\frac{m^2 + 3m}{2}$	0
overlap	$9m - 8$	$6m - 6$	$2m - 2$	m	$2m$	1
lookup	$6m - 3$	$2m - 1$	$2m - 2$	$m - 1$	$m + 3$	0

It follows from those operation counts that the plain algorithm, performing a total of

$$C_{\text{plain}}(m) = \frac{5m^2 + 9m - 8}{2}$$

basic operations per pattern of order m , has an asymptotic computational complexity of $O(m^2)$. By contrast, the overlap and lookup algorithms both scale linearly with the pattern order: their total operation counts per pattern are

$$C_{\text{overlap}}(m) = 20m - 15 \quad \text{and} \quad C_{\text{lookup}}(m) = 12m - 4,$$

so their asymptotic complexities both are $O(m)$. The above also confirms that, for any finite pattern order $m \geq 2$, the lookup algorithm avoids some of the computations that the overlap algorithm has to perform—which is the very purpose of using a lookup table in the first place. Theoretical computational complexity thus decreases from the plain algorithm to the overlap algorithm, and again from the overlap algorithm to the lookup algorithm.

In practice, however, computational complexity alone does not suffice to determine the run-time performance of an algorithm, which also depends on such aspects as memory utilisation, the execution environment, and the actual implementation. The rest of this chapter therefore focusses on the practical aspects of turning each of the three encoding algorithms into workable software.

5.6.2 Memory Alignment

The algorithms here considered represent the ordinal patterns $\Omega_m = \{\pi_1, \pi_2, \dots, \pi_m\}$ of order m by distinct integers $\{0, 1, \dots, m! - 1\}$, thereby providing a bijective map $\pi_i \mapsto i - 1$ as stipulated by the enc-function of Equation (5.1). The resulting symbols are highly memory-efficient, theoretically requiring a mere $\log_2 m!$ bit per pattern. Notice that $\log_2 m!$ is the entropy of a uniform distribution of $m!$ elements, and thus, the maximum entropy an ordinal pattern distribution of order m can possibly attain. Let aside data compression techniques, no other numerical encoding can therefore be more compact (as is assured by Shannon’s source coding theorem [85, 86]).

In practice, it makes sense to align ordinal patterns to byte boundaries, which can be accomplished by dedicating an integer power of 2 (but at least 8) bits to each ordinal pattern, such that the resulting bit width per pattern is

$$w_b = 2^k \geq \log_2 m! \quad \text{where} \quad k \in \{3, 4, \dots\}. \quad (5.19)$$

Any digital processing unit equipped with 64-bit integer registers will thus handle ordinal patterns of order $m \leq 20$ natively, that is, at hardware efficiency. For reference, Table 5.4 lists the maximum pattern orders that fit the primitive data types available on standard computer systems. Although ordinal patterns and their numerical representations are intrinsically integral, this table also references two IEEE 754 floating point formats [154, 155]. Those were included because computation environments like NumPy/Python, GNU Octave, and MATLAB use the `binary64` floating point data format by default.

Table 5.4 Maximum pattern orders representable by standard numerical data types.

Data Type	Significant Bits w_b	Maximum Order m
uint8	8	5
uint16	16	8
uint32	32	12
uint64	64	20
binary32 (single)	24	10
binary64 (double)	53	18

Previously known as `double` [154], this format features an effective mantissa length of 53 bit [155], and can therefore represent a total of 2^{53} distinct non-negative values at integer precision [156]. Whenever this limit is exceeded, mantissa truncation will silently cause unexpected results (like the erroneous $2^{53} = 2^{53} + 1$), and distinct patterns will falsely be labelled as identical. When working with patterns of order $m > 18$ in such computation environments, this bug-inviting peculiarity must be kept in mind.

Independent of the software platform used, ordinal patterns of order $m > 20$ require some additional thought. This is because current general-purpose processors do not provide native support for integers wider than 64 bits, so each pattern of order $m > 20$ has to be stored as an array of integers, and all arithmetical and logical operations need to be emulated in software. Those matters will be further discussed in Section 5.6.8.

5.6.3 Compilation versus Interpretation

Without any doubt, high-level programming languages like Python and MATLAB are essential tools in scientific computing. Featuring expressive syntaxes, and providing turnkey solutions for a plethora of analysis tasks, numerical scripting languages can considerably reduce the time researchers have to spend turning ideas into code.

Instead of being translated into machine instructions beforehand, such languages are usually interpreted or just-in-time compiled at run-time. By matter of principle, this paradigm of execution imposes a certain overhead: each line of code needs to be tokenised, parsed, error-checked, and finally mapped onto a sequence of function calls or (virtual) machine instructions. Where this overhead is detrimental, developers of numerical computation frameworks switch to compiled programming languages, and merely implement

a thin wrapper in the targeted scripting language. Among many other functions, the fast Fourier transform (FFT) provided by your preferred numerical analysis software is almost certainly implemented like that.

To investigate whether ordinal pattern encoding can benefit from this approach, the plain, overlap, and lookup algorithms (Algorithms 2, 3, and 4) were implemented and benchmarked in MATLAB and NumPy/Python, but also in the C programming language. The latter was chosen as the compiled programming language because of its tight integration with most numerical computation environments, and its widespread support on virtually any hardware platform.

5.6.4 Run-Time Test Environment

All performance testing was done on a conventional x86-64 laptop computer, equipped with an Intel Core i7-5600U processor (Intel Corporation, Santa Clara, CA, USA) and 8 GB of random access memory (RAM). An Arch Linux distribution of the GNU/Linux operating system was used, running the default kernel (`linux`, 5.2.arch2-1), and C standard library (`glibc`, 2.29-3). Pre-built binary packages of GNU Octave (`octave`, 5.1.0-4), Python 3 (`python`, 3.7.3-1), NumPy (`python-numpy`, 1.16.4-1), and FFTW (`fftw`, 3.3.8-1), as well as their respective dependencies were installed from the official repositories of the distributor. The Linux version of the proprietary MATLAB 2018b release (9.5.0.1049112 Update 3) was used. Source code written in the C programming language was compiled using the GNU Compiler Collection (`gcc`, 8.3.0-2), whereby the parameters `-march=x86-64 -mtune=generic -O3` were selected to allow for heavy compiler optimisation, while not relying on any model-specific features of the targeted processor.

5.6.5 Test Signal Generation

Based on the fact that ordinal patterns of any order are uniformly distributed in white noise [116], sequences of independent and uniformly distributed pseudo-random numbers were used for testing the performance of the algorithms. This choice ascertains that all ordinal pattern transitions (see Figure 4.4) appear at the same relative frequency, such that each possible path of execution is taken equally often.

To maintain reproducible test signals across all software environments, the *xorshift random number generator* by George Marsaglia [157] was used, with a word size of 32 bits and the standard shift parameters (13, 17, 5). In this configuration, xorshifting will produce a pseudo-random sequence of period length $2^{32} - 1$, with all elements drawn from $\{1, 2, \dots, 2^{32} - 1\}$. In other words, Marsaglia’s random number generator then yields a pseudo-random permutation of the non-negative integers $0 < i < 2^{32}$. Normalisation of those random numbers (as in zero-mean or unit-variance) was omitted, considering that ordinal patterns are invariant to order-preserving transformations anyway [1]. However, the integer-valued test signals were stored in `binary64` floating point representation, as this is the expected input format in ordinal time series analysis.

Additionally, to test for possible dependencies between the run-time of the algorithms and the ordinal complexity of the input signal, low-pass filtered (and thus, self-correlated) noise of various bandwidths was also incorporated into the test procedure. As can be observed in Figure 5.2, band-limiting white noise is a simple yet effective way of obtaining test signals with gradually decreasing pattern diversity—and thus, decreasing ordinal complexity.

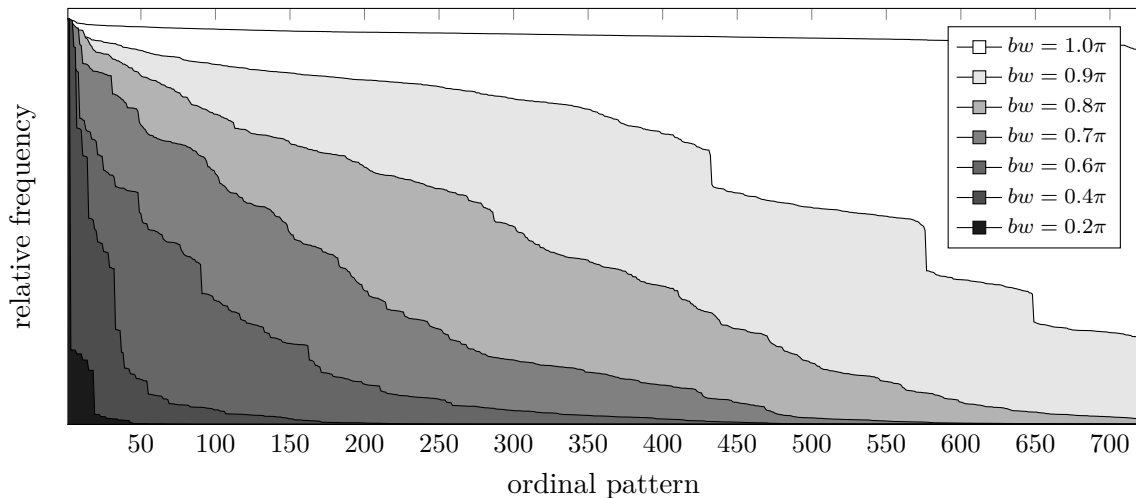


Figure 5.2 Relative frequencies of the 720 ordinal patterns of order $m = 6$ for uniform white noise ($bw = 1.0\pi$), and for noise band-limited to different fractions of the Nyquist frequency. Time series of $N = 3.6 \cdot 10^6$ samples length were used for the simulation. For each plot, the pattern counts were sorted in descending order to facilitate visualisation.

For reasons of simplicity, filtering was performed by zeroing bins in the Discrete Fourier Transform (DFT) of the full-bandwidth signal. To obtain numerically identical results

across different execution platform, the free and open-source software library FFTW3 [158] was used for all DFT computations.

5.6.6 Implementing the Plain Algorithm

The plain algorithm (Algorithm 2) is definitely the most simple among the three encoding strategies considered here. Because the algorithm makes no effort to avoid redundant operations, it may at first glance seem generally inferior to the more sophisticated overlap and lookup algorithms (Algorithms 3 and 4). Quite the contrary, the plain algorithm may actually be preferable when numerical scripting languages like GNU Octave, MATLAB or NumPy/Python are used to encode ordinal patterns. The reason is that, by contrast with the recursive Algorithms 3 and 4, the plain algorithm allows for vectorisation.

Compared to pre-compiled languages like C, scripting languages are relatively slow at iterating loops (see Section 5.6.3). This limitation clearly manifests if the plain algorithm is implemented in a straightforward manner, which then requires three levels of nested loops. See the `encode_plain` functions in the supplementary files `encode_plain.m` and `ordpat.py`, as well as the `ordpat_encode_plain` function in `ordpat.c`, respectively. As shown in Table 5.5, their run-time efficiency varies by orders of magnitude across different execution environments.

Table 5.5 Computation time (median of 20 trials) for turning $3.6 \cdot 10^5$ samples of uniform white noise into a sequence of ordinal patterns of order m , using the time lag $\tau = 1$. Straightforward iterative implementations of the plain algorithm (Algorithm 2) were tested in various computation environments.

Order	Computation Time (ms)				
	m	GNU Octave	NumPy/Python	MATLAB	C
2		$7.9 \cdot 10^3$	$2.4 \cdot 10^3$	$1.1 \cdot 10^1$	$7.8 \cdot 10^{-1}$
3		$2.0 \cdot 10^4$	$5.7 \cdot 10^3$	$2.2 \cdot 10^1$	$1.6 \cdot 10^0$
4		$3.6 \cdot 10^4$	$1.0 \cdot 10^4$	$3.5 \cdot 10^1$	$2.8 \cdot 10^0$
5		$5.6 \cdot 10^4$	$1.5 \cdot 10^4$	$5.5 \cdot 10^1$	$4.3 \cdot 10^0$
6		$8.1 \cdot 10^4$	$2.1 \cdot 10^4$	$8.5 \cdot 10^1$	$6.0 \cdot 10^0$
7		$1.1 \cdot 10^5$	$2.9 \cdot 10^4$	$1.2 \cdot 10^2$	$8.0 \cdot 10^0$
8		$1.4 \cdot 10^5$	$3.7 \cdot 10^4$	$1.6 \cdot 10^2$	$1.0 \cdot 10^1$
9		$1.8 \cdot 10^5$	$4.6 \cdot 10^4$	$2.0 \cdot 10^2$	$1.3 \cdot 10^1$

Those differences are due to the iterative nature of the plain algorithm, which forces the Octave and Python language interpreters to translate the same sequence of instructions over and over again, for each and every loop iteration. Consistently, the MATLAB just-in-time compiler performs better, but is in turn outperformed by the machine code of the fully-optimising C compiler.

To mitigate the performance penalty inherent to numerical scripting languages, a programming technique known as vectorisation can be applied in many cases. In a nutshell, vectorisation is about avoiding element-wise operations in favour of high-level instructions acting on blocks of data, like vectors (hence the name), matrices, and tensors. Vectorising the plain algorithm is a bit tricky, but the performance gain is well worth the effort. The approach is best explained by means of a practical example. MATLAB code will be used in the following, but the concepts translate to other programming environments as well.

Recall from Equation (5.7) that the map

$$(x_1, x_2, \dots, x_m) \mapsto \sum_{i=1}^{m-1} \left((m-i)! \sum_{j=i+1}^m [x_i > x_j] \right) \quad (5.20)$$

is the basis of the plain algorithm, and yields the ordinal pattern of the m -tuple (x_1, x_2, \dots, x_m) in its numerical representation. Now consider that, for any fixed pattern order m , the result of this function can be rewritten without relying on summation signs. Assuming the order $m = 5$, for example, the mathematical expression

$$\begin{aligned} & 24 \cdot \left([x_1 > x_2] + [x_1 > x_3] + [x_1 > x_4] + [x_1 > x_5] \right) \\ & + 6 \cdot \left([x_2 > x_3] + [x_2 > x_4] + [x_2 > x_5] \right) \\ & + 2 \cdot \left([x_3 > x_4] + [x_3 > x_5] \right) \\ & + 1 \cdot \left([x_4 > x_5] \right) \end{aligned}$$

is admittedly more tedious, but arithmetically equivalent to the more compact formulation in Equation (5.20). The point here is that this expression can directly be translated into a single MATLAB instruction, namely

$$\begin{aligned} & 24 * ((x1 > x2) + (x1 > x3) + (x1 > x4) + (x1 > x5)) \dots \\ & + 6 * ((x2 > x3) + (x2 > x4) + (x2 > x5)) \dots \\ & + 2 * ((x3 > x4) + (x3 > x5)) \dots \\ & + 1 * ((x4 > x5)); \end{aligned}$$

Let us assume that the time series to be analysed is represented by a $N \times 1$ vector `input` on the MATLAB workspace. Obviously, if we initialise the variables

```
x1 = input(1);
x2 = input(2);
x3 = input(3);
x4 = input(4);
x5 = input(5);
```

and call the above instruction, we obtain the ordinal pattern of the vector `input(1:5)` in its numerical representation. The underlying optimisation technique is called loop unrolling. Furthermore, consider that in numerical scripting languages, most basic operations are not limited to scalar values, but can in principle handle arrays of arbitrary dimension as their operands. If we thus set `x1`, ..., `x5` to the delay vectors

```
x1 = input(1:end-4);
x2 = input(2:end-3);
x3 = input(3:end-2);
x4 = input(4:end-1);
x5 = input(5:end-0);
```

instead, we can obtain from `input` its full sequence of ordinal patterns of order $m = 5$ and lag $\tau = 1$ by calling a single MATLAB instruction. Arbitrary time lags $\tau \geq 1$ can in turn be realised by using the generalised delay vectors

```
x1 = input(1 + 0*lag : end - 4*lag);
x2 = input(1 + 1*lag : end - 3*lag);
x3 = input(1 + 2*lag : end - 2*lag);
x4 = input(1 + 3*lag : end - 1*lag);
x5 = input(1 + 4*lag : end - 0*lag);
```

in conjunction with the exact same MATLAB expression. As a side note, the operation is also applicable to multidimensional data structures like matrices and tensors, such that multivariate time series can as well be encoded by means of a single invocation.

The downside of this approach is that each pattern order m requires a dedicated piece of code. When working with small pattern orders, manual implementation is still perfectly feasible and even yields comprehensible code—as is demonstrated by the `symbolise.m` function provided in the supplements of [77]. For higher pattern orders,

though, another convenient feature offered by many scripting languages should rather be utilised. GNU Octave, MATLAB and NumPy/Python all support self-modifying code, which is source code that can modify its own sequence of instructions at run-time. This is possible because scripts are translated during program execution anyway, so new strings of characters can easily be injected into the instruction queue of the language interpreter. Supporting languages provide built-in functions like `eval` or `exec` for this purpose, which take a string as their input argument and hand it over to the execution engine for evaluation. Such language facilities can neatly be utilised to obtain an efficient, universal implementation of the plain algorithm: one just has to write a function that dynamically creates the appropriately vectorised code for a given pattern order m and time lag τ , and subsequently executes it. Consider the functions `encode_vectorised` in the supplementary files `encode_vectorised.m` and `ordpat.py` for reference. As can be seen from Table 5.6, this form of optimisation yields a tremendous increase in run-time efficiency.

Table 5.6 Computation time (median of 20 trials) for turning $3.6 \cdot 10^5$ samples of uniform white noise into a sequence of ordinal patterns of order m , using the time lag $\tau = 1$. Vectorised implementations of the plain algorithm (Algorithm 2) were tested in various computation environments. The results of Table 5.5 were replicated for ease of comparison.

Order m	Computation Time (ms)						
	GNU Octave		NumPy/Python		MATLAB		C
	loops	vectors	loops	vectors	loops	vectors	loops
2	$7.9 \cdot 10^3$	$1.5 \cdot 10^0$	$2.4 \cdot 10^3$	$7.9 \cdot 10^{-1}$	$1.1 \cdot 10^1$	$3.8 \cdot 10^0$	$7.8 \cdot 10^{-1}$
3	$2.0 \cdot 10^4$	$6.5 \cdot 10^0$	$5.7 \cdot 10^3$	$1.3 \cdot 10^0$	$2.2 \cdot 10^1$	$9.0 \cdot 10^0$	$1.6 \cdot 10^0$
4	$3.6 \cdot 10^4$	$1.4 \cdot 10^1$	$1.0 \cdot 10^4$	$2.0 \cdot 10^0$	$3.5 \cdot 10^1$	$1.2 \cdot 10^1$	$2.8 \cdot 10^0$
5	$5.6 \cdot 10^4$	$2.0 \cdot 10^1$	$1.5 \cdot 10^4$	$2.9 \cdot 10^0$	$5.5 \cdot 10^1$	$1.5 \cdot 10^1$	$4.3 \cdot 10^0$
6	$8.1 \cdot 10^4$	$3.3 \cdot 10^1$	$2.1 \cdot 10^4$	$5.3 \cdot 10^0$	$8.5 \cdot 10^1$	$1.8 \cdot 10^1$	$6.0 \cdot 10^0$
7	$1.1 \cdot 10^5$	$4.2 \cdot 10^1$	$2.9 \cdot 10^4$	$6.8 \cdot 10^0$	$1.2 \cdot 10^2$	$2.3 \cdot 10^1$	$8.0 \cdot 10^0$
8	$1.4 \cdot 10^5$	$5.7 \cdot 10^1$	$3.7 \cdot 10^4$	$8.5 \cdot 10^0$	$1.6 \cdot 10^2$	$2.7 \cdot 10^1$	$1.0 \cdot 10^1$
9	$1.8 \cdot 10^5$	$6.8 \cdot 10^1$	$4.6 \cdot 10^4$	$1.3 \cdot 10^1$	$2.0 \cdot 10^2$	$3.2 \cdot 10^1$	$1.3 \cdot 10^1$

For any of the numerical computation environments considered, using a vectorised version of the plain algorithm allows for encoding millions of ordinal patterns in milliseconds, without having to rely on pre-compiled external libraries. Most noteworthy, the vectorised NumPy/Python implementation actually outperformed the pre-compiled C code in the majority of cases. This hints at the amount of sophistication put into the development of the free and open-source NumPy/Python framework.

5.6.7 Implementing the Overlap Algorithm

The prerequisite for vectorising an algorithm is that all input data be available in advance, such that they can be passed to the software in parallel. Therefore, and by contrast with the plain algorithm considered above, recursive solutions like the overlap algorithm (Algorithm 3) cannot be fully vectorised, but inevitably require some sort of iteration. Due to the reasons given in Section 5.6.6, implementing the overlap algorithm in a numerical scripting language thus defeats its very purpose, which is the efficient evaluation of Equation (5.12). This can be demonstrated by benchmarking a MATLAB implementation of the overlap algorithm against a vectorised implementation of the plain algorithm (Algorithm 2), written in the same programming language. While the overlap algorithm should be superior in theory (see Section 5.6.1), a vectorised implementation of the plain algorithm may actually be faster under practical conditions, as can be observed in Figure 5.3. Reference code for both implementations is provided in the supplementary files `encode_overlap.m` and `encode_vectorised.m`.

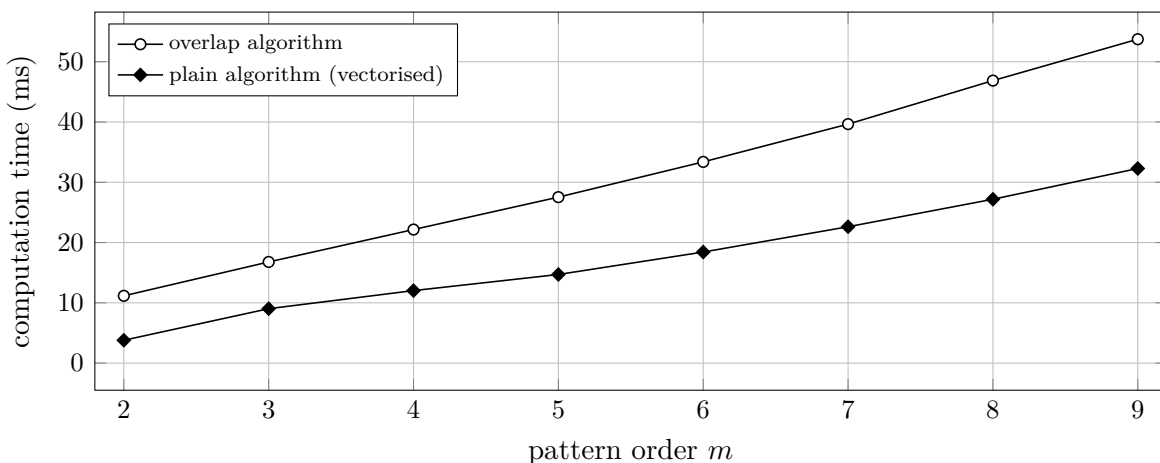


Figure 5.3 Computation time (median of 20 trials) for transforming $3.6 \cdot 10^5$ samples of uniform white noise into a sequence of ordinal patterns of order m . The lag was set to a constant $\tau = 1$, and the MATLAB functions `encode_vectorised` and `encode_overlap` from the supplementary files `encode_vectorised.m` and `encode_overlap.m` were used for the simulation.

To benefit from the overlap algorithm in terms of efficiency, it is therefore highly advisable to use a compiled programming language instead. In the following, we shall exclusively be concerned with implementing the overlap algorithm in the C programming language.

Assuming 64-bit integer support on the targeted platform, the pseudocode of Algorithm 3 can directly be translated into C code for any pattern order $m \leq 20$. Admittedly, a few minor tweaks are still possible, but those are easily understood from the reference implementation, that is, from the `ordpat_encode_overlap` function in the supplementary file `ordpat.c`. Typical run-time performances achieved by C implementations of the plain versus the overlap algorithm are depicted in Figure 5.4.

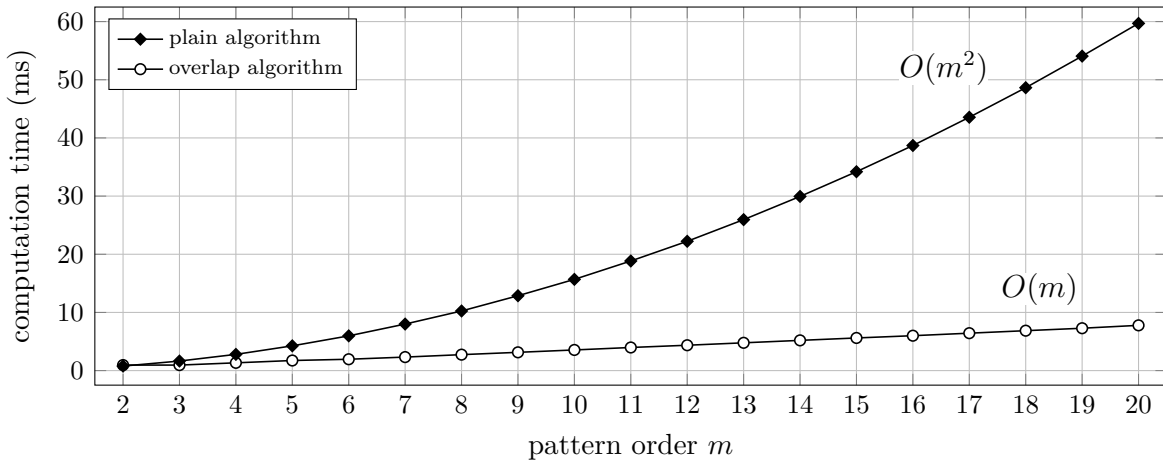


Figure 5.4 Computation time (median of 20 trials) taken for transforming $3.6 \cdot 10^5$ samples of uniform white noise into a sequence of ordinal patterns of order m . The time lag was set to $\tau = 1$, and the C functions `ordpat_encode_plain` and `ordpat_encode_overlap` from the supplementary file `ordpat.c` were used for the simulation. In line with theoretical expectation (see Section 5.6.1), the run-time complexity of the plain algorithm is $O(m^2)$, whereas the overlap algorithm scales at $O(m)$. Notice that for $m = 2$, there is no advantage over the plain algorithm: all order relations are then disjoint, such that no overlap can be exploited (see Section 4.2.3).

5.6.8 The Overlap Algorithm for High Pattern Orders

The aforementioned limitation to orders $m \leq 20$ is due to the relation $20! < 2^{64} < 21!$, which implies that ordinal patterns beyond $m = 20$ cannot be enumerated using 64-bit unsigned integers (see Section 5.6.2). Reconsidering the pseudocode of the overlap algorithm (Algorithm 3), however, it is easily confirmed that this constraint only affects the instruction

$$n_t \leftarrow (m - j)(n_t + r_{i,j}) \quad (5.21)$$

in line 27 of the listing. Specifically, if the variable n_t is limited to 64 bits of accuracy, it will eventually overflow for $m > 20$.

Fortunately, it merely takes (1) an arbitrary-precision integer representation for the variable n_t , (2) a function that adds a non-negative integer to n_t , and (3) a function that multiplies n_t with a non-negative integer to overcome this upper boundary. And although books have been written about arbitrary precision arithmetic [159, 160], a straightforward approach will fully satisfy the requirements of the present application. Let us therefore work through the above list one step at a time. Throughout the following, we shall assume 64-bit integer support on the target architecture.

Arbitrary-Precision Unsigned Integers

The maximum space required to store the numerical representation of an ordinal pattern π_i of order m amounts to $\log_2 m!$ bits, because $\text{enc}(\pi_i) \in \{0, 1, \dots, m! - 1\}$ for $\pi_i \in \Omega_m$. It is advantageous to keep the bit width constant across all patterns of a given order, and moreover, to allocate an integer multiple of the machine word size per pattern. Doing so avoids dynamic memory reallocation, and allows for iterating pattern sequences at a constant memory stride. Storing the numerical code $n = \text{enc}(\pi_i)$ of a single ordinal pattern $\pi_i \in \Omega_m$ then requires an array of $d = \lceil \log_2(m!)/64 \rceil$ unsigned 64-bit integers

$$(\nu_1, \nu_2, \dots, \nu_d) \in \{0, 1, \dots, 2^{64} - 1\}^d, \quad \text{such that} \quad n = \sum_{i=1}^d \nu_i \cdot 2^{64(i-1)}. \quad (5.22)$$

Given a sequence of N patterns of order m , its in-memory representation is then an array comprised of $d \times N$ unsigned 64-bit integer values, and each ordinal pattern is stored as a block of d consecutive integers. The resulting memory alignment is also visualised in Figure 5.5.

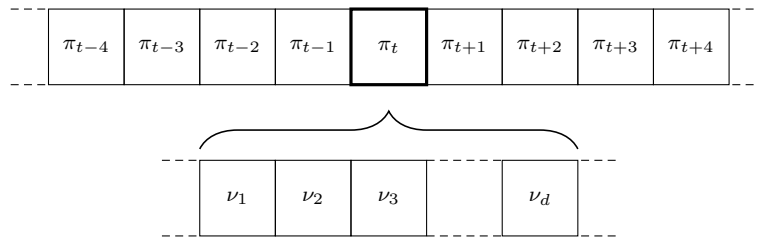


Figure 5.5 The numerical code $n = \text{enc}(\pi_t)$ of an ordinal pattern π_t of order m can be represented in memory by d consecutive 64-bit unsigned integers $(\nu_1, \nu_2, \dots, \nu_d)$, with their values set as described by Equation (5.22). The tuple length d is determined by the pattern order m , whereby it holds that $d = \lceil \log_2(m!)/64 \rceil$. Sequences of N ordinal patterns are stored by concatenation, which results in the patterns being aligned to $8d$ byte boundaries.

The Arbitrary-Precision Addition Operation

One part of the compound operation given by Equation (5.21) is an addition of the form

$$n \leftarrow n + r, \quad (5.23)$$

where n is the arbitrary-precision variable used to accumulate the numerical code of the ordinal pattern. Thus, it has to be stored by means of a tuple of 64-bit unsigned integers $(\nu_1, \nu_2, \dots, \nu_d)$ as described by Equation 5.22. By contrast, the variable r represents a right inversion count in terms of Equation (5.4), so it holds that $r \in \{0, 1, \dots, m - 1\}$, and one 64-bit unsigned integer variable is more than enough to store any practically relevant value of r . (Actually, 64 bits of memory are wildly overdimensioned in this context. Still, this bit width *is* required to implement arbitrary-precision multiplication, as will be described later on.)

Notice that the sum $n + r$ is directly written back to the variable n , and thus overwrites the previous value. Consequently, the addition operation can be implemented to act *in-place* on the operand n , or more specifically, on the block of memory $(\nu_1, \nu_2, \dots, \nu_d)$. From a strictly theoretical point of view, the following sequence of instructions will have the desired effect:

$$\begin{aligned} \nu_1 &\leftarrow (\nu_1 + r) \bmod 2^{64}, \\ r &\leftarrow \lfloor (\nu_1 + r) / 2^{64} \rfloor, \\ \nu_2 &\leftarrow (\nu_2 + r) \bmod 2^{64}, \\ r &\leftarrow \lfloor (\nu_2 + r) / 2^{64} \rfloor, \\ \nu_3 &\leftarrow (\nu_3 + r) \bmod 2^{64}, \\ r &\leftarrow \lfloor (\nu_3 + r) / 2^{64} \rfloor, \\ &\vdots \\ \nu_d &\leftarrow (\nu_d + r) \bmod 2^{64}, \\ r &\leftarrow \lfloor (\nu_d + r) / 2^{64} \rfloor. \end{aligned}$$

The above instructions basically reflect a “schoolbook” addition-with-carry, although performed in a numerical system to the base 2^{64} instead of the decimal system. Modulo

operations are required to limit the $\nu_i \in \{0, 1, \dots, 2^{64} - 1\}$ to their allowed value range, whereas floored divisions are used to obtain the respective carry values.

While this approach is formally correct, the situation is a lot different in practice, because any commonly used processing unit with 64-bit integer registers will truncate the result of $\nu_i + r$ to its least significant 64 bits, that is, the processor will effectively calculate $(\nu_i + r) \bmod 2^{64}$ in any case. Carry values can therefore not be obtained by evaluating expressions of the form $\lfloor (\nu_i + r)/2^{64} \rfloor$, because the intermediate result $\nu_i + r$ is unavailable. Thus, a different approach is required to calculate the carry values. In this respect, notice that due to $\nu_i \in \{0, 1, \dots, 2^{64} - 1\}$ and $r \in \{0, 1, \dots, 2^{64} - 1\}$, it holds that $0 \leq \nu_i + r \leq 2^{65} - 2$. Consequently,

$$\begin{aligned} \max\left(\left\lfloor (\nu_i + r)/2^{64} \right\rfloor\right) &= \left\lfloor (2^{65} - 2)/2^{64} \right\rfloor = \left\lfloor 2 - 2^{-63} \right\rfloor = 1, \\ \min\left(\left\lfloor (\nu_i + r)/2^{64} \right\rfloor\right) &= 0, \end{aligned}$$

from which it follows that

$$\forall (\nu_i, r) \in \{0, 1, \dots, 2^{64} - 1\}^2: \quad \left\lfloor (\nu_i + r)/2^{64} \right\rfloor \in \{0, 1\}.$$

In other words, the carry of any possible addition $\nu_i + r$ can be represented by a single bit, which is usually called the *carry flag*. Clearly, it further holds for this carry flag that

$$b_{\text{carry}} = \left\lfloor (\nu_i + r)/2^{64} \right\rfloor = \left[\nu_i + r \geq 2^{64} \right]. \quad (5.24)$$

Thus, the result of $\nu_i + r$ will either be smaller than 2^{64} (and therefore fit into a single 64-bit unsigned integer), or it will “wrap around” exactly one time under the modulo operation, such that

$$\nu_i \leftarrow (\nu_i + r) \bmod 2^{64} = \begin{cases} \nu_i + r, & \text{if } b_{\text{carry}} = 0, \\ \nu_i + r - 2^{64}, & \text{if } b_{\text{carry}} = 1. \end{cases}$$

Because it holds that $\nu_i + r \geq r$, whereas $\nu_i + r - 2^{64} < r$, the carry flag can be obtained by computing

$$b_{\text{carry}} = \left[(\nu_i + r) \bmod 2^{64} < r \right], \quad (5.25)$$

which, compared to the expressions given in Equation (5.24), does not require the full numerical value of $\nu_i + r$, but works with merely its 64 least significant bits.

On that basis, a computationally feasible version of the arbitrary-precision addition operation can be formulated in terms of the following sequence of instructions:

$$\begin{aligned}
\nu_1 &\leftarrow (\nu_1 + r) \bmod 2^{64}, \\
r &\leftarrow [\nu_1 < r], \\
\nu_2 &\leftarrow (\nu_2 + r) \bmod 2^{64}, \\
r &\leftarrow [\nu_2 < r], \\
\nu_3 &\leftarrow (\nu_3 + r) \bmod 2^{64}, \\
r &\leftarrow [\nu_3 < r], \\
&\vdots \\
\nu_d &\leftarrow (\nu_d + r) \bmod 2^{64}, \\
r &\leftarrow [\nu_d < r].
\end{aligned}$$

In practice, the iteration of those instructions can be terminated as soon as $r \leftarrow 0$. For instance, if $\nu_1 + r < 2^{64}$, no carry is required and the values (ν_2, \dots, ν_d) do not have to be touched at all. In summary, the arbitrary-precision addition operation can be implemented as described by Algorithm 5.

Algorithm 5. *The pseudocode below adds a 64-bit unsigned integer r to a multi-precision unsigned integer n . The latter is to be represented by an array of 64-bit unsigned integers $(\nu_1, \nu_2, \dots, \nu_d)$, such that $n = \sum_{i=1}^d \nu_i \cdot 2^{64(i-1)}$. The result of the addition is stored in-place, that is, $n \leftarrow n + r$ when the function returns. For reasons of run-time efficiency, no boundary checks are performed, so the caller has to make sure that $n + r < 2^{64}$.*

```

1 function add_mp
2 input
3      $n \equiv (\nu_1, \nu_2, \dots, \nu_d) \in \{0, 1, \dots, 2^{64} - 1\}^d$  with  $d \in \{1, 2, \dots\}$ 
4      $r \in \{0, 1, \dots, 2^{64} - 1\}$ 
5 begin
6      $i \leftarrow 1$ 
7     while  $r \neq 0$  do
8          $\nu_i \leftarrow (\nu_i + r) \bmod 2^{64}$  /* Exploit implicit truncation */
9          $r \leftarrow [\nu_i < r]$  /* Check if carry required */
10         $i \leftarrow i + 1$ 
11    end
12 end.

```

The Arbitrary-Precision Multiplication Operation

Besides the addition operation of Algorithm 5, an arbitrary-precision multiplication function is also required to evaluate the expression $n_t \leftarrow (m - j)(n_t + r_{i,j})$ given by Equation (5.21). The structure of this multiplication operation is

$$n \leftarrow c \cdot n, \quad (5.26)$$

where n is the arbitrary-precision variable storing the numerical code of the ordinal pattern. For the second operand, $c \in \{1, 2, \dots, m - 1\}$ holds true, as can be seen from the pseudocode of Algorithm 3. In analogy with the addition operation already discussed earlier, n will be stored in memory as a tuple of 64-bit unsigned integers $(\nu_1, \nu_2, \dots, \nu_d)$ in terms of Equation (5.22), that is

$$n = \sum_{i=1}^d \nu_i \cdot 2^{64(i-1)}.$$

The multiplication operation will be designed to act in-place on this block of memory. For reasons provided in the following, the second operand shall be restricted to the value range $c \in \{1, 2, \dots, 2^{32} - 1\}$, such that it can be represented by a single 32-bit unsigned integer variable. This choice allows for a maximum pattern order of $m = 2^{32}$, which—as of this writing—does not pose a limitation in practice.

With the operands c and $(\nu_1, \nu_2, \dots, \nu_d)$ defined as described above, the overall task then is to implement a map

$$(\nu_1, \nu_2, \dots, \nu_d) \mapsto (\nu'_1, \nu'_2, \dots, \nu'_d) \in \{0, 1, \dots, 2^{64} - 1\}^d,$$

such that

$$\sum_{i=1}^d \nu'_i \cdot 2^{64(i-1)} = \sum_{i=1}^d c\nu_i \cdot 2^{64(i-1)} = cn.$$

Notice that the obvious $\nu'_i = c\nu_i$ is not a solution here, because it violates the constraint that $\nu'_i < 2^{64}$.

At this point, let us recall a fundamental property of any binary multiplication operation: for an unsigned integer u_1 of d_1 bits width, and another such variable u_2 that is d_2 bits

wide, it holds that the width of their product $u_1 \cdot u_2$ does not exceed $d_1 + d_2$ bits, because

$$\max(u_1) \cdot \max(u_2) = (2^{d_1} - 1)(2^{d_2} - 1) < 2^{d_1+d_2} - 1.$$

In the present context, this means that the overall arbitrary-precision multiplication has to be constructed from sums of products of (at most) 32-bit wide unsigned integers, such that the result of each intermediate multiplication is guaranteed to fit into 64 bits of memory. In a first step, let us therefore split each of the 64-bit variables ν_i into a pair of 32-bit halfwords

$$\begin{aligned} \nu_{i,\text{high}} &= \lfloor \nu_i / 2^{32} \rfloor, \\ \nu_{i,\text{low}} &= \nu_i \bmod 2^{32}, \end{aligned} \quad \text{such that} \quad \nu_i = \nu_{i,\text{high}} \cdot 2^{32} + \nu_{i,\text{low}}.$$

The overall arithmetic value of the product cn can then be rewritten as

$$cn = \sum_{i=1}^d (c\nu_{i,\text{high}} \cdot 2^{32} + c\nu_{i,\text{low}}) \cdot 2^{64(i-1)}.$$

Notice that both $c\nu_{i,\text{high}}$ and $c\nu_{i,\text{low}}$ are guaranteed to fit into 64 bits of memory space, and are thus computable using 64-bit integer arithmetic. However, $c\nu_{i,\text{high}}$ has to be shifted by 32 bits to the right, so it is not aligned to a 64-bit boundary. To resolve this, $c\nu_{i,\text{high}}$ has to be split into a pair of 32-bit halfwords once more, which yields

$$c\nu_{i,\text{high}} = \lfloor c\nu_{i,\text{high}} / 2^{32} \rfloor \cdot 2^{32} + (c\nu_{i,\text{high}}) \bmod 2^{32},$$

and therefore allows for expressing the overall computation as

$$cn = \sum_{i=1}^d \left(\underbrace{\lfloor c\nu_{i,\text{high}} / 2^{32} \rfloor}_{< 2^{32}} \cdot 2^{64} + \underbrace{((c\nu_{i,\text{high}}) \bmod 2^{32})}_{< 2^{64}} \cdot 2^{32} + \underbrace{c\nu_{i,\text{low}}}_{< 2^{64}} \right) \cdot 2^{64(i-1)}. \quad (5.27)$$

All three summands in the above equation can then be computed using 64-bit integer arithmetic, and each is aligned to a 64-bit boundary. In particular, $\lfloor c\nu_{i,\text{high}} / 2^{32} \rfloor$ does not have to be multiplied by 2^{64} , but can simply be shifted up in memory by one 64-bit word. Based on this formulation, the arbitrary-precision multiplication can be performed as given by the following algorithm.

Algorithm 6. *The pseudocode below multiplies a 32-bit unsigned integer c with a multi-precision unsigned integer n , which is to be represented as an array of 64-bit unsigned integers $(\nu_1, \nu_2, \dots, \nu_d)$, such that $n = \sum_{i=1}^d \nu_i \cdot 2^{64(i-1)}$. The result is stored in-place, that is, $n \leftarrow cn$ when the function returns. For reasons of run-time efficiency, no boundary checks are performed, so the caller has to make sure that $cn < 2^{64d}$. The auxiliary function `add_mp` is specified by Algorithm 5.*

```

1 function multiply_mp
2 input
3      $(\nu_1, \nu_2, \dots, \nu_d) \in \{0, 1, \dots, 2^{64} - 1\}^d$  with  $d \in \{1, 2, \dots\}$ 
4      $c \in \{0, 1, \dots, 2^{32} - 1\}$ 
5 begin
6      $\nu_d \leftarrow (c \cdot \nu_d) \bmod 2^{64}$ 
7      $i \leftarrow d - 1$ 
8     while  $i \neq 0$  do
9          $w \leftarrow c \cdot \lfloor \nu_i / 2^{32} \rfloor$ 
10         $\nu_i \leftarrow c \cdot (\nu_i \bmod 2^{32})$ 
11         $w_l \leftarrow (w \bmod 2^{32}) \cdot 2^{32}$ 
12         $w_h \leftarrow \lfloor w / 2^{32} \rfloor$ 
13        add_mp $((\nu_i, \dots, \nu_d), w_l)$ 
14        add_mp $((\nu_{i+1}, \dots, \nu_d), w_h)$ 
15         $i \leftarrow i - 1$ 
16     end
17 end.

```

Some further remarks on this algorithm may be in order. Firstly, notice that $(\nu_1, \nu_2, \dots, \nu_d)$ is iterated in reverse order, starting with the most significant word ν_d , and descending to the least significant ν_1 . This is of crucial importance, because the `add_mp` functions in lines 13 and 14 overwrite (ν_i, \dots, ν_d) , but do not change $(\nu_1, \dots, \nu_{i-1})$. Furthermore, consider that many of the arithmetic operations in the above pseudocode can easily (and efficiently) be implemented using bitwise operations. In particular, for any $x \in \{0, 1, \dots, 2^{64} - 1\}$,

$$\begin{aligned}
 \lfloor x / 2^{32} \rfloor &\equiv \text{shift } x \text{ by 32 bits to the left,} \\
 (x \bmod 2^{32}) \cdot 2^{32} &\equiv \text{shift } x \text{ by 32 bits to the right,} \\
 x \bmod 2^{32} &\equiv \text{bitwise AND of } x \text{ and } 2^{32} - 1, \\
 x \bmod 2^{64} &\equiv \text{truncate } x \text{ to least significant 64 bits.}
 \end{aligned}$$

Of course, the latter is implicitly performed after any 64-bit integer arithmetic operation.

The Arbitrary-Precision Overlap Algorithm

Utilising the arbitrary-precision facilities derived in the above, the overlap algorithm can easily be extended to support pattern orders up to $m = 2^{32}$. To that end, the instruction

$$n_t \leftarrow (m - j)(n_t + r_{i,j})$$

in line 27 of Algorithm 3 is replaced by

$$\begin{aligned} (\nu_1, \nu_2, \dots, \nu_d) &\leftarrow \text{add_mp}((\nu_1, \nu_2, \dots, \nu_d), r_{i,j}), \\ (\nu_1, \nu_2, \dots, \nu_d) &\leftarrow \text{multiply_mp}((\nu_1, \nu_2, \dots, \nu_d), m - j). \end{aligned} \quad (5.28)$$

The functions `add_mp` and `multiply_mp` are described by Algorithm 5 and 6, respectively, whereas the d -tuple $(\nu_1, \nu_2, \dots, \nu_d)$ is the numerical code n_t of the ordinal pattern, mapped to the arbitrary-precision representation given by Equation (5.22). For reference, see the functions `add_mp` and `multiply_mp`, as well as the resulting multi-precision implementation of the overlap algorithm called `ordpat_encode_overlap_mp` (all to be found in the supplementary file `ordpat.c`). The performance of the arbitrary-precision approach as compared to the standard implementation is depicted in Figure 5.6 for the range of pattern orders supported by both variants.

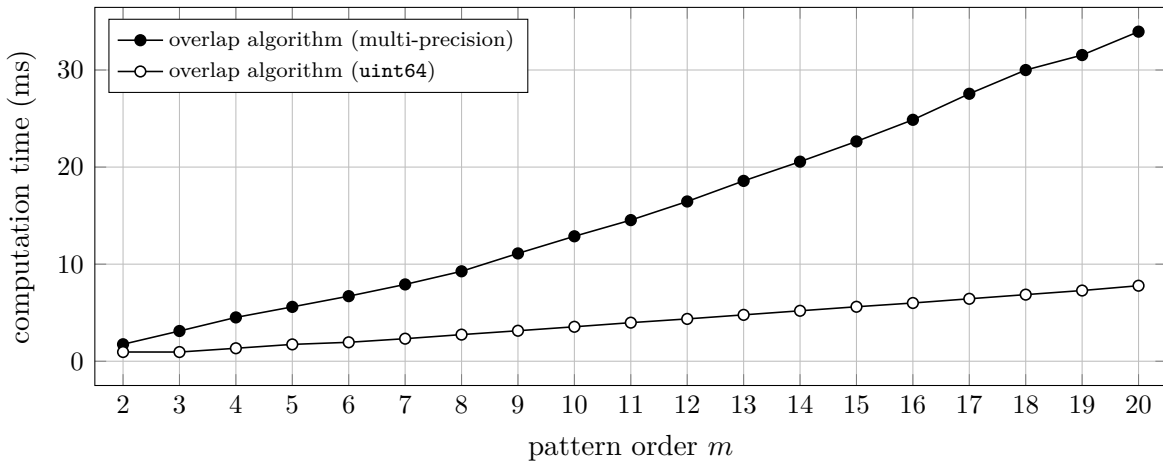


Figure 5.6 Computation time (median of 20 trials) for transforming $3.6 \cdot 10^5$ samples of uniformly distributed white noise into ordinal patterns of order m , using the time lag $\tau = 1$. The C functions `ordpat_encode_overlap` and `ordpat_encode_overlap_mp` from the supplementary file `ordpat.c` were used for the simulation. The arbitrary-precision arithmetic used in the `ordpat_encode_overlap_mp` function increases the overall run-time complexity, and the timing is less stable than for strictly hardware-based arithmetic operations.

As is to be expected, arbitrary-precision arithmetic introduces a certain performance penalty. Nevertheless, the extension enables the computation of ordinal patterns beyond the order $m = 20$, and it does so at reasonable speed. In this respect, Figure 5.7 visualises the run-time behaviour of the `ordpat_encode_overlap_mp` function for a wide range of pattern orders m .

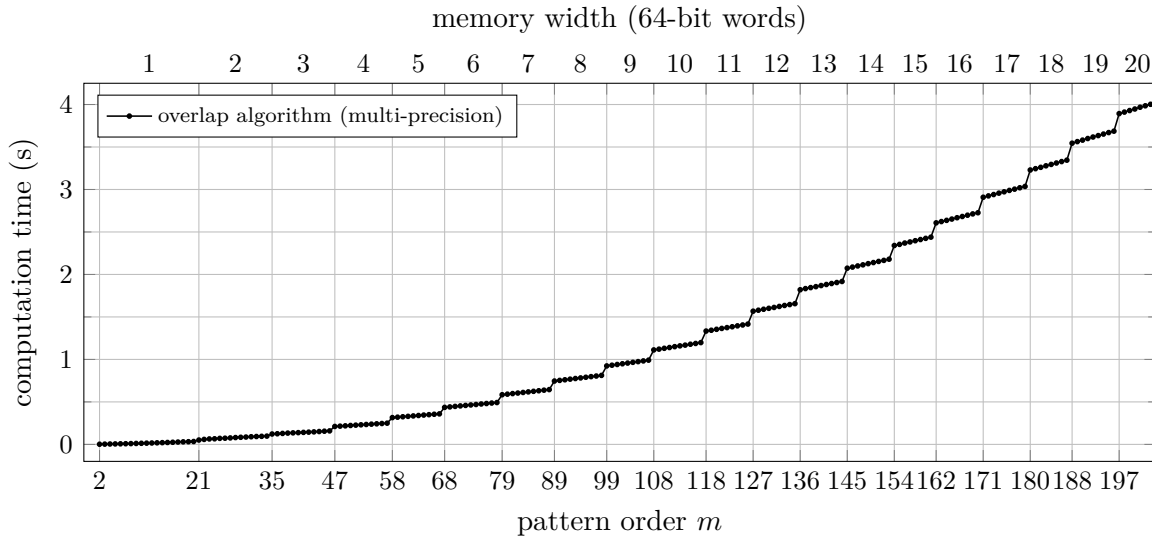


Figure 5.7 Computation time (median of 20 trials) for transforming $3.6 \cdot 10^5$ samples of uniformly distributed white noise into ordinal patterns of order m . The time lag was set to $\tau = 1$, and the function `ordpat_encode_overlap_mp` from the supplementary file `ordpat.c` was used for the simulation. The memory required per pattern is growing with m in a stepwise manner, increasing by one 64-bit word at each vertical grid line. The computational cost in turn rises linearly with the number of memory words to be iterated for each pattern, which shows as distinct jumps in the graph. Independent of that, the run-time complexity also increases linearly with the pattern order m as such. Both effects combined explain the parabolic envelope of the curve depicted.

As can be seen from Figures 5.6 and 5.7, some of the run-time efficiency of the overlap algorithm has to be traded off when allowing for pattern orders $m > 20$, which require multi-precision (and thus, multi-iteration) integer arithmetic. And although the overall computational complexity then scales with $O(m^2)$, the absolute run-time of the approach is still acceptable: encoding a one-hour sequence of data sampled at 100 Hz will take less than one second of processing time for orders as high as $m = 100$, where an inconceivable number of $100! \approx 9.3 \cdot 10^{157}$ distinct ordinal patterns do formally exist.

5.6.9 Implementing the Lookup Algorithm

In the original publication on the lookup algorithm (Algorithm 4), Unakafova and Keller proposed a MATLAB script as the reference implementation, and used this piece of code for performance benchmarks [145]. In analogy with the overlap algorithm (Algorithm 3), however, the lookup algorithm cannot be fully vectorised due to its recursive nature. In terms of run-time efficiency, numerical scripting languages are therefore at a considerable disadvantage compared to compiled programming languages. Due to this reason, and to enable meaningful comparison with the overlap algorithm, the lookup algorithm was here re-implemented in the C programming language (see the `ordpat_encode_lookup` function provided in the supplementary `ordpat.c` file). For the sake of completeness, native implementations for numerical scripting languages are still provided in the supplements, but are merely meant to allow the reader a quick confirmation of the aforementioned performance drop.

Theoretically speaking, the lookup algorithm is computationally more efficient than the overlap algorithm: for each and every pattern π_i to be encoded, the overlap algorithm has to calculate an integer representation $n = \text{enc}(\pi_i) \in \{0, 1, \dots, m! - 1\}$ from a tuple of inversion counts (r_1, r_2, \dots, r_m) , whereas the lookup algorithm can fetch the result of this operation from memory. By matter of principle, this reduces the number of computational operations to be executed per ordinal pattern (see Table 5.3).

In practice, however, the overall run-time of a piece of software is not exclusively determined by the number of operations it performs, but also depends on its memory requirements and its memory access patterns. In this regard, and as described in Section 5.5.3, Algorithm 4 requires a lookup table of $m! \times m$ elements, each holding the numerical representation of a particular ordinal pattern of order m . Thus, the size of the lookup table increases rapidly with the pattern order: conservatively assuming $\log_2 m!$ bits of storage space per pattern, the table size exceeds a gigabyte for $m = 11$, and occupies more than five terabytes of memory for the order $m = 14$. Consequently, memory latency quickly becomes prohibitive as the pattern order increases. Unakafova and Keller therefore stated that the applicability of their algorithm be limited to the pattern orders most commonly used [145], and provided precomputed lookup tables for $m \in \{2, 3, \dots, 9\}$.

Nevertheless, the principal problem of execution delays due to memory access times still persists for those lower pattern orders. In cases where the lookup table is too large

to entirely reside in the processor’s internal cache, the overall run-time efficiency of Algorithm 4 strongly depends on the nature of the input data. Time series of high ordinal complexity will then result in frequent cache misses. In other words, if the time series contains many different ordinal patterns, the processor will frequently have to reload different parts of the lookup table from main memory into cache, which stalls the processor and thus slows down the encoding process. This circumstance can be demonstrated by feeding low-pass filtered noise of increasing bandwidth (see Section 5.6.5) to the lookup algorithm, which yields results as presented in Figure 5.8.

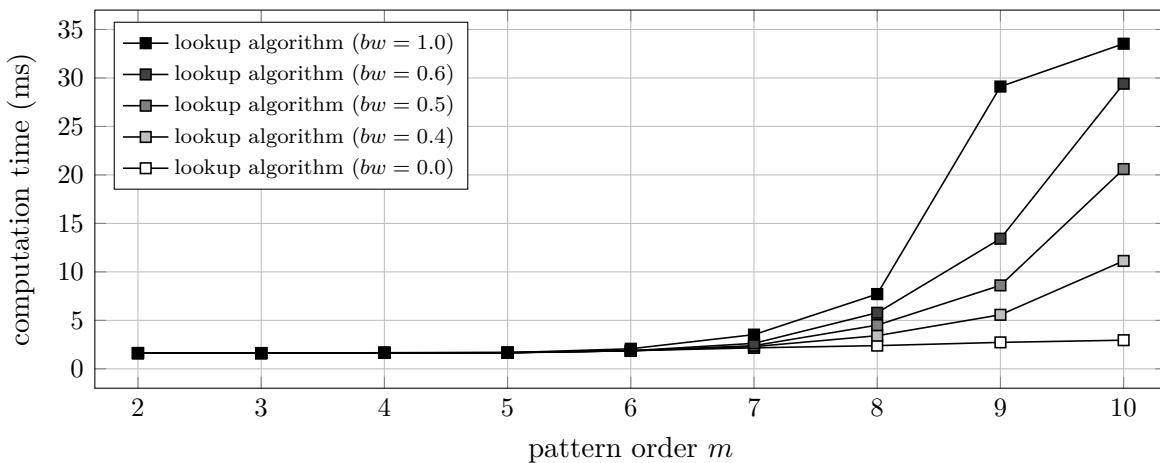


Figure 5.8 Computation time (median of 20 trials) for transforming $3.6 \cdot 10^5$ samples of uniform white noise, low-pass filtered to various relative bandwidths bw , into sequences of ordinal patterns of order m . The time lag was set to $\tau = 1$, and the function `ordpat_encode_lookup` from the supplementary file `ordpat.c` was used for the simulation. The time required for loading lookup tables from mass storage into main memory was not taken into account. The computation time increases not only with the pattern order m , but (for the most part) with the ordinal complexity of the input signal. (Also notice that filtering to $bw = 0.0$ results in an all-zero input signal, whereas $bw = 1.0$ results in white noise.)

It is therefore hard to draw a general conclusion on the run-time efficiency of the lookup algorithm. Suffice it to say that, for input data of low ordinal complexity, the lookup algorithm may outperform the overlap algorithm, as can be substantiated by using an all-zero time series as the test signal. In this idealised case, the algorithm will look up the exact same ordinal pattern again and again, and will therefore not run into cache contention issues. This is demonstrated in Figure 5.9.

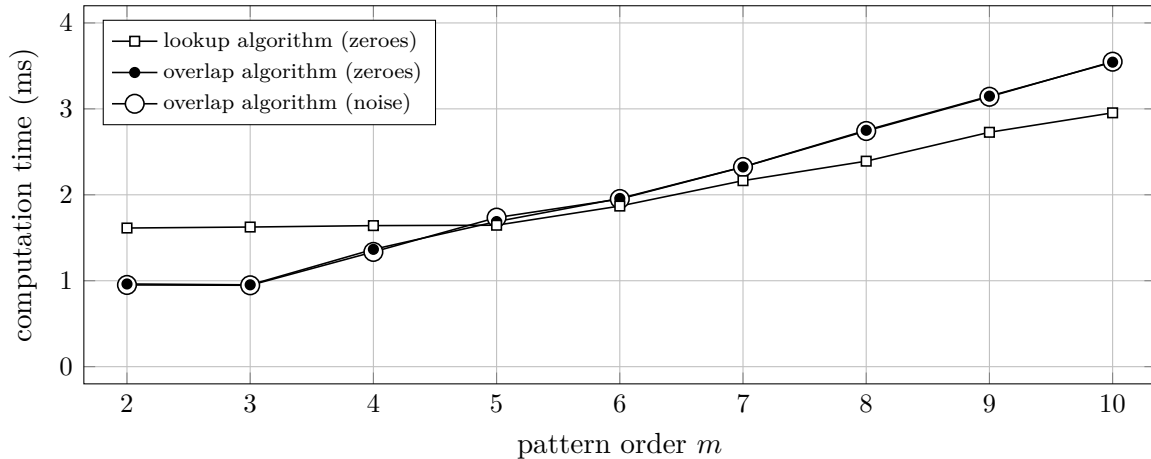


Figure 5.9 Computation time (median of 20 trials) for transforming $3.6 \cdot 10^5$ samples of data into sequences of ordinal patterns of order m . Either zeroes or uniform white noise were used as input data. The time lag was set to $\tau = 1$, and the functions `ordpat_encode_lookup` and `ordpat_encode_overlap` from the supplementary file `ordpat.c` were used for the simulation. The time required for loading lookup tables from mass storage into main memory was not taken into account. For an all-zero input signal, no cache contention occurs, and the lookup algorithm can outperform the overlap algorithm as the pattern order m (and thus, the computational workload for the overlap algorithm) increases.

Figure 5.9 also shows that the performance of the overlap algorithm is independent from the input data. It remains stable for both extreme cases: sequences of zeroes, as well as white noise. The benchmarks also convey the impression that the overlap algorithm may be at an advantage for $m \in \{2, 3, 4\}$ and any type of input signal. A possible explanation could be that addressing and accessing the lookup table in cache still imposes some constant delay, causing the processor’s execution pipeline to stall for those pattern orders with the lowest computational workload. Considering that under the above conditions both algorithms achieve a data throughput of more than 1 GB per second, this effect was not studied any further, though. In practice, loading input data from mass storage will likely take a lot longer than the actual processing times seen here.

5.6.10 Sequence Length and Time Lag

In the simulations presented so far, signals of a fixed $N = 3.6 \cdot 10^5$ samples length have been processed, using the constant time lag $\tau = 1$ throughout. A remaining question therefore is how the plain, overlap and lookup algorithms (Algorithms 2, 3 and 4) scale

with regard to the length N of the input sequence and the time lag τ under practical conditions. Fortunately, those aspects are qualitatively identical for all three algorithms, and their run-time behaviour is consistent with theoretical expectation: apart from the additional τ steps required to initialise the overlap and lookup algorithms, each of the algorithms is repeated once per ordinal pattern to be encoded, so doubling the amount of input data will coarsely double the computational effort. Additionally, the data to be encoded are iterated in a linear manner. Therefore, neither inordinate cache misses nor incorrect branch prediction should pose a problem in theory. Simulation confirms that the run-time of all three algorithms scales linearly with the sequence length, as can be observed in Figure 5.10.

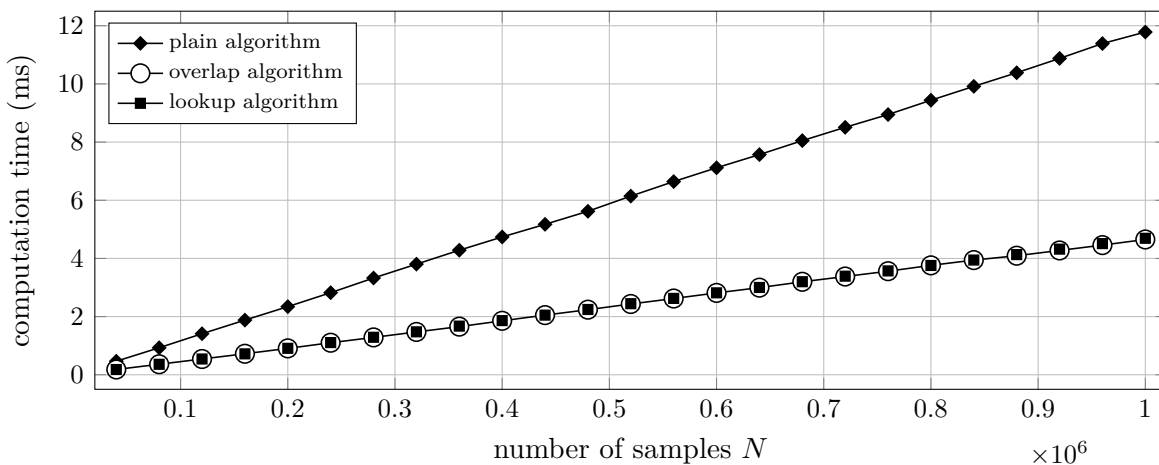


Figure 5.10 Computation time (median of 20 trials) for transforming N samples of uniform white noise into a sequence of ordinal patterns of order $m = 5$, using the time lag $\tau = 1$. The functions `ordpat_encode_plain`, `ordpat_encode_overlap` and `ordpat_encode_lookup` from the supplementary file `ordpat.c` were tested. The time required for loading lookup table data from mass storage into main memory was not taken into account. The order $m = 5$ was selected so as to operate the `ordpat_encode_lookup` function at its sweet spot with regard to cache utilisation. In good approximation, the computation time then increases linearly with the sequence length N for all three algorithms.

The relation between the time lag τ and the overall run-time is even simpler. With respect to computational effort, the time lag should not make any difference at all, because for all three algorithms, the value of τ is predominantly used to calculate memory addresses, where its absolute value cannot influence the computational workload. On the other hand, τ determines the stride of the (otherwise linear) memory access pattern. Therefore, increasing the time lag could theoretically be detrimental to the cache performance.

Under test conditions, however, the choice of τ had no noticeable influence on run-time efficiency—consider the measurements depicted in Figure 5.11.

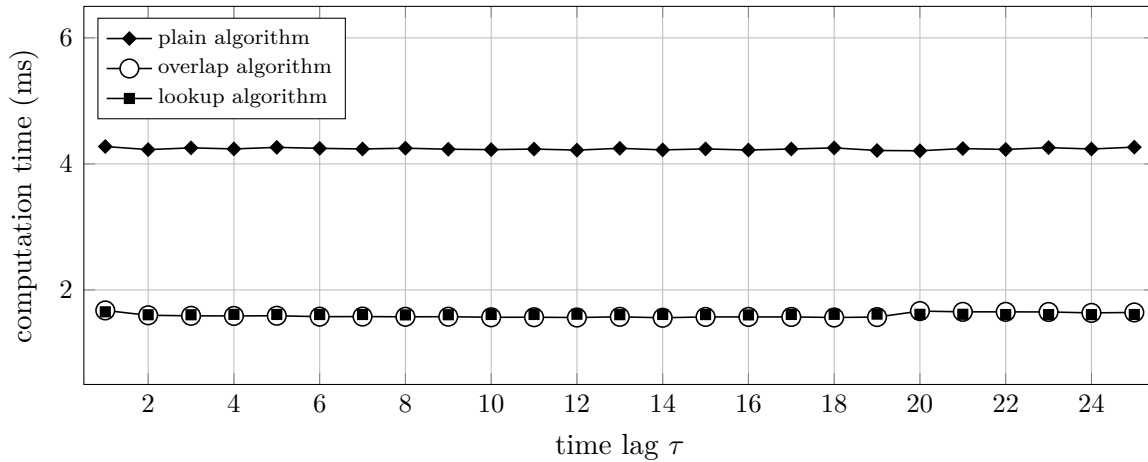


Figure 5.11 Computation time (median of 20 trials) for transforming $3.6 \cdot 10^5$ samples of uniform white noise into a sequence of ordinal patterns of order $m = 5$, using increasing time lags τ . The functions `ordpat_encode_plain`, `ordpat_encode_overlap` and `ordpat_encode_lookup` from the supplementary file `ordpat.c` were tested. The time required for loading lookup table data from mass storage into main memory was not taken into account. The order $m = 5$ was selected so as to operate the `ordpat_encode_lookup` function at its sweet spot with regard to cache utilisation. The simulations did not reveal any noticeable dependency between the time lag τ and the computation time.

5.7 Summary

The three algorithms discussed in this chapter all extract ordinal patterns from a given time series, and encode them in a computationally advantageous way, such that the patterns $\{\pi_1, \pi_2, \dots, \pi_m\}$ of order m are compactly represented by the set of non-negative integers $\{0, 1, \dots, m! - 1\}$ in a one-to-one manner. The theoretical foundations for this encoding were adopted from the Lehmer code [150], a classical approach in computational combinatorics (also see Section 5.2). From a theoretical perspective, the plain algorithm (Algorithm 2) has the highest computational complexity, followed by the overlap algorithm (Algorithm 3), and in turn followed by the lookup algorithm (Algorithm 4), which is the least computationally complex among the three (see Section 5.6.1). In practice, however, the algorithms presented are complementary with regard to their scope of application, and each can be worth considering.

Being fully vectorisable, the plain algorithm (Algorithm 2) is a particularly good choice for computational environments like MATLAB, GNU Octave or NumPy/Python, and its efficiency will likely suffice most standard applications of ordinal pattern analysis (see Table 5.6).

By contrast, the overlap algorithm (Algorithm 3) constitutes a general-purpose solution, providing high data throughput over a wide range of pattern orders m , while only requiring a small amount of extra memory. To achieve suitable run-time performance, it needs to be implemented in a compiled programming language, though. This is not an actual limitation in practice, because virtually any high-level scripting language can link against pre-compiled library functions. Under this paradigm of execution, the overlap algorithm clearly outperforms the plain algorithm. Implementing the overlap algorithm in the C programming language is straightforward, and provides plenty of opportunity for platform-specific optimisation: as demonstrated in Section 5.6.8, arbitrary-precision arithmetic can easily be incorporated to enable pattern orders $m > 20$, and (although not considered here) single-instruction-multiple-data (SIMD) processing could be employed to further boost the run-time performance on supporting architectures. With regard to real-time applications running on specialised embedded systems, it may be worth mentioning that the algorithm does not depend on floating-point arithmetic, and merely uses a few extra bytes of working memory on top of its input/output buffers.

As with the overlap algorithm, the lookup algorithm (Algorithm 4) should ideally be implemented in a compiled programming language to maximise its performance. By matter of principle, it has a narrower scope of application, though. Depending on a lookup table of $m! \times m$ elements, its memory requirements currently limit the algorithm to pattern orders $m \in \{2, 3, \dots, 10\}$. For the same reason, its run-time performance varies with the nature of the input data (see Figure 5.8). When analysing time series of comparatively low ordinal complexity, the lookup algorithm may outperform the overlap algorithm. On the other hand, time series of higher ordinal complexity will result in frequent cache misses, and may lead to a substantial drop in the overall run-time. Thus, if performance is critically important, both algorithms should be tested for the particular kind of data to be analysed.

6 Permutation Entropy of the Electroencephalogram

This chapter is based on previously published work by the same author [77].

6.1 Electroencephalography

Electroencephalography is the act of recording an electroencephalogram (EEG), which in turn is a collection of voltage signal traces originating from the neuroelectrical activity of the brain. The amplitudes of those signals are quite subtle, residing in the range of maximally a few hundred microvolts [161]. EEG is technically assessed by positioning electrodes in various locations of the subject's scalp (see Figure 6.1), and measuring the electrical voltage undulations of those electrodes over time.

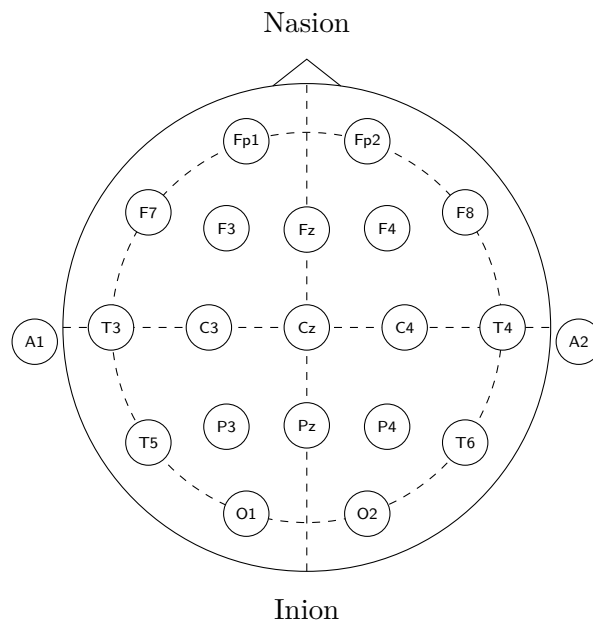


Figure 6.1 EEG electrode locations as defined by the *Ten Twenty Electrode System of the International Federation* [162], more commonly referred to as the *10–20 System*. Abbreviations used are: frontal pole (Fp), frontal (F), central (C), parietal (P), occipital (O), and auricular (A). Electrodes on the right (left) hemisphere are assigned even (odd) numbers, electrodes on the middle line are suffixed with the letter “z” (= zero).

6 Permutation Entropy of the Electroencephalogram

Differential amplifiers with high-impedance inputs and an appropriate amount of gain are used for this purpose. In the vast majority of cases, the amplified signals are then equidistantly sampled to transfer them to the digital domain. At this point, the data can conveniently be visualised, stored, and analysed. An exemplary multi-channel EEG recording is depicted in Figure 6.2.

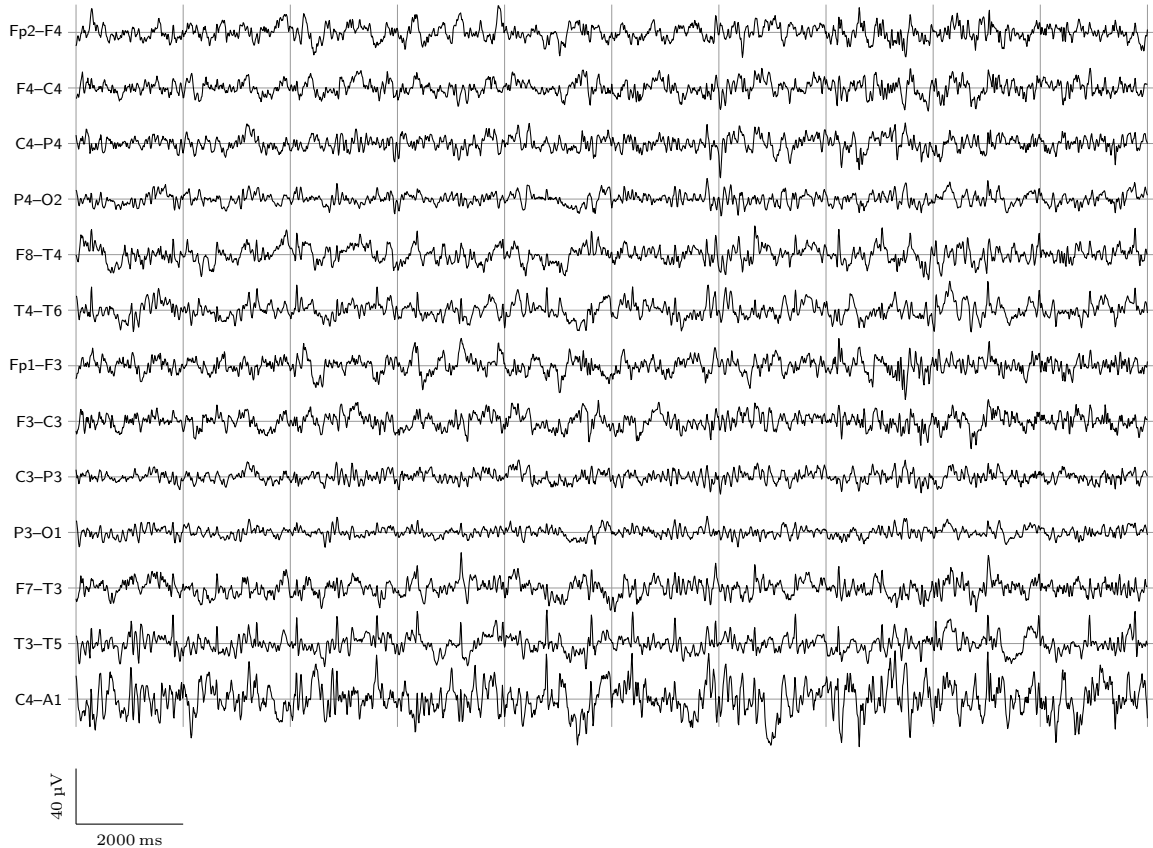


Figure 6.2 Multi-channel EEG recording. Each trace represents the voltage between a pair of electrodes (also see Figure 6.1). Data from the *CAP Sleep Database* [163], further described in Section 6.5.1, were used to create this figure.

Human EEG was for the first time recorded in the 1920s by Hans Berger [164], a German psychiatrist and strong believer in the existence of telepathy [165], who laid the grounds for electroencephalography in his well-known article *Über das Elektrenkephalogramm des Menschen* [166]. In particular, Berger found that a human subject, when lying still with eyes closed, would express rhythmical oscillations of approximately 10 Hz in the occipital EEG. He further found that those pronounced oscillations, later referred to by some as the “Berger rhythm” [167], would be replaced by electrical activity of higher frequency and lower amplitude as soon as the subject opened their eyes.

To this day, different EEG signals are most commonly distinguished by their predominant frequency content. Consistent with the conventions suggested by Berger, who called the aforementioned 10 Hz oscillations *alpha waves* and the higher-frequency oscillations *beta waves*, other frequency bands have later been assigned Greek letters of their own. This eventually lead to an arbitrary, but widely agreed upon separation of the EEG frequency spectrum in terms of delta (0.5–4 Hz), theta (4–8 Hz), alpha (8–12 Hz), beta (12–30 Hz), and gamma (> 30 Hz) activity [168].

The electrogenesis of the EEG—which tries to explain how physiological processes in different brain regions contribute to the generation of various forms of scalp electrical potentials—is a neuroscientific research field in its own right. Among other publications, the textbooks *Electric Fields of the Brain: The Neurophysics of EEG* by Paul Nunez and Ramesh Srinivasan [161], as well as *Rhythms of the Brain* by György Buzsáki [168] provide valuable insights on this field of study.

In the present work, EEG will be considered from a drastically more abstract perspective. In this regard, consider that the human brain can be seen as an extremely complex dynamical system with unknown dynamics, and endowed with a phase space of unobservable structure and dimensionality (see Chapter 3). In other words, the brain can be modelled as a *black box* in terms of systems theory. Based on this assumption, the EEG may be understood as a very low-dimensional projection of the underlying brain dynamics, and thus provides a window into those dynamics: for sure, the EEG will not reflect all the state transitions occurring in the brain, but on the other hand, any change in the EEG implies that its underlying dynamical system has progressed to some different state (also see Chapter 3). Being non-invasive, painless, cost-effective, and rather easily performed, this interrelation renders the EEG a very powerful tool.

6.1.1 Quantitative EEG Analysis

Electroencephalography is widely used in research and medical diagnostics. Regardless of its broad spectrum of applications, a common denominator of working with EEG is the assessment of motifs/patterns/features in the signals, and to relate them to observations from other modalities—the subject’s behaviour, for instance. The ultimate purpose of EEG analysis is the inference on cortical state changes, and such analyses are either performed by visual inspection, or supported by computer-based parameter extraction.

The latter is also called quantitative EEG analysis, and the general approach is usually as follows. Assume an EEG recording containing a total of k differential voltage traces (“channels”), all synchronously digitised at a constant rate of f_S samples per second, and over a recording duration of d seconds. Those data constitute a multivariate time series $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, where $N = \lfloor df_S \rfloor$, and

$$\forall t \in \{1, 2, \dots, N\}: \mathbf{x}_t = \begin{pmatrix} x_{t,1} & x_{t,2} & \cdots & x_{t,k} \end{pmatrix}^T \quad (6.1)$$

contains the voltage samples for all k channels at time index t .

Now, as a prerequisite for any meaningful analysis, pseudo-stationarity has to be assumed. In other words, the signal properties of the EEG are allowed (and often expected) to change over time, but are also required to remain reasonably stable within short time intervals Δd . The rationale is that an analysis window of Δd seconds duration can then be chosen, and the overall sequence of N data points can be segmented into overlapping subsequences, each containing $w = \lfloor \Delta df_S \rfloor$ consecutive vectors $\{\mathbf{x}_t, \mathbf{x}_{t+1}, \dots, \mathbf{x}_{t+w-1}\}$, whereby $t \in \{1, 2, \dots, N - w + 1\}$.

Based on this segmentation, a function of the form $f(\mathbf{x}_t, \mathbf{x}_{t+1}, \dots, \mathbf{x}_{t+w-1})$ can be applied to the data for each time index $t \in \{1, 2, \dots, N - w + 1\}$, which yields a new time series

$$\{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_{N-w+1}\}, \quad \text{where} \quad \mathbf{y}_t = f(\mathbf{x}_t, \mathbf{x}_{t+1}, \dots, \mathbf{x}_{t+w-1}). \quad (6.2)$$

Figuratively speaking, an analysis window of w samples length is moved along the EEG, and thus, the approach is often called the sliding-window technique (with maximum overlap, in the present case). The individual sequences of EEG resulting from this segmentation are also called EEG epochs. If the function f is chosen so as to represent a meaningful signal property, then each vector \mathbf{y}_t provides some information on the particular EEG epoch located in the time interval between t and $t + w - 1$. With regard to finding such meaningful signal properties, consider again that the EEG is a very low-dimensional projection of the underlying cerebral processes, and that its relation with higher cortical functions is not fully understood. Consequently, most techniques of quantitative EEG analysis are intrinsically empirical. Be it the traditional frequency band-related measures or more recent additions to the set of analysis tools: the mechanistic background of the vast majority of EEG parameters is not entirely known. In this inherently probabilistic framework, any given signal property that correlates with behavioural observations constitutes a parameter worth considering for EEG analysis.

6.2 Permutation Entropy as an EEG Parameter

One quantity that is receiving increasing attention in the field of quantitative EEG analysis is permutation entropy. It has been successfully used for EEG analyses in sleep scoring, general anaesthesia monitoring and research on disorders of the central nervous system, most notably epilepsy. Comprehensive overviews regarding its manifold applications have been given in Chapter 1, and can also be found in the review articles [144, 146]. Also, new applications of permutation entropy in EEG processing, and especially ways of extending the method are constantly reported on: for instance, “weighted” [108], “multiscale” [47], “multivariate multi-scale” [109], and “amplitude-aware” [110] variants of permutation entropy have so far been proposed. As literature suggests, the quantity may well be relevant to any research field that benefits from a widely accepted measure of the complexity of EEG.

Considering the great interest taken in permutation entropy, and given its virtually universal applicability in EEG analysis, a particular set of questions is surprisingly seldom addressed, though: What are the dynamic signal characteristics of EEG that permutation entropy responds to? Can the order m and time lag τ be selected on a phenomenological basis? And finally, how should the abstract notion of complexity be interpreted in EEG?

From a strictly outcome-oriented perspective, those may be minor issues. Nevertheless, a careful examination of permutation entropy in EEG might foster our general understanding of this class of electrophysiological signals. The present chapter takes a step in this direction, pioneering the following approach: instead of extending permutation entropy, try to simplify it without compromising its suitability for EEG analysis, and if successful, examine whether the result can more easily be interpreted than the initial chaos-theoretic complexity measure.

Credit for inspiring this idea is due to Bandt [16], who recently questioned the usage of terms like “complexity”, “chaos” and “disorder” in the EEG context and suggested that the permutation entropy of an epoch of EEG is equivalent to its distance from white noise—a simplification that immediately increases interpretability. In the following, we shall advance on this strategy, systematically deriving an intuitive explanation for the behaviour of permutation entropy in EEG.

6.2.1 Estimating Permutation Entropy from EEG

Consistent with Section 6.1.1, the usual approach of using permutation entropy as a quantitative EEG parameter is the following. Let $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ denote a k -channel EEG recording, comprised of N samples per channel. Then, for any suitable window length w , a sliding-window analysis in terms of Equation (6.2) yields the general result

$$\{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_{N-w+1}\}, \quad \text{where} \quad \mathbf{y}_t = f(\mathbf{x}_t, \mathbf{x}_{t+1}, \dots, \mathbf{x}_{t+w-1}).$$

In case of permutation entropy, the function f has to be defined to return a k -dimensional vector

$$\mathbf{y}_t = \left(y_{t,1} \quad y_{t,2} \quad \cdots \quad y_{t,k} \right)^\top,$$

where each element $y_{t,i}$ contains the (empirical) permutation entropy of the particular EEG epoch $\{x_{t,i}, x_{t+1,i}, \dots, x_{t+w-1,i}\}$. Given that permutation entropy is a univariate measure, let us drop the channel index i , and focus on a single EEG trace with signal windows in terms of $\{x_t, x_{t+1}, \dots, x_{t+w-1}\}$. (This is of course without loss of generality, but results in a simpler formalism.)

From each such segment, permutation entropy is obtained as has been described in Section 4.3. For a given order m and time lag τ , the EEG samples $\{x_t, x_{t+1}, \dots, x_{t+w-1}\}$ are first transformed into a sequence of ordinal patterns, then a probability vector

$$\mathbf{p}_t = \left(p_{t,1} \quad p_{t,2} \quad \cdots \quad p_{t,m!} \right)^\top$$

in terms of Equation (4.9) is estimated by counting the occurrences of each ordinal pattern $\pi_i \in \Omega_m$. Finally, the entropy $y_t = H(\mathbf{p}_t)$ as given by Equation (4.8) is computed.

By applying this procedure to all the overlapping segments of the EEG trace, a time series $\{y_1, y_2, \dots, y_{N-w+1}\}$ of permutation entropy values is obtained. This sequence reportedly (see [144, 146]) yields interesting correlations with behavioural observations: also see the introductory example given in Chapter 1.

6.3 The Dynamics of Permutation Entropy

Being an abstract complexity measure, the absolute value of permutation entropy obtained from a single epoch of EEG is quite insignificant. Much rather, the measure becomes

relevant for EEG analysis when entropies are compared across multiple EEG epochs, which is the purpose of the aforementioned sliding-window approach. Considerable contrast in value can then be observed, and be correlated with behavioural observations as is common practice. Let us call those decisive inter-epoch fluctuations the *dynamics* of permutation entropy. Their relation with the signal characteristics of EEG shall be further studied in the following. To that end, let us consider some generic properties of entropy, and derive implications for the dynamics of permutation entropy from those.

6.3.1 Probability Reallocation

In accordance with Section 6.2.1, let $\{y_t\}$ be a sequence of permutation entropy values, obtained from a trace of EEG samples $\{x_t\}$ using the pattern order m , the time lag τ , and a sliding window of length w with maximum overlap. Clearly, each of the entropy values y_t then relates to a probability vector

$$\mathbf{p}_t = \left(p_{t,1} \quad p_{t,2} \quad \cdots \quad p_{t,m!} \right)^\top, \quad \text{such that} \quad y_t = H(\mathbf{p}_t).$$

Further recall that, depending on the pattern order m and time lag τ , an analysis window of w samples length effectively spans a total of

$$\hat{w} = w - (m - 1)\tau \tag{6.3}$$

ordinal patterns, and that the probability estimate \mathbf{p}_t is based on counting pattern occurrences. In other words, when computing y_t , the vector $\hat{w}\mathbf{p}_t$ is obtained from the pattern sequence, and subsequently divided by \hat{w} to receive the probability vector \mathbf{p}_t , with $\|\mathbf{p}\|_1 = 1$.

Now observe that any pair of neighbouring probability vectors \mathbf{p}_t and \mathbf{p}_{t+1} are either identical, or differ in exactly two elements, whereby

$$p_{t+1,i} = p_{t,i} \pm \frac{1}{\hat{w}} \quad \text{and} \quad p_{t+1,j} = p_{t,j} \mp \frac{1}{\hat{w}}. \tag{6.4}$$

This is due to the fact that between the calculations of $y_t = H(\mathbf{p}_t)$ and $y_{t+1} = H(\mathbf{p}_{t+1})$, the analysis window is moved forward by one sample. Effectively, the leftmost ordinal pattern π_t is removed from the probability distribution \mathbf{p}_t , and the new rightmost ordinal pattern $\pi_{t+\hat{w}}$ takes its place in the “updated” distribution \mathbf{p}_{t+1} . It therefore holds that $\mathbf{p}_t = \mathbf{p}_{t+1}$ if $\pi_t = \pi_{t+\hat{w}}$, while otherwise, the relations given by Equation (6.4) apply.

Consequently, the dynamics of permutation entropy (that is, the value progression of the time series $\{y_t\}$) are governed in their entirety by a sequence of pairwise probability reallocations.

6.3.2 Probability Balance Coefficients

To understand how a probability reallocation in terms of Equation (6.4) affects the value of permutation entropy, a universal property of entropy can be utilised. To that end, consider that any pair of probabilities (p_i, p_j) from some probability vector \mathbf{p} contributes a fraction

$$p_i \log \frac{1}{p_i} + p_j \log \frac{1}{p_j}$$

to the overall entropy $H(\mathbf{p})$. Defining the shorthand $p_{ij} = p_i + p_j$, and introducing the notion of *probability balance coefficients* in terms of

$$\beta_{ij} = \begin{cases} p_i/p_{ij}, & \text{for } p_{ij} > 0, \\ 1/2, & \text{for } p_{ij} = 0, \end{cases} \quad \text{such that} \quad \beta_{ij} = 1/2 \iff p_i = p_j, \quad (6.5)$$

this contribution can be rewritten as a bivariate function

$$\begin{aligned} H_{\Delta}(p_{ij}, \beta_{ij}) &= p_i \log \frac{1}{p_i} + p_j \log \frac{1}{p_j} \\ &= p_i \log \frac{1}{p_{ij}} + p_i \log \frac{1}{\beta_{ij}} + p_j \log \frac{1}{p_{ij}} + p_j \log \frac{1}{\beta_{ji}} \\ &= p_{ij} \left(\log \frac{1}{p_{ij}} + \beta_{ij} \log \frac{1}{\beta_{ij}} + (1 - \beta_{ij}) \log \frac{1}{1 - \beta_{ij}} \right) \\ &= p_{ij} \left(\log \frac{1}{p_{ij}} + H_b(\beta_{ij}) \right), \end{aligned} \quad (6.6)$$

wherein H_b denotes the binary entropy function

$$H_b(\beta_{ij}) = \beta_{ij} \log \frac{1}{\beta_{ij}} + (1 - \beta_{ij}) \log \frac{1}{1 - \beta_{ij}} = \beta_{ij} \log \frac{1}{\beta_{ij}} + \beta_{ji} \log \frac{1}{\beta_{ji}}. \quad (6.7)$$

For any constant p_{ij} , the function $H_{\Delta}(p_{ij}, \beta_{ij})$ is an affine transformation of $H_b(\beta_{ij})$. Therefore, the behaviour of $H_b(\beta_{ij})$ fully determines how any probability redistribution among p_i and p_j affects the overall entropy value $H(\mathbf{p})$. As shown in Figure 6.3, $H_b(\beta_{ij})$ is a concave function, and is symmetrical around its maximum at $\beta_{ij} = 1/2$.

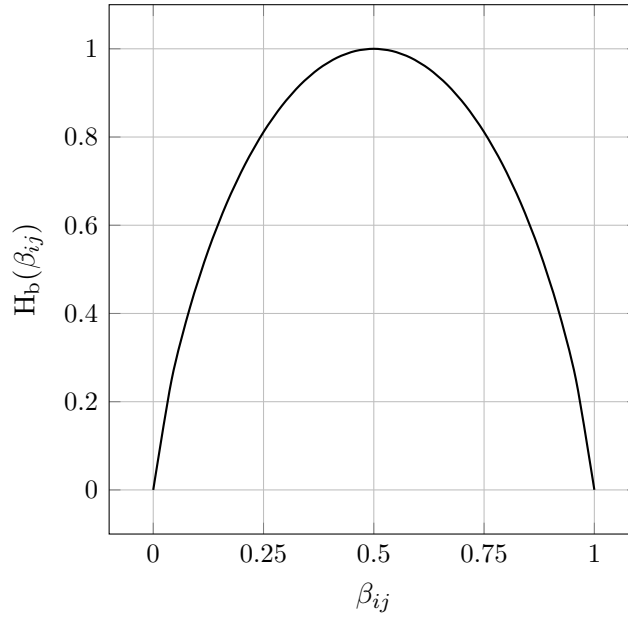


Figure 6.3 The binary entropy function of Equation (6.6), plotted using the binary logarithm.

Consequently, if the summed probability $p_{ij} = p_i + p_j$ is reallocated such that p_i and p_j approach their average, then β_{ij} approaches $1/2$ and the entropy increases. The same principle applies to any pair of probabilities drifting apart, where the entropy hence decreases. Two well-known corner cases of this property are that Shannon entropy is maximal for the uniform distribution, while it is zero if all but one p_i are zero [85].

For a probability distribution \mathbf{p} of n elements, a quadratic matrix of n^2 different balance coefficients

$$\begin{pmatrix} \beta_{11} & \cdots & \beta_{1n} \\ \vdots & \ddots & \vdots \\ \beta_{n1} & \cdots & \beta_{nn} \end{pmatrix} \quad (6.8)$$

can in principal be created. However, it is sufficient to consider the entries above its main diagonal, namely the subset

$$B = \{\beta_{ij} \mid 1 \leq i < j \leq n\}, \quad \text{such that} \quad |B| = \binom{n}{2} = \frac{n^2 + n}{2}, \quad (6.9)$$

because the remaining coefficients, β_{ij} with $i \geq j$, are easily obtained via the symmetry relation $\beta_{ji} = 1 - \beta_{ij}$. This also applies to the balance coefficients on the main diagonal, where $i = j$, and therefore, $\beta_{ij} = \beta_{ji} = 1/2$.

In conjunction with the relation $\|\mathbf{p}\|_1 = 1$, the coefficients $\beta_{ij} \in B$ constitute a system of $|B| + 1$ linear equations in the probabilities \mathbf{p} . Its augmented system matrix is

$$\left(\begin{array}{cccccc|c} 1 & 1 & 1 & 1 & \cdots & 1 & 1 \\ \beta_{12} - 1 & \beta_{12} & 0 & 0 & \cdots & 0 & 0 \\ \beta_{13} - 1 & 0 & \beta_{13} & 0 & \cdots & 0 & 0 \\ \beta_{14} - 1 & 0 & 0 & \beta_{14} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \beta_{n-1,n} & 0 \end{array} \right), \quad (6.10)$$

and Gaussian elimination confirms that the system is overdetermined but consistent. This means that any discrete probability distribution \mathbf{p} is unambiguously representable by a set of balance coefficients $\beta_{ij} \in B$, and in turn, that the entropy value $H(\mathbf{p})$ can be determined from those balance coefficients.

Permutation entropy of order m is the entropy $H(\mathbf{p})$ of an $m!$ -dimensional probability vector \mathbf{p} , and consequently, its dynamics can be studied by analysing variations within a collection of probability balances $\beta_{ij} \in B$, where $|B| = (m!)(m! + 1)/2$.

6.4 Complexity and Pseudo-Complexity

When used for EEG analysis, permutation entropy depends on a set of parameters and surrounding conditions. Besides the pattern order m and the time lag τ , also the sampling rate f_s , the window length w , as well as any filters applied to the EEG signals may affect the results. Those are a lot of parameters to decide upon, especially when considering the empirical nature of ordinal EEG analysis. Existing guidelines on parameter selection are essentially based on computational or statistical feasibility concerns [146], and are rather not motivated by (electro-)physiological considerations. Therefore, any parametrisation of permutation entropy reported suitable for EEG analysis must be regarded as the result of extensive experimentation.

With that in mind, it is remarkable that independent groups have successfully used the six ordinal patterns of order $m = 3$ for EEG encoding [169]. Most likely by experimental optimisation, investigators ended up using the lowest-possible order that still accounts for more than just the patterns “*the signal increases*” (**12**) versus “*the signal decreases*” (**21**).

Paradoxically, if six ordinal patterns suffice, most of what makes permutation entropy a complexity measure can apparently be omitted.

Accepting this immanent contradiction, one may raise the following questions: can the number of ordinal patterns be even further reduced without impairing the dynamics of permutation entropy in EEG? And if so, do the remaining patterns permit a more tangible interpretation of this analysis technique?

6.4.1 A New Class of Ordinal Patterns?

Regarding those questions, variegating the pattern order m will obviously not permit any further investigation. A finer-grained approach is necessary, and shall therefore be described in the following. It is based on the hypothetical idea that any pair of ordinal patterns may be the redundant bifurcation of a less specialised, yet more decisive kind of motif. By way of illustration, let us define an augmented class of ordinal patterns of order $m = 3$. For any tuple of pairwise distinct values (x_1, x_2, x_3) , our *extra-ordinal patterns* shall not only describe how its elements relate to one another, but additionally encode the value of the Boolean expression

$$(x_{\text{hi}} - x_{\text{mid}}) \geq (x_{\text{mid}} - x_{\text{lo}}). \quad (6.11)$$

Figuratively, we assess if the centroid of a pattern lies above or below its equatorial axis, and consistently, we shall call the respective pattern either *top-heavy* or *bottom-heavy*. The principle is visualised in Figure 6.4.

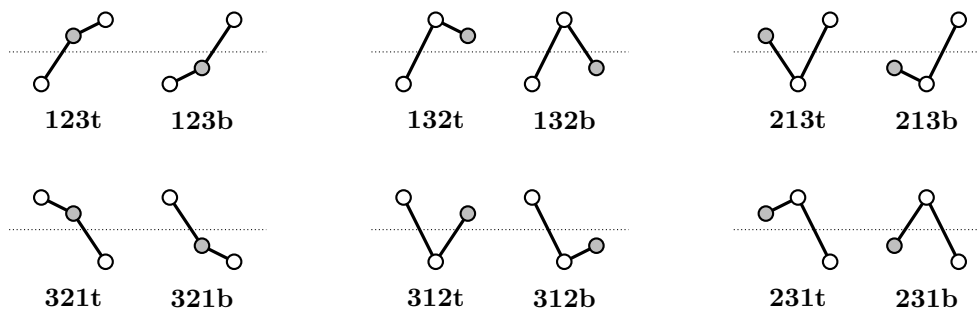


Figure 6.4 The twelve extra-ordinal patterns of order $m = 3$. Highlighted samples render patterns either top-heavy (**t**) or bottom-heavy (**b**). Each pattern pair π_i^t and π_i^b descends from a common ordinal pattern π_i .

As an example, the tuple (23, 11, 14) features the bottom-heavy pattern **312b**, while the tuple (28, 6, 17) maps onto a top-heavy **312t**. Both are derivatives of the same conventional ordinal pattern **312**, though, and it holds for their probabilities that $p_{\mathbf{312}} = p_{\mathbf{312t}} + p_{\mathbf{312b}}$. More generally speaking, a pair of extra-ordinal patterns π_i^t and π_i^b exists for each ordinal pattern π_i , and their probabilities relate to each other as in

$$p_i = p_{ti} + p_{bi}. \quad (6.12)$$

In practice, calculating the hypothetical *extra-ordinary permutation entropy* differs only marginally from the standard approach: probability distribution estimates for $2 \cdot m! = 12$ distinct patterns have to be obtained.

6.4.2 Pseudo-Complexity in Ordinal Pattern Analysis

Apart from nomenclature, extra-ordinary permutation entropy is a reasonable extension in theory. Using conventional permutation entropy of order $m = 3$, we implicitly agree to distinguish two upward-peak patterns (**132** and **231**) as well as two downward-peak patterns (**213** and **312**), so by implication, the additional distinction between top-heavy and bottom-heavy patterns is equally justified: both kinds of partitions stem from the same principle, and the degree of granularity that better matches the signal characteristics of EEG is not evident—because ultimately, the relations between EEG and ordinal patterns are not understood.

Understood are, however, the basic properties of entropy. According to Equation (6.6), and in conjunction with Equation (6.12), it holds for any pair of extra-ordinal patterns π_i^t and π_i^b that their summed contribution to the extra-ordinary permutation entropy is

$$H_{\Delta}(p_i, \beta_i) = p_{ti} \log \frac{1}{p_{ti}} + p_{bi} \log \frac{1}{p_{bi}} = p_i H_b(\beta_i) + p_i \log \frac{1}{p_i}, \quad (6.13)$$

where β_i is the probability balance coefficient $\beta_i = p_{ti}/(p_{ti} + p_{bi}) = p_{ti}/p_i$. By estimating the variance of a particular β_i in a suitably large collection of EEG epochs, the impact of the corresponding pattern pair π_i^t and π_i^b on overall entropy can be quantified. As an example as intuitive as unlikely, imagine that the variance of some balance coefficient β_i is found to be zero, such that β_i and $H_b(\beta_i)$ remain constant for any epoch of EEG. Equation (6.13) then implies that π_i^t and π_i^b can merely contribute to the dynamics (= value changes) of entropy via the sum of their probabilities, which is the probability

p_i of the conventional ordinal pattern π_i . In this case, the distinction between π_i^t and π_i^b is redundant in the first place: without impairing the dynamics, the respective patterns π_i^t and π_i^b can be merged into their common ancestor, the traditional ordinal pattern π_i .

More realistically, any balance coefficient will likely feature some variance. Nevertheless, by estimating the variances of all the coefficients $\beta_{ij} \in B$ given by Equation (6.9), the corresponding pattern pairs can be ranked by their relative contribution to the overall dynamics. In this way, one can separate decisive pattern pairs from redundant ones, and tell apart the complexity of the data from the pseudo-complexity of the method.

Admittedly, if the lengths of the EEG epochs analysed suffice for accurate probability estimation, it is not necessarily detrimental to use more ordinal patterns than the underlying signal characteristics demand for. However, doing so conveys the impression of a more intricate phenomenon than there might actually be, and could hence promote the misinterpretation of analysis results.

6.5 The Entropy of Peaks

The principle outlined above is immediately transferable to *actual* permutation entropy. As a starting point, let us consider the pattern order $m = 3$, and the time lag $\tau = 1$. Extensions for higher orders and time lags will be discussed later on. Because six different ordinal patterns of order $m = 3$ exists, their probability distribution is of the form

$$\mathbf{P} = \left(p_{123} \quad p_{132} \quad p_{213} \quad p_{231} \quad p_{312} \quad p_{321} \right)^T, \quad (6.14)$$

and their corresponding balance coefficients are

$$\begin{pmatrix} \beta_{123/123} & \beta_{123/132} & \beta_{123/213} & \beta_{123/231} & \beta_{123/312} & \beta_{123/321} \\ \beta_{132/123} & \beta_{132/132} & \beta_{132/213} & \beta_{132/231} & \beta_{132/312} & \beta_{132/321} \\ \beta_{213/123} & \beta_{213/132} & \beta_{213/213} & \beta_{213/231} & \beta_{213/312} & \beta_{213/321} \\ \beta_{231/123} & \beta_{231/132} & \beta_{231/213} & \beta_{231/231} & \beta_{231/312} & \beta_{231/321} \\ \beta_{312/123} & \beta_{312/132} & \beta_{312/213} & \beta_{312/231} & \beta_{312/312} & \beta_{312/321} \\ \beta_{321/123} & \beta_{321/132} & \beta_{321/213} & \beta_{321/231} & \beta_{321/312} & \beta_{321/321} \end{pmatrix}. \quad (6.15)$$

As discussed in Section 6.3.2, it suffices to consider the matrix entries above the main diagonal, that is, the coefficients β_{ij} with $1 \leq i < j \leq 6$. Those 15 elements shall be

aggregated in a vector of balance coefficients

$$\boldsymbol{\beta} = \left(\beta_{123/132} \quad \beta_{123/213} \quad \beta_{123/231} \quad \cdots \quad \beta_{231/312} \quad \beta_{231/321} \quad \beta_{312/321} \right)^{\top}. \quad (6.16)$$

For reasons of simplicity, linear indexing (that is, the notation β_i with $1 \leq i \leq 15$) will henceforth be used as an alternative means of notation when referring to the individual elements of the vector $\boldsymbol{\beta}$.

6.5.1 An Open-Source, Open-Data Approach

The statistics of balance coefficients in EEG have to be assessed empirically, that is, from a suitably large collection of data. The analyses presented here were carried out on the *CAP Sleep Database* [163], a set of 108 polysomnographic recordings conducted at the Ospedale Maggiore di Parma, and kindly dedicated to the public domain. Sleep EEG is particularly suitable for the task at hand, because permutation entropy reportedly varies for different stages of natural sleep [15, 16], and polysomnographic recordings usually encompass hours of contiguous data per subject. Moreover, a multitude of pathologies justify polysomnographic clarification, which increases the heterogeneity of the data—another plus for the present use case.

To aid reproducibility, using a dataset available to all of the scientific community was considered obligatory. The choice for the CAP Sleep Database was motivated by Bandt's recent publication on using ordinal patterns for sleep stage classification [16]. The dataset is hosted by PhysioNet [170], and provided free of charge. Once again in the interest of reproducible science [153], the free and open-source numerical computation environment GNU Octave 4.2 was utilised for data analysis. The code is provided in the supplements of [77], and was tested compatible with MATLAB, versions 2015b and above.

6.5.2 EEG Segmentation and Processing

The CAP Sleep Database is a collection of 108 files in European Data Format (EDF) [171]. Four of them (`brux1.edf`, `n13.edf`, `n14.edf`, and `narco3.edf`) had to be rejected due to apparent inconsistencies in their file headers. From the remaining polysomnograms, all channels containing EEG were selected, and converted to a sampling rate of $f_s = 200$ Hz. No additional preprocessing, nor any artefact correction were applied. The resampled EEG traces were split into non-overlapping epochs of 20 s duration, that is, into sequences

of $w = 4000$ samples each. To avoid inadvertent biasing of the pattern distributions, epochs containing ties (triples of values violating the constraint $x_1 \neq x_2 \neq x_3$) were discarded. The procedure yielded a total of $N_{\text{cap}} = 1.6 \cdot 10^6$ signal segments, that is, more than one year of single-channel EEG data.

Using the order $m = 3$ and time lag $\tau = 1$, the epochs were then mapped onto sequences of ordinal patterns. From each of the resulting pattern sequences, a probability vector

$$\mathbf{p}_i = \left(p_{123i} \ p_{132i} \ p_{213i} \ p_{231i} \ p_{312i} \ p_{321i} \right)^{\top} \quad (6.17)$$

was estimated, and permutation entropy was subsequently calculated. All entropy values were divided by $\log 3!$ for normalisation. As is to be expected, the sleep EEG data feature a wide range of permutation entropy values. Their non-trivial distribution is depicted in Figure 6.5.

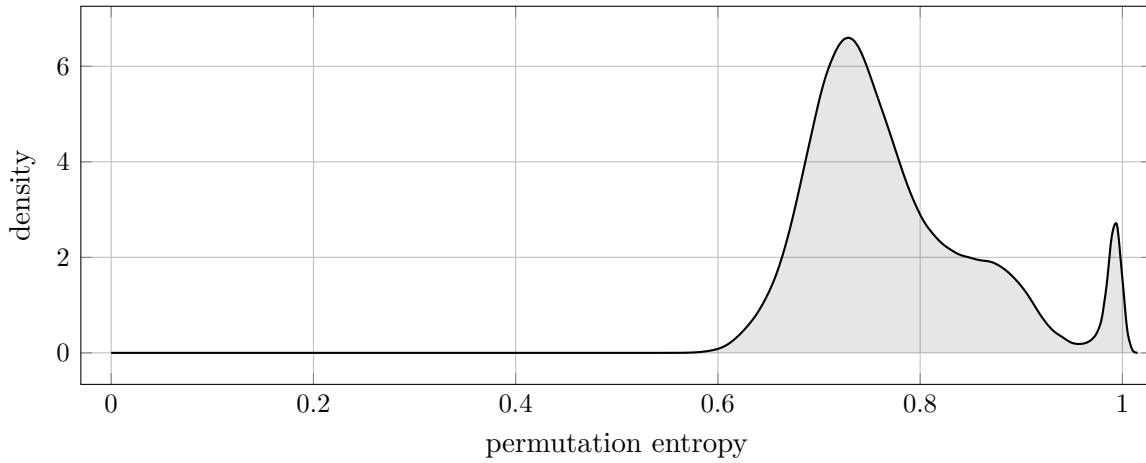


Figure 6.5 Kernel density estimate (Gaussian, $\sigma = 5 \cdot 10^{-3}$) for the $N_{\text{cap}} = 1.6 \cdot 10^6$ normalised permutation entropy values (order $m = 3$, time lag $\tau = 1$) obtained from the CAP Sleep Database.

6.5.3 Principle Components of the Probability Balances

Furthermore, a vector of balance coefficients β_i as per Equation (6.16) was computed for each ordinal pattern distribution \mathbf{p}_i . For notational purposes, let us concatenate these vectors into a $N_{\text{cap}} \times 15$ data matrix

$$\mathbf{B} = \left(\beta_1 \ \beta_2 \ \dots \ \beta_{N_{\text{cap}}} \right)^{\top}. \quad (6.18)$$

Interpreting each vector β_i (each row in \mathbf{B}) as one point in a 15-dimensional feature space, principle component analysis (PCA) was then carried out on the data matrix. PCA constitutes a perfect match for the problem at hand, given that balance coefficients shall be ranked by their contribution to the variance of the data matrix \mathbf{B} .

Let us notate PCA in terms of computing the set of eigenpairs

$$\{(\lambda_i, \mathbf{w}_i) \mid 1 \leq i \leq 15\} \quad (6.19)$$

to the 15×15 covariance matrix $\Sigma_{\mathbf{B}}$ of the data matrix \mathbf{B} . The eigenvectors \mathbf{w}_i then represent the loading vectors of the PCA, and map \mathbf{B} onto its principle components

$$\mathbf{z}_i = \mathbf{B}\mathbf{w}_i. \quad (6.20)$$

Each principle component \mathbf{z}_i is thus a linear combination of the balance coefficients β_i , and explains a fraction of the variance in the dataset \mathbf{B} . More specifically, it holds that $\lambda_i = \text{Var}(\mathbf{z}_i)$, and it is therefore common practice to enumerated the principle components according to $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_{15}$, such that \mathbf{z}_1 contributes most to the variance in \mathbf{B} .

For the actual CAP dataset, PCA revealed that

$$\frac{1}{\lambda_0} \sum_{i=1}^4 \lambda_i > 0.999, \quad \text{where} \quad \lambda_0 = \sum_{i=1}^{15} \lambda_i.$$

With more than 99.9% of the variance in \mathbf{B} being due to \mathbf{z}_1 , \mathbf{z}_2 , \mathbf{z}_3 , and \mathbf{z}_4 (see Table 6.1), all further considerations were limited to those four principal components. Their kernel density estimates, depicted in Figure 6.6, are highly instructive.

Table 6.1 Eigenvalues and explained variations of the first four principle components, and their Spearman correlation coefficients with permutation entropy (PeEn).

Component	Eigenvalue	Explained Variation	Spearman Correlation
\mathbf{z}_i	λ_i	λ_i/λ_0	$\rho(\text{PeEn}, \mathbf{z}_i)$
\mathbf{z}_1	$5.0 \cdot 10^{-2}$	94.4%	0.99996
\mathbf{z}_2	$1.5 \cdot 10^{-3}$	2.8%	-0.006
\mathbf{z}_3	$1.2 \cdot 10^{-3}$	2.2%	-0.008
\mathbf{z}_4	$2.5 \cdot 10^{-4}$	0.5%	0.02

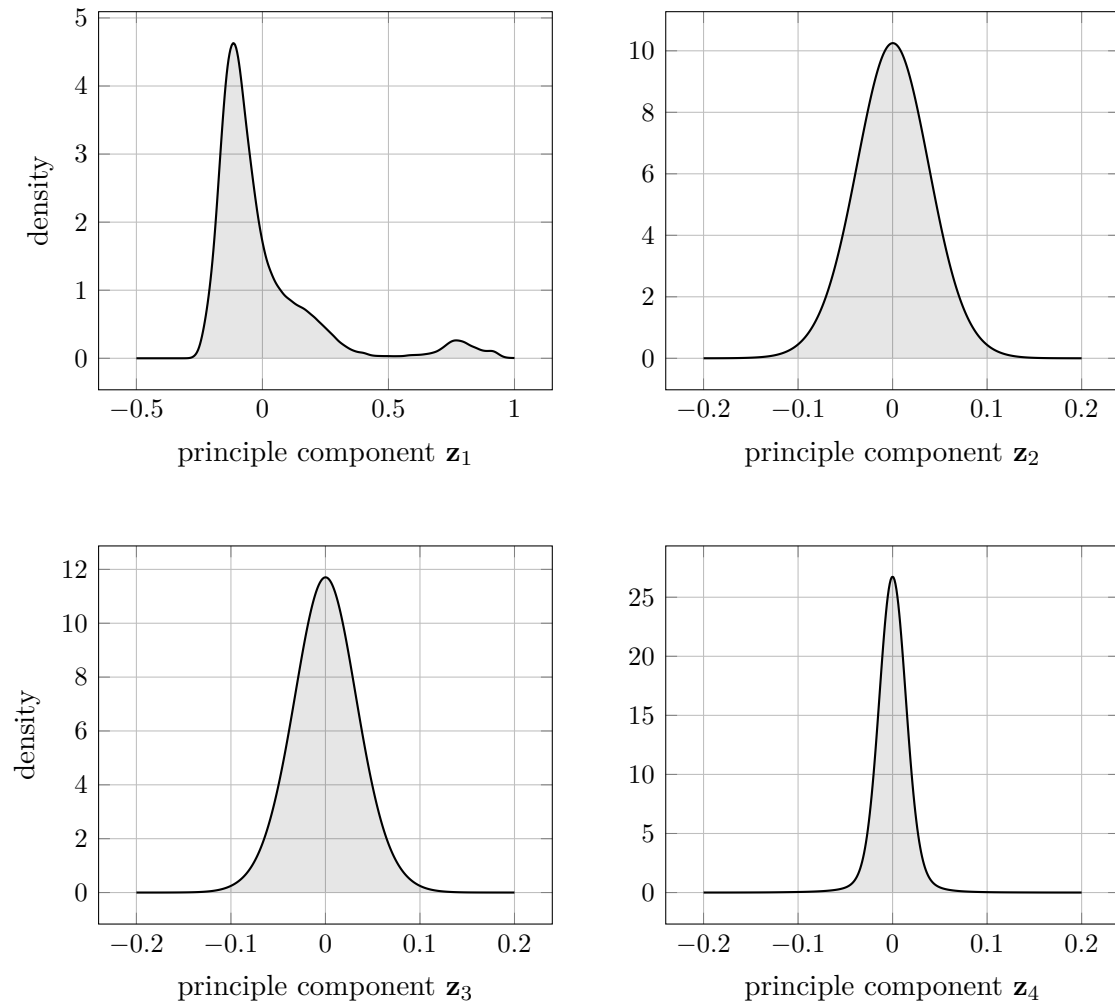


Figure 6.6 Kernel density estimates (Gaussian, $\sigma = 1 \cdot 10^{-2}$) for the first four principle components \mathbf{z}_1 , \mathbf{z}_2 , \mathbf{z}_3 and \mathbf{z}_4 of the data matrix \mathbf{B} .

Notice that the density of \mathbf{z}_1 resembles the distribution of permutation entropy values depicted in Figure 6.5, suggesting some correlation between the two. By contrast, the density estimates of \mathbf{z}_2 , \mathbf{z}_3 and \mathbf{z}_4 seem to be normally distributed around zero, that is, they appear to be random noise.

Testing for correlation between the permutation entropy values and the principle components \mathbf{z}_1 , \mathbf{z}_2 , \mathbf{z}_3 , and \mathbf{z}_4 , respectively, Spearman correlation coefficients as reported in Table 6.1 were obtained. Consistent with Figures 6.5 and 6.6, the first principle component \mathbf{z}_1 is almost perfectly correlated with permutation entropy, while \mathbf{z}_2 , \mathbf{z}_3 , and \mathbf{z}_4 clearly contain uncorrelated noise. Moreover, \mathbf{z}_1 alone explains more than 94% of the

variance in \mathbf{B} . It is therefore safe to assume that the dynamics of permutation entropy are exclusively driven by the component \mathbf{z}_1 .

6.5.4 Eliminating Pseudo-Complexity

The nature of the first principle component's loading vector \mathbf{w}_1 is *the* pivotal result of the current chapter. In accordance with the data given in Table 6.2, it holds in very good approximation that

$$\frac{\mathbf{z}_1}{w_{1,1}} = \frac{\boldsymbol{\beta}^\top \mathbf{w}_1}{w_{1,1}} \cong \begin{pmatrix} \beta_{123/132} \\ \beta_{123/213} \\ \beta_{123/231} \\ \beta_{123/312} \\ \beta_{132/321} \\ \beta_{213/321} \\ \beta_{231/321} \\ \beta_{312/321} \end{pmatrix}^\top \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \\ -1 \end{pmatrix} = \begin{pmatrix} \beta_{123/132} \\ \beta_{123/213} \\ \beta_{123/231} \\ \beta_{123/312} \\ \beta_{321/132} \\ \beta_{321/213} \\ \beta_{321/231} \\ \beta_{321/312} \end{pmatrix}^\top \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}. \quad (6.21)$$

The above is a bubble of pseudo-complexity bursting: while the analysis was designed to isolate *some* pattern pairs that are irrelevant for permutation entropy in EEG, Equation (6.21) implies that all of the probability balances among the pairs

$$\Omega_\dagger = \left\{ (\mathbf{123}, \mathbf{321}), (\mathbf{132}, \mathbf{213}), (\mathbf{132}, \mathbf{231}), (\mathbf{132}, \mathbf{312}), \right. \\ \left. (\mathbf{213}, \mathbf{231}), (\mathbf{213}, \mathbf{312}), (\mathbf{231}, \mathbf{312}) \right\} \quad (6.22)$$

contribute virtually nothing to the variance in \mathbf{z}_1 , and are thus negligible in good approximation. Those are the balances between any two patterns containing a peak, as well as the balance $\beta_{123/321}$ between the monotonously rising and the monotonously falling pattern.

Furthermore, estimating the mean, median and mode of each balance coefficient from data matrix \mathbf{B} , results as presented in Table 6.2 were obtained. In particular, it holds true for all pattern pairs in Ω_\dagger that

$$E[\beta_i] \cong \text{Md}[\beta_i] \cong \text{Mo}[\beta_i] \cong 1/2, \quad (6.23)$$

Table 6.2 The principle component $\mathbf{z}_1 = \beta^\top \mathbf{w}_1$ is a (trivially simple) linear combination of a subset of balance coefficients. Moreover, all balance coefficients not included in \mathbf{z}_1 (listed in the lower part of the table) appear to be symmetrically distributed around 1/2.

Index	Balance Coeff.	Weight		Mean	Median	Mode
		$w_{1,i}$	$w_{1,i}/w_{1,1}$			
i	β_i			$E[\beta_i]$	$\text{Md}[\beta_i]$	$\text{Mo}[\beta_i]$
1	$\beta_{\mathbf{123}/\mathbf{132}}$	-0.353	1.000	0.8588	0.8842	0.9000
2	$\beta_{\mathbf{123}/\mathbf{213}}$	-0.354	1.002	0.8586	0.8842	0.9000
3	$\beta_{\mathbf{123}/\mathbf{231}}$	-0.353	0.999	0.8588	0.8842	0.9000
4	$\beta_{\mathbf{123}/\mathbf{312}}$	-0.352	0.997	0.8590	0.8842	0.9000
9	$\beta_{\mathbf{132}/\mathbf{321}}$	0.354	-1.003	0.1415	0.1157	0.0909
12	$\beta_{\mathbf{213}/\mathbf{321}}$	0.355	-1.005	0.1417	0.1158	0.0909
14	$\beta_{\mathbf{231}/\mathbf{321}}$	0.354	-1.002	0.1414	0.1158	0.1111
15	$\beta_{\mathbf{312}/\mathbf{321}}$	0.353	-1.000	0.1412	0.1158	0.0909
5	$\beta_{\mathbf{123}/\mathbf{321}}$	0.002	-0.005	0.5004	0.5004	0.5000
6	$\beta_{\mathbf{132}/\mathbf{213}}$	-0.001	0.003	0.4997	0.5000	0.5000
7	$\beta_{\mathbf{132}/\mathbf{231}}$	0.001	-0.001	0.5001	0.5000	0.5000
8	$\beta_{\mathbf{132}/\mathbf{312}}$	0.001	-0.004	0.5004	0.5000	0.5000
10	$\beta_{\mathbf{213}/\mathbf{231}}$	0.001	-0.004	0.5004	0.5000	0.5000
11	$\beta_{\mathbf{213}/\mathbf{312}}$	0.002	-0.007	0.5007	0.5008	0.5000
13	$\beta_{\mathbf{231}/\mathbf{312}}$	0.001	-0.003	0.5003	0.5000	0.5000

which strongly suggests that their respective balance coefficients are symmetrically distributed around 1/2. Under this premise, and in conjunction with the definition as per Equation (6.5), the relations

$$E[p_{\mathbf{123}}] \cong E[p_{\mathbf{321}}], \quad \text{as well as} \quad E[p_{\mathbf{132}}] \cong E[p_{\mathbf{213}}] \cong E[p_{\mathbf{231}}] \cong E[p_{\mathbf{312}}] \quad (6.24)$$

follow immediately. Moreover, and again due to the linear combination of Equation (6.21), the balance coefficients for the pattern pairs

$$\Omega_\star = \left\{ (\mathbf{123}, \mathbf{132}), (\mathbf{123}, \mathbf{213}), (\mathbf{123}, \mathbf{231}), (\mathbf{123}, \mathbf{312}), \right. \\ \left. (\mathbf{321}, \mathbf{132}), (\mathbf{321}, \mathbf{213}), (\mathbf{321}, \mathbf{231}), (\mathbf{321}, \mathbf{312}) \right\} \quad (6.25)$$

contribute almost equally to the variance in \mathbf{B} . Those are all of the possible balances between a peak pattern and a monotonous pattern. Introducing the peak probability \hat{p} in terms of

$$\hat{p} = p_{\mathbf{132}} + p_{\mathbf{213}} + p_{\mathbf{231}} + p_{\mathbf{312}}, \quad \text{and} \quad 1 - \hat{p} = p_{\mathbf{123}} + p_{\mathbf{321}}, \quad (6.26)$$

and using the relations given by Equation (6.24), each of these balance coefficients can be approximated by

$$\beta_{\star} = \frac{(1 - \hat{p})/2}{(1 - \hat{p})/2 + \hat{p}/4} = \frac{2(\hat{p} - 1)}{\hat{p} - 2},$$

and the linear combination of Equation (6.21) be reduced to the expression

$$\frac{\mathbf{z}_1}{w_{1,1}} \approx 8\beta_{\star} = \frac{16(\hat{p} - 1)}{\hat{p} - 2}. \quad (6.27)$$

Let us assume that equality holds in the above relation, such that all variance within the decisive principle component \mathbf{z}_1 is exclusively induced by the peak probability \hat{p} . Without compromising the dynamics of permutation entropy, all four peak patterns **132**, **213**, **231** and **312** can then be merged into just one ordinal pattern—the peak pattern. Likewise, the monotonous patterns **123** and **321** can be unified, yielding a single *edge* pattern. On this basis, permutation entropy of order $m = 3$ and time lag $\tau = 1$ can be replaced by a much simpler *entropy of peaks* function

$$H(\hat{p}) = \hat{p} \log \frac{4}{\hat{p}} + (1 - \hat{p}) \log \frac{2}{1 - \hat{p}} = H_b(\hat{p}) + \hat{p} \log 2 + \log 2. \quad (6.28)$$

This function is concave, and has its maximum at $\hat{p} = 2/3$, where $p_i = 1/6$ for all $i \in \{1, 2, \dots, 6\}$, which is consistent with permutation entropy being maximal for a uniform distribution of ordinal patterns. The graph of $H(\hat{p})$ is depicted in Figure 6.7.

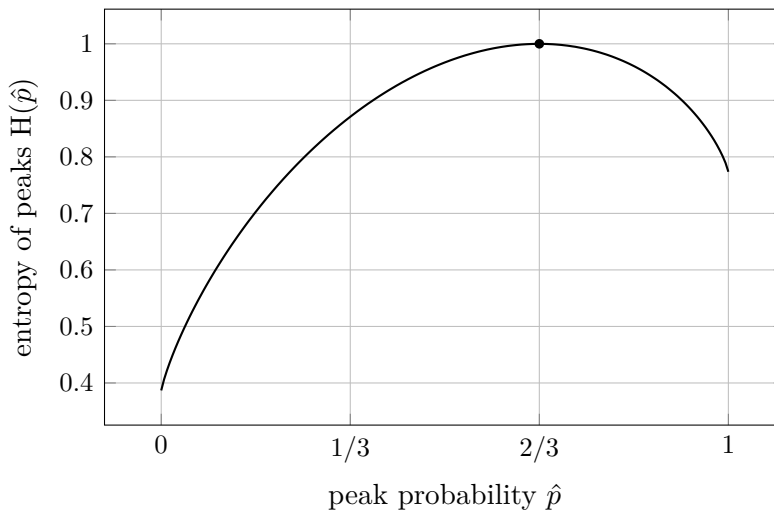


Figure 6.7 The entropy of peaks function $H(\hat{p})$ as given by Equation (6.28), divided by $\log 3!$ for normalisation. The maximum is taken on for $\hat{p} = 2/3$, where $H(\hat{p}) = 1$.

For verification purposes, the entropy of peaks $H(\hat{p})$ was calculated for all of the EEG epochs obtained from the CAP Sleep Database. Using the same overall approach as described in Section 6.5.2 for conventional permutation entropy, another sequence of $N_{\text{cap}} = 1.6 \cdot 10^6$ entropy values was obtained. Direct comparison yielded a Pearson correlation coefficient of $r = 0.99998$, and a mean relative error of $\bar{\eta} = 0.03\%$ between permutation entropy and the entropy of peaks.

In summary, the ordinal patterns of order $m = 3$ obtained from the CAP Sleep Database can be replaced by binary symbols that encode the disjoint properties “*three consecutive values form a peak*” and “*three consecutive values form an edge*”.

6.6 Linearising Permutation Entropy

The previous sections justify the working hypothesis that, in EEG analysis, permutation entropy of order $m = 3$ may behave as a univariate function of the peak probability \hat{p} . The latter can be estimated from an epoch of w EEG samples $\{x_1, x_2, \dots, x_w\}$ by computing

$$\hat{p} = \frac{\hat{w}}{w} \quad \text{with} \quad \hat{w} = \sum_{k=2}^{w-1} \left| [x_k - x_{k-1} \geq 0] - [x_{k+1} - x_k \geq 0] \right|. \quad (6.29)$$

In the following, let us look in more detail at the properties of those decisive signal peaks.

6.6.1 Signal Peaks and Spectral Bandwidth

In mathematical terms, a peak is a local extremum. Any real-valued, twice differentiable function s has a peak at t_k if it holds that $s'(t_k) = 0$ and $s''(t_k) \neq 0$. In other words, a peak in s is a zero crossing in its first derivative s' . Now, consider a real-valued function s , periodic in $T = 1/f_0$, and band-limited to frequencies including Nf_0 , such that

$$s(t) = \sum_{k=0}^N a_k \sin(2\pi k f_0 t + \phi_k), \quad \text{with } a_k \in \mathbb{R}, \text{ and } \phi_k \in \mathbb{R}. \quad (6.30)$$

Its first derivative is again a real-valued function, periodic in T and band-limited to Nf_0 , namely

$$s'(t) = \sum_{k=0}^N 2\pi a_k k f_0 \cos(2\pi k f_0 t + \phi_k) = \sum_{k=0}^N b_k \sin(2\pi k f_0 t + \vartheta_k). \quad (6.31)$$

Being the N -th partial sum of a Fourier series, the function can be alternatively be written using complex exponentials,

$$s'(t) = \sum_{k=-N}^N c_k e^{j2\pi k f_0 t}, \quad \text{with } j = \sqrt{-1}, \text{ and } c_k \in \mathbb{C}. \quad (6.32)$$

This expression constitutes a trigonometric polynomial of order N , and therefore has exactly $2N$ roots per period [172]. As for any real-valued polynomial, its zeroes

$$Z = \{t_k \mid s'(t_k) = 0, k \in \{1, 2, \dots, 2N\}\} \quad (6.33)$$

are not necessarily distinct, and are either real or occur in complex conjugate pairs. Therein, let us focus on the subset of zero crossings exclusively, which is

$$Z_\phi = \{t_k \in \mathbb{R} \mid s''(t_k) \neq 0\} \subseteq Z. \quad (6.34)$$

With Z_ϕ being a subset of Z , it obviously holds that $|Z_\phi| \leq 2N$. Moreover, the number of peaks \hat{w} in the function s equals the number of zero crossings $|Z_\phi|$ in its first derivative s' . It therefore holds that

$$\hat{w} = |Z_\phi| \leq \frac{2f_{\max}}{f_0} \quad \text{with} \quad f_{\max} = Nf_0. \quad (6.35)$$

Sampling one full cycle of the function s at a sampling rate of $f_S = wf_0$, a sequence of w values $\{x_1, x_2, \dots, x_w\}$ is obtained. Its maximum number of peaks can be expressed as

$$\hat{w} \leq \frac{2wf_{\max}}{f_S} \quad \text{with} \quad f_{\max} \leq \frac{f_S}{2}. \quad (6.36)$$

Consequently, it holds for the peak probability \hat{p} that

$$\hat{p} = \frac{\hat{w}}{w} \leq \frac{2f_{\max}}{f_S}. \quad (6.37)$$

From a practical point of view, this relation is not limited to periodic signals. To assess f_{\max} in *any* signal segment, Fourier transform needs to be applied, which by definition constitutes a periodic continuation of the analysis window. Thus, the inequality given by Equation (6.37) provides an upper bound on the number of peaks in any band-limited signal.

This bound may be naïvely conservative, but it is easily derived and sufficient to draw the following conclusion: given that the entropy of peaks function $H(\hat{p})$ of Equation (6.28) increases monotonically on the interval $0 \leq \hat{p} \leq 2/3$, it follows from Equation (6.37) that permutation entropy of order $m = 3$ is monotonic with \hat{p} for EEG epochs band-limited such that

$$f_{\max} \leq \frac{f_s}{3}. \quad (6.38)$$

As stated earlier, the absolute value of permutation entropy obtained from a single EEG epoch is insignificant. It is the contrast in value across distinct epochs, its dynamics, that renders permutation entropy a valuable tool for EEG analysis. However, this contrast persists for any parameter that grows monotonically with permutation entropy. Therefore, the following constitutes an astounding, yet sensible proposition: for an epoch of EEG, digitised at a sampling rate of f_s , and band-limited such that $f_{\max} \leq f_s/3$, permutation entropy of order $m = 3$ and time lag $\tau = 1$ can be substituted by the peak probability \hat{p} , that is, by the number of peaks per unit time.

6.6.2 Counting Zigzags in EEG

Further EEG analyses were performed to test the aforementioned proposition under real-world conditions. To rule out possible peculiarities of the CAP Sleep Database or sleep EEG in general, the data corpus was expanded with a set of EEG signals from a different clinical setting.

The *CHB-MIT Scalp EEG Database* is a collection of EEG recordings obtained from paediatric patients suffering from epileptic seizures [173]. Acquired at Boston Children’s Hospital, the data were generously put into the public domain and are available from PhysioNet [170]. In analogy with the procedure described in Section 6.5.2, a total of $N_{\text{mit}} = 4.1 \cdot 10^6$ EEG epochs of 20 s duration were obtained from this dataset—another two and a half years of single-channel EEG data. For each of those epochs, permutation entropy of order $m = 3$ and time lag $\tau = 1$ was computed. In addition, an estimate of the peak probability \hat{p} as per Equation (6.29) was obtained from every EEG epoch of the CAP and CHB-MIT datasets. Spearman correlation coefficients between peak probabilities and permutation entropies were then computed for the subsets of EEG epochs with $\hat{p} \leq 2/3$, the subsets of EEG epochs with $\hat{p} > 2/3$, as well as the respective data sets in their entirety. The results of those analyses are provided in Table 6.3.

Table 6.3 Spearman correlation coefficients between permutation entropy and peak probability, for EEG epochs obtained from the CAP Sleep Database and the CHB-MIT Scalp EEG Database. Quantities subscripted “low” and “high” correspond to results obtained from those subsets of signal epochs for which $\hat{p} \leq 2/3$ or $\hat{p} > 2/3$ respectively hold.

Data Set	Number of Epochs			Spearman Correlation		
	N	N_{low}	N_{high}	$\rho(\text{PeEn}, \hat{p})$	$\rho_{\text{low}}(\text{PeEn}, \hat{p})$	$\rho_{\text{high}}(\text{PeEn}, \hat{p})$
CAP	$1.6 \cdot 10^6$	$1.6 \cdot 10^6$	$6.0 \cdot 10^2$	0.99998	0.99998	-0.923
CHB-MIT	$4.1 \cdot 10^6$	$4.0 \cdot 10^6$	$1.1 \cdot 10^5$	0.99937	0.99993	-0.963

The results given in Table 6.3 clearly support the conjecture that permutation entropy of order $m = 3$ and time lag $\tau = 1$ increases monotonically with the peak probability for $\hat{p} \leq 2/3$. Moreover, the correlation is inverted for peak probabilities exceeding this boundary. Both results are fully in line with the function graph of Figure 6.7. When testing on the full set of peak probabilities, strong positive correlation is maintained, because the epochs for which $\hat{p} > 2/3$ holds are too seldom to make a difference. This directly relates to the bandwidth constraint of Equation (6.38), which implies that for the chosen sampling rate of $f_S = 200$ Hz, the peak probability cannot increase beyond $\hat{p} = 2/3$ for EEG band-limited to frequencies below $f_{\text{max}} \cong 66.7$ Hz.

Not only is the bandwidth $f_{\text{max}} \leq f_S/3$ seldom exceeded in the datasets considered here, but the same presumably holds true for the majority of EEG in general: sampling rates below 100 Hz are quite uncommon for EEG, while spectral content above 30 Hz is often regarded as insignificant, and thus suppressed using low-pass filters. Moreover, an investigator actually interested in “gamma-band activity” will choose a considerably higher sampling rate. It is therefore likely that the decrease in permutation entropy for peak probabilities $\hat{p} > 2/3$ is not a part of the actual effect, but a rarely observed flaw in the analysis technique.

In essence, utilising permutation entropy of order $m = 3$ and time lag $\tau = 1$ for EEG analysis appears to be an involved way of counting zigzags in the signal.

6.7 Permutation Entropy as a Spectral Estimator

While counting signal peaks may seem a trivial procedure at first glance, a profound mathematical theory actually supports this approach to signal analysis. It was formulated by Benjamin Kedem in the 1980s, and is centred around the notion of *higher order crossings*. Only a humble outline of this framework (limited to aspects that aid the interpretation of permutation entropy in EEG analysis) will here be provided.

For a large-scale, yet low-threshold introduction to the theory, the reader is referred to Kedem's comprehensive article *Spectral Analysis and Discrimination by Zero-Crossings* [174], which also served as the primary source for the section to follow.

6.7.1 Zero Crossings and the Dominant Frequency Principle

Signal analysis by means of higher order crossings is a generalisation of the commonly known *zero crossing rate*, a classic method for fundamental frequency estimation [175]. Self-descriptively, the zero crossing rate z_ϕ of a signal is its average number of x -axis intercepts per unit time. It is obtained from a discrete sequence of samples $\{x_1, x_2, \dots, x_w\}$ by computing

$$z_\phi = \frac{1}{w-1} \sum_{k=2}^w |[x_k \geq 0] - [x_{k-1} \geq 0]|. \quad (6.39)$$

Kedem refined the meaning of this quantity, demonstrating that the zero crossing rate of a discrete-time signal constitutes an approximation of the *centroid* of its power spectrum

$$c_{S_{xx}} = \frac{1}{m_{S_{xx}}} \int_0^\infty f S_{xx}(f) df, \quad \text{where} \quad m_{S_{xx}} = \int_0^\infty S_{xx}(f) df. \quad (6.40)$$

Here, S_{xx} denotes the power spectral density of the signal, and $m_{S_{xx}}$ is its (single-sided) spectral mass. Intuitively, if S_{xx} contains a predominant frequency f_c , the spectral centroid $c_{S_{xx}}$ will gravitate towards this particular frequency, and the more $S_{xx}(f_c)$ outweighs the rest of the spectrum, the shorter the distance between $c_{S_{xx}}$ and f_c will be. Kedem termed this the *dominant frequency principle*, and further deduced that

$$c_{S_{xx}} \cong \frac{z_\phi}{2}. \quad (6.41)$$

A mathematical proof for this interrelation was given in [174], and further supported by an example quite similar to the following Example 8.

Example 8. Consider the time-continuous sinusoidal signal $s(t) = A \sin(2\pi f_c t + \varphi)$, which has the power spectral density

$$S_{xx}(f) = \begin{cases} A^2/4, & \text{for } f = \pm f_c, \\ 0, & \text{otherwise.} \end{cases}$$

Because f_c is the only frequency in the spectrum, it is obviously the dominant frequency. Calculating the spectral centroid of the signal by means of Equation (6.40), we see that $c_{S_{xx}} = f_c$ holds. Hence, this is a showcase for the dominant frequency principle.

Just as comprehensible, the relation $z_\emptyset = 2f_c = 2c_{S_{xx}}$ also applies, because any sinusoidal signal has exactly two zero crossings per cycle.

6.7.2 Higher Order Crossings

In the framework of higher order crossings, the peak probability \hat{p} in terms of Equation (6.29) is closely related to the zero crossing rate z_\emptyset .

Let $\{x_1, x_2, \dots, x_w\}$ be a sequence of values, sampled from a continuous-time signal $s(t)$ at discrete time steps $t \in \{1/f_s, 2/f_s, \dots, w/f_s\}$. Estimating the peak probability \hat{p} of this sequence as per Equation (6.29) is equivalent to obtaining the zero crossing rate z_\emptyset of the difference sequence $\{\nabla x_2, \nabla x_3, \dots, \nabla x_w\}$, whereby the difference operator ∇ is defined such that

$$\nabla x_k = x_k - x_{k-1} = s(k/f_s) - s((k-1)/f_s). \quad (6.42)$$

In terms of Kedem's theory, the generalised zero crossings obtained by recursively applying the ∇ -operator k times in a row are called: higher order crossings of the D_{k+1} type. Thus, *actual* zero crossings are higher order crossings of the D_1 type, while local extrema (that is, the peak patterns **132**, **213**, **231**, and **312**) are of type D_2 .

Kedem's approach is highly convenient in that the ∇ -operator represents a linear time-invariant system with the power transfer function

$$|H(f)|^2 = |(1 - e^{-j2\pi f/f_s})|^2 = 2 - 2 \cos(2\pi f/f_s). \quad (6.43)$$

It thus acts as a high-pass filter with a frequency response as depicted in Figure 6.8.

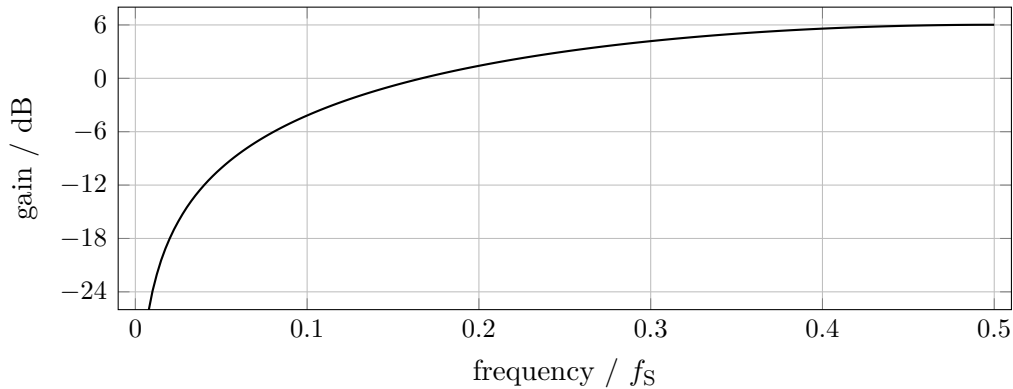


Figure 6.8 Power transfer function $|H(f)|^2$ of the ∇ -operator.

Intuitively, the filter compresses the spectral mass, shifting the spectral centroid further to the right with each recurrent application of the ∇ -operator.

6.7.3 The Spectral Estimation Hypothesis

In summary, the chain of results that explains the dynamics of permutation entropy in EEG for pattern order $m = 3$ and time lag $\tau = 1$ is the following:

1. Due to their specific distribution in EEG, it may suffice to distinguish not six, but merely two kinds of ordinal patterns, namely the peaks $\{\mathbf{132}, \mathbf{213}, \mathbf{231}, \mathbf{312}\}$, and the edges $\{\mathbf{123}, \mathbf{321}\}$, with their respective probabilities \hat{p} and $1 - \hat{p}$.
2. For EEG epochs with $\hat{p} \leq 2/3$, permutation entropy increases monotonously with the peak probability \hat{p} , and both can therefore be used interchangeably in comparative analyses.
3. The peak probability \hat{p} is the zero crossing rate z_\circ of a high-pass filtered version of the signal, whereas the zero crossing rate z_\circ of a signal is an estimate of its power spectral centroid.

On that basis, it is a rational conjecture that applying permutation entropy of order $m = 3$ and time lag $\tau = 1$ to EEG epochs with a peak probability $\hat{p} \leq 2/3$ effectively means high-pass filtering the signal according to Equation (6.43), and estimating the centroid of the resulting weighted power spectrum. Both steps are standard procedures of linear signal processing, possibly rendering this EEG parameter a spectral estimator in disguise.

6.8 Higher Pattern Orders and Time Lags

Up to this point, all considerations and propositions on the behaviour of permutation entropy in EEG were limited to the pattern order $m = 3$ and time lag $\tau = 1$. Regarding the latter, a general discussion was postponed for reasons of simplicity only, and will be presented in Section 6.8.2. Conversely, formulating a universal interpretation for patterns of orders $m > 3$ is a subject still under investigation—and hopefully one to be reported on in the future. For the time being, the reader is invited to consider the following preliminary results.

6.8.1 Prospects for Patterns of Higher Order

While an ordinal pattern of order $m = 3$ is either a peak or an edge, there is no such simple duality for higher pattern orders. In the general case, a single pattern of order m can contain up to $m - 2$ peaks, however, the peaks of consecutive ordinal patterns may overlap. Moreover, the marginal probability relation of Equation (6.24) does not translate to the “sub-patterns” of order $m = 3$ that form a pattern of higher order: conditional probabilities have to be considered instead.

Despite such intricacies, the principle behaviour of permutation entropy does apparently not change when increasing the order m within statistically reasonable limits. Calculating permutation entropy of orders $m \in \{3, 4, 5\}$ for the EEG epochs obtained from the CAP and CHB-MIT datasets (see Sections 6.5.1 and 6.6.2), results as depicted in Figure 6.9 were obtained. In particular, the strong monotonic correlation between permutation entropy and the peak probability \hat{p} is maintained for higher pattern orders. This hints at a possible generalisation of our spectral estimation conjecture.

6.8.2 Sampling, Resampling and Aliasing

In comparison to higher pattern orders m , the interpretation of arbitrary time lags τ is straightforward. Under the premise of permutation entropy acting as a spectral estimator in EEG analysis, the principles of linear systems theory fully apply. Using a time lag $\tau > 1$ is then a means of downsampling (and thus, of band limiting), but may also result in a violation of the Nyquist–Shannon sampling theorem. Let us substantiate.

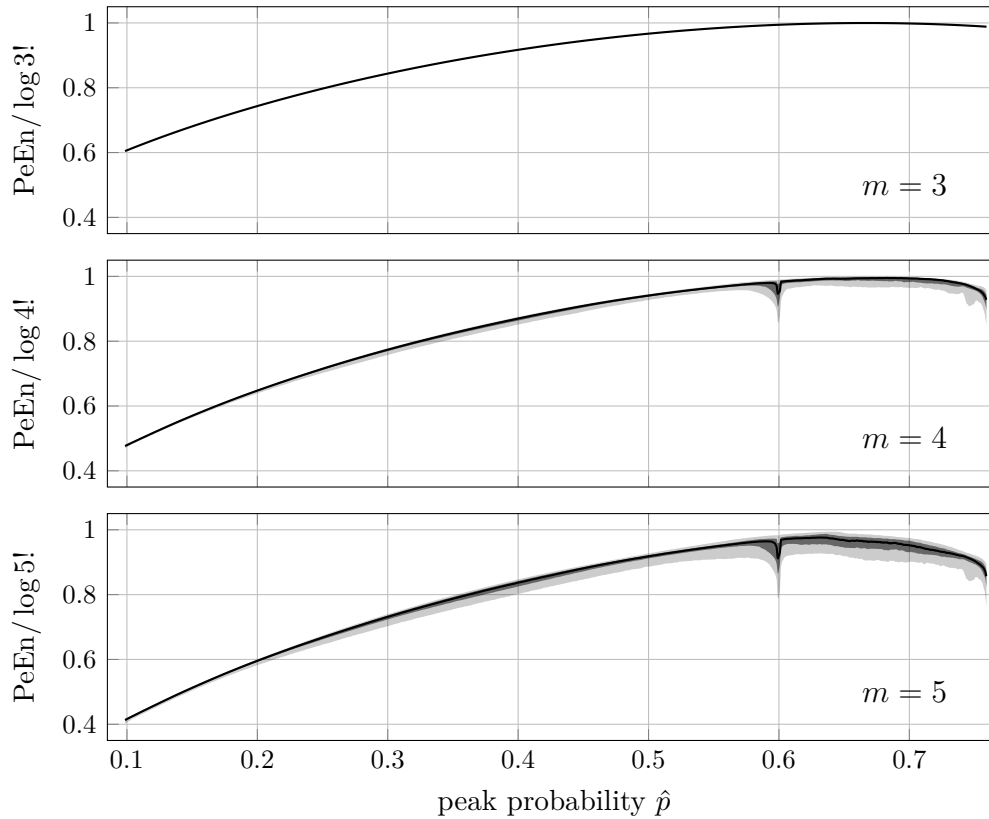


Figure 6.9 Normalised permutation entropy (PeEn) of orders $m \in \{3, 4, 5\}$, plotted over the peak probability \hat{p} . Solid curves represent the median. Where graphically resolvable, inner shaded bands correspond to the 25th–75th, outer shaded bands to the 5th–95th percentiles. Peak probabilities occurring less than 100 times were discarded and missing values linearly interpolated. A moving average filter of bandwidth $2.5 \cdot 10^{-3}$ was applied for data smoothing.

The Nyquist–Shannon sampling theorem [176] states that any time-discrete representation of an analogue signal is implicitly band limited to frequencies below and including half its sampling rate f_S . Due to this fundamental constraint of digital signal processing, an analogue low-pass filter is commonly applied during signal acquisition. This filter prevents distortions known as *aliasing*, that is, spectral content beyond the Nyquist frequency $f_{ny} = f_S/2$ folding back into the usable frequency range as in

$$f \mapsto \frac{f_S}{2} - \left| (f \bmod f_S) - \frac{f_S}{2} \right|, \quad \text{where } f \geq 0. \quad (6.44)$$

As displayed in Figure 6.10, this function describes a triangle waveform with its extrema at multiples of $f_S/2$.

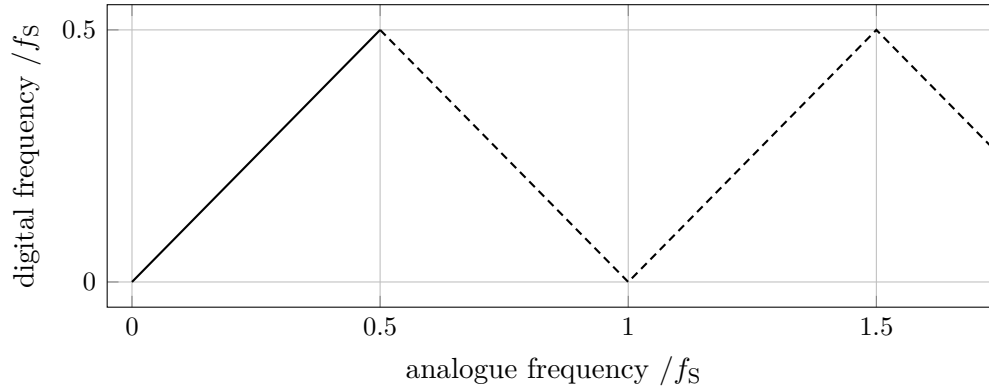


Figure 6.10 The aliasing relation as specified by Equation (6.44). Any spectral content exceeding the Nyquist rate $f_{ny} = f_s/2$ is periodically folded back into the representable frequency range.

The same constraint applies to sampling rate conversion in the digital domain. Correctly reducing the sampling rate f_s of a digital signal by a factor k therefore involves two steps: limit the bandwidth to $f_s/2k$, then omit all but every k -th sample. If the band limiting is skipped, downsampling is prone to aliasing, because any spectral content exceeding $f_s/2k$ is incompatible with the targeted sampling rate f_s/k , and folds back into the representable frequency range, corrupting the resulting signal.

6.8.3 The Time Lag as a Downsampling Factor

According to Section 6.2.1, the usual approach of estimating pattern distributions from time series is as follows. Given a finite time series $\{x_1, x_2, \dots, x_N\}$ of N elements, its ordinal pattern π_t of order m and time lag τ at time index t is fully determined by the m -tuple $(x_t, x_{t+\tau}, x_{t+2\tau}, \dots, x_{t+(m-1)\tau})$. To estimate the ordinal pattern distribution of the time series, the sequence $\{\pi_t\}$ is obtained for all $t \in \{1, 2, \dots, N - (m-1)\tau\}$, and the relative number of occurrences of each $\pi_i \in \Omega_m$ is then counted.

For time lags $\tau > 1$, consider an alternative, yet entirely equivalent estimation algorithm: first, partition the time series $\{x_1, x_2, \dots, x_N\}$ into τ sub-sequences of the form

$$\forall \lambda \in \{1, 2, \dots, \tau\}: \{x_\lambda, x_{\lambda+\tau}, x_{\lambda+2\tau}, \dots, x_{\lambda+k\tau}\}, \quad \text{with } k = \left\lfloor \frac{N - \lambda}{\tau} \right\rfloor + 1. \quad (6.45)$$

Next, estimate from each sub-sequence its ordinal pattern distribution \mathbf{p}_λ , now using the time lag $\tau^* = 1$. The ordinal pattern distribution \mathbf{p} of the initial sequence can then be

obtained by cumulation, that is

$$\mathbf{p} = \sum_{\lambda=1}^{\tau} \mathbf{p}_{\lambda} = \mathbf{p}_1 + \mathbf{p}_2 + \cdots + \mathbf{p}_{\tau}. \quad (6.46)$$

Observe that generating a sub-sequence as per Equation (6.45) is identical to down-sampling the time series $\{x_1, x_2, \dots, x_N\}$ by a factor τ , whereby the index λ merely specifies which sample to keep per block of τ consecutive samples.

This circumstance is both the explanation and the fundamental problem of using $\tau > 1$ for pattern encoding. We implicitly, but no less effectively, downsample the signal without any anti-aliasing measures taken. Inevitably, all spectral content exceeding $f_s/2\tau$ folds back into the representable frequency range. Therefore, calculating permutation entropy for $\tau > 1$ is equivalent to computing permutation entropy for the time lag $\tau^* = 1$ from a set of τ downsampled, but also non-linearly distorted versions of the actual signal.

Notice that this is not a conceptual flaw of permutation entropy. One has to keep in mind that the quantity was developed as a complexity measure for *time series*. Working with less generic data, additional constraints and requirements may arise. In particular, while any digital signal can be described as a time series, not all time series are digital signals in terms of linear systems theory: only if the requirements of the sampling theorem are adhered to can a discrete sequence of voltage measurements be a valid representation of analogue EEG.

6.8.4 Frequency Aliasing in Permutation Entropy

The aliasing introduced by using time lags $\tau > 1$ can easily be observed by computing the entropy of peaks $H(\hat{p})$ as per Equation (6.28) from sinusoidal signals of varying frequency $f_c \leq f_s/2$. In accordance with the frequency mapping of Equation (6.44), and depending on the effective sampling rate f_s/τ , the signal that is actually fed into the estimator is a sinusoidal wave of frequency

$$\tilde{f}_c = \frac{f_s}{2\tau} - \left| \left(f_c \bmod \frac{f_s}{\tau} \right) - \frac{f_s}{2\tau} \right|. \quad (6.47)$$

Any sinusoidal signal has two peaks per cycle, so it further holds that $\hat{p} = 2\tilde{f}_c$, and in

conjunction with Equation (6.28), the relation

$$H(\hat{p}) = H(2\tilde{f}_c) = H_b(2\tilde{f}_c) + 2\tilde{f}_c \log 2 + \log 2 \quad (6.48)$$

is obtained. Variegating the time lag τ , the periodic foldback effect manifests in the graphs of this family of functions. This is depicted and further discussed in Figure 6.11.

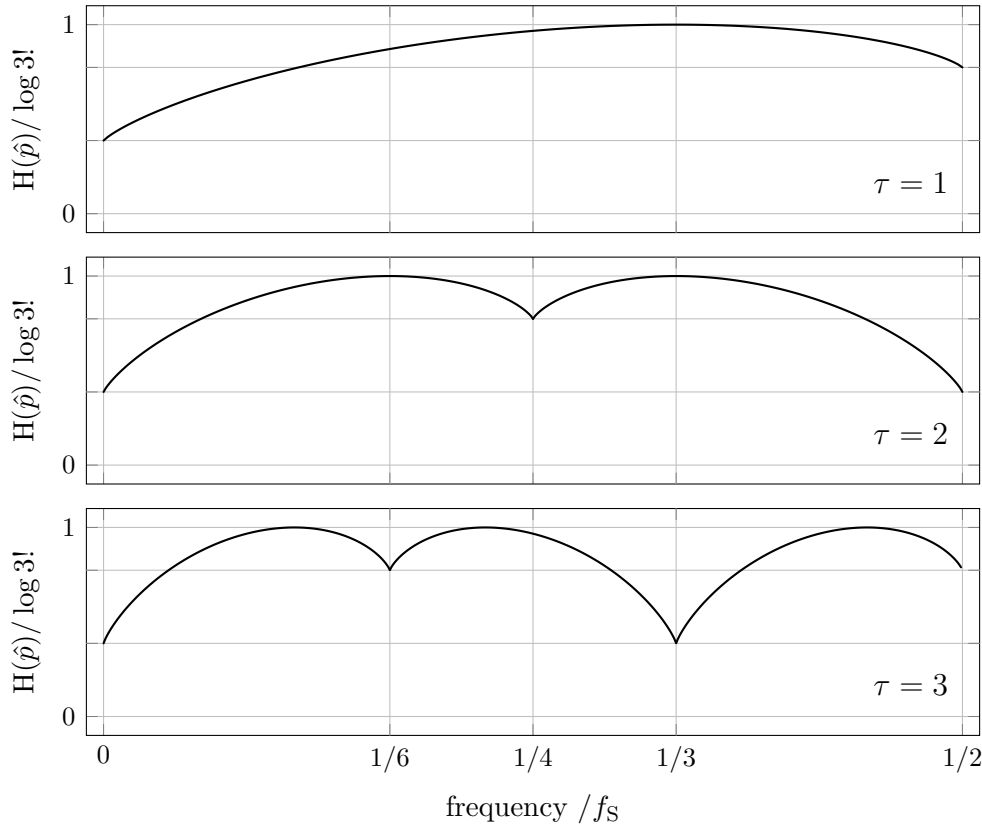


Figure 6.11 The entropy of peaks function $H(\hat{p})$, calculated from a sine wave of increasing frequency. For the time lag $\tau = 1$, the function behaves as discussed for Figure 6.7. When using a time lag $\tau > 1$, aliasing is provoked for frequencies exceeding $f_S/2\tau$. Those frequencies periodically fold back into the representable spectrum, which leads to the symmetries depicted.

The presented approach is based on the work of Erik Olofsen, Jamie Sleight and Albert Dahan. To experimentally investigate its dependence on frequency, the authors calculated permutation entropy of order $m = 3$ and time lags $\tau \in \{1, 2\}$ for sinusoidal signals of increasing frequency [44]. The here proposed spectral estimation conjecture conclusively explains the authors’ observations—in particular, the “markedly different, and more complex” [44] frequency dependence for $\tau = 2$.

6.8.5 Anti-Aliased Permutation Entropy

As demonstrated in the previous section, using time lags $\tau > 1$ for EEG encoding may cause unintended signal distortions. Nevertheless, a multitude of applications in EEG analysis reportedly depend on such time lags [169]. In practice, the impact of aliasing may be less detrimental than the graphs of Figure 6.11 suggest: EEG is anything but sinusoidal, and while permutation entropy literally locks onto a single sine wave, the discrepancy will be more subtle for actual EEG. Moreover, EEG is commonly oversampled, and of course, $f_S \gg 2f_{\max}$ mitigates the issue to some extent (see Section 6.6).

Nothing is lost by resolving a source of inaccuracy, though. This can easily be achieved by means of sample rate conversion. Instead of using $\tau > 1$, one would *correctly* convert the EEG to a new sampling rate $f_S^* = f_S/\tau$, and compute permutation entropy therefrom, now choosing $\tau^* = 1$ as the time lag. An apparent disadvantage of this approach is that it reduces the number of ordinal patterns available for probability estimation by a factor τ . However, the sampling theorem implicitly assures that a single rate-reduced sequence suffices to fully describe the band-limited EEG (including its signal peaks), so nothing is gained by counting each peak τ times in a row.

At least in the anti-aliased case, a very tangible interpretation of the time lag τ immediately follows. Permutation entropy effectively splits the single-sided frequency spectrum of the EEG into τ spectral bands of equal width, then exclusively responds to the content of the lowest band. Given our spectral estimation hypothesis, permutation entropy of order $m = 3$ and time lag $\tau \geq 1$ thus approximates the weighted power spectral centroid of the EEG frequency band ranging from 0 to $f_S/2\tau$. For applications depending on $\tau > 1$, it may hence be worthwhile to further investigate the influence of anti-aliasing. This could be a chance of concisely relating ordinal analysis techniques with classic band-spectral methods of EEG analysis.

6.9 Summary

EEG has long become an indispensable tool in research and medical diagnostics, even more so as the progress in digital technology has revolutionised our general approach towards data—biomedical data being no exception.

When performing EEG analysis, one has to keep in mind its predominantly probabilistic nature: any signal property that statistically correlates with some behavioural observation is a perfectly valid signal property, because ultimately, none of them is fully explainable mechanistically. In an ongoing effort, investigators thus dedicate themselves to the experimental identification of new parameters adaptable to EEG processing. But with a plethora of different analysis techniques already proposed, the application of a more sophisticated method does not necessarily yield *additional* information, because there are only so many complementary signal characteristics. According to the results of the current chapter,

- permutation entropy of order $m = 3$ and time lag $\tau = 1$,
- the power spectral centroid of the signal's first derivative, and
- Kedem's higher order crossings of the D_2 type

can all be used interchangeably in EEG analysis. It comes to no surprise that both the spectral centroid [177, 178], as well as higher order crossings [179] have been reported as suitable EEG parameters in their own rights. The property underlying all three of those methods has been known and used for more than 50 years now: in 1964, Neil Burch and colleagues reported on transferring the EEG analysis method of counting “baseline crossing[s]” from analogue circuitry to a digital computer, and explicitly pointed out the “well-known accentuation of the higher frequency component in the derivatives” [180]. Back then, the procedure was called *period analysis*.

On the other hand, it cannot be over-emphasized that EEG analysis is mostly driven by experimentation, and this obviously includes the results presented in this chapter. What applies for the CAP and CHB-MIT databases does not necessarily hold true for EEG in general. Hence, the above results neither falsify the existence of ordinal complexity in the EEG, nor its observability using permutation entropy. It merely demonstrates that the permutation entropy of the EEG does not exclusively relate to this phenomenon.

Conclusively, the interrelations described extend the framework of ordinal pattern-based EEG processing. For a given analysis task, concurrently computing both permutation entropy and the peak probability \hat{p} provides a means of rejecting the null hypothesis of permutation entropy acting as a spectral estimator. This may enable the investigator to narrow down on cases that truly demand for complexity considerations, while otherwise resorting to a simpler and more time-tested analysis technique. From that perspective, the peak probability \hat{p} given by Equation (6.29) integrates seamlessly into the framework of ordinal pattern analysis. A convenient side effect: if, for a particular analysis task,

permutation entropy is found to be replaceable by the peak probability \hat{p} , the issue of selecting an appropriate pattern order m does not arise.

In closing, implementing the peak probability parameter in software is straightforward:

```
y = mean(abs(diff(diff(x) >= 0)));
```

suffices to make the parameter available in GNU Octave, MATLAB, or similar numerical computation environments. An extended version, supporting sliding window analysis, is provided by the `zztop.m` function in the supplements of [77]. While not extensively optimised, its execution speed should meet the requirements of major analysis campaigns: the main challenge in data analysis today is not computational feasibility, but the interpretation of the manifold results obtainable.

7 Conclusions

The research project described in this dissertation started with a very simple question: Why does permutation entropy work as a signal parameter in quantitative EEG analysis? Investigating this question yielded results that increase the *interpretability* of permutation entropy in EEG, and the *efficiency* of ordinal time series analysis in general. To conclude the present journey through ordinal pattern spaces, both aspects are briefly summarised in the following.

7.1 Efficiency

To allow for studying permutation entropy in large EEG databases, fast algorithms for turning time series into sequences of ordinal patterns have been developed, and are described in Chapter 5. The algorithms utilise the classical Lehmer code to enumerate ordinal patterns by consecutive non-negative integers, starting from zero. This encoding can be performed at comparatively low computational cost, and the resulting compact representation considerably simplifies working with ordinal patterns in the digital domain. The algorithms described may improve the efficiency of virtually any analysis method involving ordinal patterns—among them quantitative measures like permutation entropy and symbolic transfer entropy, but also techniques like forbidden pattern identification [116, 181]. One of the algorithms studied stands out in terms of scalability: its run-time increases linearly with both pattern order and sequence length, while its memory footprint is practically negligible. Those properties enable the study of high-dimensional pattern spaces, and may allow for putting ideas into practice that had previously been hindered by computational burden. To that end, a cross-platform software library has been published as a supplement to [78]. It includes reference implementations for all of the algorithmic variants considered here, and supports various programming languages commonly used in scientific computing, namely NumPy/Python, GNU Octave, MATLAB, as well as the C programming language. All source code is provided under the permissive terms of a three-clause Berkeley Software Distribution (BSD) license.

7.2 Interpretability

Since the introduction of permutation entropy in 2002, its high performance as a signal parameter for EEG classification has been demonstrated by many different studies [3–75]. While the contrasts observed are usually attributed to the varying *complexity* of the EEG, the results of the present work allow for a different interpretation.

The empirical analyses reported in Chapter 6 suggest characteristic regularities in the ordinal pattern distributions of the EEG. In all brevity, those regularities imply that for oversampled EEG signals, permutation entropy increases monotonously with the relative frequency of *local extrema* in the signal. In other words, the more peaks an epoch of EEG contains, the higher its permutation entropy is. Counting the peaks of a signal is equivalent to counting the zero crossings of its first derivative. In turn, the zero crossing rate is known as an estimator of the spectral centroid of a signal. Consequently, the findings discussed in Chapter 6 bridge the gap between permutation entropy and the Fourier transform, and thus establish an immediate relation with classical EEG analysis methods.

The above conjecture is based on empirical observations from a finite collection of EEG. Future work should therefore consider both permutation entropy and the number of signal peaks in parallel, and use the latter to assess the relevance of the first. In cases where the behaviour of permutation entropy can actually be reproduced by counting signal peaks, a very useful interpretation follows immediately. Consider that differentiating a signal $x(t)$ is the same as weighting its amplitude spectrum $|X(f)|$ with $2\pi f$. Also recall that EEG amplitude spectra coarsely follow a power-law in terms of $|X(f)| \sim 1/f^\alpha$, whereby the exponent is usually close to $\alpha = 1$ [168, 182]. Differentiating an EEG signal thus compensates for the decay of its spectrum, and ultimately, the spectral centroid of this “whitened” signal reflects the dominant signal frequency relative to the $1/f$ -like background.

In conjunction with the many reports on the advantageous behaviour of permutation entropy as an EEG parameter, the results of the present work confirm that deviations from $1/f$ -like noise are highly significant for quantitative electroencephalography. Those should definitely be further investigated in the future.

Bibliography

- [1] C. Bandt and B. Pompe, “Permutation Entropy: A Natural Complexity Measure for Time Series”, *Physical Review Letters*, vol. 88, no. 17, 174102, 2002. DOI: 10.1103/PhysRevLett.88.174102.
- [2] C. E. Spearman, “The Proof and Measurement of Association between Two Things”, *The American Journal of Psychology*, vol. 15, no. 1, pp. 72–101, 1904. DOI: 10.2307/1412159.
- [3] Y. Cao, W.-w. Tung, J. B. Gao, V. A. Protopopescu and L. M. Hively, “Detecting dynamical changes in time series using the permutation entropy”, *Physical Review E*, vol. 70, no. 4, 046217, 2004. DOI: 10.1103/PhysRevE.70.046217.
- [4] X. Li, G. Ouyang and D. A. Richards, “Predictability analysis of absence seizures with permutation entropy”, *Epilepsy Research*, vol. 77, no. 1, pp. 70–74, 2007. DOI: 10.1016/j.eplepsyres.2007.08.002.
- [5] A. A. Bruzzo, B. Gesierich, M. Santi, C. A. Tassinari, N. Birbaumer and G. Rubboli, “Permutation entropy to detect vigilance changes and preictal states from scalp EEG in epileptic patients. A preliminary study”, *Neurological Sciences*, vol. 29, no. 1, pp. 3–9, 2008. DOI: 10.1007/s10072-008-0851-3.
- [6] C. C. Jouny and G. K. Bergey, “Characterization of early partial seizure onset: Frequency, complexity and entropy”, *Clinical Neurophysiology*, vol. 123, no. 4, pp. 658–669, 2012. DOI: 10.1016/j.clinph.2011.08.003.
- [7] N. Mammone, D. Labate, A. Lay-Ekuakille and F. C. Morabito, “Analysis of Absence Seizure Generation Using EEG Spatial-Temporal Regularity Measures”, *International Journal of Neural Systems*, vol. 22, no. 06, 1250024, 2012. DOI: 10.1142/S0129065712500244.
- [8] G. Ouyang, J. Li, X. Liu and X. Li, “Dynamic characteristics of absence EEG recordings with multiscale permutation entropy analysis”, *Epilepsy Research*, vol. 104, no. 3, pp. 246–252, 2013. DOI: 10.1016/j.eplepsyres.2012.11.003.

- [9] E. Ferlazzo, N. Mammone, V. Cianci, S. Gasparini, A. Gambardella, A. Labate, M. A. Latella, V. Sofia, M. Elia, F. C. Morabito and U. Aguglia, “Permutation entropy of scalp EEG: A tool to investigate epilepsies”, *Clinical Neurophysiology*, vol. 125, no. 1, pp. 13–20, 2014. DOI: 10.1016/j.clinph.2013.06.023.
- [10] Z. Yang, Y. Wang and G. Ouyang, “Adaptive Neuro-Fuzzy Inference System for Classification of Background EEG Signals from ESES Patients and Controls”, *The Scientific World Journal*, vol. 2014, 140863, 2014. DOI: 10.1155/2014/140863.
- [11] J. Li, X. Liu and G. Ouyang, “Using Relevance Feedback to Distinguish the Changes in EEG During Different Absence Seizure Phases”, *Clinical EEG and Neuroscience*, vol. 47, no. 3, pp. 211–219, 2016. DOI: 10.1177/1550059414548721.
- [12] D. M. Mateos, R. Guevara Erra, R. Wennberg and J. L. Perez Velazquez, “Measures of entropy and complexity in altered states of consciousness”, *Cognitive Neurodynamics*, vol. 12, no. 1, pp. 73–84, 2018. DOI: 10.1007/s11571-017-9459-8.
- [13] Y. Yang, M. Zhou, Y. Niu, C. Li, R. Cao, B. Wang, P. Yan, Y. Ma and J. Xiang, “Epileptic Seizure Prediction Based on Permutation Entropy”, *Frontiers in Computational Neuroscience*, vol. 12, 55, 2018. DOI: 10.3389/fncom.2018.00055.
- [14] G. Li, Y. Fan and Q. Pang, “[A study of sleep stage classification based on permutation entropy for electroencephalogram].”, *Sheng wu yi xue gong cheng xue za zhi [Journal of Biomedical Engineering]*, vol. 26, no. 4, pp. 869–872, 2009. URL: <https://www.ncbi.nlm.nih.gov/pubmed/19813629>.
- [15] N. Nicolaou and J. Georgiou, “The Use of Permutation Entropy to Characterize Sleep Electroencephalograms”, *Clinical EEG and Neuroscience*, vol. 42, no. 1, pp. 24–28, 2011. DOI: 10.1177/155005941104200107.
- [16] C. Bandt, “A New Kind of Permutation Entropy Used to Classify Sleep Stages from Invisible EEG Microstructure”, *Entropy*, vol. 19, no. 5, 197, 2017. DOI: 10.3390/e19050197.
- [17] T. Wielek, J. Lechinger, M. Wislowska, C. Blume, P. Ott, S. Wegenkittl, R. del Giudice, D. P. J. Heib, H. A. Mayer, S. Laureys, G. Pichler and M. Schabus, “Sleep in patients with disorders of consciousness characterized by means of machine learning”, *PLOS ONE*, vol. 13, no. 1, e0190458, 2018. DOI: 10.1371/journal.pone.0190458.

- [18] J. González, M. Cavelli, A. Mondino, C. Pascovich, S. Castro-Zaballa, P. Torterolo and N. Rubido, “Decreased electrocortical temporal complexity distinguishes sleep from wakefulness”, *Scientific Reports*, vol. 9, no. 1, 18457, 2019. DOI: 10.1038/s41598-019-54788-6.
- [19] T. Wielek, R. Del Giudice, A. Lang, M. Wislowska, P. Ott and M. Schabus, “On the development of sleep states in the first weeks of life”, *PLOS ONE*, vol. 14, no. 10, e0224521, 2019. DOI: 10.1371/journal.pone.0224521.
- [20] A. B. Thul, J. Lechinger, J. Donis, G. Michitsch, G. Pichler, E. F. Kochs, D. Jordan, R. Ilg and M. Schabus, “EEG entropy measures indicate decrease of cortical information processing in Disorders of Consciousness”, *Clinical Neurophysiology*, vol. 127, no. 2, pp. 1419–1427, 2016. DOI: 10.1016/j.clinph.2015.07.039.
- [21] J. Claassen, A. Velazquez, E. Meyers, J. Witsch, M. C. Falo, S. Park, S. Agarwal, J. Michael Schmidt, N. D. Schiff, J. D. Sitt, L. Naccache, E. Sander Connolly and H.-P. Frey, “Bedside quantitative electroencephalography improves assessment of consciousness in comatose subarachnoid hemorrhage patients”, *Annals of Neurology*, vol. 80, no. 4, pp. 541–553, 2016. DOI: 10.1002/ana.24752.
- [22] L. Zhu, G. Cui, J. Cao, A. Cichocki, J. Zhang and C. Zhou, “A Hybrid System for Distinguishing between Brain Death and Coma Using Diverse EEG Features”, *Sensors*, vol. 19, no. 6, 1342, 2019. DOI: 10.3390/s19061342.
- [23] Y. Wang, Y. Bai, X. Xia, Y. Yang, J. He and X. Li, “Spinal cord stimulation modulates complexity of neural activities in patients with disorders of consciousness”, *International Journal of Neuroscience*, 2019. DOI: 10.1080/00207454.2019.1702543.
- [24] B. Deng, L. Liang, S. Li, R. Wang, H. Yu, J. Wang and X. Wei, “Complexity extraction of electroencephalograms in Alzheimer’s disease with weighted-permutation entropy”, *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 25, no. 4, 043105, 2015. DOI: 10.1063/1.4917013.
- [25] B. Deng, L. Cai, S. Li, R. Wang, H. Yu, Y. Chen and J. Wang, “Multivariate multi-scale weighted permutation entropy analysis of EEG complexity for Alzheimer’s disease”, *Cognitive Neurodynamics*, vol. 11, no. 3, pp. 217–231, 2017. DOI: 10.1007/s11571-016-9418-9.

- [26] D. O’Hora, S. Schinkel, M. J. Hogan, L. Kilmartin, M. Keane, R. Lai and N. Upton, “Age-Related Task Sensitivity of Frontal EEG Entropy During Encoding Predicts Retrieval”, *Brain Topography*, vol. 26, no. 4, pp. 547–557, 2013. DOI: 10.1007/s10548-013-0278-x.
- [27] L. Waschke, M. Wöstmann and J. Obleser, “States and traits of neural irregularity in the age-varying human brain”, *Scientific Reports*, vol. 7, no. 1, 17381, 2017. DOI: 10.1038/s41598-017-17766-4.
- [28] N. K. Al-Qazzaz, S. Ali, S. A. Ahmad and J. Escudero, “Classification enhancement for post-stroke dementia using fuzzy neighborhood preserving analysis with QR-decomposition”, in *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Seogwipo, South Korea: IEEE, 2017, pp. 3174–3177. DOI: 10.1109/EMBC.2017.8037531.
- [29] N. K. Al-Qazzaz, S. H. B. M. Ali, S. A. Ahmad, M. S. Islam and J. Escudero, “Discrimination of stroke-related mild cognitive impairment and vascular dementia using EEG signal analysis”, *Medical & Biological Engineering & Computing*, vol. 56, no. 1, pp. 137–157, 2018. DOI: 10.1007/s11517-017-1734-7.
- [30] F. Li, Y. Fan, X. Zhang, C. Wang, F. Hu, W. Jia and H. Hui, “Multi-Feature Fusion Method Based on EEG Signal and its Application in Stroke Classification”, *Journal of Medical Systems*, vol. 44, no. 2, 39, 2020. DOI: 10.1007/s10916-019-1517-9.
- [31] R. J. Kakumanu, A. K. Nair, R. Venugopal, A. Sasidharan, P. K. Ghosh, J. P. John, S. Mehrotra, R. Panth and B. M. Kutty, “Dissociating meditation proficiency and experience dependent EEG changes during traditional Vipassana meditation practice”, *Biological Psychology*, vol. 135, pp. 65–75, 2018. DOI: 10.1016/j.biopsycho.2018.03.004.
- [32] S. Keshmiri, H. Sumioka, J. Nakanishi and H. Ishiguro, “Bodily-Contact Communication Medium Induces Relaxed Mode of Brain Activity While Increasing Its Dynamical Complexity: A Pilot Study”, *Frontiers in Psychology*, vol. 9, 1192, 2018. DOI: 10.3389/fpsyg.2018.01192.
- [33] A. Martínez-Rodrigo, B. García-Martínez, L. Zunino, R. Alcaraz and A. Fernández-Caballero, “Multi-Lag Analysis of Symbolic Entropies on EEG Recordings for Distress Recognition”, *Frontiers in Neuroinformatics*, vol. 13, 40, 2019. DOI: 10.3389/fninf.2019.00040.

- [34] H. R. Khairuddin, A. S. Malik, W. Mumtaz, N. Kamel and L. Xia, “Analysis of EEG signals regularity in adults during video game play in 2D and 3D”, in *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Osaka, Japan: IEEE, 2013, pp. 2064–2067. DOI: 10.1109/EMBC.2013.6609938.
- [35] W. Mumtaz, Likun Xia, A. S. Malik and M. A. M. Yasin, “EEG classification of physiological conditions in 2D/3D environments using neural network”, in *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Osaka, Japan: IEEE, 2013, pp. 4235–4238. DOI: 10.1109/EMBC.2013.6610480.
- [36] S. Aydin, N. Arica, E. Ergul and O. Tan, “Classification of Obsessive Compulsive Disorder by EEG Complexity and Hemispheric Dependency Measurements”, *International Journal of Neural Systems*, vol. 25, no. 3, 1550010, 2015. DOI: 10.1142/S0129065715500100.
- [37] N. Behzadfar, S. M. P. Firoozabadi and K. Badie, “Low-Complexity Discriminative Feature Selection From EEG Before and After Short-Term Memory Task”, *Clinical EEG and Neuroscience*, vol. 47, no. 4, pp. 291–297, 2016. DOI: 10.1177/1550059416633951.
- [38] F. C. Morabito, M. Campolo, N. Mammone, M. Versaci, S. Franceschetti, F. Tagliavini, V. Sofia, D. Fatuzzo, A. Gambardella, A. Labate, L. Mumoli, G. G. Tripodi, S. Gasparini, V. Cianci, C. Sueri, E. Ferlazzo and U. Aguglia, “Deep Learning Representation from Electroencephalography of Early-Stage Creutzfeldt-Jakob Disease and Features for Differentiation from Rapidly Progressive Dementia”, *International Journal of Neural Systems*, vol. 27, no. 02, 1650039, 2017. DOI: 10.1142/S0129065716500398.
- [39] G. S. Yi, J. Wang, B. Deng and X. L. Wei, “Complexity of resting-state EEG activity in the patients with early-stage Parkinson’s disease”, *Cognitive Neurodynamics*, vol. 11, no. 2, pp. 147–160, 2017. DOI: 10.1007/s11571-016-9415-z.
- [40] R. Shalbaf, C. Brenner, C. Pang, D. M. Blumberger, J. Downar, Z. J. Daskalakis, J. Tham, R. W. Lam, F. Farzan and F. Vila-Rodriguez, “Non-linear Entropy Analysis in EEG to Predict Treatment Response to Repetitive Transcranial Magnetic Stimulation in Depression”, *Frontiers in Pharmacology*, vol. 9, 1188, 2018. DOI: 10.3389/fphar.2018.01188.

- [41] J. Kang, H. Chen, X. Li and X. Li, “EEG entropy analysis in autistic children”, *Journal of Clinical Neuroscience*, vol. 62, pp. 199–206, 2019. DOI: 10.1016/j.jocn.2018.11.027.
- [42] D. Jordan, G. Stockmanns, E. F. Kochs, S. Pilge and G. Schneider, “Electroencephalographic Order Pattern Analysis for the Separation of Consciousness and Unconsciousness”, *Anesthesiology*, vol. 109, no. 6, pp. 1014–1022, 2008. DOI: 10.1097/ALN.0b013e31818d6c55.
- [43] X. Li, S. Cui and L. J. Voss, “Using Permutation Entropy to Measure the Electroencephalographic Effects of Sevoflurane”, *Anesthesiology*, vol. 109, no. 3, pp. 448–456, 2008. DOI: 10.1097/ALN.0b013e318182a91b.
- [44] E. Olofsen, J. Sleigh and A. Dahan, “Permutation entropy of the electroencephalogram: a measure of anaesthetic drug effect”, *British Journal of Anaesthesia*, vol. 101, no. 6, pp. 810–821, 2008. DOI: 10.1093/bja/aen290.
- [45] J. Kortelainen, M. Koskinen, S. Mustola and T. Seppanen, “Effect of remifentanyl on the nonlinear electroencephalographic entropy parameters in propofol anesthesia”, in *2009 31st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS)*, Minneapolis, MN, USA: IEEE, 2009, pp. 4994–4997. DOI: 10.1109/IEMBS.2009.5334612.
- [46] A. Silva, H. Cardoso-Cruz, F. Silva, V. Galhardo and L. Antunes, “Comparison of Anesthetic Depth Indexes Based on Thalamocortical Local Field Potentials in Rats”, *Anesthesiology*, vol. 112, no. 2, pp. 355–363, 2010. DOI: 10.1097/ALN.0b013e3181ca3196.
- [47] D. Li, X. Li, Z. Liang, L. J. Voss and J. W. Sleigh, “Multiscale permutation entropy analysis of EEG recordings during sevoflurane anesthesia”, *Journal of Neural Engineering*, vol. 7, no. 4, 046010, 2010. DOI: 10.1088/1741-2560/7/4/046010.
- [48] A. Anier, T. Lipping, V. Jäntti, P. Puumala and A.-M. Huotari, “Entropy of the EEG in transition to burst suppression in deep anesthesia: Surrogate analysis”, in *2010 32nd Annual International Conference of the IEEE Engineering in Medicine and Biology (EMBS)*, Buenos Aires, Argentina: IEEE, 2010, pp. 2790–2793. DOI: 10.1109/IEMBS.2010.5626366.
- [49] A. Silva, D. A. Ferreira, C. Venâncio, A. P. Souza and L. M. Antunes, “Performance of electroencephalogram-derived parameters in prediction of depth of anaesthesia in a rabbit model”, *British Journal of Anaesthesia*, vol. 106, no. 4, pp. 540–547, 2011. DOI: 10.1093/bja/aeq407.

- [50] A. Silva, S. Campos, J. Monteiro, C. Venâncio, B. Costa, P. Guedes de Pinho and L. Antunes, “Performance of Anesthetic Depth Indexes in Rabbits under Propofol Anesthesia”, *Anesthesiology*, vol. 115, no. 2, pp. 303–314, 2011. DOI: 10.1097/ALN.0b013e318222ac02.
- [51] N. Nicolaou, S. Houris, P. Alexandrou and J. Georgiou, “Entropy Measures for Discrimination of ‘awake’ Vs ‘anaesthetized’ State in Recovery from General Anesthesia”, in *2011 33rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS)*, Boston, MA, USA: IEEE, 2011, pp. 2598–2601. DOI: 10.1109/IEMBS.2011.6090717.
- [52] Z. Liang, Y. Wang, G. Ouyang, L. J. Voss, J. W. Sleigh and X. Li, “Permutation auto-mutual information of electroencephalogram in anesthesia”, *Journal of Neural Engineering*, vol. 10, no. 2, 026004, 2013. DOI: 10.1088/1741-2560/10/2/026004.
- [53] D. Jordan, R. Ilg, V. Riedl, A. Schorer, S. Grimberg, S. Neufang, A. Omerovic, S. Berger, G. Untergehrer, C. Preibisch, E. Schulz, T. Schuster, M. Schröter, V. Spormaker, C. Zimmer, B. Hemmer, A. Wohlschläger, E. F. Kochs and G. Schneider, “Simultaneous Electroencephalographic and Functional Magnetic Resonance Imaging Indicate Impaired Cortical Top–Down Processing in Association with Anesthetic-induced Unconsciousness”, *Anesthesiology*, vol. 119, no. 5, pp. 1031–1042, 2013. DOI: 10.1097/ALN.0b013e3182a7ca92.
- [54] R. Shalhaf, H. Behnam, J. W. Sleigh, A. Steyn-Ross and L. J. Voss, “Monitoring the depth of anesthesia using entropy features and an artificial neural network”, *Journal of Neuroscience Methods*, vol. 218, no. 1, pp. 17–24, 2013. DOI: 10.1016/j.jneumeth.2013.03.008.
- [55] D. Li, Z. Liang, Y. Wang, S. Hagihira, J. W. Sleigh and X. Li, “Parameter selection in permutation entropy for an electroencephalographic measure of isoflurane anesthetic drug effect”, *Journal of Clinical Monitoring and Computing*, vol. 27, no. 2, pp. 113–123, 2013. DOI: 10.1007/s10877-012-9419-0.
- [56] G. Untergehrer, D. Jordan, S. Eyl and G. Schneider, “Effects of propofol, sevoflurane, remifentanil, and (s)-ketamine in subanesthetic concentrations on visceral and somatosensory pain-evoked potentials”, *Anesthesiology*, vol. 118, no. 2, pp. 308–317, 2013. DOI: 10.1097/ALN.0b013e318279fb21.
- [57] M. Kreuzer, E. F. Kochs, G. Schneider and D. Jordan, “Non-stationarity of EEG during wakefulness and anaesthesia: advantages of EEG permutation entropy

- monitoring”, *Journal of Clinical Monitoring and Computing*, vol. 28, no. 6, pp. 573–580, 2014. DOI: 10.1007/s10877-014-9553-y.
- [58] G. Schneider, D. Jordan, G. Schwarz, P. Bischoff, C. J. Kalkman, H. Kuppe, I. Rundshagen, A. Omerovic, M. Kreuzer, G. Stockmanns and E. F. Kochs, “Monitoring Depth of Anesthesia Utilizing a Combination of Electroencephalographic and Standard Measures”, *Anesthesiology*, vol. 120, no. 4, pp. 819–828, 2014. DOI: 10.1097/ALN.000000000000151.
- [59] A. Silva, C. Venâncio, A. L. Ortiz, A. P. Souza, P. Amorim and D. A. Ferreira, “The effect of high doses of remifentanyl in brain near-infrared spectroscopy and in electroencephalographic parameters in pigs”, *Veterinary Anaesthesia and Analgesia*, vol. 41, no. 2, pp. 153–162, 2014. DOI: 10.1111/vaa.12091.
- [60] Z. Liang, Y. Wang, X. Sun, D. Li, L. J. Voss, J. W. Sleight, S. Hagihira and X. Li, “EEG entropy measures in anesthesia”, *Frontiers in Computational Neuroscience*, vol. 9, 16, 2015. DOI: 10.3389/fncom.2015.00016.
- [61] Z. Liang, X. Duan, C. Su, L. Voss, J. Sleight and X. Li, “A Pharmacokinetics-Neural Mass Model (PK-NMM) for the Simulation of EEG Activity during Propofol Anesthesia”, *PLOS ONE*, vol. 10, no. 12, e0145959, 2015. DOI: 10.1371/journal.pone.0145959.
- [62] R. Shalhaf, H. Behnam and H. Jelveh Moghadam, “Monitoring depth of anesthesia using combination of EEG measure and hemodynamic variables”, *Cognitive Neurodynamics*, vol. 9, no. 1, pp. 41–51, 2015. DOI: 10.1007/s11571-014-9295-z.
- [63] P. J. Kim, H. G. Kim, G. J. Noh, Y. S. Koo and T. J. Shin, “Usefulness of permutation entropy as an anesthetic depth indicator in children”, *Journal of Pharmacokinetics and Pharmacodynamics*, vol. 42, no. 2, pp. 123–134, 2015. DOI: 10.1007/s10928-015-9405-5.
- [64] T. Li and P. Wen, “Depth of anaesthesia assessment based on adult electroencephalograph beta frequency band”, *Australasian Physical and Engineering Sciences in Medicine*, vol. 39, no. 3, pp. 773–781, 2016. DOI: 10.1007/s13246-016-0459-5.
- [65] A. Ranft, D. Golkowski, T. Kiel, V. Riedl, P. Kohl, G. Rohrer, J. Pientka, S. Berger, A. B. Thul, M. Maurer, C. Preibisch, C. Zimmer, G. A. Mashour, E. F. Kochs, D. Jordan and R. Ilg, “Neural Correlates of Sevoflurane-induced Unconsciousness Identified by Simultaneous Functional Magnetic Resonance Imaging

- and Electroencephalography”, *Anesthesiology*, vol. 125, no. 5, pp. 861–872, 2016. DOI: 10.1097/ALN.0000000000001322.
- [66] C. Su, Z. Liang, X. Li, D. Li, Y. Li and M. Ursino, “A Comparison of Multiscale Permutation Entropy Measures in On-Line Depth of Anesthesia Monitoring”, *PLOS ONE*, vol. 11, no. 10, e0164104, 2016. DOI: 10.1371/journal.pone.0164104.
- [67] Q. Liu, Y.-F. Chen, S.-Z. Fan, M. F. Abbod and J.-S. Shieh, “Quasi-Periodicities Detection Using Phase-Rectified Signal Averaging in EEG Signals as a Depth of Anesthesia Monitor”, *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 25, no. 10, pp. 1773–1784, 2017. DOI: 10.1109/TNSRE.2017.2690449.
- [68] A. Gaskell, D. Hight, J. Winders, G. Tran, A. Defresne, V. Bonhomme, A. Raz, J. Sleight and R. Sanders, “Frontal alpha-delta EEG does not preclude volitional response during anaesthesia: prospective cohort study of the isolated forearm technique”, *British Journal of Anaesthesia*, vol. 119, no. 4, pp. 664–673, 2017. DOI: 10.1093/bja/aex170.
- [69] H. Lee, Z. Huang, X. Liu, U. Lee and A. Hudetz, “Topographic Reconfiguration of Local and Shared Information in Anesthetic-Induced Unconsciousness”, *Entropy*, vol. 20, no. 7, 518, 2018. DOI: 10.3390/e20070518.
- [70] A. Fleischmann, S. Pilge, T. Kiel, S. Kratzer, G. Schneider and M. Kreuzer, “Substance-Specific Differences in Human Electroencephalographic Burst Suppression Patterns”, *Frontiers in Human Neuroscience*, vol. 12, 368, 2018. DOI: 10.3389/fnhum.2018.00368.
- [71] Y. Gu, Z. Liang and S. Hagihira, “Use of Multiple EEG Features and Artificial Neural Network to Monitor the Depth of Anesthesia”, *Sensors*, vol. 19, no. 11, 2499, 2019. DOI: 10.3390/s19112499.
- [72] N. Lange, S. Schleifer, M. Berndt, A.-K. Jörger, A. Wagner, S. M. Krieg, D. Jordan, M. Bretschneider, Y.-M. Ryang, B. Meyer and J. Gempt, “Permutation entropy in intraoperative ECoG of brain tumour patients in awake tumour surgery— a robust parameter to separate consciousness from unconsciousness”, *Scientific Reports*, vol. 9, no. 1, 16482, 2019. DOI: 10.1038/s41598-019-52949-1.

- [73] A. Shalbaf, M. Saffar, J. W. Sleigh and R. Shalbaf, “Monitoring the Depth of Anesthesia Using a New Adaptive Neurofuzzy System”, *IEEE Journal of Biomedical and Health Informatics*, vol. 22, no. 3, pp. 671–677, 2018. DOI: 10.1109/JBHI.2017.2709841.
- [74] A. Shalbaf, R. Shalbaf, M. Saffar and J. Sleigh, “Monitoring the level of hypnosis using a hierarchical SVM system”, *Journal of Clinical Monitoring and Computing*, 2019. DOI: 10.1007/s10877-019-00311-1.
- [75] Z. Liang, S. Shao, Z. Lv, D. Li, J. W. Sleigh, X. Li, C. Zhang and J. He, “Constructing a Consciousness Meter Based on the Combination of Non-Linear Measurements and Genetic Algorithm-Based Support Vector Machine”, *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 28, no. 2, pp. 399–408, 2020. DOI: 10.1109/TNSRE.2020.2964819.
- [76] R. D. Sanders, G. Tononi, S. Laureys and J. W. Sleigh, “Unresponsiveness \neq Unconsciousness”, *Anesthesiology*, vol. 116, no. 4, pp. 946–959, 2012. DOI: 10.1097/ALN.0b013e318249d0a7.
- [77] S. Berger, G. Schneider, E. F. Kochs and D. Jordan, “Permutation Entropy: Too Complex a Measure for EEG Time Series?”, *Entropy*, vol. 19, no. 12, 692, 2017. DOI: 10.3390/E19120692.
- [78] S. Berger, A. Kravtsiv, G. Schneider and D. Jordan, “Teaching Ordinal Patterns to a Computer: Efficient Encoding Algorithms Based on the Lehmer Code”, *Entropy*, vol. 21, no. 10, 1023, 2019. DOI: 10.3390/e21101023.
- [79] K. E. Iverson, *A Programming Language*. New York City, NY, USA: Wiley, 1962.
- [80] D. E. Knuth, “Two Notes on Notation”, *The American Mathematical Monthly*, vol. 99, no. 5, pp. 403–422, 1992. DOI: 10.2307/2325085.
- [81] A. Gnedin and G. Olshanski, “The two-sided infinite extension of the Mallows model for random permutations”, *Advances in Applied Mathematics*, vol. 48, no. 5, pp. 615–639, 2012. DOI: 10.1016/J.AAM.2012.01.001.
- [82] G. Cantor, “Ueber die einfachen Zahlensysteme”, *Zeitschrift für Mathematik und Physik*, vol. 14, pp. 121–128, 1869. URL: <https://books.google.de/books?id=T1NaAAAAYAAJ>.
- [83] G. B. Arfken, H. J. Weber and F. E. Harris, *Mathematical Methods for Physicists*, 7th ed. Amsterdam, The Netherlands, EU: Elsevier, 2013. DOI: 10.1016/C2009-0-30629-7.

- [84] C. D. Meyer, *Matrix Analysis and Applied Linear Algebra*. Philadelphia, PA, USA: SIAM, 2000.
- [85] C. E. Shannon, “A Mathematical Theory of Communication”, *Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948. DOI: 10.1002/j.1538-7305.1948.tb01338.x.
- [86] C. E. Shannon, “A Mathematical Theory of Communication”, *Bell System Technical Journal*, vol. 27, no. 4, pp. 623–656, 1948. DOI: 10.1002/j.1538-7305.1948.tb00917.x.
- [87] M. Tribus and E. C. McIrvine, “Energy and Information”, *Scientific American*, vol. 225, no. 3, pp. 179–188, 1971. DOI: 10.1038/scientificamerican0971-179.
- [88] T. Schreiber, “Measuring Information Transfer”, *Physical Review Letters*, vol. 85, no. 2, pp. 461–464, 2000. DOI: 10.1103/PhysRevLett.85.461.
- [89] A. Kaiser and T. Schreiber, “Information transfer in continuous processes”, *Physica D: Nonlinear Phenomena*, vol. 166, no. 1-2, pp. 43–62, 2002. DOI: 10.1016/S0167-2789(02)00432-3.
- [90] J.-P. Eckmann and D. Ruelle, “Ergodic theory of chaos and strange attractors”, *Reviews of Modern Physics*, vol. 57, no. 3, pp. 617–656, 1985. DOI: 10.1103/RevModPhys.57.617.
- [91] H. Kantz and T. Schreiber, *Nonlinear Time Series Analysis*, 2nd ed. Cambridge, UK: Cambridge University Press, 2004.
- [92] P. Faure and H. Korn, “Is there chaos in the brain? I. Concepts of nonlinear dynamics and methods of investigation”, *Comptes Rendus de l’Académie des Sciences - Série III, Sciences de la Vie*, vol. 324, no. 9, pp. 773–793, 2001. DOI: 10.1016/S0764-4469(01)01377-4.
- [93] H. Korn and P. Faure, “Is there chaos in the brain? II. Experimental evidence and related models”, *Comptes Rendus Biologies*, vol. 326, no. 9, pp. 787–840, 2003. DOI: 10.1016/j.crvi.2003.09.011.
- [94] L. Arnold, *Random Dynamical Systems*. Berlin/Heidelberg, Germany, EU: Springer, 1998. DOI: 10.1007/978-3-662-12878-7.
- [95] A. Katok and B. Hasselblatt, *Introduction to the Modern Theory of Dynamical Systems*. Cambridge, UK: Cambridge University Press, 1997.
- [96] L. O. Chua, C. A. Desoer and E. S. Kuh, *Linear and Nonlinear Circuits*. New York City, NY, USA: McGraw-Hill, 1987.

- [97] B. van der Pol, “On “Relaxation-Oscillations””, *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 978–992, 1926. DOI: 10.1080/14786442608564127.
- [98] E. N. Lorenz, “Deterministic Nonperiodic Flow”, *Journal of the Atmospheric Sciences*, vol. 20, no. 2, pp. 130–141, 1963. DOI: 10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2.
- [99] D. Ruelle and F. Takens, “On the Nature of Turbulence”, *Communications in Mathematical Physics*, vol. 20, no. 3, pp. 167–192, 1971. URL: <https://projecteuclid.org/euclid.cmp/1103857186>.
- [100] T.-Y. Li and J. A. Yorke, “Period Three Implies Chaos”, *The American Mathematical Monthly*, vol. 82, no. 10, pp. 985–992, 1975. DOI: 10.2307/2318254.
- [101] N. H. Packard, J. P. Crutchfield, J. D. Farmer and R. S. Shaw, “Geometry from a Time Series”, *Physical Review Letters*, vol. 45, no. 9, pp. 712–716, 1980. DOI: 10.1103/PhysRevLett.45.712.
- [102] H. Whitney, “Differentiable Manifolds”, *The Annals of Mathematics*, vol. 37, no. 3, pp. 645–680, 1936. DOI: 10.2307/1968482.
- [103] T. Sauer, J. A. Yorke and M. Casdagli, “Embedology”, *Journal of Statistical Physics*, vol. 65, no. 3–4, pp. 579–616, 1991. DOI: 10.1007/BF01053745.
- [104] F. Takens, “Detecting strange attractors in turbulence”, in *Dynamical Systems and Turbulence, Warwick 1980*, Berlin, Heidelberg: Springer Berlin Heidelberg, 1981, pp. 366–381.
- [105] L.-S. Young, “Dimension, entropy and Lyapunov exponents”, *Ergodic Theory and Dynamical Systems*, vol. 2, no. 1, pp. 109–124, 1982. DOI: 10.1017/S0143385700009615.
- [106] H. D. I. Abarbanel, R. Brown, J. J. Sidorowich and L. S. Tsimring, “The analysis of observed chaotic data in physical systems”, *Reviews of Modern Physics*, vol. 65, no. 4, pp. 1331–1392, 1993. DOI: 10.1103/RevModPhys.65.1331.
- [107] K. Keller, M. Sinn and J. Emonds, “Time Series From the Ordinal Viewpoint”, *Stochastics and Dynamics*, vol. 7, no. 2, pp. 247–272, 2007. DOI: 10.1142/S0219493707002025.
- [108] B. Fadlallah, B. Chen, A. Keil and J. Príncipe, “Weighted-permutation entropy: A complexity measure for time series incorporating amplitude information”, *Physical Review E*, vol. 87, no. 2, 022911, 2013. DOI: 10.1103/PhysRevE.87.022911.

- [109] F. C. Morabito, D. Labate, F. La Foresta, A. Bramanti, G. Morabito and I. Palamara, “Multivariate Multi-Scale Permutation Entropy for Complexity Analysis of Alzheimer’s Disease EEG”, *Entropy*, vol. 14, no. 12, pp. 1186–1202, 2012. DOI: 10.3390/e14071186.
- [110] H. Azami and J. Escudero, “Amplitude-aware permutation entropy: Illustration in spike detection and signal segmentation”, *Computer Methods and Programs in Biomedicine*, vol. 128, pp. 40–51, 2016. DOI: 10.1016/j.cmpb.2016.02.008.
- [111] A. M. Unakafov and K. Keller, “Conditional entropy of ordinal patterns”, *Physica D: Nonlinear Phenomena*, vol. 269, pp. 94–102, 2014. DOI: <https://doi.org/10.1016/j.physd.2013.11.015>.
- [112] J. R. King, J. D. Sitt, F. Faugeras, B. Rohaut, I. El Karoui, L. Cohen, L. Naccache and S. Dehaene, “Information sharing in the brain indexes consciousness in noncommunicative patients”, *Current Biology*, vol. 23, no. 19, pp. 1914–1919, 2013. DOI: 10.1016/j.cub.2013.07.075.
- [113] M. Staniek and K. Lehnertz, “Symbolic Transfer Entropy”, *Physical Review Letters*, vol. 100, no. 15, 158101, 2008. DOI: 10.1103/PhysRevLett.100.158101.
- [114] A. Groth, “Visualization of coupling in time series by order recurrence plots”, *Physical Review E*, vol. 72, no. 4, 046220, 2005. DOI: 10.1103/PhysRevE.72.046220.
- [115] C. Bandt and F. Shiha, “Order patterns in time series”, *Journal of Time Series Analysis*, vol. 28, no. 5, pp. 646–665, 2007. DOI: 10.1111/j.1467-9892.2007.00528.x.
- [116] J. M. Amigó, *Permutation Complexity in Dynamical Systems*. Berlin/Heidelberg, Germany, EU: Springer, 2010. DOI: 10.1007/978-3-642-04084-9.
- [117] C. Bandt, “Small Order Patterns in Big Time Series: A Practical Guide”, *Entropy*, vol. 21, no. 6, 613, 2019. DOI: 10.3390/e21060613.
- [118] L. Zunino, F. Olivares, F. Scholkmann and O. A. Rosso, “Permutation entropy based time series analysis: Equalities in the input signal can lead to false conclusions”, *Physics Letters A*, vol. 381, no. 22, pp. 1883–1892, 2017. DOI: 10.1016/J.PHYSLETA.2017.03.052.
- [119] R. Vicente, M. Wibral, M. Lindner and G. Pipa, “Transfer entropy—a model-free measure of effective connectivity for the neurosciences”, *Journal of Computational Neuroscience*, vol. 30, no. 1, pp. 45–67, 2011. DOI: 10.1007/s10827-010-0262-3.

- [120] M. Lindner, R. Vicente, V. Priesemann and M. Wibral, “TRENTOOL: A Matlab open source toolbox to analyse information flow in time series data with transfer entropy”, *BMC Neuroscience*, vol. 12, no. 1, 119, 2011. DOI: 10.1186/1471-2202-12-119.
- [121] P. Wollstadt, M. Martínez-Zarzuela, R. Vicente, F. J. Díaz-Pernas and M. Wibral, “Efficient Transfer Entropy Analysis of Non-Stationary Neural Time Series”, *PLoS ONE*, vol. 9, no. 7, e102833, 2014. DOI: 10.1371/journal.pone.0102833.
- [122] M. Staniek and K. Lehnertz, “Symbolic transfer entropy: inferring directionality in biosignals”, *Biomedizinische Technik/Biomedical Engineering*, vol. 54, no. 6, pp. 323–328, 2009. DOI: 10.1515/BMT.2009.040.
- [123] S.-W. Ku, U. Lee, G.-J. Noh, I.-G. Jun and G. A. Mashour, “Preferential Inhibition of Frontal-to-Parietal Feedback Connectivity Is a Neurophysiologic Correlate of General Anesthesia in Surgical Patients”, *PLoS ONE*, vol. 6, no. 10, e25155, 2011. DOI: 10.1371/journal.pone.0025155.
- [124] M. Martini, T. A. Kranz, T. Wagner and K. Lehnertz, “Inferring directional interactions from transient signals with symbolic transfer entropy”, *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, vol. 83, no. 1, 011919, 2011. DOI: 10.1103/PhysRevE.83.011919.
- [125] S. Blain-Moraes, G. A. Mashour, H. Lee, J. E. Huggins and U. Lee, “Altered cortical communication in amyotrophic lateral sclerosis”, *Neuroscience Letters*, vol. 543, pp. 172–176, 2013. DOI: 10.1016/j.neulet.2013.03.028.
- [126] H. Dickten and K. Lehnertz, “Identifying delayed directional couplings with symbolic transfer entropy”, *Physical Review E*, vol. 90, no. 6, 062706, 2014. DOI: 10.1103/PhysRevE.90.062706.
- [127] G. Untergerher, D. Jordan, E. F. Kochs, R. Ilg and G. Schneider, “Fronto-parietal connectivity is a non-static phenomenon with characteristic changes during unconsciousness”, *PLoS ONE*, vol. 9, no. 1, e87498, 2014. DOI: 10.1371/journal.pone.0087498.
- [128] J. M. Amigó, R. Monetti, N. Tort-Colet and M. V. Sanchez-Vives, “Infragranular layers lead information flow during slow oscillations according to information directionality indicators”, *Journal of Computational Neuroscience*, vol. 39, no. 1, pp. 53–62, 2015. DOI: 10.1007/s10827-015-0563-7.

- [129] K. Lehnertz and H. Dickten, “Assessing directionality and strength of coupling through symbolic analysis: an application to epilepsy patients”, *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 373, no. 2034, 20140094, 2015. DOI: 10.1098/rsta.2014.0094.
- [130] S. Dimitriadis, Y. Sun, N. Laskaris, N. Thakor and A. Bezerianos, “Revealing Cross-Frequency Causal Interactions During a Mental Arithmetic Task Through Symbolic Transfer Entropy: A Novel Vector-Quantization Approach”, *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 24, no. 10, pp. 1017–1028, 2016. DOI: 10.1109/TNSRE.2016.2516107.
- [131] U. Lee, S. Blain-Moraes and G. A. Mashour, “Assessing levels of consciousness with symbolic analysis”, *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 373, no. 2034, 20140117, 2015. DOI: 10.1098/rsta.2014.0117.
- [132] F. Vallone, E. Vannini, A. Cintio, M. Caleo and A. Di Garbo, “Time evolution of interhemispheric coupling in a model of focal neocortical epilepsy in mice”, *Physical Review E*, vol. 94, no. 3, 032409, 2016. DOI: 10.1103/PhysRevE.94.032409.
- [133] D. Pal, B. H. Silverstein, H. Lee and G. A. Mashour, “Neural Correlates of Wakefulness, Sleep, and General Anesthesia”, *Anesthesiology*, vol. 125, no. 5, pp. 929–942, 2016. DOI: 10.1097/ALN.0000000000001342.
- [134] D. Li, V. S. Hambrecht-Wiedbusch and G. A. Mashour, “Accelerated Recovery of Consciousness after General Anesthesia Is Associated with Increased Functional Brain Connectivity in the High-Gamma Bandwidth”, *Frontiers in Systems Neuroscience*, vol. 11, 16, 2017. DOI: 10.3389/fnsys.2017.00016.
- [135] D. Rathee, H. Cecotti and G. Prasad, “Propofol-induced sedation diminishes the strength of frontal-parietal-occipital EEG network”, in *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Seogwipo, South Korea: IEEE, 2017, pp. 4463–4466. DOI: 10.1109/EMBC.2017.8037847.
- [136] Y. Gao, L. Ren, R. Li and Y. Zhang, “Electroencephalogram-Electromyography Coupling Analysis in Stroke Based on Symbolic Transfer Entropy”, *Frontiers in Neurology*, vol. 8, no. JAN, 716, 2018. DOI: 10.3389/fneur.2017.00716.
- [137] L. Kuhlmann, K. Lehnertz, M. P. Richardson, B. Schelter and H. P. Zaveri, “Seizure prediction — ready for a new era”, *Nature Reviews Neurology*, vol. 14, no. 10, pp. 618–630, 2018. DOI: 10.1038/s41582-018-0055-2.

- [138] S. Stefan, B. Schorr, A. Lopez-Rolon, I. T. Kolassa, J. P. Shock, M. Rosenfelder, S. Heck and A. Bender, “Consciousness Indexing and Outcome Prediction with Resting-State EEG in Severe Disorders of Consciousness”, *Brain Topography*, vol. 31, no. 5, pp. 848–862, 2018. DOI: 10.1007/s10548-018-0643-x.
- [139] F. Zubler, A. Seiler, T. Horvath, C. Roth, S. Miano, C. Rummel, H. Gast, L. Nobili, K. A. Schindler and C. L. Bassetti, “Stroke causes a transient imbalance of interhemispheric information flow in EEG during non-REM sleep”, *Clinical Neurophysiology*, vol. 129, no. 7, pp. 1418–1426, 2018. DOI: 10.1016/J.CLINPH.2018.03.038.
- [140] S. Ye, K. Kitajo and K. Kitano, “Information-theoretic approach to detect directional information flow in EEG signals induced by TMS”, *Neuroscience Research*, 2019. DOI: 10.1016/j.neures.2019.09.003.
- [141] M. Wibral, N. Pampu, V. Priesemann, F. Siebenhühner, H. Seiwert, M. Lindner, J. T. Lizier and R. Vicente, “Measuring Information-Transfer Delays”, *PLoS ONE*, vol. 8, no. 2, e55809, 2013. DOI: 10.1371/journal.pone.0055809.
- [142] M. Staniek, “Symbolische Transferentropie: Charakterisierung gerichteter Interaktionen in nichtlinearen dynamischen Netzwerken”, Dissertation, Universität Bonn, Bonn, Germany, EU, 2010. URL: <http://hss.ulb.uni-bonn.de/2010/2369/2369.htm>.
- [143] B. Frank, B. Pompe, U. Schneider and D. Hoyer, “Permutation entropy improves fetal behavioural state classification based on heart rate analysis from biomagnetic recordings in near term fetuses”, *Medical & Biological Engineering & Computing*, vol. 44, no. 3, pp. 179–187, 2006. DOI: 10.1007/s11517-005-0015-z.
- [144] M. Zanin, L. Zunino, O. A. Rosso and D. Papo, “Permutation Entropy and Its Main Biomedical and Econophysics Applications: A Review”, *Entropy*, vol. 14, no. 8, pp. 1553–1577, 2012. DOI: 10.3390/e14081553.
- [145] V. A. Unakafova and K. Keller, “Efficiently Measuring Complexity on the Basis of Real-World Data”, *Entropy*, vol. 15, no. 12, pp. 4392–4415, 2013. DOI: 10.3390/e15104392.
- [146] J. M. Amigó, K. Keller and V. A. Unakafova, “Ordinal symbolic analysis and its application to biomedical recordings”, *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 373, no. 2034, 20140091, 2014. DOI: 10.1098/rsta.2014.0091.

- [147] A. M. Unakafov and K. Keller, “Change-Point Detection Using the Conditional Entropy of Ordinal Patterns”, *Entropy*, vol. 20, no. 9, 709, 2018. DOI: 10.3390/e20090709.
- [148] M. Staniek and K. Lehnertz, “Parameter selection for permutation entropy measurements”, *International Journal of Bifurcation and Chaos*, vol. 17, no. 10, pp. 3729–3733, 2007. DOI: 10.1142/S0218127407019652.
- [149] C.-A. Laisant, “Sur la numération factorielle, application aux permutations”, *Bulletin de la Société Mathématique de France*, vol. 16, pp. 176–183, 1888. DOI: 10.24033/bsmf.378.
- [150] D. H. Lehmer, “Teaching combinatorial tricks to a computer”, in *Proceedings of Symposia in Applied Mathematics*, vol. 10, Providence, RI, USA: American Mathematical Society, 1960, pp. 179–193. DOI: 10.1090/psapm/010.
- [151] D. E. Knuth, “Structured Programming with go to Statements”, *ACM Computing Surveys*, vol. 6, no. 4, pp. 261–301, 1974. DOI: 10.1145/356635.356640.
- [152] T. E. Oliphant, “Python for Scientific Computing”, *Computing in Science & Engineering*, vol. 9, no. 3, pp. 10–20, 2007. DOI: 10.1109/MCSE.2007.58.
- [153] J. W. Eaton, “GNU Octave and reproducible research”, *Journal of Process Control*, vol. 22, no. 8, pp. 1433–1438, 2012. DOI: 10.1016/j.jprocont.2012.04.006.
- [154] *IEEE 754-1985: IEEE Standard for Binary Floating-Point Arithmetic*. New York City, NY, USA: IEEE Computer Society, 1985. DOI: 10.1109/IEEESTD.1985.82928.
- [155] *IEEE 754-2008: IEEE Standard for Floating-Point Arithmetic*. New York City, NY, USA: IEEE Computer Society, 2008. DOI: 10.1109/IEEESTD.2008.4610935.
- [156] D. Goldberg, “What every computer scientist should know about floating-point arithmetic”, *ACM Computing Surveys*, vol. 23, no. 1, pp. 5–48, 1991. DOI: 10.1145/103162.103163.
- [157] G. Marsaglia, “Xorshift RNGs”, *Journal of Statistical Software*, vol. 8, 14, 2003. DOI: 10.18637/jss.v008.i14.
- [158] M. Frigo and S. Johnson, “The Design and Implementation of FFTW3”, *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005. DOI: 10.1109/JPROC.2004.840301.

- [159] T. St. Denis and G. Rose, *BigNum Math: Implementing Cryptographic Multiple Precision Arithmetic*. Rockland, MA, USA: Syngress Publishing, 2006. DOI: 10.1016/B978-1-59749-112-9.X5000-X.
- [160] R. P. Brent and P. Zimmermann, *Modern Computer Arithmetic*. Cambridge, UK: Cambridge University Press, 2010. DOI: 10.1017/CB09780511921698.
- [161] P. L. Nunez and R. Srinivasan, *Electric Fields of the Brain: The Neurophysics of EEG*, 2nd ed. New York City, NY, USA: Oxford University Press, 2006.
- [162] H. H. Jasper, “Report of the Committee on Methods of Clinical Examination in Electroencephalography”, *Electroencephalography and Clinical Neurophysiology*, vol. 10, no. 2, pp. 370–375, 1958. DOI: 10.1016/0013-4694(58)90053-1.
- [163] M. G. Terzano, L. Parrino, A. Sherieri, R. Chervin, S. Chokroverty, C. Guilleminault, M. Hirshkowitz, M. Mahowald, H. Moldofsky, A. C. Rosa, R. Thomas and A. Walters, “Atlas, rules, and recording techniques for the scoring of cyclic alternating pattern (CAP) in human sleep”, *Sleep Medicine*, vol. 2, no. 6, pp. 537–553, 2001. DOI: 10.1016/S1389-9457(01)00149-6.
- [164] D. Millett, “Hans Berger: From Psychic Energy to the EEG”, *Perspectives in Biology and Medicine*, vol. 44, no. 4, pp. 522–542, 2001. DOI: 10.1353/pbm.2001.0070.
- [165] H. Berger, *Psyche*. Jena, Germany, EU: Fischer, 1940.
- [166] H. Berger, “Über das Elektrenkephalogramm des Menschen”, *Archiv für Psychiatrie und Nervenkrankheiten*, vol. 87, no. 1, pp. 527–570, 1929. DOI: 10.1007/BF01797193.
- [167] E. D. Adrian and B. H. C. Matthews, “The Berger Rhythm: Potential Changes from the Occipital Lobes in Man”, *Brain*, vol. 57, no. 4, pp. 355–385, 1934. DOI: 10.1093/brain/57.4.355.
- [168] G. Buzsáki, *Rhythms of the brain*. New York City, NY, USA: Oxford University Press, 2006.
- [169] M. Riedl, A. Müller and N. Wessel, “Practical considerations of permutation entropy”, *The European Physical Journal Special Topics*, vol. 222, no. 2, pp. 249–262, 2013. DOI: 10.1140/epjst/e2013-01862-7.

- [170] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng and H. E. Stanley, “PhysioBank, PhysioToolkit, and PhysioNet”, *Circulation*, vol. 101, no. 23, e215–e220, 2000. DOI: 10.1161/01.CIR.101.23.e215.
- [171] B. Kemp, A. Värri, A. C. Rosa, K. D. Nielsen and J. Gade, “A simple format for exchange of digitized polygraphic recordings”, *Electroencephalography and clinical Neurophysiology*, vol. 82, no. 5, pp. 391–393, 1992. DOI: 10.1016/0013-4694(92)90009-7.
- [172] A. A. G. Requicha, “The Zeros of Entire Functions: Theory and Engineering Applications”, *Proceedings of the IEEE*, vol. 68, no. 3, pp. 308–328, 1980. DOI: 10.1109/PROC.1980.11644.
- [173] A. H. Shoeb, “Application of Machine Learning to Epileptic Seizure Onset Detection and Treatment”, Dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA, 2009. URL: <https://hdl.handle.net/1721.1/54669>.
- [174] B. Kedem, “Spectral Analysis and Discrimination by Zero-Crossings”, *Proceedings of the IEEE*, vol. 74, no. 11, pp. 1477–1493, 1986. DOI: 10.1109/PROC.1986.13663.
- [175] S. O. Rice, “Statistical Properties of a Sine Wave Plus Random Noise”, *Bell System Technical Journal*, vol. 27, no. 1, pp. 109–157, 1948. DOI: 10.1002/j.1538-7305.1948.tb01334.x.
- [176] C. E. Shannon, “Communication in the Presence of Noise”, *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, 1949. DOI: 10.1109/JRPROC.1949.232969.
- [177] N. Sulaiman, Mohd Nasir Taib, Siti Armiza Mohd Aris, Noor Hayatee Abdul Hamid, S. Lias and Zunairah Haji Murat, “Stress features identification from EEG signals using EEG Asymmetry & Spectral Centroids techniques”, in *2010 IEEE EMBS Conference on Biomedical Engineering and Sciences (IECBES)*, IEEE, 2010, pp. 417–421. DOI: 10.1109/IECBES.2010.5742273.
- [178] N. Sulaiman, M. N. Taib, S. Lias, Z. H. Murat, S. A. M. Aris and N. H. A. Hamid, “EEG-based Stress Features Using Spectral Centroids Technique and k-Nearest Neighbor Classifier”, in *2011 UkSim 13th International Conference on Computer Modelling and Simulation*, IEEE, 2011, pp. 69–74. DOI: 10.1109/UKSIM.2011.23.
- [179] P. C. Petrantonakis and L. J. Hadjileontiadis, “Emotion Recognition From EEG Using Higher Order Crossings”, *IEEE Transactions on Information Technology in Biomedicine*, vol. 14, no. 2, pp. 186–197, 2010. DOI: 10.1109/TITB.2009.2034649.

Bibliography

- [180] N. R. Burch, W. J. Nettleton, J. Sweeney and R. J. Edwards, “Period Analysis of the Electroencephalogram on a General-Purpose Digital Computer”, *Annals of the New York Academy of Sciences*, vol. 115, no. 2, pp. 827–843, 1964. DOI: 10.1111/j.1749-6632.1964.tb00061.x.
- [181] J. M. Amigó, S. Zambrano and M. A. F. Sanjuán, “True and false forbidden patterns in deterministic and random dynamics”, *Europhysics Letters*, vol. 79, no. 5, p. 50 001, 2007. DOI: 10.1209/0295-5075/79/50001.
- [182] M. A. Colombo, M. Napolitani, M. Boly, O. Gosseries, S. Casarotto, M. Rosanova, J.-F. Brichant, P. Boveroux, S. Rex, S. Laureys, M. Massimini, A. Chiaregato and S. Sarasso, “The spectral exponent of the resting EEG indexes the presence of consciousness during unresponsiveness induced by propofol, xenon, and ketamine”, *NeuroImage*, vol. 189, pp. 631–644, 2019. DOI: 10.1016/j.neuroimage.2019.01.024.