universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

A Python Toolbox for Neuronal Manifold Learning
Easy access to the NC-MCM framework

verfasst von | submitted by

Michael Hofer BSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2025

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 875
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Bioinformatik
Betreut von | Supervisor:	Univ.-Prof. Dipl.-Ing. Dr. -Ing. Moritz Grosse-Wentrup

Contents

List of Figures

List of Tables

1 Abstract (deutsch)

Eine zentrale Herausforderung der modernen Neurowissenschaft besteht in der Interpretation neuronaler Aktivitäten. Das Neuro-Kognitive Multilevel Causal Modeling (*NC-MCM*) schließt diese Lücke durch die Modellierung kognitiver Prozesse als Markov-Prozesse erster Ordnung. Diese Arbeit stellt ein *Python*-Paket vor, das die Anwendung des *NC-MCM*-Frameworks zur Analyse neuronaler und Verhaltensdaten ermöglicht. Es umfasst Visualisierungstools (kognitive Graphen, Zeitreihen), zwei statistische Tests zur Bewertung von Markov-Prozessen, deren Methodik detailliert in der begleitenden Publikation beschrieben wird, sowie die Integration von *scikit-learn*-Klassifikatoren und Dimensionalitätsreduktion für Manifold-Visualisierungen. Das Paket bietet eine umfassende Dokumentation zur Unterstützung der Nutzer und erleichtert durch die Nutzung bekannter Bibliotheken wie *numpy* sowie die Bereitstellung auf *GitHub* und *PyPI* den Zugang und die Weiterentwicklung. Eine Power-Analyse und Fallstudien veranschaulichen das Potenzial des Pakets, neue Erkenntnisse über die Dynamik des Gehirns zu gewinnen.

Abstract 2

A fundamental challenge in neuroscience is interpreting neuronal activity. The Neuro-Cognitive Multilevel Causal Modeling (*NC-MCM*) framework addresses this by modeling cognition as a first-order Markov process. This thesis introduces a *Python*-based toolbox to apply the *NC-MCM* framework for analyzing neuronal and behavioral data. Key features include visualizations (cognitive graphs, time series), two statistical tests for assessing Markov properties, with their methodologies thoroughly described in the accompanying publication, integration of *scikit-learn* classifiers for behavioral predictions, and dimensionality reduction for manifold visualizations. The package includes comprehensive documentation to guide users and is built with widely used libraries like *numpy*. Hosted on *GitHub* and *PyPI*, it ensures accessibility and fosters collaboration. Case studies demonstrate its potential to advance our understanding of brain dynamics.

Introduction 3

3.1 Background

At the beginning of the 20th century Santiago Ramón y Cajal created the first drawings of neurons using a silver staining technique called Golgi's method [3]. Using Camillo Golgi's staining technique, Cajal discovered that the nervous system is composed of billions of individual neurons that communicate via synapses, a concept that became a cornerstone of modern neuroscience (see Figure 3.1). Since then neuroscience has come a long way to understand the inner workings of the brain. For *Drosophila melanogaster* researcher were able to create conceptual circuit models of visual guidance control mechanisms using only a few simulated neurons [4]. "Rewired" ferrets show the brain's remarkable ability to adapt and reorganize in response to sensory deprivation or altered sensory inputs. By rewiring the auditory pathways of ferrets to process visual information instead, scientists have revealed the brain's capacity for cross-modal plasticity in the development of corresponding brain regions and their processing [5]. These studies highlight how neurons are organized into effective circuits, but also the dynamic nature of neuronal circuits and their capacity for reorganization in response to environmental changes.

It has been shown that so-called place- and grid cells encode for highly selective stimuli within the brain. The former are neurons located in the entorhinal cortex and exhibit spatially periodic firing patterns, forming a hexagonal grid-like representation of the environment and are thought to enable organisms to navigate through complex spatial environments with precision [7, 8]. The latter are predominantly found in the hippocampus, which fires selectively when an animal occupies a specific location within its environment. They are thought to be essential for episodic memory formation and spatial cognition [8, 9, 10]. These findings were crucial for the current understanding of the mechanism for episodic memory formation [9, 10] in rodents [11, 12] and primates [13]. Encoding of more complex stimuli in single cells has been debated a lot among neuroscientists as in the jokingly called grandmother-cells [14, 15, 16, 17]. These cells are said to respond selectively to highly complex and specific stimuli, such as faces or specific individuals. All these important findings helped us to understand how the integration of information in the brain works.

Unlike previously mentioned highly selective sensory integration [7, 8, 14, 15, 16], most sensory stimuli, behavior and movements are thought to be encoded by larger ensembles of neurons [18, 19, 20]. Research on mice brains shows us that visual signals are propagated across hierarchically organized cortical and thalamic visual areas [21]. The traditional perspective posited a strict segregation in the visual cortex between the processing of form and motion. Recent investigations have unveiled a higher degree of interconnectedness within the visual cortex than previously conceived. Specifically, sensory areas such as V1 show

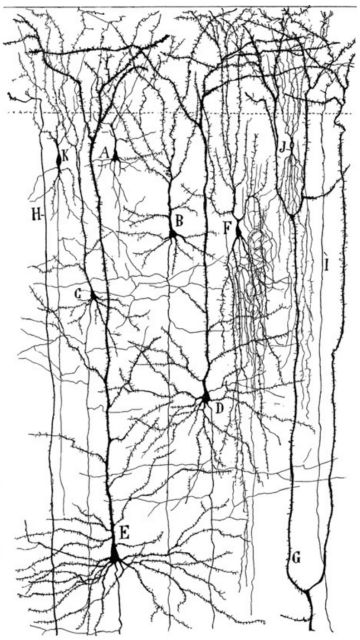


Figure 3.1: Adapted from *Estudios sobre la cortexa cerebral humana* by Ramon y Cayal et al. 1899: A drawing of the human cerebral cortex using the Golgi method, depicting interconnected pyramidal cells. This image was published by Santiago Ramón y Cajal in 1899 [6].

conjunctive, multi-modal tuning, suggesting a distributed encoding pattern across cortical areas [21, 22, 23, 24, 25]. Allen et al. demonstrated that motivational states such as thirst exert brain-wide modulatory effects on neuronal populations in mice [26]. Additionally, they illustrated how decision-making processes exhibit brain-wide task-related cell activity, which is regulated by the premotor cortex [27]. Visual information has been shown to be integrated with movement signals in the posterior cortex. This overlapping of information was strongest in the posterior parietal cortex, which indicates shifted transitions of retinotopic boundaries [22]. It is known that even simple reaching movements involve the activity of a large population of cells in the motor cortex in macaques [28]. Additionally, rhythmic EEG activities in the brain, often referred to as brain waves, are proposed to facilitate information transfer among brain regions [29, 30]. These and other recent studies suggest that even the encoding of variables, simpler actions, movements and decision-making is more distributed than modular across the cortex [19, 20, 22, 26, 27, 31, 32]. This distributed encoding observed within cortical regions [18] and across the entire brain [29, 30] underscores the importance of integrating information across the brain for a more comprehensive understanding of its inner workings.

3.2 Motivation

To truly grasp the complexities of how we think and act, it seems to be imperative to delve into ensemble and brain-wide activities. Explaining how neuronal activity gives rise to cognition arguably remains the most significant challenge in cognitive neuroscience.

Building on the knowledge of a wider encoding of variables in the brain, studies have employed functional magnetic resonance imaging (fMRI). The number of publications on fMRI studies has been exploding since the introduction of machine learning (ML) into the field. An extensive systematic review on the use of ML and deep learning (DL) in fMRI [34], including 241 studies, shows that simple ML methods like support vector machine (SVM) or ensemble classifiers are the most commonly used. In recent history, the use of deep learning is on the rise, with the first publications included in the study from 2013 [35, 36], whereas the earliest ML studies included are from 2004 [37]. Among the DL methods convolutional neural networks (CNN) [35] and variational auto encoders (VAE) [36] are the most used. Nearly half of all the studies focus on medical outcome detection, with the rest mainly focusing on connectome analysis [38] or functional dynamics in the resting state [39].

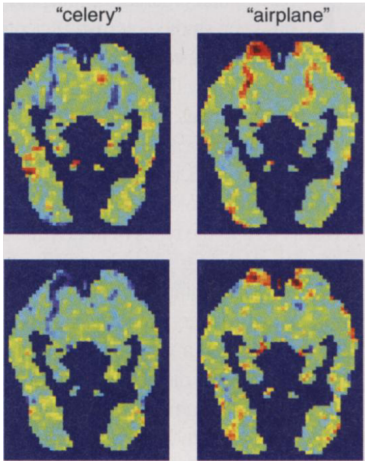


Figure 3.2: Adapted from *Predicting Human Brain Activity Associated with the Meanings of Nouns* by Tom M. Mitchell et al. 2008: The image displays predicted voxel activation signatures for the 2 left-out words (top panel), and the observed fMRI image (bottom panels). Each panel represents a horizontal brain slice [33].

Some studies utilize whole-brain fMRI data to predict mental states in humans [33, 40]. A study in 2008 aimed at predicting brain states from semantic features presented participants with five word-picture pairs from each of 12 categories. Using cross-validation, the researchers repeatedly trained a model on 58 of the 60 word-picture pairs to predict fMRI images and match them to the correct word or picture. While a 0.5 accuracy for the two left-out images would be at chance level, the researchers achieved a mean accuracy of 0.72, being significantly higher than the threshold of 0.62 (p=0.05), with significant results for each participant (see Figure 3.2) [33]. More recent work [41] on the

semantic reconstruction of language from fMRI data shows promising results for continuous language decoding, measured by BERTScores—an evaluation metric used to assess the quality of text generation systems. The researchers used a special toolkit called *pycortex* [42] to map the 3D fMRI data onto a surface flatmap, which served as the training input for the ML model. The authors note that there are challenges in differentiating between a stimulus and its paraphrases since their inputs are directly related to each other. That is why they argue that motor features could provide more control over the desired decoder output, as their correlation to perception and memory is likely to be weaker. Such studies, among others, highlight the potential of machine learning techniques in uncovering cognitive processes and the underlying brain mechanisms, while also suggesting the differences in which sensory and motor inputs are integrated and processed.

While fMRI imaging is reliable and widely used, new and improved imaging technologies [20, 43, 44, 45, 46] enable neuroscientists to record neuronal activity with greater detail and percision, as they are matching the broad spatial coverage of fMRI essential for understanding the whole brain [46]. Additionally, new methods for neuronal information processing, such as the previously mentioned ML methods, enhance sensitivity and the ability to extract rich information from patterns of neuronal activity [40, 34].

The increasing availability of large-scale neuroscience datasets provides an ideal foundation for ML-based analyses [47, 48], which thrive on large volumes of data. This intersection of neuroscience, computer science, and mathematics has given rise to neuroinformatics—a field dedicated to developing software tools for analyzing, modeling, and organizing complex brain data. Platforms such as NWB, XNAT, and REDCap have emerged to manage and process this data, facilitating collaboration and advancing our understanding of the brain [49, 50, 51].

The vast space of neural data in combination with the developing field of information processing provides many potential opportunities to deepen our understanding of the relationship between brain activity and cognition.

3.3 Approaches in Neuroinformatics

As one might expect, there exist many different concepts of how the brain is believed to integrate information. One widely accepted approach focuses on predictive coding, which posits that the brain minimizes prediction errors through processes like updating beliefs or suppressing sensory signals, effectively concentrating on unexpected changes in the environment [52]. Neuroinformatics employs various approaches and tools from the spheres of computer science and information processing to unravel individual processes as well as broader integrative concepts [51, 53, 54, 55, 56]. Among these, machine learning (ML) techniques, including supervised and unsupervised learning methods, play a central role in advancing the analysis and interpretation of complex neural data.

Supervised Learning
Supervised learning focuses on solving engineering problems by generating outputs that can be one-dimensional, categorical, continuous, or high-dimensional. The primary goal is accuracy in the generated outputs, often taking precedence over the specific methodology used to achieve these results. Practical applications include estimating behavior from data [53, 57], distinguishing healthy brains from those affected by diseases [58, 54, 59, 60], image denoising [61, 62], and deconvolution [63].

Deep learning models [60, 61, 62, 63] have achieved remarkable accuracy in many tasks; however, they often lack interpretability and fail to provide the stochastic insights [20, 53, 57] required to fully understand processes like neuronal integration. Despite these limitations, supervised learning remains a cornerstone of neuroinformatics, utilizing diverse data types to address a wide range of challenges.

An example of such a ML model was used to estimate behavior from video recordings across species, such as mice, adults, and larval Drosophila [53]. Silicon electrode array data enables the prediction of arm movements in macaques using linear models [57]. Imaging modalities like functional and structural MRI, PET, and diffusion tensor imaging (DTI) scans are employed to differentiate healthy from diseased brains. These approaches have demonstrated exceptional accuracy in detecting Alzheimer's, with studies reporting accuracies as high as 98.8% [64] and 99.2% [65]. Similarly, supervised learning has been applied to identify other neurological conditions such as schizophrenia, depression, autism, and ADHD [58, 54, 59].

Supervised learning is also used to predict future neural activity. This is particularly relevant for recognizing imminent seizures in epilepsy [66, 67, 68]. Once trained, neural networks can swiftly and accurately predict such events [56]. Recurrent neural networks (RNNs) further enhance brain-computer interfaces (BCIs), leveraging data from spike recordings [69, 70], electrocorticography (ECoG) [71], and EEG [72]. These methods highlight the transformative potential of supervised learning in addressing complex neuroinformatic problems using.

Unsupervised Learning
In neuroinformatics unsupervised learning techniques play a pivotal role, particularly in the analysis of neural information integration. Clustering methods are commonly used to visualize neural data and differentiate processing steps within the brain, offering insights into the underlying mechanisms of neural computation and organization [73, 18, 74, 51]. These approaches complement supervised learning by providing structure and patterns in data without requiring labeled outcomes. Notably, the brain's ability to self-organize task-relevant representations could emerge from such unsupervised learning processes, where input features are mapped into distinguishable activity clusters. This adaptive partitioning of input space into functional clusters enhances the extraction of essential patterns, enabling robust performance across a wide range of tasks, including reinforcement learning scenarios [74].

3.3.1 Methods & Tools

Hand-crafted models as well as processing toolboxes are often employed for preprocessing, denoising, and cleaning data. ML and DL methods assist in denoising [61, 62] and deconvolution [63] of neuronal video recordings. In calcium imaging data, deconvolution algorithms are often employed [75]. The use of generic toolboxes in these fields is rising. Research in DL techniques aims to minimize artifacts, with toolboxes like CASCADE becoming more useful for analyzing neuronal activity [76]. CASCADE has outperformed existing toolboxes like Peeling and MLSpike in benchmark tests with ground truth datasets [77, 78].

As access to lasers and microscope components becomes more affordable, other open-source toolboxes specialize in creating software to analyze two-photon calcium imaging data. User-friendly toolkits such as EZcalcium with its graphical user interface are accessible to non-programming researchers, utilizing MATLAB as its base platform [79]. The open-source tool DeepNeuron [80] and others [81] are used to label, process, and analyze smaller neuronal populations. DeepNeuron, a DL-based approach for neuron tracing, allows for automatic detection of neurite signals within a neuron image stack. This significantly reduces manual effort, enabling researchers to swiftly identify neurites in their image data. The module streamlines the formation of continuous neurite structures, facilitating a more complete and accurate representation of neurons without extensive manual tracing. Filter processes ensure that reconstructed neuron images are precise and reliable, with researchers able to engage via a user interface for interactive labeling and annotation.

While the aforementioned toolkits primarily focus on filtering and synthesizing the captured data into usable, descriptive, and objective datasets, the interpretation of this data requires expert knowledge and specialized frameworks that can help make sense of the findings. While some discoveries have been made using individually crafted models [4], a more unified model or framework seems essential for comparing and testing results as well as driving future advancements, such as brain-computer interfaces (BCIs). A key advantage of DL models for BCIs is their ability to provide real-time output within milliseconds following an initial, time-intensive training period. This rapid response is crucial for reliably decoding and predicting neuronal activity, enabling timely and accurate interactions in BCI applications [82]. All of these advancements underscore the importance of integrating computational tools in neuroinformatics to bridge the gap between raw data and meaningful cognitive insights.

3.3.2 Available Datasets

The range of neuroinformatics software available for processing imaging data is extensive, often involving datasets from costly experiments and machinery. The neuroscience community is increasingly emphasizing the principles of FAIR (Findable, Accessible, Interoperable, and Reusable) data [47]. Open-source software projects like NWB (Neurodata Without Borders) [48] aim to make brain imaging datasets more accessible. These datasets can be hosted on different platforms such as XNAT [83, 84],

DANDI Archive [48] and web-based REDCap [85] facilitate data management for researchers.

NWB provides a standardized format for neurophysiology data, supporting various data types such as electrophysiology, optical physiology, and behavioral data. XNAT allows researchers to manage and share neuroimaging datasets, integrating with major analysis platforms. REDCap, widely used in clinical research, supports extensive metadata annotation and collaboration features.

These tools collectively address the growing demand for accessible, well-documented datasets, fostering greater reproducibility and collaboration within the neuroscience community. A significant challenge in medical research, particularly in neuroscience, is the limited sample size, which is often a major barrier to statistical power and generalization of findings [58]. While the high cost of acquiring large datasets remains a key issue, advancements in molecular biology, such as the development of genetically encoded calcium indicators (e.g., GCaMP6), offer promising solutions. These innovations reduce the reliance on expensive equipment, making large-scale, high-quality data collection more feasible and lowering the barriers to entry for many researchers [86].

3.3.3 Manifolds

Despite theories like the before mentioned "predictive coding" paradigm [52], the fundamental mechanisms underlying information processing and behavior generation in the brain remain elusive. The sheer complexity of the brain, housing millions to billions of neurons, poses a computational challenge beyond current technology. Consequently, neuroscience and neuroinformatics employ various dimensionality reduction techniques to distill information from vast neuronal populations into manageable neuronal Manifolds, capturing the essence of brain dynamics in a low-dimensional space [87]. Building on this concept, the information extracted from embeddings can furthers our understanding of the function of the brain, paving the way for enhanced medical interventions. Frameworks under the umbrella of Neuronal Manifold techniques hold considerable promise for advancing medical research and clinical practice [88, 89].

According to the Manifold hypothesis many high-dimensional processes in the real world are actually explainable by a lower-dimensional latent Manifold, that resides inside the higher dimensional space [90]. An example that helps explain this phenomenon is shown with the “Swiss-Roll” dataset (see Figure 3.3). In it, the color differences between the points in the original three-dimensional dataset (left) can also be explained by only the x-axis in the non-linear 2D projection (middle), while in the linear 2D projection (Principal Component Analysis (PCA)) both axes are needed to explain the color of the points.

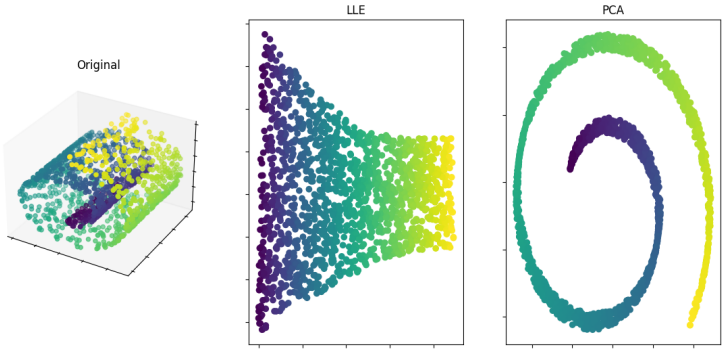


Figure 3.3: An example of a Manifold from the „Swiss-Roll " dataset (left) projected using a locally linear embedding (middle) and principal component analysis (right).

If one considers each neuron in the brain as a dimension n in the high dimensional space N and follows each neurons activity over discrete time-points $t \in T$, one would get a dataset of shape $N \times T$ of points in a neuronal activity space across time. If now one assumes that the brain operates with a set of states significantly smaller in scale than the total possible states it can potentially exhibit and the sampling rate is higher than the rate of the system dynamics, the trajectory of the points through neuronal activity space could reveal such a Manifold. In practicality, this would mean that by revealing such a "neuronal Manifold", one can explain most of the variance of the true system in a much lower dimension [87]. In this embedding, one can try to understand concepts and generate interventions that can be applied to the system. It is crucial to recognize that reducing the dimensionality of a Manifold through projection can result in structural distortions. These distortions may arise due to the inherently "local" linearity of non-linear embeddings as depicted in Figure 3.3 (middle), or due to the loss of complexity inherent in linear models, as illustrated in Figure 3.3 (right). An illustrative example is the Mercator world map, where projecting a spherical globe onto a rectangular surface results in inconsistent representations of country sizes, particularly evident in regions near the poles and the equator. Despite maintaining local consistency in shape, the size disparities between countries are altered.

In neuroscience linear Manifold learning techniques, such as PCA and Multidimensional Scaling (MDS), offer advantages in their simplicity and ease of interpretation. By preserving the properties of the data through linear transformations of observed variables, these methods yield low-dimensional embeddings that are readily interpretable. However, they are limited to uncovering only linear or approximately linear structures within neuronal population activity matrices. In contrast, non-linear techniques aim to capture the broader, more complex non-linear structure of the data, providing insights into the true topology of the Manifold. While non-linear algorithms offer the potential for a more accurate representation of neural dynamics, they often sacrifice biological interpretability as the discovered Manifold may not be directly related to individual neurons or observable variables.

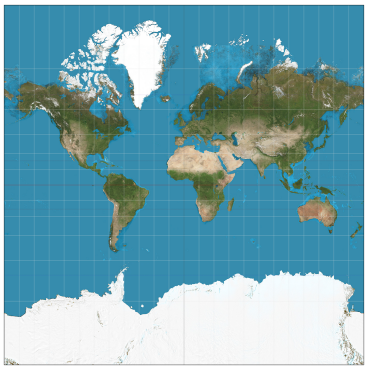


Figure 3.4: Adapted from *Mercator projection Square* by Daniel R. Strebe et al. 2011: Mercator projection of the world between 85°S and 85°N. Note the sizes of Africa, Greenland and Antarctica [91].

3.4 Bridging the Research Gap: Tools for Linking Neuronal Activity and Cognition

To address this challenge, researchers have been developing tools that balance the trade-off between capturing non-linear structures and maintaining interpretability. These approaches aim at discovering aspects of such a Manifold that represent cognitive processes generated by the underlying neuronal dynamics in the lower-dimensional space. If one steps into this higher-level realm of cognition, examining the interplay between neuronal activity and behavior, the availability of toolboxes specifically tailored for interpreting cognitive processes from neuronal data remains relatively scarce. While there exist models or atlases of cognition [92, 93], that focus on brain maps and cognitive ontology, few bottom-up built models seem to exist.

Some innovative methods use connectivity measures (Connectivity Hyper-Alignment) to create maps that are selective for specific categories, such as faces versus objects [94]. In this context, toolkits such as *BrainSpace*, *BrainStat*, and *CEBRA* represent some of the most promising approaches to bridging the gap between neuronal activity and cognition. These toolboxes adopt distinct strategies to connect neuronal dynamics with cognitive phenomena. For instance, *BrainSpace* [2] and *BrainStat* [1] are tailored for fMRI datasets and focus on cortical gradients and connectivity patterns to model cognition, visualizing large-scale brain activity without assuming a manifold structure. On the other hand, *CEBRA* [51] introduces a novel approach by discovering underlying manifolds in granular datasets. Its latent low-dimensional embeddings aim to be robust and comparable across subjects, offering a unique avenue for identifying cognitive representations in neuronal activity. Together, these toolkits exemplify the diversity of methods currently being developed to advance our understanding of the relationship between neural data and cognition and are worth exploring a little further before introduction the *NC-MCM* framework.

3.4.1 *BrainStat* [1] and *BrainSpace* [2]

BrainSpace [2] was developed to explore and analyze cortical gradients, which are essential for understanding how cognitive functions arise from the structural organization of the brain using fMRI data. *BrainSpace* employs multivariate machine learning techniques to compute these gradients, producing a compact and interpretable representation of large-scale brain activity. Specifically, the toolbox enables researchers to (i) identify cortical gradients, (ii) align them across individuals or species, and (iii) visualize their spatial distribution.

BrainStat [1], in contrast, extends this framework by adding statistical analysis and multimodal feature associations to brain imaging datasets and is based on the widely-used *SurfStat* package [95]. It provides researchers with the ability to use univariate and multivariate linear models to explore statistical relationships in both surface-based and volumetric datasets. This includes associations with other neuroimaging features, such as gene expression maps, task-based meta-analyses, and resting-state fMRI motifs. Both *BrainSpace* and *BrainStat* are implemented

in Python and MATLAB, making them accessible to a broad range of researchers. The availability of both interfaces ensures compatibility with a wide variety of workflows commonly used in the neuroimaging community.

While fMRI-based methods are invaluable for studying macroscale brain activity, they are limited by their relatively coarse spatial resolution. To address this limitation, future models could benefit from incorporating data obtained through high-resolution techniques [44, 20, 43, 44, 45, 46]. Such datasets can provide far more detailed and temporally precise insights into neural dynamics than fMRI alone. Additionally, the growing availability of tools that facilitate the inference of neuronal spikes or enhance signal denoising [75, 76, 77, 78, 79], suggests that high-resolution imaging data will become increasingly accessible. Integrating such granular imaging data into existing gradient and statistical models has the potential to reveal a deeper understanding of the cellular mechanisms driving the large-scale patterns observed in fMRI.

In conclusion, the *BrainSpace* and *BrainStat* tools offer a powerful and comprehensive toolkit for studying macroscale brain gradients and their relationship to neural function. However, functional magnetic resonance imaging (fMRI), while being widely-used, inherently lacks the precision and temporal resolution of other imaging techniques.

3.4.2 *CEBRA* [51]

One toolkit taking advantage of high percision data is *CEBRA* (*Contrastive Encoder for Behavioral and Neural Analysis*), which offers a novel approach to analyzing high-dimensional neural data using single-neuron precision datasets, such as two-photon calcium imaging or electrophysiology recordings [51]. By leveraging contrastive learning, *CEBRA* generates low-dimensional latent embeddings that combine neural activity with auxiliary behavioral or temporal data. This allows for a more integrated understanding of neural dynamics.

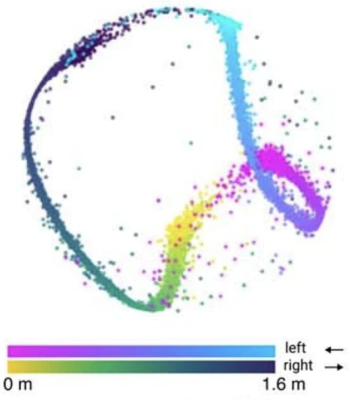


Figure 3.5: Adapted from *Learnable latent embeddings for joint behavioral and neuronal analysis* by Steffen Schneider et al. 2023: Neuronal Manifold produced by *CEBRA* from the left & right turning behavior of a rat [51].

Similar to the *Neuro-Cognitive Multilevel Causal Modeling* (*NC-MCM*) framework, *CEBRA* can merge neuronal and user-defined behavioral data to create joint embeddings for visualization and interpretation (Figure 3.5). For example, in one test, researchers input neural recordings from the V1 area of a mouse (collected via Neuropixels [44] and two-photon calcium imaging) into *CEBRA*. The model was able to predict, with 95% accuracy, the video frames the mouse was viewing, provided that the decoded frame was within a 1-second window of the actual frame [51]. This demonstrates *CEBRA*'s precision in generating embeddings for high-level neural analysis. Unlike standard methods such as PCA, which do not incorporate time information, *CEBRA* and *NC-MCM* leverage temporal ordering to enhance the accuracy of their embeddings. This is a significant improvement over popular dimensionality reduction methods like t-SNE, UMAP, and non-linear techniques such as Locally Linear Embedding (LLE), which often ignore the temporal dynamics present in neural data (see section 3.3). *CEBRA* integrates time into its contrastive learning model, ensuring that the latent space reflects the dynamic nature of neural recordings [51].

CEBRA's use of contrastive learning allows it to generate consistent embeddings across different animals, modalities, and sessions, even in the face of distributional shifts. Unlike generative models (e.g.: variational autoencoders (VAEs)), *CEBRA* does not rely on strict assumptions about data structure. This flexibility was demonstrated when *CEBRA* successfully handled neural data from two modalities—calcium imaging and Neuropixels—without loss of performance [51]. The consistency between these embeddings highlights *CEBRA*'s ability to integrate diverse datasets and maintain high decoding accuracy.

A key aspect of *CEBRA* is its time-contrastive learning, which uses the temporal structure of neural data to capture latent dynamics associated with behavioral or environmental stimuli. This was evident when the model decoded visual stimuli presented to a mouse during passive viewing of videos. *CEBRA* achieved high consistency between the embeddings from different modalities, confirming its robustness in integrating diverse data sources while preserving the underlying structure [51]. However, the authors acknowledge that there remains a gap in understanding how these latent spaces map onto specific neural computations.

3.4.3 Bridging Neural Activity & Cognition: *NC-MCM* [96]

The *Neuro-Cognitive Multilevel Causal Modeling* (*NC-MCM*) framework represents a significant advancement in bridging the explanantory gap between neural activity and cognition [96] and addresses some of the shortcommings of the previously mentioned toolkits. Its key innovation lies in constructing a manifold that inherently preserves causality by integrating the temporal context of cognitive processes. Unlike traditional manifold techniques (e.g., PCA, t-SNE, or UMAP), which are effective at producing low-dimensional representations but often neglect temporal relationships in neural data, *NC-MCM* incorporates temporal dynamics into its modeling process. This ensures a causally coherent representation of the neural mechanisms constitutive to cognitive processes. By leveraging highly precise single-neuron two-photon calcium imaging, *NC-MCM* generates latent embeddings that more accurately capture the intricate relationships between neural activity and cognition.

While frameworks like *CEBRA* also account for temporal ordering, *NC-MCM*'s unique contribution lies in its multilevel modeling approach. It treats the neural and cognitive planes as parallel time series (see Figure 3.6), which both are governed by an individual underlying causal structure. This design enables researchers to analyze both levels simultaneously and independently, with the neural level acting as the substrate and the cognitive level functioning as a lower-dimensional abstraction for higher-level reasoning. Notably, the neural level does not causally influence the cognitive level directly but is constitutive of it, reflecting a fundamental integration of these domains within the framework.

By enforcing causality within latent embeddings through Markovian constraints, *NC-MCM* ensures that cognitive state sequences respect the temporal and causal dynamics inherent in the data. This is achieved through a *causally consistent transformation* into the embedding space.

Through this integral transformation, the framework enables causal reasoning within the resulting manifold. While statistical knowledge suffices for prediction and diagnosis, causal knowledge is indispensable for action and intervention [97]. By exploring cognitive sequences through generated embeddings (see Section 4.2.3) or cognitive graphs (see Figures 5.22 and 5.23), the framework provides novel tools for hypothesis testing and deepens our understanding of how neural activity underpins cognition.

Although *NC-MCM* offers a powerful framework for examining the interplay between neural and cognitive processes, it does not resolve the ultimate question of why neural activity gives rise to cognition. This challenge lies in the inherent complexity of linking physical neural substrates to emergent cognitive phenomena. While cognition is understood to arise from neural processes, the precise mechanisms behind this emergence remain elusive within the physicalist framework. By providing a structured method to model and study these relationships, *NC-MCM* bridges this gap, enabling researchers to explore causal relationships, identify patterns, and test hypotheses with greater precision.

In this sense, *NC-MCM* represents a significant step forward. By integrating temporal dynamics, causal reasoning, and multilevel modeling, it serves as a valuable tool for advancing our understanding and lays critical groundwork for future discoveries by systematically connecting neural and cognitive domains in neuroscience research. Before delving into the details of the *NC-MCM* framework, it is essential to first introduce the concept of Markov processes, as they form a foundational component of the modeling approach.

3.5 Markov Process

As mentioned above, the *causally consistent transformation* is facilitated by the constraint of modeling the cognitive level's sequences as a Markov process. This is why, before describing the details of the *NC-MCM* framework, the definition and some aspects of a Markov process will be introduced, which are crucial to the reasons and background of this modeling approach.

A Markov process normally refers to a 1^{st}-order Markov process. Henceforth, unless otherwise specified, any mention of a Markov process implies a 1^{st}-order process. It can be described as a stochastic model that consists of a countable set of states, in which each transition of states depends only on the state attained in the previous event. This property, known as the Markov property, makes Markov processes particularly suitable for modeling dynamic systems characterized by sequential dependencies. In higher-order Markov processes, the dependency extends to multiple previous states.

The key properties of Markov processes are highlighted below:

State Space: A finite or countably infinite set of possible states or configurations s that the system can occupy at any given time:

$$S = s_1, s_2, s_3, \ldots, s_m.$$

Markov Property: The probability of transitioning to a future state depends only on the current state and is independent of the history of the process. This property is present in all Markov processes of n^{th} order and can be written as:

$$P(X_t|X_{t-1},\ldots,X_0) = P(X_t|X_{t-1},\ldots,X_{t-n}),$$

where:

$$n \in \{1,\ldots,t\}$$

Since it allows for the modeling of complex systems with evolving states and dependencies, Markov processes find widespread application in various fields such as finance [98], biology [99, 100], and natural language processing [101]. Sequences of states resulting from Markov processes are generally referred to as Markov Chains. Depending on the type of process they result from, these can be further subdivided into homogeneous and inhomogeneous Markov Chains. In homogeneous Markov Chains, the probability to transition from one state to another $p_{ij} = P(X_t = j|X_{t-1} = i)$ does not depend on t, while in inhomogeneous Markov Chains the probabilities to transition between states can vary over time. For instance, one might need to use inhomogeneous chains to model up and down ticks in the stock market as the financial environment changes between bull and bear markets. However, in other use cases where the system dynamics remain stable over time, a homogeneous Markov process is more adequate.

In *NC-MCM*, while it is not required for the theoretical framework, a time-homogeneous process with a stationary distribution is assumed, as the time-homogeneity simplifies empirical inference on the sequences [96].

Time-Homogeneous Process: Transition probabilities governing the process stay constant over time. These probabilities can be represented in a matrix of size $m \times m$, where each entry remains constant and is therefore independent of t:

$$p_{ij} := P(X_{t+1} = s_j|X_t = s_i),$$

where:

$$i, j \in \{1,\ldots,m\}$$

Another attribute of most time-homogeneous Markov processes important for the framework presented here is the concept of a stationary distribution. A stationary distribution refers to a probability distribution over states that remains unchanged as the system evolves.

Stationary Distribution: A Markov process has a stationary distribution, if the probabilities of being in specific states stabilize over time, regardless of the initial distribution. This distribution is formally given by $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$ where each π_j stabilizes over time.

$$\pi_j = \sum_{i=1}^{m} \pi_i \cdot p_{ij},$$

If m is finite a stationary distribution exists under the conditions of:

- **Irreducibility**: Each state can reach every other state in x steps
- **Aperiodicity**: Non cyclic revisiting of states at fixed intervals

for all:

$$j, x \in 1, \ldots, m$$

Stationary distributions are particularly important in applications where long-term equilibrium behavior is of interest, such as in many biological systems [102, 103]. These processes differ from periodic or time-inhomogeneous Markov processes, where the system either cycles between specific configurations at regular intervals or undergoes changes in its transition probabilities over time, thus preventing convergence to a stationary distribution.

In the *NC-MCM* framework the assumptions of homogeneity and the presence of stationary distributions are crucial as they enable robust modeling of sequential data with consistent temporal dependencies and predictable long-term behavior.

3.5.1 Application of Markov Processes in Neuroscience

Markov processes have been suggested as potential models to study the principles of sequential processes in the brain [103]. The dynamics of neuronal activity often exhibit temporal dependencies, where the firing of neurons at a given time depends on their previous activity. While this temporal dependency could likely be modeled as a Markov process, the sheer magnitude of all potential states that a brain can manifest exceeds the capacity of current tools. To address this challenge, reducing the complexity of the model by making it less granular can be beneficial. In neuroscience, this can be achieved by using embeddings of neuronal activity patterns as the states of the model, capturing essential dynamics while simplifying the state space.

Brain activity is often modeled as a continuous trajectory through an embedding, like the before mentioned Manifolds. In this continuous case, the model changes transiently from one state to the next and follows certain paths through the embedded space [104]. This fits very well with the trajectories observed in simpler model animal brains [73] or the decision processes of more complex brains [18] and would constitute a neuronal Manifold. The *NC-MCM* framework has been used to create discrete cognitive states [96], which can be easier to interpret, as well as continuous state models [104].

3.6 NC-MCM

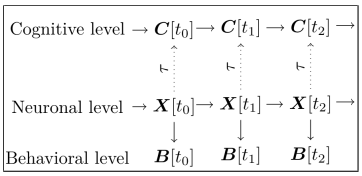


Figure 3.6: Adapted from *Neuro-Cognitive Multilevel Causal Modeling: A Framework that Bridges the Explanatory Gap between Neuronal Activity and Cognition* by Moritz Grosse-Wentrup et al. 2023: Different levels modeled as a 1st order Markov process. Solid lines represent causal relations, while dotted lines are constitutive relations [96].

When explaining how cognition arises from neuronal activity some entertain notions of non-measurable effects or quantum processes, suggest indeterminacy [105, 106], others propose a deterministic sequence of reactions within the brain leading to fixed outcomes [107, 108, 109]. If one adopts the deterministic viewpoint, the brain could in theory be modeled as a Markov process. Building upon this, the *NC-MCM* framework was developed [96]. It delineates an agent's function and actions into three levels: *neuronal*, *cognitive* and *behavioral* (see Figure 3.6).

Firstly the *neuronal* level represents actual measurable brain activity. It is modeled in the framework by a Dynamic Structural Causal Model (dSCM) as described in the original paper [96]. The dSCM described in the paper is a first-order Markov process, which means each state is causally sufficient for the probability distribution over the states at the next time point. The authors mention this must be reconsidered if the sampling rate is slower than the system dynamics. Regarding findings that some neuronal dynamics happen in time-frames shorter than 50 ms [110, 111], it is important to note that data using sampling methods slower than that could break this assumption.

As mentioned earlier in studies like the wired-ferrets experiment [5], the brain not only allows adaptation to the environment but also adapts itself—a phenomenon known as neuroplasticity [5, 112, 113]. Neuroplasticity can influence the transition probabilities between different neuronal activity states, potentially resulting in non-stationary or higher-order Markov processes. While processes such as synaptogenesis and changes in dendritic structure typically occur over hours to weeks [113, 5], they generally do not affect neural recordings taken during brief experimental contexts lasting only a few minutes.

However, it is essential to recognize that not all neuroplastic effects are fully understood. Recent research on drugs such as psilocybin and ketamine has shown promising results in accelerating neuroplasticity, reducing the time needed for these effects to manifest [114, 115]. Additionally, the process of habituation was studied in many animals and discovered by Castelucci V. et al. in 1970 [112]. The authors could show that the effect of habituation already effected neural signaling after a few minutes of stimulation. This said it seems crucial to consider these and other less known processes and subsequently test the resulting cognitive level for its time-homogeneity if one wants to model it as a time-homogeneous Markov process.

This *cognitive* level encompasses mental states such as hunger or fear, influencing behavior. It builds on top of the *neuronal* level. In other words, neuronal activity is constitutive of these mental states and therefore they should be deducible from the underlying neuronal activity. The authors reason that such a mental state space must also be causal. They use an example to illustrate this assumption:

> *"... when we say I am unhappy because I am bored, we express the causal relation boredom → happiness. This causal statement implies the empirically testable prediction that if we reduce boredom in a person, we increase their probability of being happy. It is further consistent with the view that boredom*

and happiness are two separate processes that can be manipulated independently. . . " [96]

Ultimately one wants to use causal reasoning to understand the systems' trajectories over time and to make informed interpretations, decisions or interventions. To facilitate a causal relationship between following cognitive states, strict Markovian behavior is enforced upon the sequence of cognitive states in the model. This is achieved by conducting multiple cognitive clusterings and selecting the outcomes with the least evidence contradicting certain properties of a first-order Markov process. Additionally, *BundDLe-Net* [104] employs a different technique, leveraging neuronal activity to generate a low-dimensional causal embedding. This is accomplished by ensuring at each step that the subsequent cognitive state is predictable from the preceding one. Causal relationships can be observed within neuronal processes and cognitive states, suggesting that causal statements about mental processes allow for testable predictions and manipulation of these states independently to understand and influence cognitive phenomena. While the *neuronal* and *cognitive* levels are causal within themselves, the authors argue that a causal relationship between neuronal state and cognitive state ($N \rightarrow C$) is not compatible with a physicalist standpoint. In this hypothetical causal relationship changes in N (neuronal state) affect C (cognitive state). It further implies that by isolating C it would be theoretically possible to alter the cognitive state without changing N, which seems contradictory to our understanding of physicalism. Such a relationship would rather suggest some form of dualism where mind (C) and body (N) are separate. Hence the authors reason that the relationship is constitutive and not causal.

The solution they propose to surpass the gap between these internally causal levels are *causally consistent transformations*. These mappings are chosen in a way that one can reason about the observational effects of causal interventions consistently across both models. In other words, neuronal states do not cause but are constitutive of cognitive states. This concept terms the name *multilevel causal modeling* (MCM).

Lastly, the *behavioral level* is defined as the observable action taken. These behaviors are "caused" by neuronal activity. This is highlighted by the high accuracy (group-average 92.2%) of a simple logistic regression, predicting behavior from neuronal activity, in their *C. elegans* dataset [96]. The relations between the *neuronal* level and the *behavioral* level can be considered causal because one can, in principle, manipulate behavior independently of neuronal activity, by fixating an animal and constraining movement.

3.7 Interpretability & Usage

As mentioned before, tools specifically designed for interpreting cognitive processes remain limited in neuroinformatics. The *NC-MCM* framework constructs a causal cognitive process from neuronal and behavioral data as seen in section **??**. In an example from its original paper [96], the authors demonstrate how the *NC-MCM* framework addresses the interpretation of the cognitive process sequences, using data from two-photon calcium imaging data from *C. elegans* [73].

Important decision points within the cognitive process of the worm are identified using behavioral transition diagrams akin to Figures Figures 5.22 and 5.23. Subsequently, neurons responsible for different decisions are pinpointed through global neuronal perturbations (see Figure 3.7), which quantify deviations in neuron activity relative to their mean activation energy within specific cognitive states (e.g., cognitive state 1 and sustained reversal, C1). In their study, the researchers examined perturbations across recordings of five worms, identifying neurons with significantly higher or lower activation energies at decision points —such as sustained reversal (blue), ventral turn (orange), or dorsal turn (green) (see Figure 3.7). These neurons are argued to play a crucial role in determining the worm's behavioral outcomes.

This example illustrates the powerful potential of *NC-MCM* in mapping causal relationships between cognitive state transitions and neuronal dynamics, offering new insights into the neural basis of decision-making. This capability is largely attributable to the *causally consistent transformation* employed in *NC-MCM*, enabling causal reasoning about cognitive transitions. In contrast, the *CEBRA* framework does not establish this direct causal link, highlighting a fundamental distinction between the two approaches.

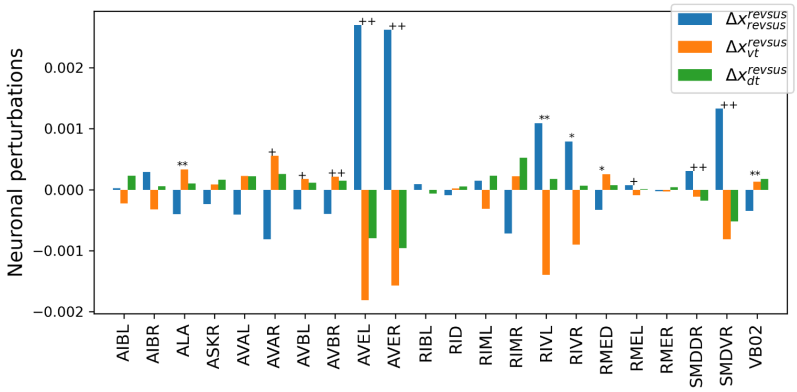


Figure 3.7: Adapted from *Neuro-Cognitive Multilevel Causal Modeling: A Framework that Bridges the Explanatory Gap between Neuronal Activity and Cognition* by MoritzGrosse-Wentrup et al. 2023: Neuronal perturbations across all worms of selected neurons which are predicted to induce a cognitive state transition that increases the probability of maintaining a sustained reversal movement (blue), initiating a ventral turn (orange), or initiating a dorsal turn (green) [96].

More sophisticated *NC-MCM* models could illuminate intricate cognitive processes across diverse organisms. Expanding on this approach, as proposed by Grosse-Wentrup et al. [96], would require capturing more nuanced behaviors. This broader spectrum of cognitive states would provide deeper insights into organismal experiences and elucidate the complex interactions among cognitive processes. Furthermore, in intervention studies, these models could evaluate the cognitive effects of medications, offering a clearer understanding of how substances modify both neuronal activity and behavior.

3.8 Aim of the toolbox

As there do not exist many easy-to-use toolboxes to study the bridge between cognition and neuronal activity, we aimed to fill this gap with the development of a toolbox for the *NC-MCM* framework. The main goal is to create a compact, open-access *Python* toolbox for the identification and analysis of low-dimensional representations from brain recordings.

Key requirements are that the tool needs to be lightweight, user-friendly, and capable of generating visualizations useful to researchers. It should be accessible to users with limited programming knowledge and facilitate quick visualizations of complex data structures. Additionally, it should incorporate the already proposed statistical test as well as create an additional one to verify the main assumptions of the *NC-MCM* framework is necessary. Thus, the toolbox provides an entry point into the framework for researchers interested in studying cognition as a window into brain organization and function.

Methods 4

4.1 Overview of the *NC-MCM* toolbox

The Methods and Results are mainly split up into three parts:

- The showcase and performance of the toolbox;
- The statistical tests and power analysis;
- The use of the toolbox on old and new data;

Python was selected as the language for the toolbox primarily due to its accessibility, infrastructure and how it manages data through a reference-based model [116], among other reasons. It allows for easy distribution using the *PyPI* repository, as well as *GitHub* (*https://github.com/DriftKing1998/NC-MCM-Visualizer*). Additionally, *Python* serves as a common interface for researchers delving into machine learning and statistical analysis, as it is used by many other toolboxes in the field [51, 117, 2]. This choice facilitates the implementation of *NC-MCM* approaches researchers in the future. The libraries utilized to ensure robustness and efficiency are listed below:

- *numpy* for robust data processing [118, 119]
- *matplotlib* for visually striking plots [120]
- *scikit-learn* for efficient clustering, scoring and decomposition [121]
- *networkx* combined with *pyvis* to generate interactive plots [122, 123]
- *tensorflow* for the neural network architecture in *BunDLe-Net* [124]

These libraries ensure seamless integration of custom code, maintaining the package's relevance and effectiveness over time as they are regularly updated, apart from the *pyvis* package.

4.1.1 Design Philosophy

The *NC-MCM* toolbox is designed with an emphasis on simplicity and usability, which sometimes comes at the cost of strict modularity. The idea was that the main components of the framework should be part of a class orientated structure, while auxilliary methods are be stored separately. This will allow first users of the toolbox to get to know it quickly by avoiding a very modular design sometimes present in *Python* libraries. For example, a close coupling between the main classes of the package, the *Database* and *Visualizer* classes, allows data updates to directly affect downstream visualizations. This prioritization of a straightforward workflow benefits researchers at different skill levels by keeping the code easy to navigate and understand. For early-career researchers, this approach offers a gentle learning curve, while advanced users benefit from quick access to key functionalities without complex class dependencies. The relatively flat directory structure helps novice users easily locate files and functions. Future expansions should be made in the form of new helper functions or as new classes added as a separate file in corresponding directory.

Package Tree

```
ncmcm
|
|--- ncmcm_classes
| |--- Database.py
| |--- Visualizer.py
| |--- Loader.py
| |--- CustomEnsembleModel.py
|
|--- helpers
| |--- markov_functions.py
| |--- plotting_functions.py
| |--- processing_functions.py
| |--- sequence_functions.py
| |--- visualizer_creation.py
|
|--- bundlenet
| |--- BunDLeNet.py
|
|--- statistical_test_plots
| |--- Viability_Testing.py
| |--- Viability_Chunk.py
| |--- power_curve_discrete.py
| |--- power_curve_memoryless.py
| |--- power_curve_pseudo_cont.py
|
|--- data
  |--- default_physic_static.json
  |--- default_physic_bounce.json
  |--- datasets
    |--- c_elegans.mat
```

4.2 *NC-MCM* Toolbox

As depicted in the package tree in Section 4.1.1 the *NC-MCM* toolbox is organized into several main directories, each housing distinct components that serve specific roles in the package's functionality:

- **`ncmcm_classes`**: Contains the core *Python* classes, including *Database*, *Visualizer*, *Loader*, and *CustomEnsembleModel*. These classes encapsulate the primary functionality of the toolbox, managing tasks like data handling, visualization, and model creation.
  - *Database.py*: Functions as a container for the data to analyze and offers application of *NC-MCM*
  - *Visualizer.py*: Handles the rendering of three dimensional visual representations of neural data
  - *Loader.py*: Responsible for loading the demo dataset into memory for processing
  - *CustomEnsembleModel.py*: Implements the custom ensemble model (see section 4.2.4)
- **`helpers`**: Includes utility scripts that define auxiliary functions for statistical testing, data processing, sequence generation, and specific plotting functions. This directory supports the modular design by containing reusable components that can be invoked across different classes or workflows as needed.
  - *markov_functions.py*: Contains the statistical tests (see sections 4.3.1 and 4.3.2)
  - *plotting_functions.py*: Functions used to create certain plots to evaluate statistical tests
  - *processing_functions.py*: Contains data preprocessing and other useful methods
  - *sequence_functions.py*: Contains all methods to simulate sequences
  - *visualizer_creation.py*: Contains methods to load or create the Visualizer instances
- **`bundlenet`**: Houses the *BunDLeNet* model architecture [104], a specialized deep learning model integrated into the *NC-MCM* toolbox. It includes supplementary functions for data preprocessing and model training, allowing for integrated use of the model.
  - *BunDLeNet.py*: Contains the BunDLe-Net code and auxilliary methods
- **`statistical_test_plots`**: The individual scripts in this directory were used to check the capabilities of the statistical tests as well as generate some of the plots seen in the Results.
  - *Viability_Testing.py*: Viability testing of both tests (used to create Figures 5.1, 5.2, 5.6 and 5.7)
  - *Chunk_Testing.py*: Viability testing of chunk amounts (used to create Figure 5.8)
  - *power_curve_memoryless.py*: Script for power curves shown in Figures 5.3 to 5.5
  - *power_curve_discrete.py*: Script for power curves shown in Figures 5.10, 5.12 and 5.14
  - *power_curve_pseudo_cont.py*: Script for power curves shown in Figures 5.9, 5.11 and 5.13
- **`data`**: Provides example datasets and configuration files (e.g., *default_physic_static.json*, *c_elegans.mat*) to demonstrate the toolbox's usage with a real-world dataset.
  - *default_physic_bounce.json*: Contains physics settings for interactive plots with static nodes
  - *default_physic_static.json*: Contains physics settings for interactive plots with repelling nodes
  - `datasets`:
    - *c_elegans.mat*: *C. elegans* example data
    - *mouse_10.nwb*: *Mus musculus* example data (not on *GitHub*)

The *Database* and *Visualizer* classes form the central components of the *NC-MCM* toolbox, each serving specific roles to support a comprehensive data analysis workflow. Since, the *Visualizer* builds upon the *Database*, the data used in linked instances stays the same. Proponents of strict modularity might argue that the resulting tight coupling of these classes could be problematic for future expansion and reuse of the toolbox [125, 126], particularly since the *Visualizer* always requires a *Database* object. However, from an analytical perspective, the *Visualizer* is specifically designed for generating three-dimensional visualizations of data stored in the *Database*. As a tightly coupled extension of the data container, it is an endpoint of analysis and focuses solely on three-dimensional plots and movies. This tight integration supports a streamlined, intuitive workflow, ensuring ease of use even for users with limited programming experience.

4.2.1 Database

ncmcm → ncmcm_classes → Database.py
Database(**neuron_traces**, **behavior**, *neuron_names*, *behavioral_states*, *fps*, *name*, *colors*)

Within this framework, the *Database* and *Visualizer* classes serve distinct yet interdependent functions that streamline data management, analysis and visualization for researchers in the context of the *NC-MCM* framework.

The *Database* object in the *NC-MCM* framework functions as the main container for managing neuronal activity data and behavioral labels. The underlying concepts for the calculations performed in this class were outlined in prior work [96]. However, their implementation in a *Python* environment was developed specifically for this work. It is designed to be user-friendly and intuitive, aiming to give researchers a comprehensive and immediate overview of the analysis workflow. By centralizing data storage and analysis within a single object, the *Database* object supports efficient and comprehensive customization of each of the analysis steps, generic modeling and two-dimensional (2D) plots of data.

- **Data Storage:** The *Database* is initialized using neuronal data (as *neuron_traces*) and behavioral labels (as *behavior*). The reference based storage model of *Python* allows memory efficient storage of data [116] —an important feature for handling large datasets. Modifications to the data within this object by hand or using methods like *exclude_neurons()*, will affect results downstream.

- **Analysis Methods:** The class provides methods for model fitting, cognitive state clustering and statistical testing. Generic methods such as *fit_model()* enable users to apply and evaluate various classification models, while in the clustering step users have influence over the type of clustering used as well as the statistical tests used for testing against non-markovian behavior. The *cluster_BPT()* and *cluster_single_BPT()* give users the necessary parameter choices.

- **Visualization and Plotting:** For quick exploratory data analysis, the *Database* includes methods like *step_plot()*, which generates a visual summary of the *NC-MCM* workflow. Additionally, the *behavioral_state_diagram()* method creates cognitive graphs that map behaviors, cognitive states and their transitions, aiding in the initial interpretation of the data. The graphs can be displayed as interactive *pyvis* graphs or be exported as XML-files for use in other softwares.

***NC-MCM* Workflow**
While more detailed documentations of the *Database* analysis options can be found in the Supplementary Material (Subsection 7.3.1) or in the project's GitHub repository, the general analysis workflow for applying *NC-MCM* is outlined here. This workflow was already used in the analysis of the original presentation of the framework [96].

First, a user creates a *Database* instance using the necessary input data. With this instance, they can fit a model to the data using the *fit_model()* method. The model must implement a few standard methods (*fit()*, *predict()*, and *predict_proba()*) that are traditionally available in most *scikit-learn* classifiers. This step provides the accuracy of predicting behavioral data from the neural data used to instantiate the *Database* object. Additionally, a cross-validation score is computed if the *cv_folds* parameter specifies the number of folds to use. The key point, however, is the possibility of projection of each time-point of neural activity into a probability space for behaviors, referred to as the *behavioral probability space*.

Next, the user can cluster these *behavioral probabilities* into groups of cognitive states using either the *cluster_BPT()* or *cluster_BPT_single()* methods. This is achieved via an indeterministic clustering algorithm (the default is k-means), resulting in a sequence of cognitive states corresponding to the neural activity time series. This clustering process is repeated for a specified number of iterations, and each resulting cognitive sequence is evaluated for its Markovian properties. The cognitive sequence that exhibits the least evidence against specific Markov process properties is considered a good fit for the *cognitive level* of the *NC-MCM* framework.

Using these selected sequences, users can generate various plots, including cognitive graphs and diagnostic visualizations, to support further analysis.

4.2.2 Visualizer

ncmcm → ncmcm_classes → Visualizer.py
Visualizer(**data**, *mapping*, *transform*)

The *Visualizer* object in the *NC-MCM* package specializes in rendering three-dimensional (3D) visual representations of the neuronal data stored in the *Database*. While the *Database* object manages 2D plotting, data storage and analysis, the *Visualizer* handles 3D visualizations and animations, making it easier to explore complex neural patterns and behavioral states in a more immersive manner. The *Visualizer* requires an instance of the *Database* object, creating a tight coupling that ensures any changes made in the *Database* (e.g., updates to data or model configurations) are automatically reflected in the visualizations.

- **Generic 3D Mapping and Dimensionality Reduction:** The *Visualizer* enables the transformation of high-dimensional neuronal data into 3D space. This is achieved by selecting a dimensionality reduction technique when initializing the object. The generic constructor supports various dimensionality reduction techniques, such as PCA and t-SNE, among others. This feature allows researchers to explore spatial representations of neural activity in linear as well as non-linear models.

- **GIF and Animation Capabilities:** The *Visualizer* can create animated representations of the 3D mappings, allowing researchers to generate customizable GIFs that display dynamic changes in neuronal and behavioral patterns. This capability is valuable for presenting complex temporal data in an accessible and visually appealing format.

The *Visualizer* class does not perform calculations related to the basic framework. Instead, as its name suggests, it is primarily used for visualization purposes, employing various dimensionality reduction techniques to project neural activity into 3D space. Its most notable methods include *plot_mapping()*, *make_comparison()*, and *make_movie()*, which are described in detail in Supplementary Subsection 7.3.2.

Another valuable method is *use_mapping_as_input()*, which allows users to create a new *Database* instance with the same behaviors but using the latent embedding of neural activity as input for the *neuron_traces* parameter. This method enables the creation of datasets with drastically reduced size from a high-dimensional input dataset for further analysis.

4.2.3 BunDLe-Net

ncmcm → bundlenet → BunDLeNet.py
BundDLeNet(**latent_dim**, **behaviors**, *discrete*)

This class is defined in a separate script with all its auxiliary functions. The models' architecture will only be briefly summarized here, since a more comprehensive explanation is found in the original paper [104]. The model consists of three smaller models that are trained simultaneously – the τ model, the T_Y model, and the *predictor*. The τ model projects the input parameters into a latent embedding, the T_Y model will predict the difference from the predicted embedding at time t to $t+1$, while the predictor simply predicts labels from this embedding. The difference to a simple latent space-based prediction is that each time point in the embedding is forced to be causally sufficient for predicting the next state. This is because the labels are predicted from an embedding that has been forwarded one-time step by the T_Y model. Thereby, helping to capture the chronological and causal properties of the data. The architecture of the model can be seen in Figure 4.1. In a single training step, the feature space X_t is projected into an embedding Y_t^L by τ. Then the T_Y model forwards Y_t^L into Y_{t+1}^L which is used to compute part of the loss (mean squared error from Y_{t+1}^L and Y_{t+1}^U). The second part of the loss is calculated from the sparse categorical cross-entropy of the true label and the predicted label by the predictor from upper Y_{t+1}.

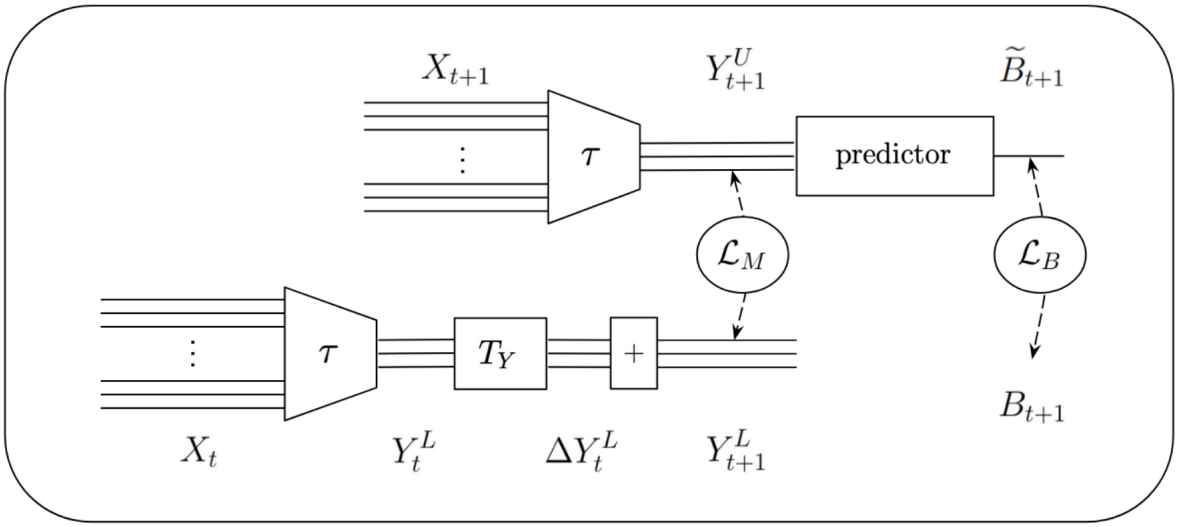


Figure 4.1: Adapted from *BunDLe-Net: Neuronal Manifold Learning Meets Behaviour* by Akshey Kumar et al. 2023: Basic architecture of the BunDLe-Net consisting of the τ model, T_Y model and the predictor [104].

4.2.4 CustomEnsembleModel

ncmcm → ncmcm_classes → CustomEnsembleModel.py
CustomEnsembleModel(**base_model**)

The *fit_model()* method (see section 7.3) of the *Database* class fits a classification model to the neuronal activity data and behavioral labels. This model is used to map the neural activity into a causally consistent This method has the option to create a *CustomEnsembleModel*, which creates k copies of the model specialized in differentiating each of the n behavioral labels from every other label separately. This increases the dimensions of its behavioral probabilities saved in *yp_map* from n to k.

$$k := \frac{n-1}{2} * n$$

Creating a *CustomEnsembleModel* is set as the standard if a model is fit on a *Database* object since these models showed better results regarding *markov_property_test()* behavior of the resulting cognitive state sequences.

4.2.5 Performance of the toolbox

The computational performance of some of the methods and functions explained in Subsections 4.2.1, 4.2.2 and 4.2.4 as well as Subsections 4.3.1 and 4.3.2 are evaluated.

Each one method was run 30 times with a set of fixed parameters and random data of the same shape and size to test the mean time elapsed when executed. For this, the *timeit* package from *Python's* standard library is used to calculate the seconds elapsed. Although this method will not give an objective measure of computational work done, since it only measures time and not computations made, it should provide a easy to understand measure of the speed at which these methods are executed.

All of the methods that were tested are listed in 4.1 and were run on a *Mac Book Pro* with an *Apple M2 chip* and 16 GB of RAM running *macOS 15.1*

To generate a better picture of how the complexity of the computationally more complex statistical tests described below (see Subsections 4.3.1 and 4.3.2) grows, they were run with different parameter sets – combinations of sequence lengths (1000, 2000, 3000, 5000 and 10000), state spaces (5, 10, 15, 20, 35 and 50), sequence simulations (200, 400, 600, 800 and 1000) and chunks (5 or self-determined) were used.

Table 4.1: A list of all methods tested for their computation time and their parameters. The methods were run exactly 30 times to create an average runtime. All tests were run on a *Mac Book Pro* with an *Apple M2 chip* and 16 GB of RAM running *macOS 15.1*

Method
fit_model()
cluster_BPT()
cluster_BPT_single()
make_movie()
plot_mapping()
make_comparison()
markov_property_test()
stationary_property_test()

4.3 Statistical Tests

4.3.1 Markov Property Test (*markov_property_test()*)

ncmcm → helpers → markov_functions.py → markovian()

The primary method for testing cognitive state sequences for non-Markovian behavior is the *markov_property_test()* function, which evaluates the Markov property of a process (see Section 3.5). This test was conceptualized by the authors of the original *NC-MCM* paper [96] and implemented in *Python* for this work. Based on the assumptions of the Markov property of a first-order Markov process:

$$P(X_t|X_{t-1}, X_{t-2}, ..., X_0) = P(X_t|X_{t-1})$$

and a 2nd order Markov process,

$$P(X_t|X_{t-1}, X_{t-2}, ..., X_0) = P(X_t|X_{t-1}, X_{t-2})$$

one can use the definition of the conditional probability,

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

to calculate the empirical 1st order and 2nd order conditional transition matrices given a sequence of length T. This is done by first creating a joint probability matrix by counting the transitions:

$$Y = (X_{t-2} = s_i, X_{t-1} = s_j, X_t = s_k)$$

where:

$$i, j, k \in \{1, ..., n\} \qquad \text{and} \qquad t \in \{2, ..., T\}$$

with:

$$n = \text{numbers of states}$$

A three-dimensional matrix is filled with the counts divided by all transitions, to get values between 0 and 1. In this joint probability matrix, the value at position [x, y, z] will give us $P(X_{t-2} = s_x, X_{t-1} = s_y, X_t = s_z)$ or in other words the joint probability for the sequence s_x to s_y to s_z. From this matrix, one can simply get the joint probability matrices for:

$P(X_{t-1} = s_y, X_t = s_z)$	by summing the values over the x-axis.
$P(X_{t-2} = s_x, X_{t-1} = s_y)$	by summing the values over the z-axis.
$P(X_{t-1} = s_y)$	by summing the values over the x- and z-axes.

Using these matrices one can finally calculate the empirical 1st and 2nd order conditional transition matrices using the conditional probability definition:

$$P(X_t|X_{t-1}) = \frac{P(X_{t-1}, X_t)}{P(X_{t-1})}$$

$$P(X_t|X_{t-1}, X_{t-2}) = \frac{P(X_{t-2}, X_{t-1}, X_t)}{P(X_{t-2}, X_{t-1})}$$

Now multiple sequences are simulated to generate a test statistic. The amount of simulated sequences is given by the parameter *sim_m* (=*N*) with the default value 200. Using the empirical 1st conditional transition matrix of the given sequence, the test generates N new 1st order Markov sequences. For each of these sequences, the

empirical 2nd order conditional transition matrix is calculated. Using the variance over the x-axis ($t-2$) of these matrices one can generate a test statistic of size N.

Assuming that the original sequence originates from a 1st order Markov process, the variance over the x-axis from its 2nd order conditional transition matrix should not be significantly different from the variance of the simulated sequences. If the variance from the initial sequence is bigger than 95% of the simulated sequences one would reject the null hypothesis, which states that the sequence is not breaking the 1st order Markov property.

It is important to emphasize that this method does not conclusively determine whether the sequence is a 1st-order Markov process or not. Instead, it can provide evidence against the null hypothesis (H_0), which posits that the sequence follows a 1st-order Markov process. Sequences exceeding the 95th percentile of the test statistic are classified as not satisfying H_0, while sequences below this threshold may be consistent with such processes.

Example 1: Given the sequence:

A → B → B → A → C → C → C → A → B → A → C → A → B → B → C → B → A → C → B → A → A → B → A → A

One starts with calculating the 2nd order joint transition matrix and from there one follows the steps as described to end up with a critical value calculated from the distribution of variances over the $t-2$ axis of the simulated sequences. This is compared to the test-statistic calculated from the variance of the original 2nd order conditional transition matrix.

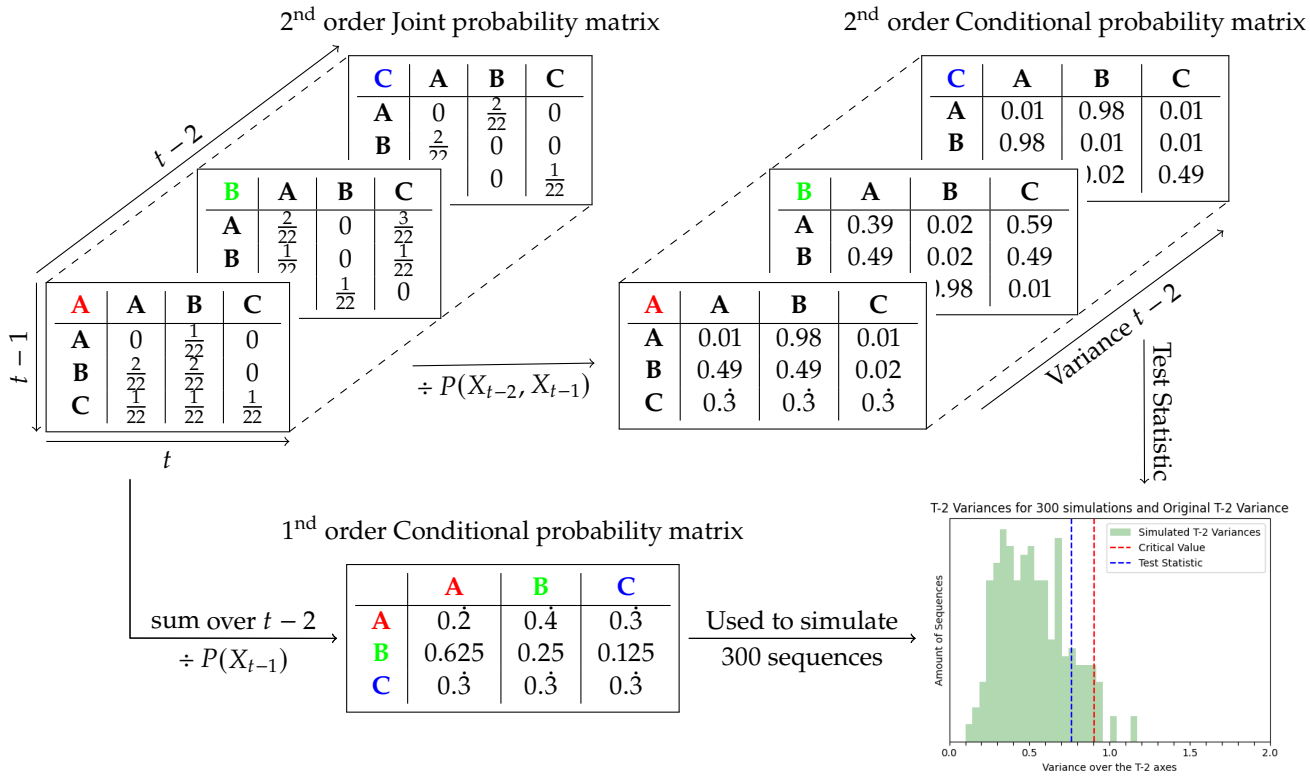


Figure 4.2: Workflow of the *markov_property_test()* function using a sequence with three states (**A**, **B**, and **C**). After generating the 2nd-order joint transition matrices (top left), the 1st- and 2nd- order conditional transition matrices can be calculated (bottom left and top right). Simulated sequences provide a distribution of variances over the $t-2$ axis, from which a critical value is derived. The test-statistic, calculated from the $t-2$ variance of the original 2nd-order conditional matrix (top right), is compared to the critical value (bottom right). *Note: Values are simplified or rounded for illustration.*

In the example a distribution of 300 variance over the $t-2$ axis of sequences generated by the 1st-order Conditional probability matrix is created. The p-value resulted from comparing

the variance of the original sequence is **0.143**, therefore the null hypothesis is not rejected. The sequence could come from a 1st-order Markov process.

Computational Complexity

For each run of this function, the computational complexity is mainly determined by simulating values for the test statistic. The process is repeated s times and in each iteration:

- s is the number of simulated sequences generated
- m is the sequence length
- n is the number of states

1. A sequence of length m is simulated $\rightarrow$ O(m)
2. The 2nd order joint probability transition matrix is populated with values $\rightarrow$ O(m)
3. Zeros in the matrix are replaced with $\epsilon \rightarrow$ O(n^3)
4. A new joint transition probability matrix is created by summing over the z-axis $\rightarrow$ O(n^3)
5. The conditional probability matrix is created by dividing the prior matrix with the new one $\rightarrow$ O(n^3)
6. The variance is computed along the 3rd dimension ($t-2$) and summed $\rightarrow$ O(n^3)

When the time complexities of each step are added together, the total time complexity per iteration becomes O($2m + 4n^3$). Thus, the upper bound for the complexity of s iterations is O($s * (2m + 4n^3)$) or O(n^3), where:

In practice, since m is often much larger than n, the complexity should be able to be simplified to $O(sm)$, assuming n^3 is relatively small compared to m. Otherwise, with an increasing number of states n the an upper bound of O(sn^3) is more adequate.

4.3.2 Stationary Property Test (*stationary_property_test()*)

ncmcm → helpers → markov_functions.py → stationary_property_test()

A second statistical test was developed within this work to test for other properties of sequences assumed by the framework. This test tries to postulate a H_0 that incorporates aspects of time-homogeneity and stationarity of Markov processes (see Section 3.5). This test relies on the assumption that, in a time-homogeneous Markov process, the conditional transition probabilities p_{ij} remain constant over time (see Section 3.5). By comparing transition probabilities over chunks of the time-series the test looks for meaningful changes in transition probabilities over time. The test as it is postulated here has a weakness to detect periodic or cyclic Markov processes, however, it should be able to detect any evidence against the time-homogeneity of a process.

By dividing the sequence into chronological parts and estimating their transition matrices, the differences over time are examined. To facilitate this, the transitions from each state are collected and divided into c equal-sized chronological chunks. Each chunk contains an equal portion of transitions from every state, ensuring a balanced representation. The transitions from each state are partitioned into c parts based on their chronological occurrence in the sequence, preserving the temporal order of transitions for each individual state. If the transitions from a particular state cannot be evenly divided into c chunks, the remainder (of size r) is distributed across the first r chunks. Using these chunks, the algorithm generates empirical 1^{st} order conditional transition matrices similarly as it was outlined in section 4.3.1.

This splitting method – referred to from now on as *State-Balanced splitting* – was chosen for two reasons. The primary reason is to ensure that each chunk contains transitions from every state, which is crucial for producing reliable and representative transition matrices. If the sequence were split purely based on its chronological occurrence, certain chunks might lack transitions from some states, especially those visited infrequently. Such missing transitions often lead to likely incorrectly estimated transition probabilities, as demonstrated in *Example 2*. Secondly, the method mitigates the use of pseudo-counts in most cases when calculating the conditional transition matrices, while maintaining chronological order of the resulting chunks and capturing representative probabilities for each one. Both these considerations are particularly important for lower numbers of states, which often result in very long sections dominated by a single state, followed by sections dominated by another state. This effect was especially apparent during initial tests with cognitive state sequences of the *C. elegans* dataset.

Example 2: Given the same sequence like this:

B → A → B → B → A → A → B → B → A → B → B → C → C → C → C → B

> Splitting a sequences based on length, in cases with rare states would result in non representative estimated transition matrices. Given the sequence above, a split into 2 or 3 chunks would result in transition probabilities of $\frac{1}{3}$ for all possible transitions (C → A, C → B and C → C) in the first 1 or 2 chunks. This is likely a bad approximation of the transition probabilities within the first two thirds of the sequence. Because C is visited very rarely its transitions will only be captured within some chunks, in term resulting in non-representative chunks. The *State-Balanced splitting* is demonstrated in more detail in *Example 3*.

The obtained transition matrices for each of the chunks are compared to each other using the Frobenius norm of the difference matrix, which is obtained by subtracting one from the other. Since the ordering of matrices has no influence on the Frobenius norms, this process results in $c\frac{c-1}{2}$ comparisons per sequence :

$$A_{diff_{1,2}} = A_{chunk1} - A_{chunk2} \qquad ||A_{diff}||_F = \sqrt{\sum_{i,j} |a_{i,j}|}$$

The test aggregates these differences between each possible pair of matrices into the sample d. This represents the Frobenius norms of the difference matrices for this one sequence. This means the sample (d) is bigger if

one has more chunks and it has a minimal size of 1 for only 2 chunks.

$$|d| = \sum_{i=1}^{c-1} i$$

Example 3: Given the same sequence as in *Example 1*:

A → B → B → A → C → C → C → A → B → A → C → A → B → B → C → B → A → C → B → A → A → B → A → A
→ →

The workflow of *State-Balanced splitting* entails the creation of a chronological list of transitions from each state, where each row in Table 4.2 represents transitions from a particular state. When individually splitting each of these rows, the algorithm guarantees that each chunk inherits transitions from each state. This should lead to more representative transition probabilities within each chunk.

Table 4.2: Chronologically listed transitions per state (rows) split into roughly equal sized chunks (columns).

State	Chunk 1	Chunk 2	Chunk 3
A	(A, B); (A, C); (A, B);	(A, C); (A, B); (A, C);	(A, A); (A, B); (A, A);
B	(B, B); (B, A); (B, A);	(B, B); (B, C); (B, A);	(B, A); (B, A);
C	(C, C); (C, C);	(C, A); (C, A);	(C, B); (C, B);

Chunk 1

	A	B	C
A	0	0.6̇	0.3̇
B	0.6̇	0.3̇	0
C	0	0	1

Chunk 2

	A	B	C
A	0	0.3̇	0.6̇
B	0.3̇	0.3̇	0.3̇
C	1	0	0

Chunk 3

	A	B	C
A	0.6̇	0.3̇	0
B	1	0	0
C	0	1	0

Chunk 1 - Chunk 2

	A	B	C
A	0	0.3̇	-0.3̇
B	0.3̇	0	-0.3̇
C	-1	0	1

Chunk 1 - Chunk 3

	A	B	C
A	-0.6̇	0.3̇	0.3̇
B	-0.3̇	0.3̇	0
C	0	-1	1

Chunk 2 - Chunk 3

	A	B	C
A	-0.6̇	0	0.6̇
B	-0.6̇	0.3̇	0.3̇
C	1	-1	0

Figure 4.3: Workflow of the first part of *stationary_property_test()* function. The transitions from a state are split equally among chunks. In cases with no equal distribution (e.g.: **B**), the first chunk(s) will get one more transition. The 1st-order conditional transition matrices of each chunk are calculated, then each combinations difference matrix is generated. These are compared to each other later using the Frobenius norm.

The difference matrices are created by subtracting the conditional transition matrices of each chunk matrix from all others. From these one can now calculate the corresponding Frobenius norms (**1.56**, **1.7** and **1.89**).

The objective of the test is to compare the distribution of the sample d to the expected distribution D of a time-homogeneous 1st-order Markov process. To achieve this, the empirical conditional transition matrix of the original sequence is used to generate N (set by the *simulations* parameter) newly simulated sequences which will be used to approximate the underlying distribution of a time-homogeneous process. These simulated sequences are then similarly split into the same number of chunks as the original sequence, and the differences between the chunks within each sequence are calculated. The combined results over all N simulations are aggregated in a one-dimensional array of size $N * |d|$, which serves as the underlying

distribution of Frobenius norms for sequences stemming from a time-homogeneous 1st-order Markov process with the same *stationary* distribution. This approximation of the underlying distribution D is called $\hat{D}$.

Depending on the *test_mode* parameter, the *stationary_property_test()* can return one or two p-values (KS-test and T-test). Using the samples d and $\hat{D}$ up to two H_0 are tested. The first H_0 (*test_mode* = *'ks'*) states that the cumulative distribution function (CDF) of Frobenius norms from the chunk-matrices of the sequence is less than or equal to the CDF of the sample $\hat{D}$. The alternative hypothesis (H_A) states that the CDF of d is greater than the CDF of $\hat{D}$. The second H_0 (*test_mode* = *'ttest'*) states that the mean of the Frobenius norms calculated from d are on average smaller or equal to the average of those calculated from $\hat{D}$. While a rejection would mean, that the mean of d is bigger than the mean of $\hat{D}$.

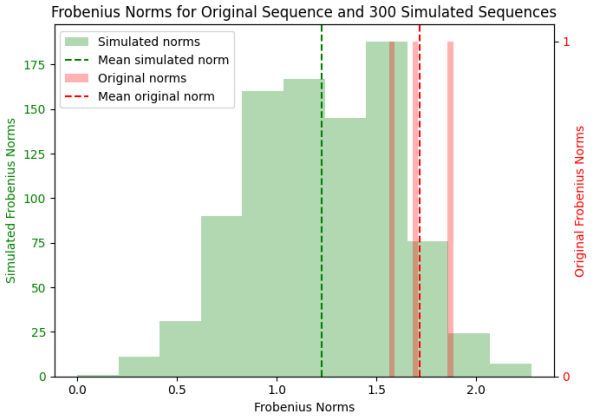
These H_0 aim to represent a consequence of time-homogeneity, which often implies a stationary distribution, except in cases like periodicity or reducibility. Significant results for either test mode should prompt researchers to consider whether a sequence violates time-homogeneity and lacks a stationary distribution.

Example 3: Continue example from above:

> In this example the Frobenius norms of the difference matrices (d) are: **1.56**, **1.7** and **1.89**. These norms are now compared to the distribution of Frobenius norms generated from sequences stemming from simulated sequences ($\hat{D}$) using the estimated conditional transition matrix of the sequence above:

	A	B	C
A	$0.\dot{2}$	$0.\dot{4}$	$0.\dot{3}$
B	0.625	0.25	0.125
C	$0.\dot{3}$	$0.\dot{3}$	$0.\dot{3}$

used to simulate 300 sequences



The KS-Test and T-Test resulted in significant p-values (**0.007** and **0.009**), indicating sufficient evidence to reject the null hypothesis and accept the alternative hypothesis. This would mean the sequence likely does not originate from a time-homogeneous Markov process.

Figure 4.4: Workflow for the second part of the *stationary_property_test()* function. The 1st-order conditional transition matrix derived from the complete sequence is used to simulate multiple sequences. For each simulated sequence, the Frobenius norms between chunks are calculated and aggregated into an approximated underlying distribution. This distribution is then compared to the Frobenius norms from the original sequence using two statistical tests: the Kolmogorov-Smirnov (KS) test and the T-test.

The KS-Test is used to assess whether the samples come from the same distribution. The H_A is that the distribution of the original sequence's sample is shifted to the right (indicating higher Frobenius norms) or does not follow the expected normal distribution, which would suggest that transition probabilities are not constant signaling a time-inhomogeneous process. Otherwise, one can conclude that the sequence shows no signs of time-inhomogeneous behavior and the H_0 is kept. The test is conducted using the *scipy.stats.ks_2samp* function with the following settings:

```
ks_2samp(sample_D, sample_d, alternative='greater')
```

The T-test does simply compare the means of both samples d and $\hat{D}$ to determine if the $\hat{\mu}_d$ is significantly bigger than $\hat{\mu}_{\hat{D}}$. This test is conducted using the *scipy.stats.ttest_ind* function with the following settings:

```
ttest_ind(sample_d, sample_D, alternative='greater')
```

The difference between the order is resulting from different implementations of *ks_2samp* and *ttest_ind*. In the analysis generally a significance level of α= 0.05 is used. The H_0 is rejected if one of the p-values is below 0.05, suggesting the sequence likely is not originating from a stationary first-order Markov process.

Chunk Algorithm

As an addition, I created an algorithm that determines an adequate number of chunks if the *chunks* parameter is not specified. This is done using the following formula:

$$c = max(\lceil \sqrt{\frac{R}{n}} \rceil, 2)$$

R = amount of occurrences of the rarest state in the sequence
n = amount of different states in the sequence

The formula determines the number of chunks (c) based on the occurrence of the least frequent state (R) and the number of distinct states (n). It first estimates the mean number of transitions from the least frequent state to each distinct state by dividing its total occurrences by n. To account for variability in these transition probabilities, the square root of the mean is taken, ensuring that this variability is reflected in each *chunk*. Although edge cases with very sparsely distributed transition probabilities may not be fully captured by the algorithm, this method consistently produces a reasonable value for c. Each *chunk* should exhibit transition probabilities somewhat similar to those of the complete sequence, provided that the process is stationary. Finally, a ceiling function is applied to guarantee a minimum value of two, ensuring the *stationary_property_test()* algorithm functions as intended, as it requires at least two *chunks* to operate.

It is important to note that more chunks are not always better. Fewer chunks are a conservative choice, effectively capturing slow changes in transition probabilities. In contrast, more chunks can detect faster changes but increase sensitivity, amplifying the influence of outliers. With more chunks the possibility for outliers is increased, while their effect on the ample d is increased because of the higher number of comparisons. Thus, the choice of c should balance these considerations to suit the specific application.

Example 3: Continue example above:

For the previous example, the optimal number of chunks according to the algorithm is given by:

$$c = max(\lceil \sqrt{\frac{R}{n}} \rceil, 2) = 2 \qquad \text{since} \quad R = 6, n = 3$$

Repeating the calculation with only 2 chunks yields a different result, as the resulting p-values of **0.196** and **0.172** do not provide sufficient evidence to reject the null hypothesis. This indicates that here the amount of chunks chosen does make a difference.

Computational Complexity

Also here the computational complexity is mainly determined by simulating the test statistic. The process is repeated s times and in each repetition:

- m is the sequence length
- s is the number of simulated sequences generated
- n is the number of states
- c is the number of chunks

1. A sequence of transitions of length m is simulated $\rightarrow$ O(m)
2. Transitions are distributed by looping over chunks c and states $n \rightarrow$ O(cn)
3. A nested loop over the chunks c creates empirical conditional transition matrices $\rightarrow$ O(m)
4. Finally, the Frobenius norms are calculated for each pair of matrices $\rightarrow$ O(c^2n^2)

When all these complexities are added together, the total computational complexity results to $O(s * (m + cn + m + c^2n^2))$, which would simplify to $O(sn^2c^2)$ or $O(n^2c^2)$.

In practice, since m is typically much larger than n or c, the complexity should generally be closer to $O(sm)$. However, because the n^2 and c^2 terms grow quadratically, they will eventually surpass m as they increase, so the true upper bound becomes $O(n^2c^2)$.

4.3.3 Viability of the Statistical Tests

The statistical tests (see Subsections 4.3.1 and 4.3.2) were tested on their ability to detect sequences originating from processes that violate any of the definitions of a Markov process (see Section 3.5) or the assumptions of the *NC-MCM* framework.

Since the state space is predefined by the user through the specified number of clusters, it does not require testing. The *Markov Property* is evaluated using the *markov_property_test()*, while the assumptions of time-homogeneity—and by extension, the likelihood of a stationary distribution—are assessed with the *stationary_property_test()*. Although the latter does not explicitly test for the presence of a stationary distribution, it examines evidence against constant transition probabilities.

Both newly proposed and existing tests were applied to sequences generated from a variety of processes: a random process, 1st-, 2nd- and 3rd- order Markov processes, two distinct types of non-stationary 1st-order Markov process, a discretized Ornstein-Uhlenbeck process and a discretized Random Walk. To evaluate the general ability of the tests to distinguish sequences of different origins, 100 sequences per origin of length 3000 and 5000 were tested across state spaces ranging from 1 to 15 states. These sequence lengths and state space sizes were chosen to align with the intended application of the framework—analyzing neural activity data. The study conducted by Grosse-Wentrup et al. [96] on a dataset of C. elegans consists of recording of approximately 3000 frames, with most analyses focusing on 3–5 cognitive states. In the *stationary_property_test()* the amount of chunks was fixed at 5 for all sequences, while the simulations were set to 50 in both tests.

4.3.4 Power Curves of Statistical Tests

After this, the power and viability of the statistical tests was measured by creating multiple power curves using different parameters. Typically these plots are generated by keeping a fixed sample size and altering the effect size. The effect size is a value describing the aspect that should be detected by the test. However, in the case of (i) testing if a sequence is not violating the Markov property and (ii) if a sequence shows signs against time-homogeneity, it is not straight forward to gradually change the effect size. For this reason, an aspect that relates to the effect size – the magnitude of the difference between the distribution of the null hypothesis (H_0) and the alternate hypothesis (H_A) – was selected.

(i) The *markov_property_test()*, tests if a sequence could stem from a Markov process of 1st order. Its H_0 states that the sequence does not violate the

Markov property by comparing variance over the $t-2$ axis of transition probability tensors to variances of true 1st order Markov processes. While the H_A states that the process violates the Markov property of a 1st order process and likely comes from a higher order Markov process or a different non-Markovian process. In this case, different effect sizes were approximated by using Markov processes of different orders, with a 2nd order process being the maximal effect size achievable and higher order processes becoming harder and harder to detect as they approach a random process[1] which should not be rejected[2]. A significance level of $\alpha = 0.05$ was used for all analyses.

(ii) The *stationary_property_test()* is testing for the constant transition probabilities or time-homogeneity of a Markov process, with the H_0 being that the sequence could originate from a process with constant transition probabilities and the H_A would mean the process seems to violate time-homogeneity and likely originates from a time-inhomogeneous process. Different effect sizes were equated by either (1) adding more or less discrete transition probability changes within the process or (2) making directed compounding changes to the transition probabilities of the process (approaching a continuous change - *pseudo continuous*). The way these were simulated is described in Subsection 4.4.3. For the initial analysis, the significance was also set to $\alpha = 0.05$, however, for clearer results another run with $\alpha = 0.01$ was done.

Generated Sequences for Power Curve Plots

The power curves for both tests – *markov_property_test()* and *stationary_property_test()* – were created for sequences of lengths 1000, 2000, 4000 and 8000. Each curve was generated by averaging the results of 50 cycles of randomly independently generated sequences coming from different processes. These processes were chosen to reflect changing effect sizes as discussed above, while a constant value of 50 was chosen for the *simulations* parameter.

For the *markov_property_test()* a measure for the effect size was calculated using the *Cohen's D* of the 50 cycles for each of the different sequence types. It is a measure for the difference between the underlying distribution and the sample to test. For this two groups are needed, the seq-group consists of the results for the variance over $t-2$ of each of the 50 randomly generated sequences. While, the $dist$-group consists of the distributions of variances over $t-2$ generated by the simulations of 1st-order Markov sequences (see Subsection 4.3.1), which means:

$$|dist| = |seq| * 50$$

Since, J. Cohen initially proposed the effect measure only for equal sized groups [127], J. Hartung et al. proposed a pooled standard deviation for unequal groups sizes [128]. Since it this use case, the $dist$-group is 50-times bigger (for each simulation), the pooled standard deviation is used in further calculations.

$$\text{Cohen's D} = \frac{\mu_{seq} - \mu_{dist}}{\sigma_{pooled}} \qquad \text{were} \qquad \sigma_{pooled} = \sqrt{\frac{(n_1 - 1) * \sigma_{seq}^2 - (n_2 - 1) * \sigma_{dist}^2}{n_1 + n_2 - 2}}$$

1: This is because as a Markov process increases in order the influence of each past state on the future becomes more diluted. If one considers the following general definition of a Markov process of n^{th} order:

$$P(X_t|X_{t-1}, ..., X_0) = P(X_t|X_{t-1}, ..., X_{t-n})$$

and one increases n towards infinity, the influence of each past state of the sequence approaches 0.

$$\lim_{n\to\infty} \left| \frac{\partial P(X_t|X_{t-1}, ..., X_{t-n})}{\partial X_{t-x}} \right| = 0$$

$$x \in \{1, 2, 3, ..., n\}$$

This would mean that the future state X_t becomes increasingly independent of any individual past state, indicating that the process is approaching a random process, characterized by the independence of the past.

2: On the other hand, a random process can also be seen as a Markov process with equal transition probabilities. Such a Markov process of 1st order should not be rejected by the test.

The effect sizes for the *stationary_property_test()* test were given by the values of the *KS-statistic* or *T-values* returned by the corresponding method. For the KS-test the values can range from 0 – no difference between distributions – to 1 – big difference between distributions. The T-test calculates *T-values* ranging from -∞ to ∞. If a sequence's Frobenius norms (seq-group) are smaller than the Frobenius norms of stationary processes (=$dist$-group), the *T-values* will be below zero. In this case high negative values will not be seen as violating the null hypothesis of the test as explained in Subsection 4.3.2.

Power curves were created for 50 sequences per origin with 3, 5 and 10 states. Power is defined as 1 − Type II error for all sequences. Effect sizes were evaluated based on the processes of origin[3] for the tests:

3: For a detailed description of how the processes are simulated see Subsection 4.4.3

- (i) *markov_property_test()*
  - 1st order Markov process (no effect)
  - Random process (no effect)
  - 5st order Markov process
  - 4st order Markov process
  - 3st order Markov process
  - 2th order Markov process random process
- (ii) *stationary_property_test()* with 5 chunks
  - (1) Different amount of changes (*discrete*)
    - 1st order Markov process (no effect)
    - Random process (no effect)
    - Markov process with 200 changes
    - Markov process with 100 changes
    - Markov process with 50 changes
    - Markov process with 5 changes
  - (2) More gradual changes (*pseudo continuous*)
    - 1st order Markov process (no effect)
    - Random process (no effect)
    - Markov process with 5 changes and $\epsilon = 0.02$
    - Markov process with 5 changes and $\epsilon = 0.04$
    - Markov process with 5 changes and $\epsilon = 0.06$
    - Markov process with 5 changes and $\epsilon = 0.08$

4.4 Data

The functionalities of the toolbox are demonstrated using two datasets. The first dataset was already used in the original paper describing the *NC-MCM* framework [96] and includes imagings of a big part of the brain and behavioral labels of *C. elegans* [73]. The second one is a dataset downloaded from the DANDI archive, which is a recording of mouse neurons during a Y-maze decision task [18].

4.4.1 *Caenorhabditis elegans*

Firstly, on brain-wide calcium imaging in *Caenorhabditis elegans* [73] from the research paper by Kato et al. in 2015 which recorded approximately 1/3 of all neurons. The dataset consists of neuronal activity data from 5

different worms over 18 minutes with a frame rate between 2.7 – 3.1 Hz (approximately 3000 time-points). The young adult worms were recorded in a specialized microscope while immobilized in a microfluidic device. Each time point has been labeled with a certain behavior using a set of characteristic neurons. The same dataset was also used in some previous papers where the framework was presented [96, 104]. While plots were created for each worm, the results presented here are limited to plots created for worm 4. Images were acquired using two CCD cameras connected via a beam splitter and a 63× objective, streaming unbinned images at 30.3 frames per second. Simultaneous behavior recordings under infrared illumination were made using a separate CCD camera at 4× magnification and 10 frames per second [73].

4.4.2 *Mus musculus*

For the successful application of NCMCM, brain-wide imagings are preferred. However, open-source brain-wide imaging datasets are challenging to find. Therefore, the second dataset used in this study, downloaded from the DANDI archive [48], is not a brain-wide imaging dataset but is limited to the optical cortex of a mouse. This dataset was chosen to demonstrate the toolbox's viability as well as its limitations when dealing with incomplete imagings. It consists of two-photon calcium imaging data recorded from four different planes within the optical cortex of mice (*Mus musculus*) [18]. The researchers studied decision-making within the mouse brain. The behavioral and neural recordings were done on eight 3-6 month old C57BL/6J mice from Jackson Laboratory (stock no. 000664). Each mouse underwent 141 sessions (only 52 sessions were found on DANDI archive for mouse 10), where a typical session for a mouse consisted of 350-450 decision-trials.

Mice were trained to navigate a virtual reality Y-maze using visual cues to choose rewarded locations based on changing rules, without explicit signaling. Rule associations were switched periodically, requiring mice to combine visual cues with rule estimates to update their choices and rule expectations for optimal reward acquisition. For demonstration purposes, data from the session on the 21st September 2021, from the first mouse 1st mouse (referred to as Mouse 10) is presented, as including data from all the mice would extend the scope of the work. The imaging is restricted to layers 2/3 of the V1 area. In this analysis the 4 recording planes were simple concatenated to form a single dataset of shape $(37748, 444)$, consisting of 444 neurons and 37748 time points.

The fluorescent traces were deconvolved using the constrained AR-1 OASIS method [129] after a motion correction and constrained non-negative matrix factorization for source identification. For a more in-depth description of the methods used in acquiring the data see the original publication [18].

In the dataset, behavior was limited to frame-aligned position of the head. As these behavioral labels cannot be used as discrete behavioral labels, movement or behavior had to be decoded from the frame-aligned position of the animal at each time-point (≈6 fps). This way the ten following behaviors can be deduced[4], with *invisible* meaning the position of the mouse is not known or recorded. As no standard decodings were found

4:

- *moving forward*
- *moving backward*
- *going right*
- *going left*
- *moving forward & right*
- *moving forward & left*
- *moving backward & right*
- *moving backward & left*
- *standing still*
- *invisible*

in the literature, the thresholds for lateral or frontal movement were set to a change of ≈0.85 cm and ≈0.085 cm between frames, indicating a frontal speed of 0.05 m/s or 0.18 km/h and a lateral speed of 0.005 m/s or 0.018 km/h.

As noted, this dataset is not optimal for the application of the *NC-MCM* framework, since the non-availability of discrete behavioral labels as is often the case with open-source neuronal data, is not ideal. Nonetheless, using a second dataset to demonstrate the potential use of the toolbox on new data is important. Furthermore, there have already been attempts of creating an implementation of the *NC-MCM* framework for continuous labels [104], which would allow the use of many open-source dataset with continuous behavioral labels.

4.4.3 Synthetic Sequences

As stated, the *NC-MCM* models cognitive state sequences as Markov processes. This is a constraint set by the framework to ensure that the sequences are causal, as Markovian processes inherently rely on present states to determine future transitions. While the constant state space property (see Section 3.5) can be satisfied by fixing the amount of states in the *cognitive level*, the Markov and the Stationary property need to be tested for. To evaluate the efficacy of the statistical tests developed in this work, synthetic sequences were employed. Among others, the processes chosen were the Random Walk and the Ornstein-Uhlenbeck (OU) process. These choices are rooted in their relevance to cognitive modeling and their distinct statistical properties.

The Random Walk, while not a Markov process due to its dependency on the entire sequence history for determining future states, is widely used to simulate accumulative processes and noise-driven dynamics often observed in behavioral and neural data [130, 131]. Its non-Markovian nature makes it a useful benchmark for testing whether the developed methods can identify deviations from strict causality assumptions. In contrast, the OU process is a classical Markovian process with exponential decay in its auto-correlation, making it ideal for modeling neural and decision-making dynamics [132, 133]. The OU process has been extensively used to describe psychological discrimination tasks and response times, where its Markovian properties provide a tractable mathematical framework for understanding the relationship between neural activity and behavior.

By testing against both processes among others, this study ensures the robustness of the statistical tests across sequences with diverse dependency structures and cognitive relevance. These synthetic sequences also bridge the gap between theoretical model assumptions and real-world cognitive phenomena, reinforcing the applicability of the proposed methods.

Performance of the Statistical Tests

To test the performance of the statistical tests (see Subsections 4.3.1 and 4.3.2), multiple different synthetic sequences were generated. These sequences represent a variety of process types, which allow to test violations of Markov properties under diverse conditions. All sequences were generated using a standardized approach, employing *Numpy.random* methods for randomization.

Since is hard to identify a qualitative source for cognitive state sequences in nature, in this Subsection performance of the tests on many different sequences will be investigated. The eight different sequences used for preliminary testing included a 1^{st}-, 2^{nd}- and 3^{rd}- order Markov process, two different custom time-inhomogeneous processes, a random process, a discretized Random Walk as well as a Ornstein-Uhlenbeck process.

The Markov processes were a obvious pick, since *NC-MCM* postulates a Markov process for cognitive states. However, it is hard to know from which distribution the transition probabilities of the cognitive state sequences are picked from. There are different explanation to why [134] distributions in nature are often normal. If the reason is because the conditions underlying the Central Limit Theorem (CLT) are frequently met or the power of normality may stem from the maximum entropy property of normal distributions is not central to the decision here. Ultimately transition probabilities for n states were either sampled from a normal distribution $\mathcal{N}(\frac{1}{n}, 1)$ with values truncated between 0 and 1 or a uniform distribution, which is normalized in a second step .

The marginal probability distributions for the transition probabilities of each state, sampled from uniform or truncated normal distributions, are depicted in Figures 4.5 and 4.6. Each curve represents the empirical distribution from which the probabilities were sampled for each state. As expected, the curves are identical within each plot. The distributions sampled from the uniform distribution are slightly skewed to the right, favoring above-average probabilities. In contrast, those sampled from the truncated normal distribution behave similarly in smaller state spaces but skew slightly to the left as the number of states increases. Following these preliminary results, it was chosen to use the uniform distribution in Chapter 5, as the behavior across the state spaces considered (5, 10, 15) is fairly consistent between the two sampling methods.

Next two custom time-inhomogeneous processes were developed, which allowed to easily change effect sizes. This was needed to be able to create useful power plots, as explained later in Subsection 4.3.4. The random process was added as a quality control, as it can ultimately be seen as a Markov process with equal transition probabilities.

Lastly two additional processes – a Ornstein-Uhlenbeck process and a Random Walk – were included. While the latter has been argued to be a model for neural state drift over multiple trials [131], the former is has been employed to simulate the human decision process in cognitive decision tasks [133]. Therefore, the different processes were included in testing as potential models of the evolution of cognitive states over time, with one of them being a Markov process and the other one not.

- **Markov processes:** The x^{th} order Markov processes are generated by using probabilities from randomly generated transition matrices with shape $n * n * x$ (n =number of states). The transitions are picked from a uniform distribution from 0 to 1 (or a truncated normal distribution between 0 and 1 with $\mu = 1/n$ and $\sigma = 1$). The values are then normalized so the probabilities from each state sum up to exactly 1. This way symmetric marginal distribution over the sampled transition probabilities to the next state are achieved (see Figures 4.5 and 4.6). This means that each states transition probabilities are ultimately sampled from the same distribution.

In each of the simulated Markov sequences the first x state(s) are chosen randomly from a uniform distribution.

- **Random processes:** The random sequence is generated by randomly picking the next state from state space at each time-point.

- **Time-inhomogeneous Markov processes:** Two different types of time-inhomogeneous processes are tested. A *discrete* time-inhomogeneous process as well as a *pseudo-continuous* time-inhomogeneous process were created. Both types swap out the transition matrix for sequence simulation with a new one. The difference lies in the new matrix used.

  - ***Discrete* time-inhomogeneous process:** In this case the new matrix is a completely new random 1st order. The changes are equally split over the length of the sequence generated.

  - ***Pseudo-continuous* time-inhomogeneous process:** Here the new matrix is created by adding or subtracting a constant random value from each of the values of the previous matrix. The sum of all the changes per row sums up to 0 (ensuring that no transition probabilities from a single state exceeded 1) while the maximum and minimum changes achievable are $\pm\epsilon$. After each perturbation the transition probabilities are clipped below 0 and above 1 and normalized. The changes are equally split over the length of the sequence generated.

- **Ornstein-Uhlenbeck stationary process:** This process is generated by discretizing a continuous Ornstein-Uhlenbeck process, characterized by reversion to a long-term mean ($\mu = 0$) with a reversion strength controlled by θ (here 0.5). At each step, a Gaussian noise term with standard deviation σ is added $\sigma \sim \mathcal{N}(0, 1)$. The Ornstein-Uhlenbeck process is time-homogeneous because transitions do not depend on t beyond $t - 1$ and the mean reversion which limits its total variance. The continuous process – characterized by $X_0 \sim \mathcal{N}(\mu_0, \sigma_0^2)$ – is digitized into N discrete states by dividing its range into N equally spaced intervals and assigning each value to the nearest interval. The continuous process is given by:

$$X_t = \theta * (\mu - X_{t-1}) + \sigma$$

- **Random-walk processes:** This process is modeled as a simple Random Walk, where the value at each time step is the previous value plus a random variable with a noise term. The Random Walk is inherently time-inhomogeneous, as it lacks any mean-reverting behavior and its total variance increases over time. At each step, a Gaussian noise term $\sigma \sim \mathcal{N}(0, 1)$ is drawn and added to the current state. The continuous process is digitized into N discrete states by dividing its range into N equally spaced intervals and assigning each value to the nearest interval. The continuous Random Walk evolves according to the following equation:

$$X_t = X_{t-1} + \sigma$$

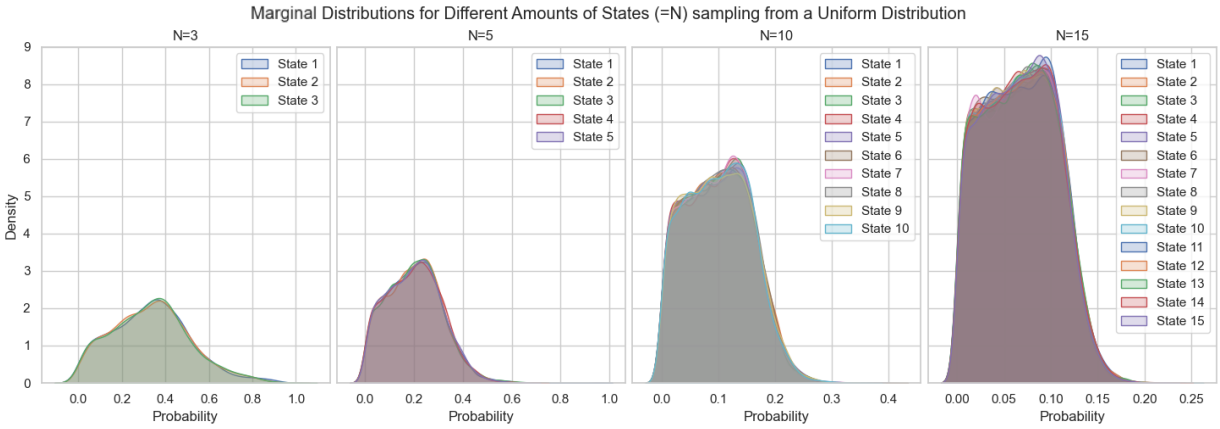


Figure 4.5: Marginal distributions of probabilities sampled from a uniform distribution for 1000 transition matrices. The probabilities between 0 and 1 are normalized after sampling so each row of the transition matrix sums up to 1.

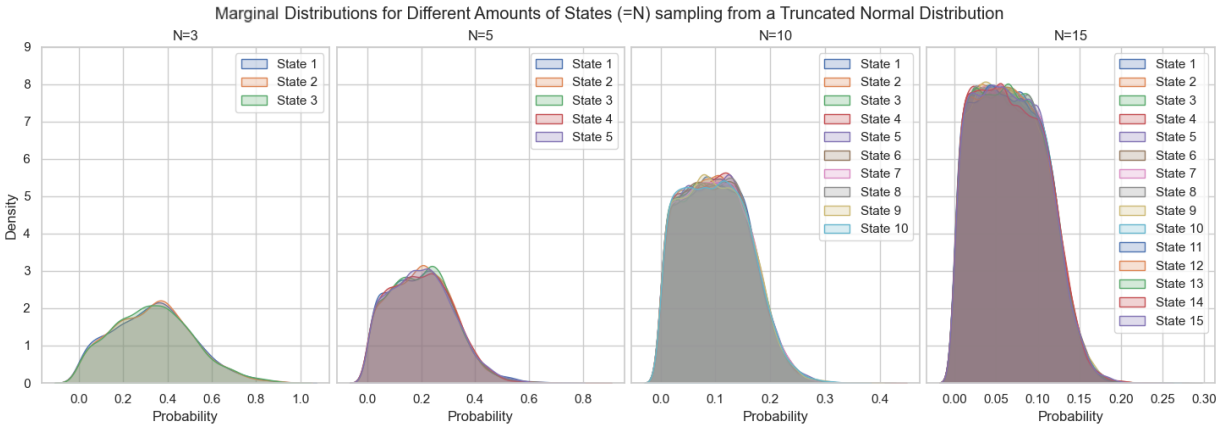


Figure 4.6: Marginal distributions of probabilities sampled from a truncated normal distribution for 1000 transition matrices. The probabilities are sampled from a distribution with $\mu = 1/N$ and a standard deviation of 1. This is accomplished using the *scipy.stats.truncnorm* function, which is allows to truncate the sampling between 0 and 1.

Results 5

As already mentioned the results will also be split up into three parts:

- The showcase and performance of the toolbox;
- The statistical tests and power analysis;
- The use of the toolbox on old and new data;

5.1 Toolbox

The results presented in this section show the capabilities and performance of the package. First, in Subsection 5.1.1 the performance of the main methods was tested including the statistical tests, while in Sections 5.2 and 5.3 the viability of both of the newly created statistical tests is assessed. As there were only minor differences in marginal distributions between transition probabilities sampled from uniform- or truncated normal distribution (see Figures 4.5 and 4.6), for all the following results the uniform distribution was used when simulating synthetic data. Lastly, the capabilities of the toolbox are shown by creating similar plots as those from the original *NC-MCM* paper [96] using the same data. More precisely plots shown in Subsection 5.4.1 use data from worm 4 of the dataset. In Subsection 5.4.2, the toolbox' usage of new data from the optical cortex of a mouse will be tested. For a better look at the resulting toolbox visit github.com/DriftKing1998/NC-MCM-Visualizer.

5.1.1 Testing Runtimes

To evaluate computational performance, each method was run 30 times to calculate the mean runtime for a given parameter set. Table 5.1 summarizes results for methods from Subsections 4.2.1 and 4.2.2 using the data of worm 4. Parameters for methods are listed in the rows of the corresponding column. In these tests, an *CustomEnsembleModel* of logistic regressions (*LR*) (see section 4.2.4) was used for all methods to project into probability space, while principal component analysis (PCA) was used to project into three-dimensional space for visualization. The remaining non-default parameter values for the methods (*cluster_num*, *nrep*, *chunks*, *legend* and *quivers*) are listed as well. Most methods had average runtimes under five seconds for the given parameters, except for *cluster_BPT()* (2m 46.75s) and *cluster_BPT_single()* (32.74s). In Table 5.1 the *stationary* parameter was set to *True* for both methods. With the *stationary* parameter set to *False*, runtimes decreased by roughly one-third (*cluster_BPT()*: 1m 51.25 s; *cluster_BPT_single()*: 22.151s).

To get a more precise look at the functions responsible for these computation times separate runtime measurements were done for the statistical tests used in the *cluster_BPT()* and *cluster_BPT_single()* methods. The tables with average runtimes for different parameters of the *markov_property_test()* and *stationary_property_test()* methods are presented in Tables 5.2 to 5.5.

Function	fit_model	cluster_BPT	cluster_BPT_single
Mean time	0.16 s	166.75 s	32.74 s
model	LR	LR	LR
nrep	/	5	5
cluster_num	/	5	5
chunks	/	5	5
stationary	/	True	True
Function	**plot_mapping**	**make_movie**	**make_comparison**
Mean time	1.17 s	1.20 s	3.49 s
mapping	PCA	PCA	PCA
legend	True	True	True
quivers	True	True	True

Table 5.1: Mean time elapsed for a set of methods of the *Database* and *Visualizer* classes. For all measurement an ensemble *LogisticRegression* (*LR*) model for clustering and a PCA as a mapping were used. The data was taken from worm 4. In both clustering methods both statistical tests were employed with default simulation counts (*markov_property_test()*: 200; *stationary_property_test()*: 100).

Each has sequence lengths indicated by rows and state space sizes or amount of simulations in columns (see Tables 5.2 to 5.5 - for color gradients see Table 7.1). By looking at the elapsed time in tables Tables 5.2 and 5.3 it is clear that the computational complexity of both methods grow mainly with the sequence length and slightly with state space size in this example. The observed computational complexity of the tests aligns with the expected theoretical complexity O(sm) (see Subsections 4.3.1 and 4.3.2), with the sequence length and the number of simulations (here constant) being the primary contributors to runtime. In Tables 5.4 and 5.5 the size of the test statistic also demonstrates a similar linear complexity growth, with a doubling of simulations taking double the time, as expected (see Subsections 4.3.1 and 4.3.2). Results with a self-determined *chunks* parameter are similar to a constant amount of *chunks* (see Table 5.5).

Length	5 states	10 states	15 states	20 states	50 states
1000	6.58 s	6.53 s	6.58 s	6.63 s	7.03 s
2000	13.13 s	13.12 s	13.13 s	13.19 s	13.89 s
3000	19.75 s	19.66 s	19.75 s	19.63 s	20.68 s
10000	65.66 s	65.52 s	65.66 s	65.24 s	68.58 s

Table 5.2: Mean elapsed time after 30 runs of the parameters as specified for the *markov_property_test()* method. Each iteration had a test statistic of size 1000 (*sim_m*).

chunks	Length	15 states	20 states	35 states	50 states
5	1000	6.66 s	6.87 s	7.04 s	7.19 s
5	2000	13.21 s	13.73 s	13.91 s	14.21 s
5	3000	19.79 s	20.79 s	21.07 s	21.26 s
5	10000	65.73 s	68.84 s	69.50 s	70.84 s
None	1000	6.69 s	6.73 s	6.85 s	7.01 s
None	2000	13.49 s	13.55 s	13.70 s	19.81 s
None	3000	20.24 s	20.41 s	20.68 s	36.31 s
None	10000	67.94 s	67.97 s	69.19 s	82.92 s

Table 5.3: Mean elapsed time after 30 runs of the parameters as specified for the *stationary_property_test()* method. Each iteration had a test statistic of size 1000 (*sim_s*) and the *chunks* parameter was set to 5 or chosen by the algorithm (*chunks=None*).

In Tables 5.2 and 5.3 the parameters *sim_m* and *sim_s* used the value of 1000. The results in Table 5.1 for the methods *cluster_BPT()* and *cluster_BPT_single()* used default test statistics of size 200 and 100 for *markov_property_test()* and *stationary_property_test()* respectively. Lastly, Tables 5.4 and 5.5 highlight that the amount of simulations (*sim_m* and *sim_s*) influence the duration as expected (see Section 6.1), with linear regressions using sequence length and simulations as input explaining most of the variance witnessed in runtime ($R^2 = 0.8721$; $R^2 = 0.8723$).

Length	Simulations 200	400	600	800	1000
1000	1.34 s	2.64 s	3.99 s	5.35 s	6.58 s
2000	2.66 s	5.32 s	7.86 s	10.48 s	13.13 s
3000	3.99 s	7.98 s	11.79 s	15.83 s	19.75 s
5000	6.72 s	13.48 s	20.24 s	26.99 s	33.71 s
7500	10.03 s	19.79 s	29.79 s	39.88 s	49.24 s
10000	13.27 s	26.38 s	39.85 s	52.64 s	65.66 s

Table 5.4: Mean elapsed time after 30 runs of the parameters as specified for the *markov_property_test()* method. Each sequence had 15 different states, while the size of the test statistic (*sim_m*) is given by the column headers (200, 400, 600, 800 and 1000).

Length	Simulations 200	400	600	800	1000
1000	1.33 s	2.67 s	3.99 s	5.33 s	6.66 s
2000	2.67 s	5.30 s	7.94 s	10.56 s	13.21 s
3000	4.02 s	8.00 s	12.07 s	16.00 s	19.79 s
5000	6.64 s	13.30 s	19.92 s	26.41 s	33.13 s
7500	9.96 s	19.72 s	29.62 s	39.96 s	49.57 s
10000	13.30 s	26.49 s	39.73 s	52.55 s	65.73 s

Table 5.5: Mean elapsed time after 30 runs of the parameters as specified for the *stationary_property_test()* method. Each iteration used 5 *chunks* and the sequence had 15 different states, while the size of the test statistic (*sim_s*) is given by the column headers (200, 400, 600, 800 and 1000).

5.2 Markov Property – *markov_property_test()*

The initial assessments results of the *markov_property_test()* using sequences as specified in Subsection 4.3.3 are shown in Figures 5.1 and 5.2. The test rejected nearly all 2nd order Markov sequences up to 15 states and most 3rd order Markov sequences of up to 10 states. While the sequences originating from a 1st order Markov process are seldom rejected (length 3000: 3.43% ± 2.53; length 5000: 3.36% ± 2.21;). The exact number of rejections for the 100 sequences of length 3000 of 3, 5, 10 and 15 states can bee seen in Table 5.6, while exact rejections for sequences of length 5000 are shown in Supplementary Table 7.3. When looking at the results displayed in Figures 5.1 and 5.2 one can see that the significant results for 3rd-order Markov processes as well as *discrete* time-inhomogeneous process are shifted to the right, meaning that sequences with more states are more likely showing significant results in longer sequences. The 2nd order Markov sequences with 15 states that show some non-significant outliers for 3000 unit long sequences, result in only significant results in 5000 unit sequences, experiencing a similar right-shift (more clearly seen in Figures 7.3 and 7.4). Apart from these differences and both random sequence types displaying greater p-values in the longer sequences the results for 3000 and 5000 long sequences show no major differences.

5.2.1 Power analysis

After this initial assessment, power curves were created as outlined in Subsection 4.3.4 for 3, 5 and 10 states (see Figures 5.3 to 5.5). The results show the power of the *markov_property_test()* on the Y-axis, while for test sequences with no effect size (i.e., originating from a 1st-order Markov process or a random process[1]), the value instead reflects the Type I error rate[2]. The mean power on the Y-axis was calculated as $1 - \beta$, with β being the Type II error averaged over all 50 sequences per process type. On the X-axis the effect size is shown as *Cohen's D* with pooled variance.

1: A random process can be seen as a Markov process with equal transition probabilities. Such a Markov process of 1st order has no effect size and should not be rejected by the test.

2: To recapitulate, **Type I error (α)** happens if one falsely rejects the null hypothesis, while **Type II error (β)** happens when one incorrectly accepts the null hypothesis. In this case, a **Type I error** occurs when one rejects the H_0 (p-value <= α) even though the sequence originates from a 1st order Markov process. A **Type II error**, on the other hand, happens if the H_0 is kept when the sequence does not come from a 1st order Markov process.

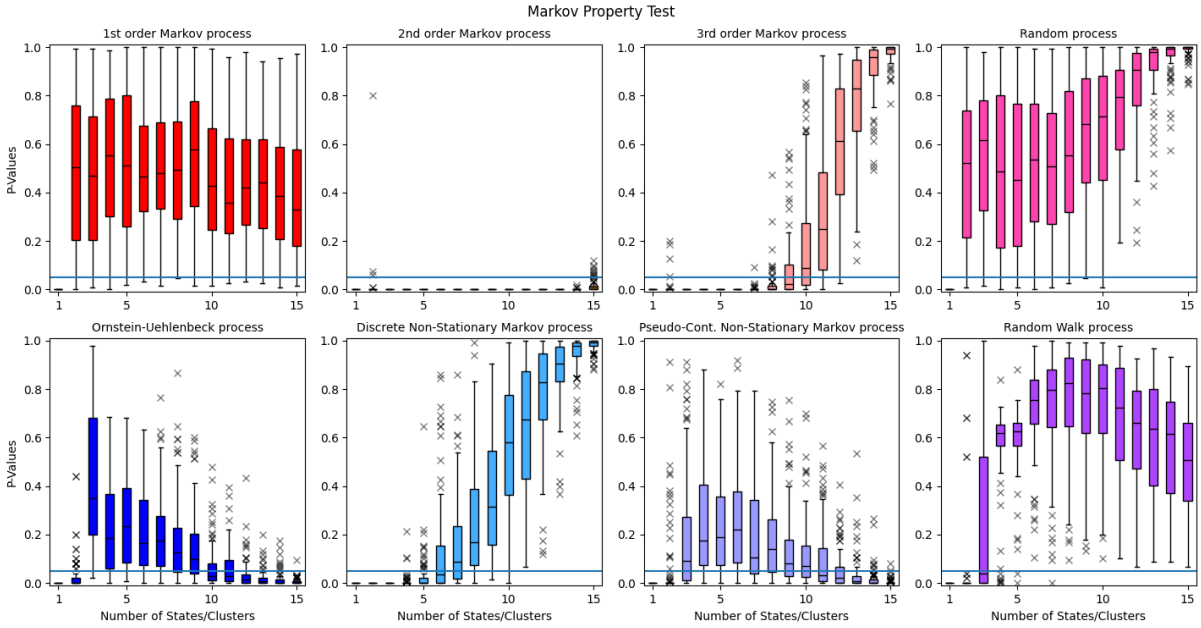


Figure 5.1: This plot shows results of the test for the Markov property of a 1st-, 2nd- and 3rd- order Markov process, a random process, a Ornstein-Uhlenbeck process, a *discrete* and *pseudo-continuous* non-stationary process and a discretized Random Walk. All sequences were 3000 units long and the amount of different states are listed on the X-axis, while p-values are indicated by the Y-axis. Each boxplot visualizes results from 100 sequences and the parameter *simulations* was fixed at 150 for all runs.

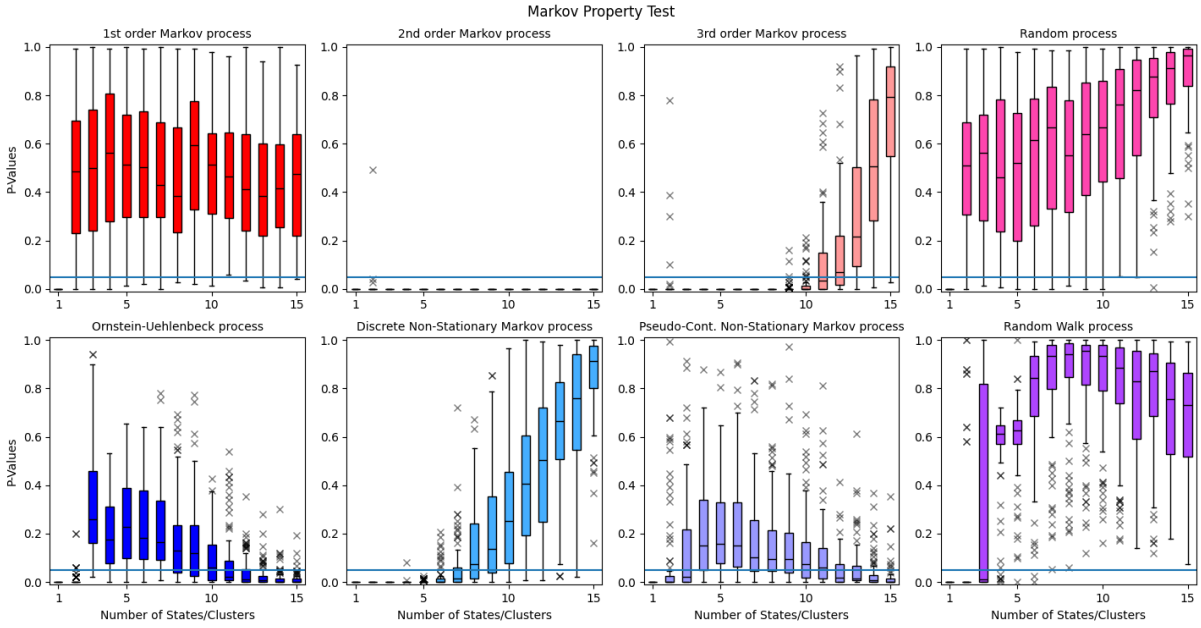


Figure 5.2: This plot shows results of the test for the Markov property of a 1st-, 2nd- and 3rd- order Markov process, a random process, a Ornstein-Uhlenbeck process, a *discrete* and *pseudo-continuous* non-stationary process and a discretized Random Walk. All sequences were 5000 units long and the amount of different states are listed on the X-axis, while p-values are indicated by the Y-axis. Each boxplot visualizes results from 100 sequences and the parameter *simulations* was fixed at 150 for all runs.

Table 5.6: This table lists the exact rejection counts for the *markov_property_test()* at a significance level of $\alpha = 0.05$, based on 100 simulated sequences of 3000 units each. The sequences were generated from processes including 1st-, 2nd-, and 3rd-order Markov processes, a random process, an Ornstein-Uhlenbeck process, a *discrete* and *pseudo-continuous* non-stationary process, and a Random Walk. Results are visualized in Figure 5.1.

3 States	HO kept	H_0 rejected
1st order M.	92	8
2nd order M.	0	100
3rd order M.	0	100
Random	94	6
OU	92	8
Discrete non-stat.	0	100
Pseudo-Cont. non-stat.	61	39
Random Walk	41	59

5 States	HO kept	H_0 rejected
1st order M.	96	4
2nd order M.	0	100
3rd order M.	0	100
Random	92	8
OU	96	4
Discrete non-stat.	16	84
Pseudo-Cont. non-stat.	84	16
Random Walk	98	2

10 States	HO kept	H_0 rejected
1st order M.	98	2
2nd order M.	0	100
3rd order M.	64	36
Random	98	2
OU	40	60
Discrete non-stat.	97	3
Pseudo-Cont. non-stat.	56	44
Random Walk	100	0

15 States	HO kept	H_0 rejected
1st order M.	94	6
2nd order M.	9	91
3rd order M.	100	0
Random	100	0
OU	4	96
Discrete non-stat.	100	0
Pseudo-Cont. non-stat.	98	2
Random Walk	100	0

The power curve for 3 states (see Figure 5.3) shows that while the 1st-order Markov process and the random process similarly indicate a Type I error at 5% on average ($\sigma = 0.98$), the power for 2nd-, 3rd- and 4th- order Markov processes consistently is over 80%. Even the 5th-order sequences showed significance above 90% for the longer sequences (4000 and 8000).

The results of 5 states (see Figure 5.4) of the 2nd- and 3rd-order Markov processes – signaled by | and ▲ – demonstrate a consistent power of >90% for all sequence lengths tested. The longer sequences (length 4000 and 8000) also reach similar power for 4th-order Markov processes – marked as a ◆ in the plot. Type I error for sequences with no effect size is 4.75% on average ($\sigma = 1.51$).

The state space of size 10 (see Figure 5.5) in general shows lower power for most processes, indicating higher Type II errors. Mean power for 2nd-order sequences for most lengths tested (2000, 4000 and 8000) was consistently at 100%, while also 3rd-order sequences had power >90% on average for the 8000 unit long sequences and >70% on average for 4000 unit sequences. Type I error for 10 state sequences with no effect size is 1.25% on average ($\sigma = 1.19$).

In general results show power curves that reflect increasing power with stronger effect sizes most apparent in the smaller state spaces Figures 5.3 and 5.4. Furthermore, average Type I errors for the 1st-order Markov and the random process are consistently below 10% for all lengths and state spaces tested.

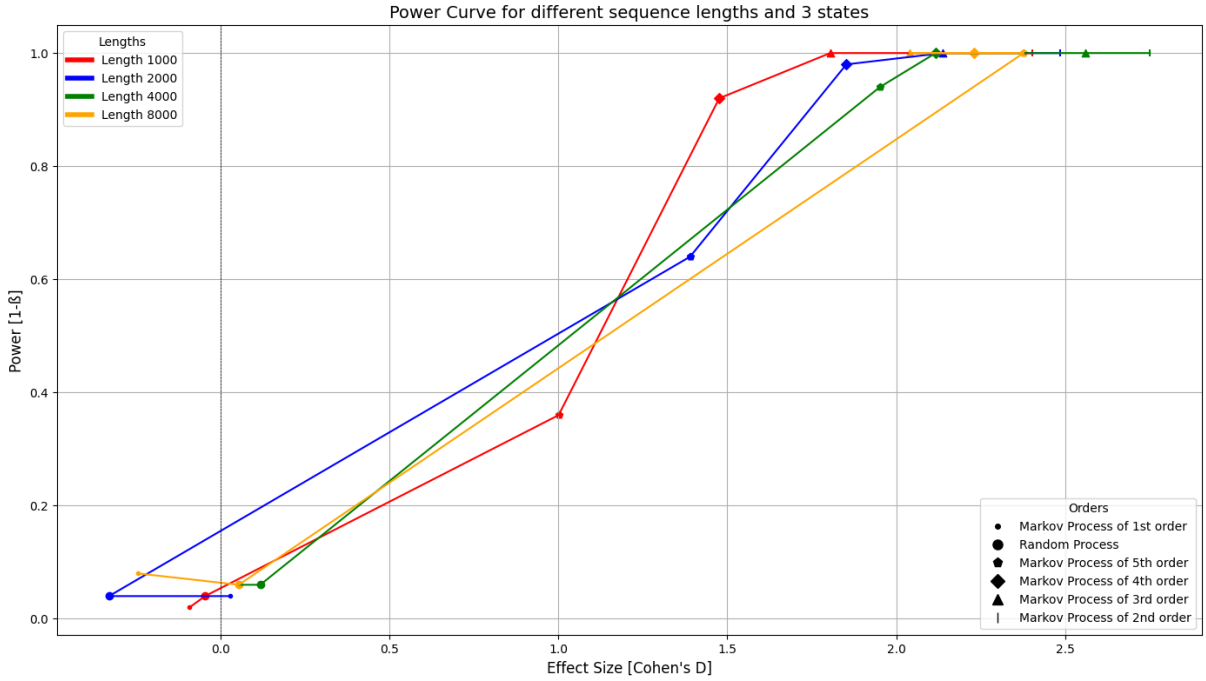


Figure 5.3: Power curve for the *markov_property_test()* method for sequences of lengths 1000, 2000, 4000 and 8000 with 3 states, using different orders of Markov processes as effect sizes. Effect sizes are listed on the X-axis as *Cohen's D* with pooled variance, while the power $(1 - \beta)$ or the type I error rate (for 1^{st} order Markov and random process) is listed by the Y-axis. Each point shows the averaged power over 50 independent simulated sequences for a given process type.

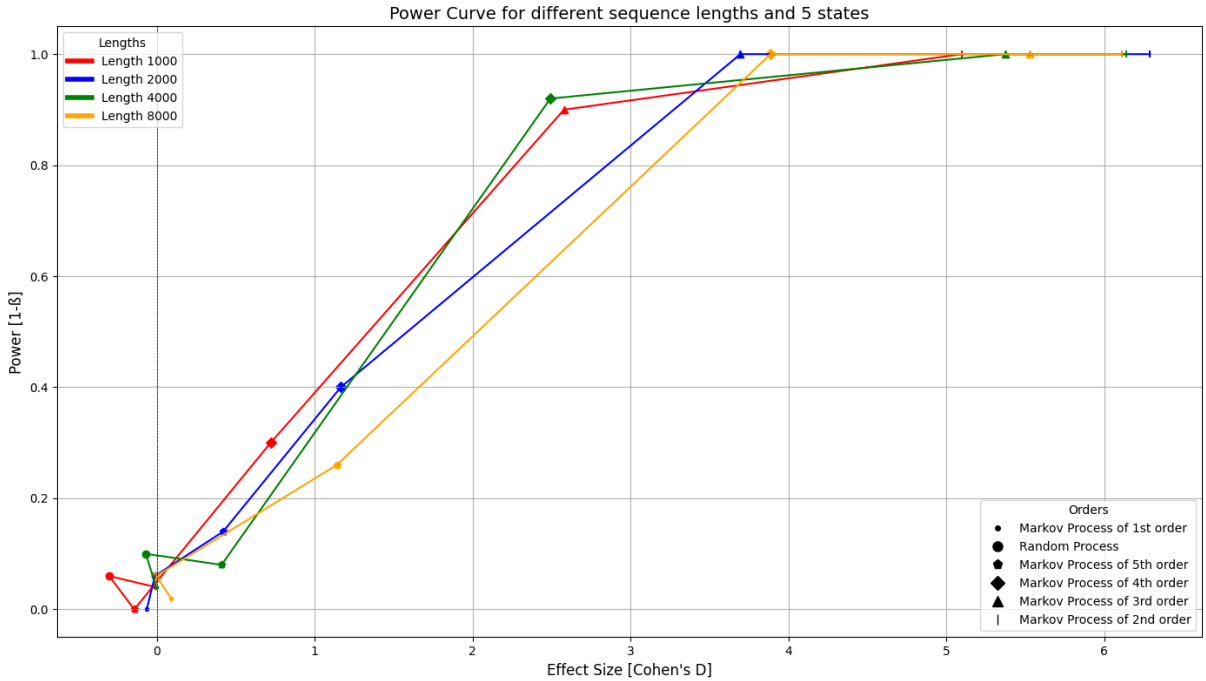


Figure 5.4: Power curve for the *markov_property_test()* method for sequences of lengths 1000, 2000, 4000 and 8000 with 5 states, using different orders of Markov processes as effect sizes. Effect sizes are listed on the X-axis as *Cohen's D* with pooled variance, while the power $(1 - \beta)$ or the type I error rate (for 1^{st} order Markov and random process) is indicated by the Y-axis. Each point shows the averaged power over 50 independent simulated sequences for a given process type.

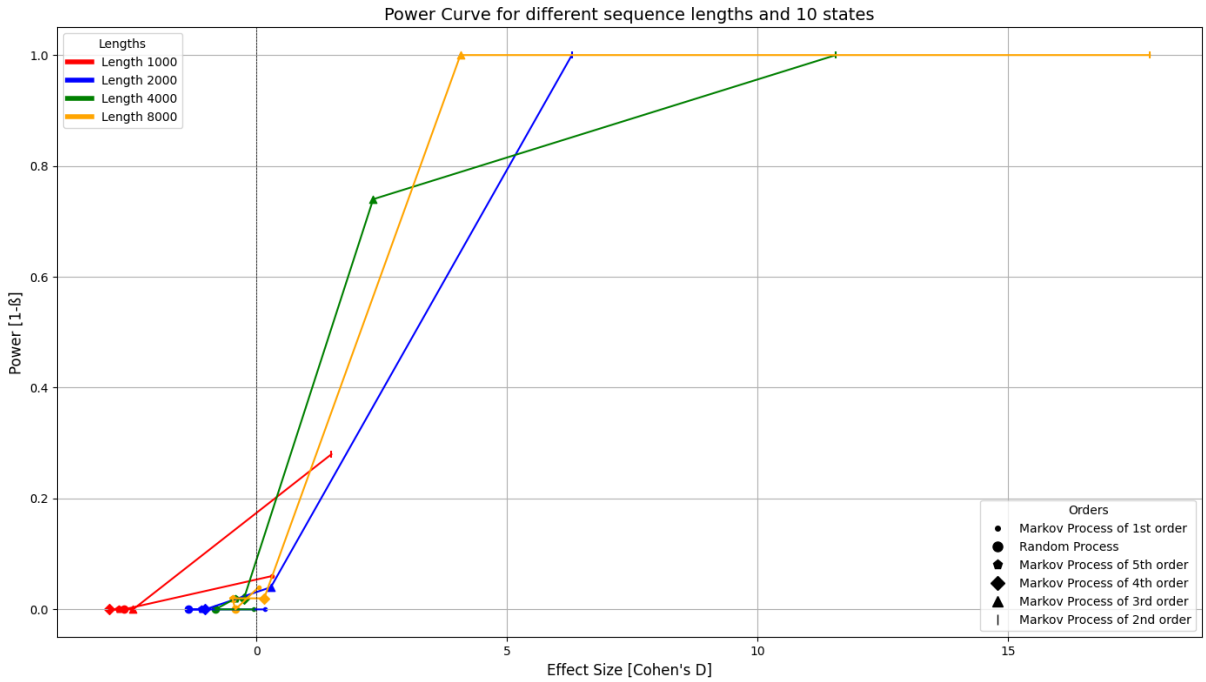


Figure 5.5: Power curve for the *markov_property_test()* method for sequences of lengths 1000, 2000, 4000 and 8000 with 10 states, using different orders of Markov processes as effect sizes. Effect sizes are listed on the X-axis as *Cohen's D* with pooled variance, while the power $(1 - \beta)$ or the type I error rate (for 1st order Markov and random process) is indicated by the Y-axis. Each point shows the averaged power over 50 independent simulated sequences for a given process type.

5.3 Time-Homogeneity – *stationary_property_test()*

The initial assessment of the *stationary_property_test()* included the same sequences as for the *markov_property_test()*. Results for different sequence lengths – as specified in Subsection 4.3.3 – are shown in Figures 5.6 and 5.7. The test rejected all *discrete* time-inhomogeneous sequences, except the sequences with only 1 state. It also rejected most of the *pseudo-continuous* time-inhomogeneous sequences and Random Walk sequences. Here the spread of p-values for the KS-Test seems to be higher at the ends of the X-axis with fewer significant results at state spaces bigger than 10. The p-values for Ornstein-Uhlenbeck sequences show less variance than 1st-, 2nd- and 3rd- order Markov and random sequences. Furthermore, the mean p-values decrease with higher state spaces, with a average p-value of barely non-significance at 15 states. Sequences originating form a random process as well as 1st-, 2nd- and 3rd- order Markov processes are seldom rejected (see Figure 5.6). The results fro 5000 long sequences in Figure 5.7 are very similar. The only notable difference is the decrease in non-significant outliers for *pseudo-continuous* time-inhomogeneous sequence. The exact number of rejections of the 100 tests for 3, 5, 10 and 15 states for 3000 unit sequences can be seen in Table 5.7 and for 5000 unit sequences Supplementary Table 7.4.

This method is testing the time-inhomogeneous assumption of sequences, by assuming that in time-inhomogeneous processes, chunks have constant transition probabilities. To see how the amount of chunks influence results further tests with changing chunks were done and visualized in Figure 5.8. The figure displays KS-Test results of the *stationary_property_test()* for

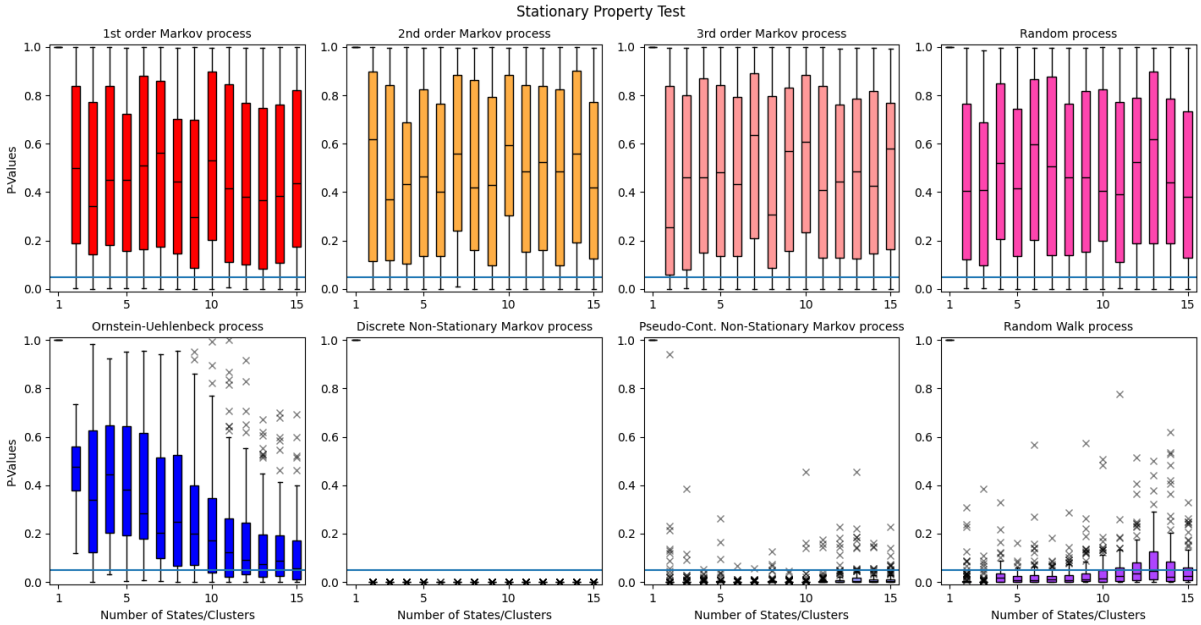


Figure 5.6: This plot shows results of the *stationary_property_test()* method with 5 chunks for simulated sequences of length 3000 with p-values on the Y-axis and number of states on the X-axis. The plot is created from 100 of each 1st order Markov (*A*), 2nd order Markov (*B*), random (*C*) and 1st order non-stationary Markov sequences (*D*). Below 0.05 p-values would indicate that the sequence is following a stationary process. Each boxplot visualizes results from 100 sequences and the parameter *simulations* was fixed at 50 for all runs.

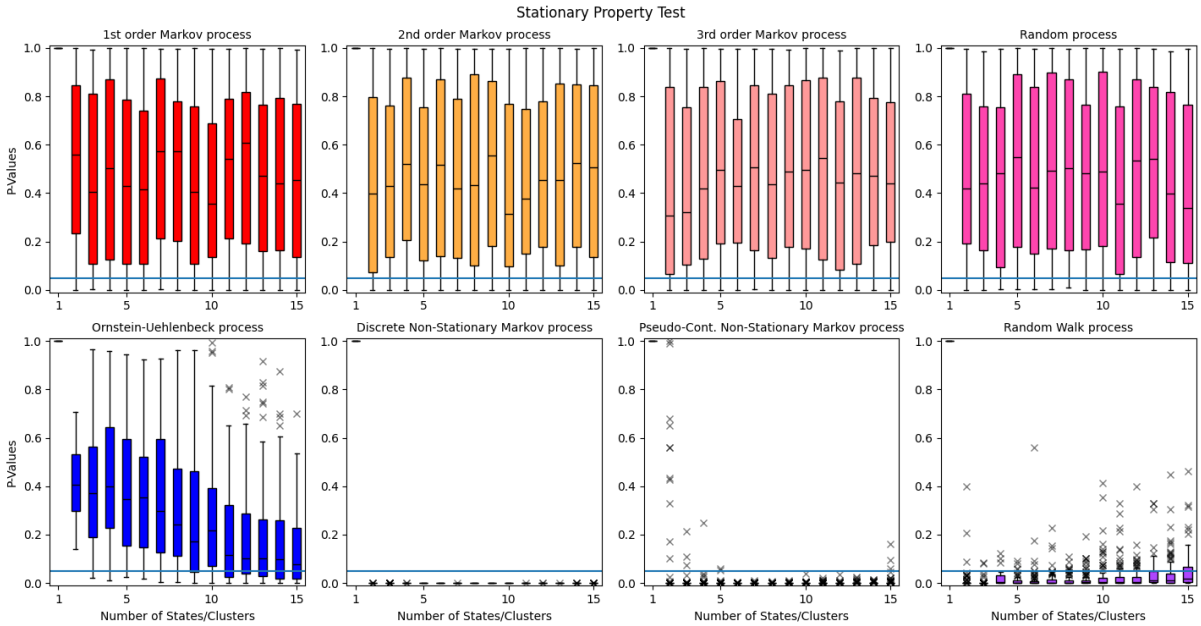


Figure 5.7: This plot shows results of the *stationary_property_test()* method with 5 chunks for simulated sequences of length 5000 with p-values on the Y-axis and number of states on the X-axis. The plot is created from 100 of each 1st order Markov (*A*), 2nd order Markov (*B*), random (*C*) and 1st order non-stationary Markov sequences (*D*). Below 0.05 p-values would indicate that the sequence is following a stationary process. Each boxplot visualizes results from 100 sequences and the parameter *simulations* was fixed at 50 for all runs.

Table 5.7: The table lists the exact rejection counts for the KS-Test in the *stationary_property_test()* at a significance level of $\alpha = 0.05$, based on 100 simulated sequences of 3000 units each. The amount of chunks was set to 5 and the *simulations* to 50. Sequences were generated from processes including 1st-, 2nd-, and 3rd- order Markov processes, a random process, an Ornstein-Uhlenbeck process, a *discrete* and *pseudo-continuous* non-stationary process and a discretized Random Walk. Results are visualized in Figure 5.6.

3 States	HO kept	H_0 rejected
1st order M.	87	13
2nd order M.	84	16
3rd order M.	79	21
Random	84	16
OU	95	5
Discrete non-stat.	0	100
Pseudo-Cont. non-stat.	3	97
Random Walk	6	94

5 States	HO kept	H_0 rejected
1st order M.	87	13
2nd order M.	88	12
3rd order M.	84	16
Random	84	16
OU	93	7
Discrete non-stat.	0	100
Pseudo-Cont. non-stat.	3	97
Random Walk	9	91

10 States	HO kept	H_0 rejected
1st order M.	89	11
2nd order M.	90	10
3rd order M.	88	12
Random	81	9
OU	73	27
Discrete non-stat.	0	100
Pseudo-Cont. non-stat.	3	97
Random Walk	25	75

15 States	HO kept	H_0 rejected
1st order M.	87	13
2nd order M.	88	12
3rd order M.	83	17
Random	79	21
OU	52	48
Discrete non-stat.	0	100
Pseudo-Cont. non-stat.	8	92
Random Walk	32	68

different state spaces with 2 to 15 chunks. In the range form 2 to 10 states the blue lines (*discrete*, *pseudo-continuous* time-inhomogeneous process and discretized Random Walk) are displaying stable significant results for all chunks tested, with lower chunks in general showing less clear significance. In both time-inhomogeneous sequences 10 changes were done with an ϵ of 0.05. Starting at 12 states the 75% confidence intervals of the Random Walk starts to spread, mainly at lower amounts of chunk. Similarly, significance disappears with bigger state spaces (see Supplementary Figure 7.5) for *discrete* and *pseudo-continuous* time-inhomogeneous processes.

When looking at the red lines (1st-, 2nd- order Markov process and Ornstein-Uhlenbeck process) one can also see similarities within this group, as the spread of p-values is much bigger compared to the other group while most results seem to show no significance. The Ornstein-Uhlenbeck is a bit of an outlier in this group because it start to display lower mean p-values for state spaces bigger than 6. It also demonstrates more non-significant results at higher chunk numbers for these smaller state spaces. This continues until the p-values sink into significance and the results are not differentiable from the blue group from 16 to 18 states onward. The 1st-order and 2nd order Markov processes p-values are more constant, with a big 75% CI spread and a mean p-value around 0.5 for all the state spaces tested.

In general higher amounts of chunks result in lower p-values for the blue group and the confidence intervals of the blue group are narrower, while the red group experiences wider confidence intervals.

5.3.1 Power analysis

For this test, a multiple power curves were created as described in Subsection 4.3.4 for 3, 5 and 10 states for the results of the t-test and the KS-test. The effect sizes were simulated using two different approaches, the *discrete* transition probability changes (see Figures 5.10, 5.12 and 5.14) and the *pseudo-continuous* transition probability changes (see Figures 5.9, 5.11 and 5.13). The results show the power of the test on the Y-axis, while for test sequences with no effect size (i.e., originating from a 1st-order Markov process), the value instead reflects the Type I error rate. On the X-axis the effect size is shown using either the mean t-statistic or the mean KS-statistic returned by the respective tests. The Y-value of a point is calculated as the mean power over the 30 sequences, while the Y-value is the mean effect size, with individual results shown as translucent markers to demonstrate the distribution of effect sizes in a group.

In Figures 5.9, 5.11 and 5.13 and Figures 5.10, 5.12 and 5.14 the power curves for effect sizes as *discrete* and *pseudo-continuous* changes in transition probabilities are depicted. Across all curves the power does increase with a higher effect size apart from few outliers which only occurred in short sequences with big state spaces (Figures 5.13 and 5.14).

***Discrete* time-inhomogeneous sequences**

The power to detect the *discrete* time-inhomogeneous sequences with 5 changes was above 80% for both tests and all lengths. More changes for the *discrete* time-inhomogeneous sequences showed lower powers with decreasing effect size, particularly in shorter sequences (lengths 1000 or 2000) and greater state spaces (5 or 10 states) this decline in power occurs earlier. The sequences with 100 and 200 changes display powers at or below 80% in 1000 long sequences with 5 states, while the 1000 or 2000 unit long sequences with 10 states do not surpass the 50% power mark. In the 1000 unit long sequence with 10 states power stays below 40% for all sequence except the one with the strongest effect size in both t-test and KS-test. The Type I error for the stationary 1st-order Markov and random process do increase with longer sequences in both tests with an $\alpha = 0.05$ for KS- and t- Test from 5-20% in shorter sequences to as much as 35% in some longer sequences without any effect size.

In all KS-Test curves the spread of results for effect size within process types in shorter sequences is greater than with longer sequences. This is underlined by the slopes of linear regressions fit on the lengths and the standard deviations of effect sizes for the different processes. In *discrete* sequences with 3 states these slopes for the 1st-order Markov process, random process, process with 200-, 100-, 50- and 5- changes equate to **-0.0025** ($R^2 = 0.4$), **-0.0045** ($R^2 = 0.99$), **-0.0082** ($R^2 = 0.66$), **-0.0074** ($R^2 = 0.88$), **-0.0124** ($R^2 = 0.88$) and **-0.0071** ($R^2 = 0.71$) resulting in a mean decrease in standard deviation of **0.005** for every 1000 units of length. For 5 states the mean slope is **-0.01** (*-0.0086* ($R^2 = 0.82$), *-0.0099* ($R^2 = 0.96$), *-0.0071* ($R^2 = 0.87$), *-0.0107* ($R^2 = 0.98$), *-0.0164* ($R^2 = 0.86$), *-0.0051* ($R^2 = 0.69$)), while for 10 states the mean slope is **-0.012** (*-0.0105* ($R^2 = 0.86$), *-0.0092* ($R^2 = 0.76$), *-0.0077* ($R^2 = 0.76$), *-0.0149* ($R^2 = 0.76$), *-0.0215* ($R^2 = 0.98$), *-0.0094* ($R^2 = 0.49$)). The standard deviations of

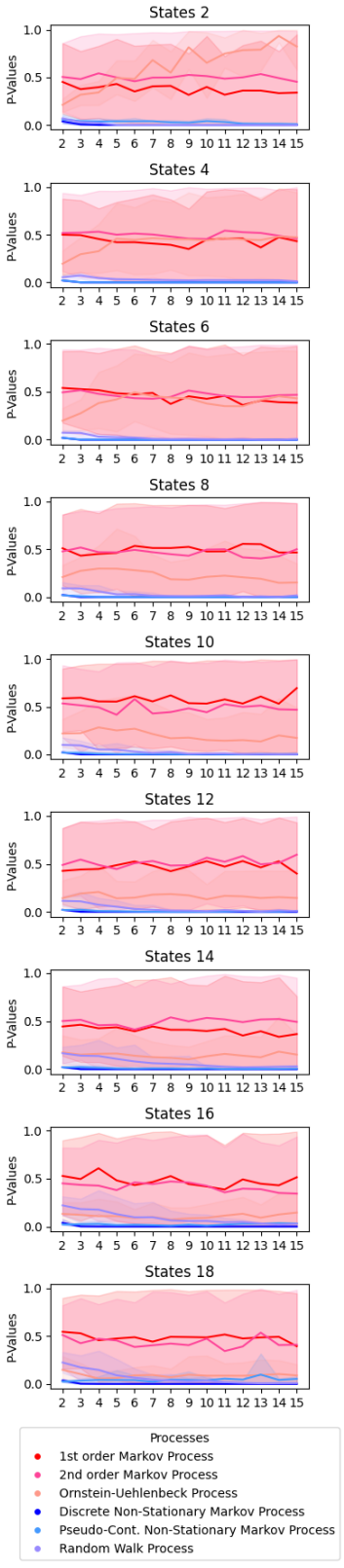


Figure 5.8: Results of the *stationary_property_test()* for 50 simulated sequences each of a 1st-, 2nd- order Markov process, a Ornstein-Uhlenbeck process, a *discrete* and *pseudo-continuous* time-inhomogeneous process as well as a discretized Random Walk. All sequences were of length 3000. Different subplots were made for 2, 4, 6, 8, 10, 12, 14, 16, 18 & 20 states. The amount of chunks used in the test is displayed on the X-axis, while p-values are displayed on the Y-axis. A 75% confidence interval is given for all results.

t-statistics can not be analyzed similarly since, the effect sizes are open ended in the t-test.

***Pseudo-continuous* time-inhomogeneous sequences**

In *pseudo-continuous* time-inhomogeneous sequences the power of sequences with 2000 units or longer was consistently over 80%. The short sequences with only 1000 showed lower powers for state spaces of 5 or 10. In curves with 4000 or 8000 units all sequences with a effect scored a power of 75% or higher, while Type I errors for sequences without effect were consistently below 40% for $\alpha = 0.05$ for the KS- and t- Test with a maximum error rate for sequences with 3 states.

By aggregating the results from sequences with no true effect in both *discrete* and *pseudo-continuous* datasets, an overall Type I error rate was calculated for each state space configuration. For datasets with 3 states, the average Type I error rate was 26.63% and 27.75% for the KS-test and t-test, respectively. For datasets with 5 states, the average Type I error rate decreased to 21% and 19.88%, and for datasets with 10 states, it further decreased to 14.5% and 16%. These Type I error rates were achieved by interpreting p-values < 0.05 by the KS- and t- Test as significant.

In a secondary analysis (see Supplementary Figures 7.6 to 7.11), where results from the KS-test and t-test were deemed significant only if p-value < 0.005, the Type I error rates were reduced with similar powers for longer sequences. Specifically, for datasets with 3 states, the rates dropped to 12.88% and 17.88%; for datasets with 5 states, to 9.75% and 11.25%; and for datasets with 10 states, to 4.5% and 4.75%, respectively.

The spread of KS-test results is greater than with longer sequences also in *pseudo-continuous* non-stationary sequences. For 3 states the slopes of linear regressions for the 1st-order Markov process, random process, process with 200-, 100-, 50- and 5- changes equate to -0.0069 ($R^2 = 0.94$), -0.0034 ($R^2 = 0.42$), 0.0014 ($R^2 = 0.95$), -0.0073 ($R^2 = 0.88$), -0.0074 ($R^2 = 0.94$) and -0.0038 ($R^2 = 0.24$). Which means that on average with every 1000 units increase in length the standard deviation of the KS-statistics decreases by 0.007. For 5 states the mean slope is -0.008 (*-0.0082* ($R^2 = 0.70$), *-0.0046* ($R^2 = 0.72$), *-0.0068* ($R^2 = 0.96$), *-0.0115* ($R^2 = 0.89$), *-0.0083* ($R^2 = 0.72$), *-0.0099* ($R^2 = 0.92$)), while for 10 states it is -0.013 (*-0.01* ($R^2 = 0.76$), *-0.0144* ($R^2 = 0.58$), *-0.0138* ($R^2 = 0.92$), *-0.0158* ($R^2 = 0.88$), *-0.0138* ($R^2 = 0.97$), *-0.0113* ($R^2 = 0.91$)).

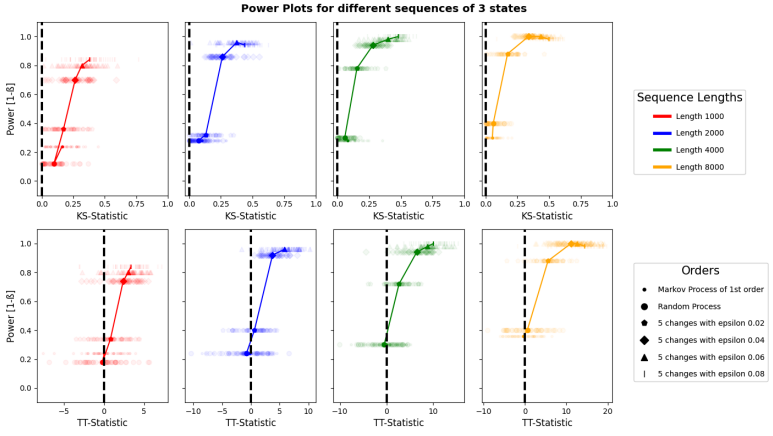


Figure 5.9: Power curve for the *stationary_property_test()* method with 5 chunks for sequences of different lengths with **3 states** and $\alpha = 0.05$ for the KS- or t-Tests, using 5 directed perturbations in the transition probabilities with different amplitudes (epsilon) as effect sizes. Effect sizes are listed on the X-axis, while the power $(1 - \beta)$ or Type I error probability is indicated by the Y-axis.

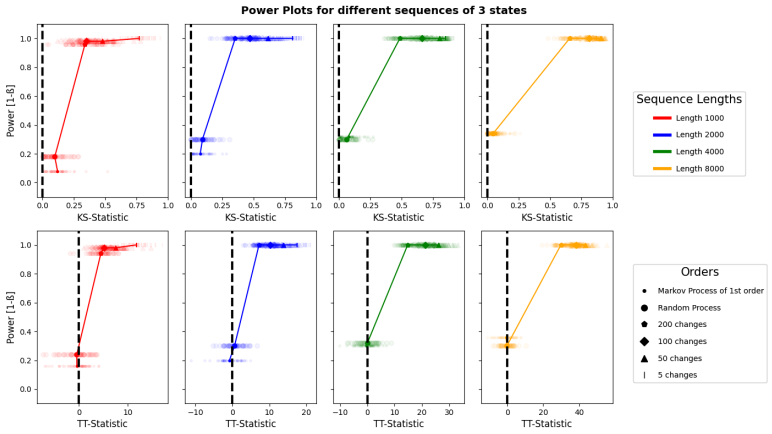


Figure 5.10: Power curve for the *stationary_property_test()* method with 5 chunks for sequences of different lengths with **3 states** and $\alpha = 0.05$ for the KS- or t- Tests, using different numbers of changes in the transition probabilities as effect sizes. Effect sizes are listed on the X-axis, while the power $(1 - \beta)$ or Type I error probability is indicated by the Y-axis.

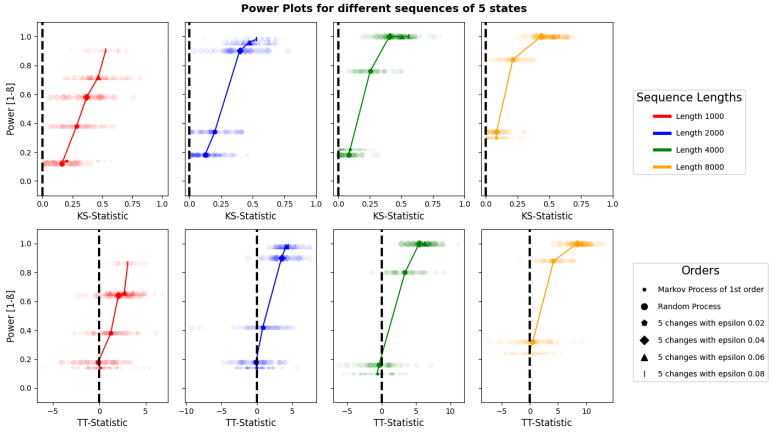


Figure 5.11: Power curve for the *stationary_property_test()* method with 5 chunks for sequences of different lengths with **5 states** and $\alpha = 0.05$ for the KS- or t-Tests, using 5 directed perturbations in the transition probabilities with different amplitudes (epsilon) as effect sizes. Effect sizes are listed on the X-axis, while the power $(1 - \beta)$ or Type I error probability is indicated by the Y-axis.

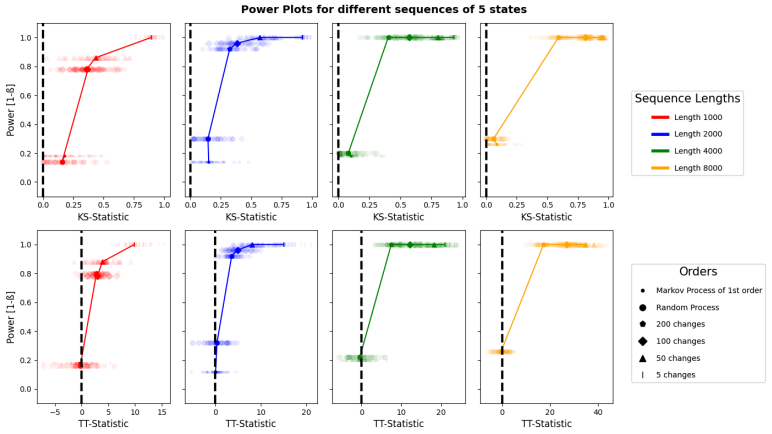


Figure 5.12: Power curve for the *stationary_property_test()* method with 5 chunks for sequences of different lengths with **5 states** and $\alpha = 0.05$ for the KS- or t- Tests, using different numbers of changes in the transition probabilities as effect sizes. Effect sizes are listed on the X-axis, while the power $(1 - \beta)$ or Type I error probability is indicated by the Y-axis.

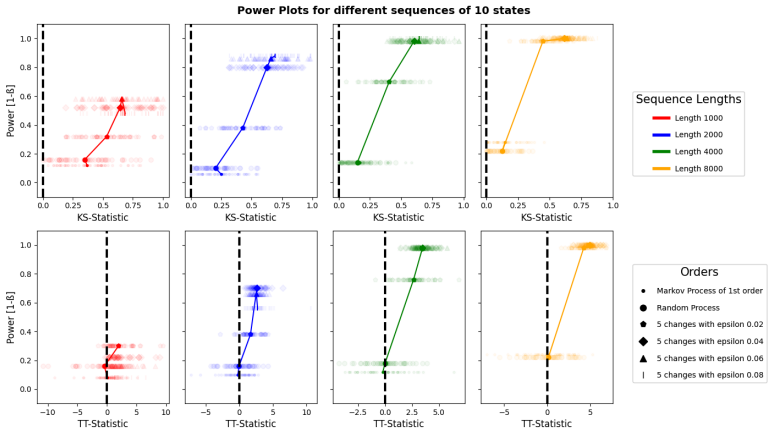


Figure 5.13: Power curve for the *stationary_property_test()* method with 5 chunks for sequences of different lengths with **10 states** and $\alpha = 0.05$ for the KS- or t-Tests, using 5 directed perturbations in the transition probabilities with different amplitudes (epsilon) as effect sizes. Effect sizes are listed on the X-axis, while the power $(1 - \beta)$ or Type I error probability is indicated by the Y-axis.

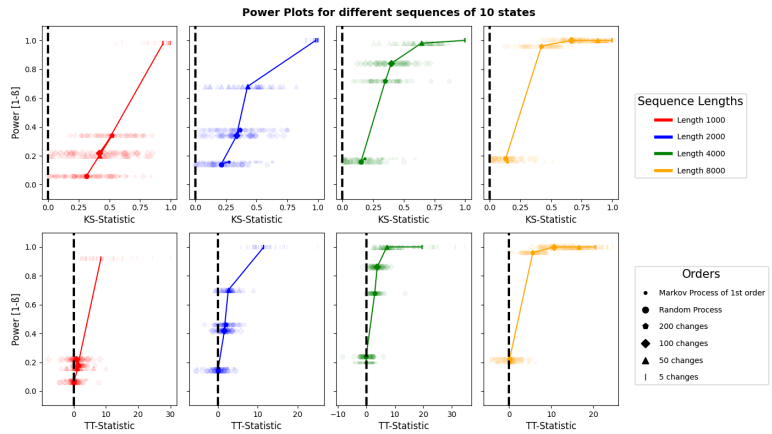


Figure 5.14: Power curve for the *stationary_property_test()* method with 5 chunks for sequences of different lengths with **10 states** and $\alpha = 0.05$ for the KS- or t- Tests, using different numbers of changes in the transition probabilities as effect sizes. Effect sizes are listed on the X-axis, while the power $(1 - \beta)$ or Type I error probability is indicated by the Y-axis.

5.4 Application on Data

In this Section a quick presentation of an analysis workflow using the toolbox is shown. In Subsection 5.4.1 the use of the toolbox on the data from the original paper [96, 73] is demonstrated, while Subsection 5.4.2 shows the application on a new dataset. The plots are depicted in the order they would be generated in a workflow of neural activity data with behavioral labels.

5.4.1 *Caenorhabditis elegans*

The start of an analysis workflow can be challenging. The get an overview over the dataset a researcher can generate the behavioral and neuronal heat maps shown in Figure 5.16 after a *Database* instance has been created using the data. The upper plot shows every neuron listed on the Y-axis over time (X-axis), while the lower plot depicts behavioral labels over the same time period. To help to understand the process of *NC-MCM* a step-by-step plot of its application can be generated next. Figure 5.17 shows this plot created with the *step_plot()* method applied on worm number 4 from the dataset, showing the basic *NC-MCM* workflow described in the second part of Subsection 4.2.1.

```
data = Database(neuron_traces, behavior, neuron_names=neurons)
data.plotting_neuronal_behavioral()
data.step_plot()
```

Now a logistic regression model is fit on the dataset as the *step_plot()* method does by default, after the neurons used to classify the behaviors are excluded from the dataset. The model is then trained to predict the behavioral labels from the neural activity of the remaining neurons. The *fit_model()* method is executed with the *cv_folds* parameter set on 10 to assess the models' ability to generalize through cross validation. The resulting printout shows a mean accuracy of 0.87 for cross-validation and an in sample accuracy of 0.94 when training on the complete dataset. By setting the parameter *ensemble* to *True* the logistic regression is used as the base model for a *CustomEnsembleModel*, allowing for higher dimensional *behavioral probability space* (see Subsection 4.2.4).

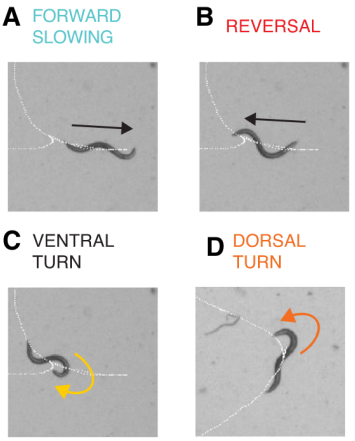


Figure 5.15: Adapted from *Global Brain Dynamics Embed the Motor Command Sequence of Caenorhabditis elegans* by Saul Kato et al. 2015: Images of the different behaviors of *C. elegans*. The white dotted lines show the crawling trajectory, while the arrows indicate crawling direction [73].

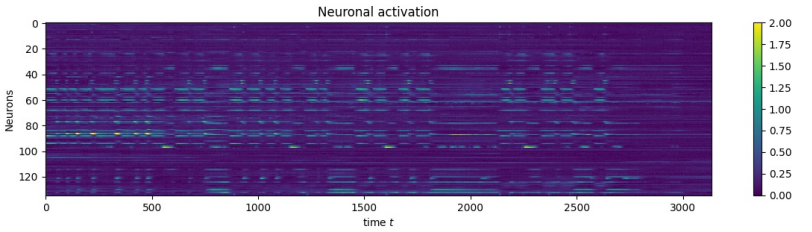


Figure 5.16: This plot shows neuronal activation pattern and the sequence of behaviors over time. It is generated by using the *plotting_neuronal_behavioral* method of the *Database* class.

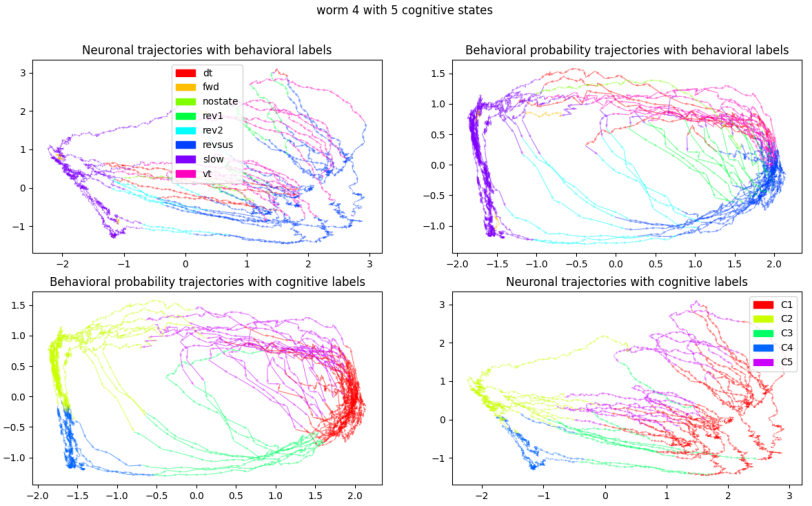


Figure 5.17: A step-by-step depiction of the implementation of the *NC-MCM* framework on data from worm 4 with 5 different cognitive states. This plot is created with the *step_plot()* method which uses a *CustomEnsembleModel* to project into behavioral probability space.

```
data.exclude_neurons(neurons_used_to_label_behaviors)
data.fit_model(LogisticRegression, ensemble=True, cv_folds=10)
data.cluster_BPT(nrep=30, max_clusters=25, stationary=True,
                 sim_m=150, sim_s=50, chunks=7,
                 plot_markov=True)
```

If a model has been fit on the data, *cluster_BPT()* can be used to cluster the *behavioral probability trajectories* of the worm in the 28 dimensions. After the clustering using a selected algorithm, in this case k-means, the sequences of clusters are tested for their Markovian behavior. In Figure 5.18 these results are plotted, with time-homogeneity results also displayed for self determined amounts of chunks.

On one hand, the blue boxplots show the p-value results for all clustering repetitions of the *markov_property_test()*, on the other hand, the red boxplots visualize results for the *stationary_property_test()*. According to the results, no signs against the Markov property were found for 50% or more of state spaces with size 2 up to 9 cognitive clusters. There is a general trend of not contradicting the Markov property for all sequences 2 to 7 cognitive clusters (>75%). The results of the *stationary_property_test()* with 7 *chunks*, suggests that only a few state spaces (2, 4, 7, 8, 9, 10 and 11) have 50% or more sequences that do not violate the time-homogeneous assumption. While, only clusterings of 2, 4 and 10

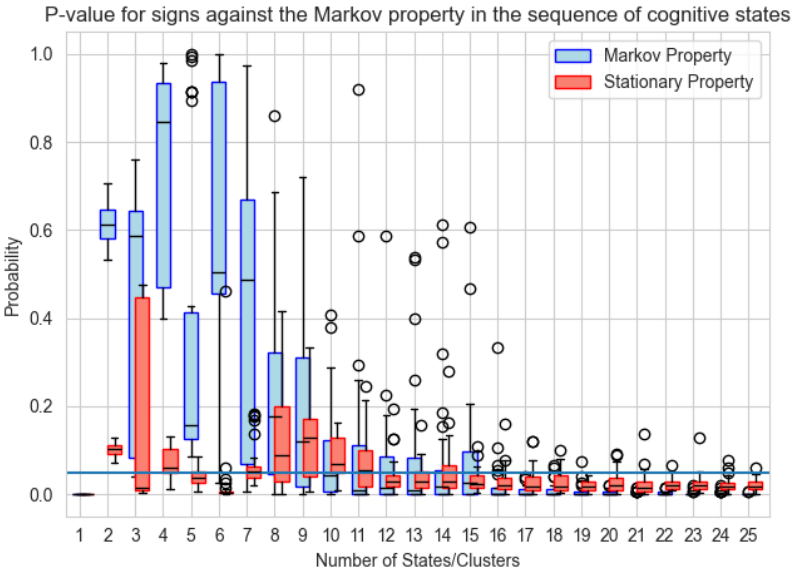


Figure 5.18: Results of the test for the Stationary (red) and Markov (blue) property on the sequence through cognitive state space for worm 4. The amount of cognitive states is listed on the x-axis, while p-values for both tests are indicated by y-values. Each boxplot indicates results for 30 repetitions of clustering and subsequent testing. The *cluster_BPT()* method creates this plot if *plot_markov* is set to *True*.

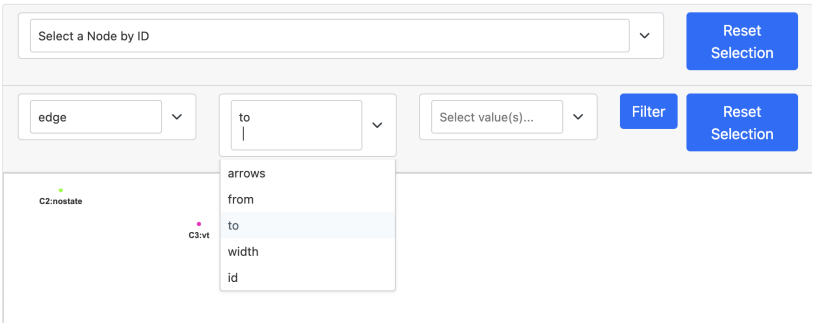


Figure 5.19: The plot shows the settings for the interactive HTML file. The menu allows the user to filter nodes or edges based on their attributes, such as behavior or cognitive state for nodes and source, target or width for edges. Selection of single nodes is also possible in the node-selection at the top of the page.

cognitive states result in 75% or more of the sequences not violating time-homogeneous behavior. Results with self determined chunks are shown in Supplementary Figure 7.1 and show less violations against the H_0 for tests with more then 8 states.

After the initial processing the researcher can reason about the number of cognitive states, which best harmonize with the Markovian constraint of the *NC-MCM* framework. By choosing certain numbers of cognitive clusters, a cognitive graph is created with the *behavioral_state_diagram()*. It visualizes the behavioral and cognitive states and their transitions of the clustered cognitive state sequence with the least evidence against the Markov properties by default. Two plots are created for two and five cognitive clusters respectively using data from worm 4. By setting the *interactive* parameter to *True* an HTML file is generated using *pyvis*. The interactive plots give the user options to interact with the layout by moving nodes and change the visualized parts of the graph (see Figures 5.19 to 5.21) as well as display additional information. Figure 5.22 with two cognitive states demonstrates the hovering option for the ventral turn within the 2nd cognitive cluster. It shows all the transitions from the current state, including transitions to itself. The size of the nodes indicates the number of occurrences of a certain cognitive cluster. Figure 5.23 shows a more granular depiction of the same dataset with five cognitive states, while in Figure 5.19 the additional settings are shown. The top menu displayed in each of the interactive HTML files, allows users to filter nodes or edges based on their attributes, which can

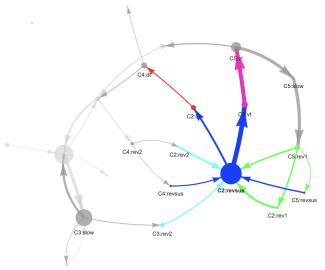


Figure 5.20: The plot shows the selection option of the interactive cognitive graph file. In the plot the node C2:revsus is selected, which will color only nodes that are directly connected to C2:revsus and their edges.

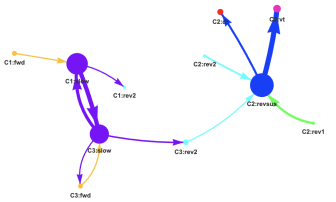


Figure 5.21: The plot shows the filter option of the interactive cognitive graph file. In this case only nodes for the cognitive states C1, C2 and C3 are displayed with their edges.

help make a crowded plot more interpretable. While the filter option will make non-selected nodes transparent (see Figure 5.21), the selection option will only gray them out(see Figure 5.20).

```
data.behavioral_state_diagram(cog_stat_num=3, interactive=True)
data.behavioral_state_diagram(cog_stat_num=5, interactive=True)
```

From the *Database* instance, a *Visualizer* instance using *BunDLe-Net* can be created with the *create_visualizer()* method with default parameters. Then the *make_comparison()* method is used to create Figure 5.24. The right subplot shows the true labels defined by behavioral labels, while the left subplot colors the points according to the in-sample prediction from the selected model – in this case a *BunDLe-Net* predictor. The central part, shows differences between the true values and in-sample predictions. The *BunDLe-Net* was able to reach a in-sample prediction accuracy of 97%.

```
bundle_net = data.create_visualizer()
bundle_net.make_comparison()

pca = data.create_visualizer(mapping=PCA(n_components=3))
pca.plot_mapping(show_legend=True, quivers=True)
```

Furthermore, the *create_visualizer()* method allows to specify the dimensionality reduction technique to be used using the *mapping* parameter. For demonstration purposes the projections of a selection of them for worm 4 are plotted in Figure 5.25. The toolbox allows all *sklearn* dimensionality reductions and others, not limited but including: Multidimensional Scaling (MDS), non-negative Matrix Factorization (NMF), Uniform Manifold Approximation and Projection (UMAP), t-Distributed Stochastic Neighbor Embedding (t-SNE), PCA, Isometric Mapping (Isomap) and Spectral Embedding.

Additionally, a plot of the empirical marginal probability distributions of the estimated transition matrix for cognitive state sequences generated for all 5 worms are shown in Figure 5.26. The plots show symmetric marginal distributions with many very low probabilities and some very high ones.

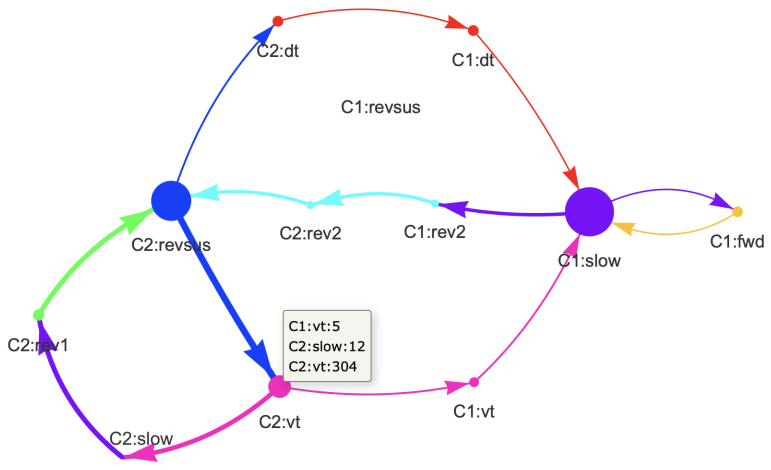


Figure 5.22: A plot showing the HTML plot with 2 cognitive clusters for worm 4. The transitions for the ventral turn within the 2nd cognitive state are shown by hovering above the node with the mouse. The *cognitive graph* is generated by setting the *interactive* parameter of *behavioral_state_diagram()* to *True*.

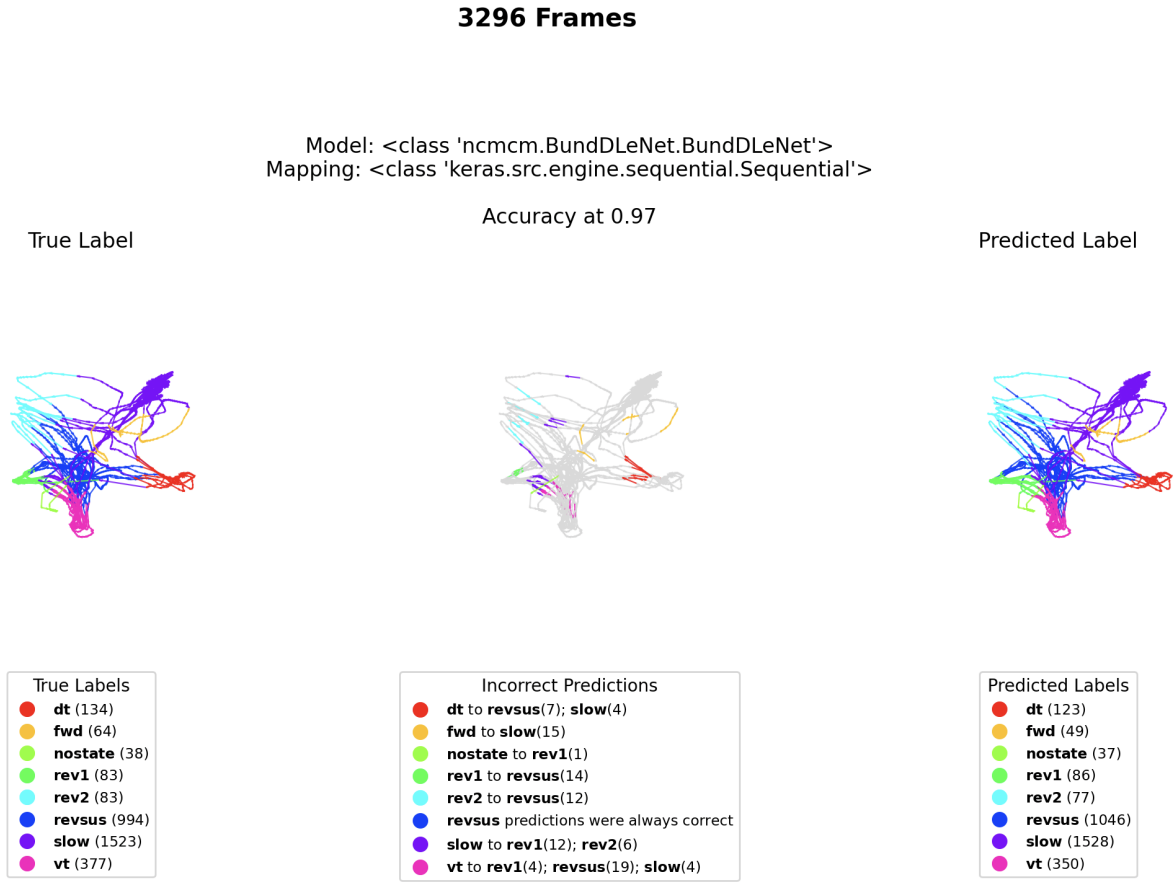


Figure 5.24: A comparison between true behaviors and *BunDLe-Net's* predicted behaviors, created with the *make_comparison()* method with *quivers* and *show_legend* each set to *True*. The points are mapped into three-dimensional space by *BunDLe-Net's* τ model embedding.

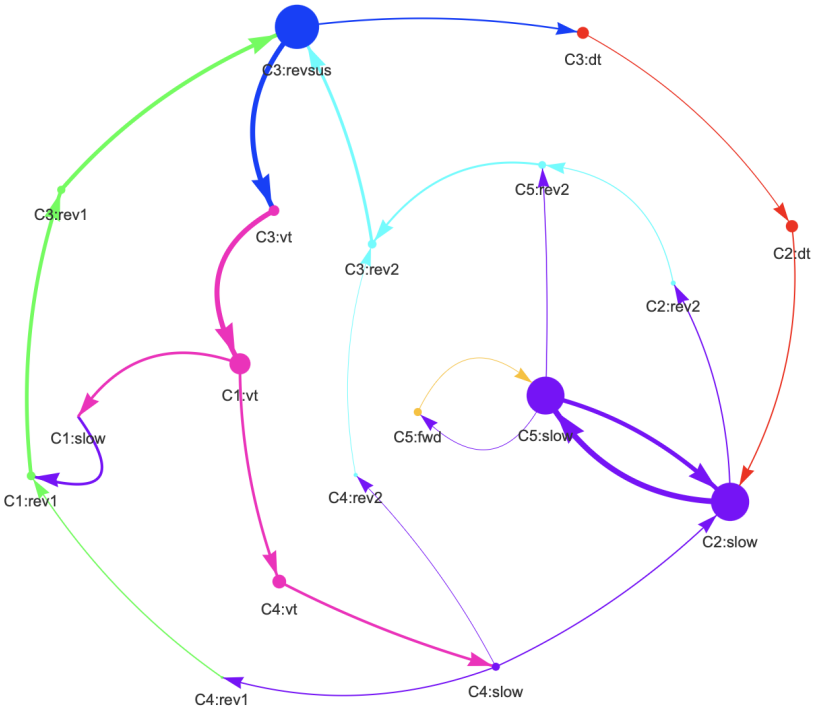


Figure 5.23: A plot showing the HTML plot with 5 cognitive clusters for worm 4. The *cognitive graph* is generated by setting the *interactive* parameter of *behavioral_state_diagram()* to *True*.

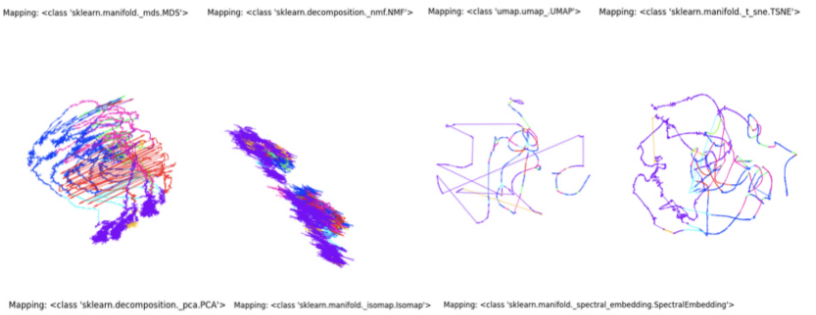


Figure 5.25: A comparison between some of the dimensionality reduction techniques usable in the *Visualizer* class. The two-dimensional projections of worm 4 are plotted. Top row left to right: MDS, NMF, UMAP and t-SNE. Bottom row left to right: PCA, Isomap and SpectralEmbedding.

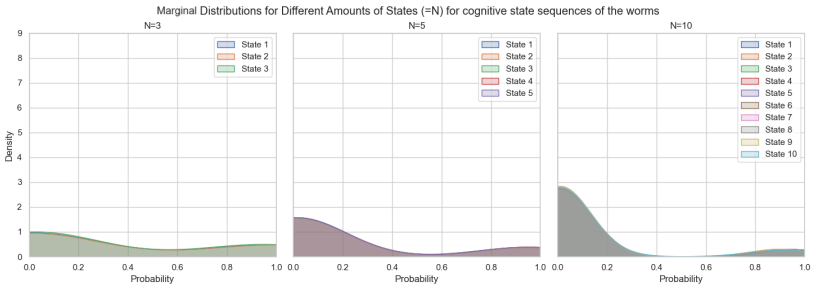


Figure 5.26: Marginal distributions of probabilities sampled from cognitive state transitions of all worms. The probabilities were taken from estimated transition matrices created from cognitive state sequences.

5.4.2 V1 area of a mouse

In this next section, the toolbox is applied to new data. For this the calcium imaging data from the V1 area of the mouse brain was used [18]. Since, there were no behavioral labels included in this dataset, they needed to be imputed as described in Subsection 4.4.2. Using the thresholds for lateral and frontal movements as mentioned, resulted in the following behavioral labels (with frame-count): *standing still* (10732), *moving forward* (8767), *going right* (702), *going left* (1818), *invisible* (13748), *moving forward-right* (656) and *moving forward-left* (1325).

Here all 4 the planes recorded in the V1 area combined with these behaviors were used as input for the *Database* instance. Similarly to before, behavioral and neuronal heat maps are created to first visualize the data (see Figure 5.27) and a logistic regression has been fit on the data, resulting in an in-sample accuracy of behavior prediction of 82.6%. After applying *cluster_BPT()*, results for the properties of the Markov process are shown (see Figure 5.28). It is clear by looking at the plot, that the behavioral trajectories (or cognitive states) do not follow a 1^{st} order Markov process, since all state spaces tested show significant evidence against the Markov Property. This was also the case when stratifying the dataset into smaller strata of 5000 length each. The results for time-homogeneity indicate a process with constant transition probabilities for all states tested. However, these results alone do not suggest Markovian behavior.

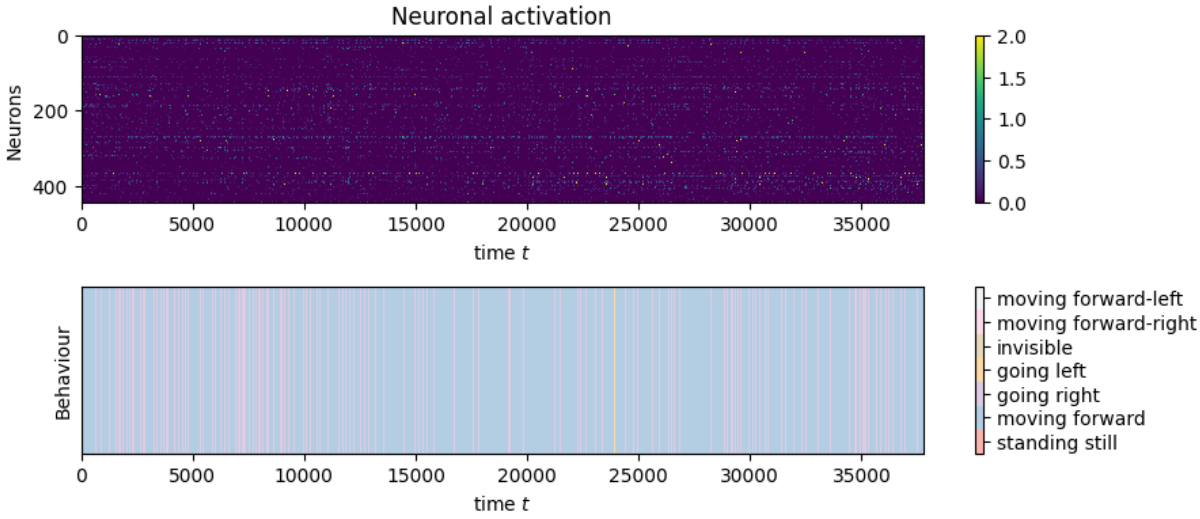


Figure 5.27: This plot shows neural activation pattern of recordings of area V1 and the sequence of behaviors over time for mouse 10.

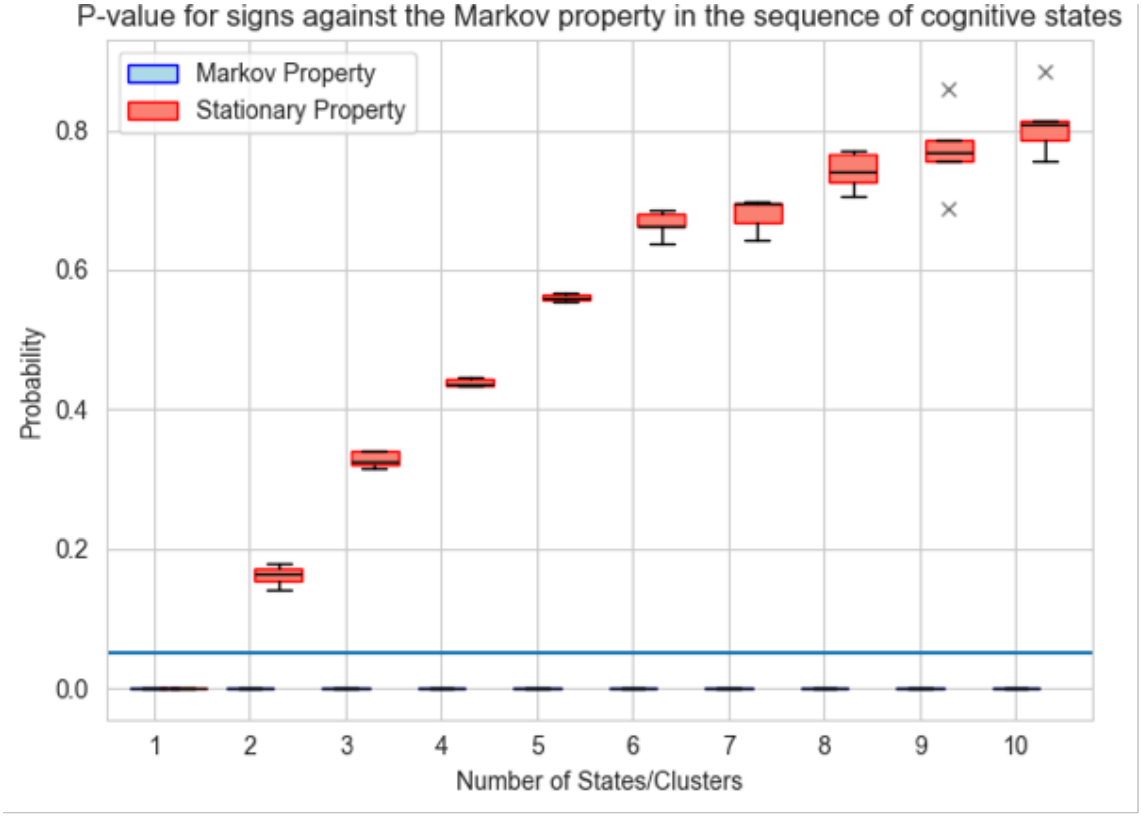


Figure 5.28: Results of the test for the Markov property on the sequence through cognitive state space created from neural activity of mouse 10. The amount of cognitive states is listed on the X-axis (up to 10), while p-values for both tests are indicated by Y-values. Each boxplot indicates results for 10 repetitions of clustering and subsequent testing.

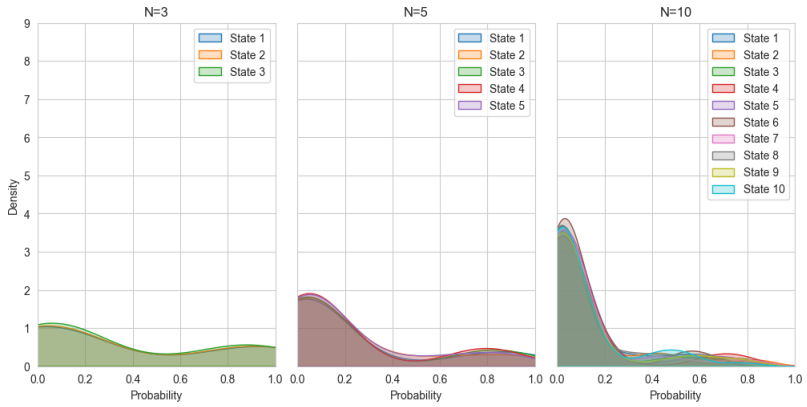


Figure 5.29: Marginal distributions of probabilities sampled from cognitive state transitions of mouse 10. The probabilities were taken from estimated transition matrices created from cognitive state sequences.

Figure 5.30: A comparison between true behaviors and *BunDLe-Net's* predicted behaviors for mouse 10. The central plot shows labels that are predicted incorrectly in the right subplot. The points are mapped into three-dimensional space by *BunDLe-Net's* τ model embedding.

The marginal distributions on the transition probabilities were also screened, demonstrating a bimodal distribution similar to the ones on the worms, but with a nearly unimodal distribution for bigger state spaces (see Figure 5.29). Furthermore, an HTML plot displaying the behavioral state diagram is created for 2 and 5 cognitive states (see Figures 5.31 and 5.32). The diagrams do have more transitions than the corresponding ones for worm 4 (see Figures 5.22 and 5.23) and show the splitting of cognitive clusters into unconnected sub-graphs, as is the case for *C1* in both plots (see Figures 5.31 and 5.32). Lastly, a comparison plot is created using *BunDLe-Net's* τ model (see Figure 5.30). It signals a good accuracy in behavior prediction of 93% and also separates behaviors into distinct sections of the embedding.

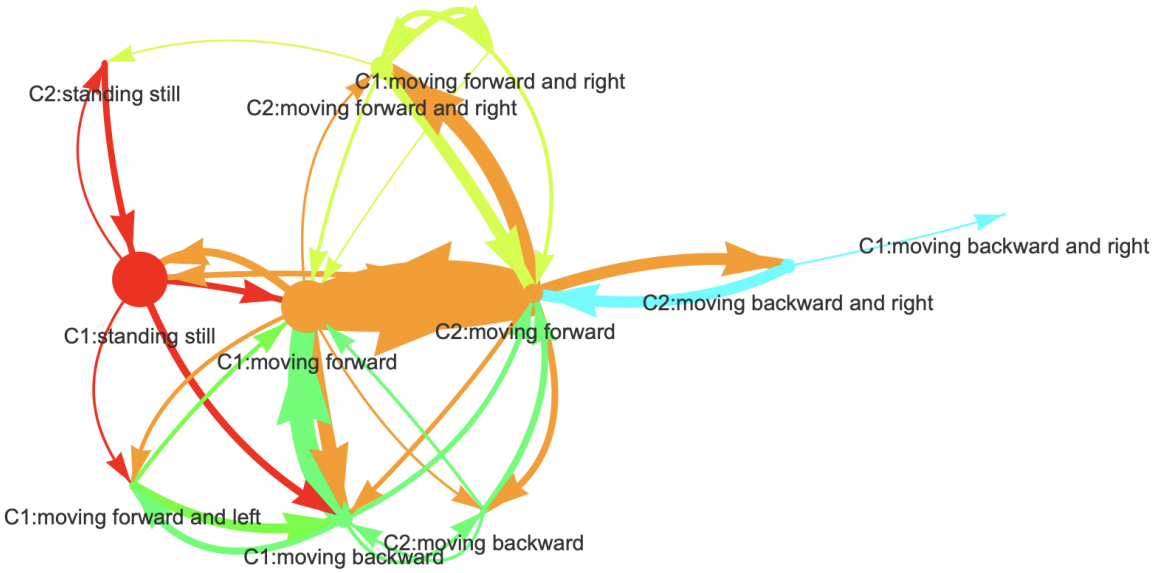


Figure 5.31: A behavioral state diagram showing state transitions of mouse 10 with 2 cognitive states. The *cognitive graph* is generated by setting the *interactive* parameter of *behavioral_state_diagram()* to *True*.

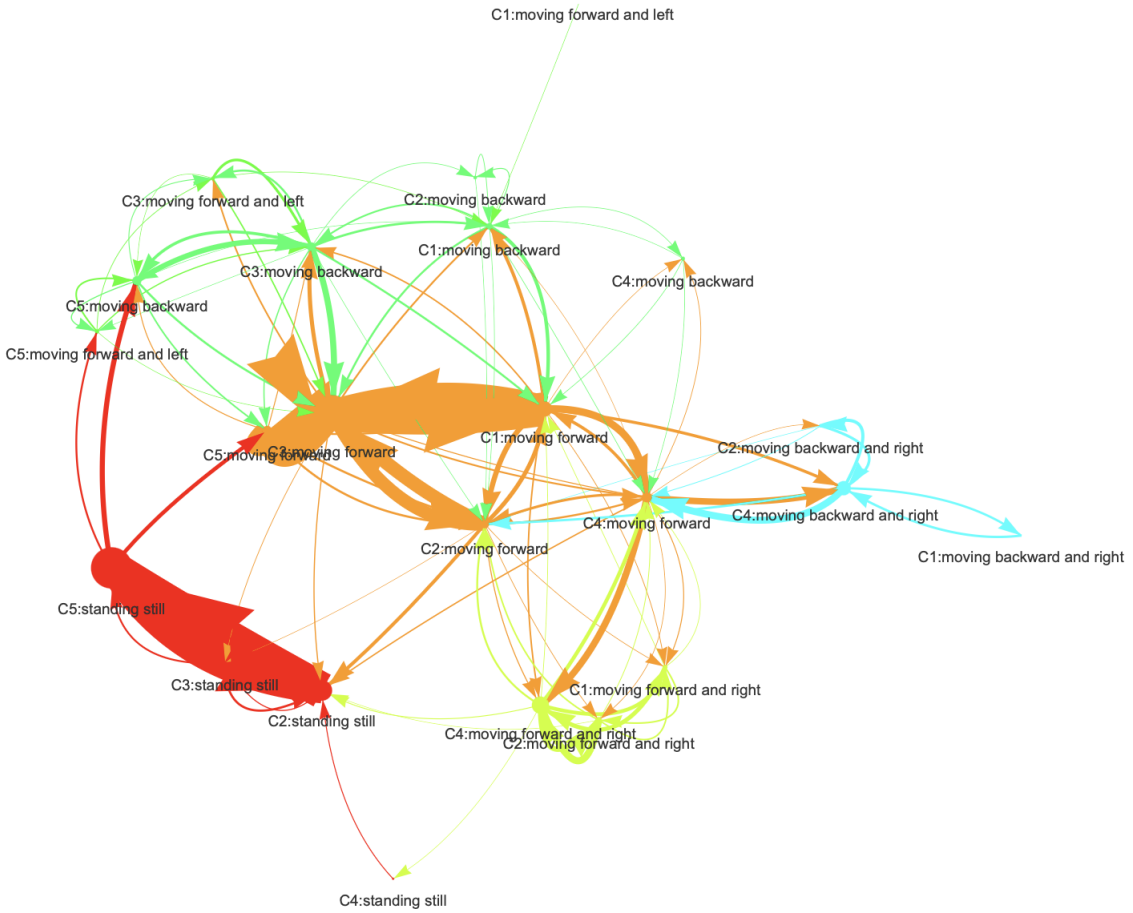


Figure 5.32: A behavioral state diagram showing state transitions of mouse 10 with 5 cognitive states. The *cognitive graph* is generated by setting the *interactive* parameter of *behavioral_state_diagram()* to *True*.

6 Discussion

It is essential for the *NC-MCM* framework that the cognitive state sequences proposed by it are causal, or more precisely, follow a Markov process. While the relationship between *neuronal level* and the *cognitive level* is constitutive and not causal, the levels in themselves are modeled as causal Markov chains. To achieve this, two statistical tests were implemented to identify the clustered cognitive state sequence from all repetitions that exhibits the fewest violations of the Markov properties and assumptions of the framework. This constraint on cognitive state sequences is integral to the framework and was therefore given the highest priority.

6.1 Statistical Tests

It is fundamentally impossible to definitively determine whether a sequence originates from a Markov process. Consequently, no statistical test can conclusively prove a Markovian origin. However, one can test for violations of the key properties of a Markov process (see Section 3.5). The statistical tests developed for the *NC-MCM* framework evaluate these properties to ensure compliance. While the *markov_property_test()* has already been proposed by previous work (see **??**), here a power analysis was done for it as well as a test of its viability. The *stationary_property_test()*, on the other hand, was developed from the ground up to address key aspects necessary for the effective use of the framework, which were not adequately covered by the previous test. This new test ensures a more comprehensive evaluation of the underlying assumptions, enabling a more accurate assessment of the data within the context of the framework.

To recapitulate, the constant state space property is inherently set by the framework when the number of cognitive states is defined. Second, the Markov property, specifically 1^{st}-order Markov property, is addressed by the *markov_property_test()* (see Subsection 4.3.1). Lastly, the *stationary_property_test()* evaluates the constancy of transition probabilities over time (see Subsection 4.3.2). If no violations against any of these properties are found one can assume with a high likelihood that a given sequence is a time-homogeneous Markov chain of 1^{st} order.

The sequences that will be analyzed by these tests in the future are likely to differ significantly from those generated synthetically in simulations (see Subsection 4.4.3). For instance, the marginal distributions of the worms (Figure 5.26) and the mouse (Figure 5.29) are qualitatively distinct from the simulated Markov processes (see Figures 4.5 and 4.6). Unlike the unimodal distributions observed in simulated sequences, the transition probabilities of the real animals seem multimodal (at lower state spaces). These estimated transition matrices feature a some high transition probabilities and many very low ones, resulting in the two peaks observed in Figures 5.26 and 5.29. It remains unclear whether this multimodal pattern reflects intrinsic characteristics of the *cognitive level* or arises from

the small state spaces, which may fail to capture the full complexity of cognitive behavior. Larger state spaces might reduce the second peak by splitting high-probability transitions into multiple smaller ones as can be seen in the 10 states model of the mouse (see Figure 5.29). To resolve this question, comparisons with brain-wide imaging datasets from other animals might be necessary. In the absence of deeper insight into the cognitive state sequences identified by the *NC-MCM* framework, the current power analysis represents a preliminary step toward identifying measures to constrain clustered sequences within reasonable bounds.

markov_property_test()

The results in Figures 5.1 and 5.2 suggest that the *markov_property_test()* can effectively distinguish first-order Markov sequences of lengths 3000 or 5000 from second-order processes and higher-order processes to a certain extent. Smaller state spaces are more easily detected than larger ones, although this relationship is mitigated by longer sequences. Intuitively, longer sequences capture more motif combinations, enabling pattern detection even with a higher number of states. There appears to be an optimal state space size—depending on sequence length—for this test to perform effectively. Simulations with even larger state spaces reveal that even first-order Markov processes exhibit significant results (see Supplementary Figure 7.3).

The power curves (Figures 5.3 to 5.5) for smaller state spaces exhibit a clear gradient of increasing power with higher effect sizes. This gradient becomes less pronounced for larger state spaces, likely due to the exponential growth in motif possibilities while keeping the sequence length constant. A notable limitation is the low statistical power for short sequences with large state spaces, which restricts the analysis to smaller cognitive states in brief neural recordings. The Type I error rates for no-effect sequences align well with the specified α value of 0.05, ensuring a robust detection rate. It is important to note the general decrease in Type I error for larger state spaces (Figure 5.5), suggesting that a smaller α value might partially mitigate the aforementioned power limitation.

Since the Ornstein-Uhlenbeck process is a Markov process with a stationary distribution [135], it should in theory not be rejected by either test. This is because it fulfills the Markov property and the time-homogeneous assumption. Indeed, this holds true for smaller state spaces (3 to 7) for the *markov_property_test()*. However, the tendency to produce significant results for a non-Markovian process emerges more quickly in the Ornstein-Uhlenbeck process than in a standard synthetic Markov process with transition probabilities sampled from a uniform or normal distribution (see Figures 5.1 and 5.2 and Supplementary Figure 7.3). The main reason most likely lies in the sampling technique and the mean reverting behavior of the Ornstein-Uhlenbeck process[1], which gives it "inertia". Since the process starts as a continuous process, the evolution of the state is influenced not only by the present but also by its proximity to the long-term mean. After discretization, the process still carries remnants of this "inertia," now manifesting as varying strengths of the mean reversion behavior across different time steps (for the same discrete state). This transition from continuous to discrete form may explain the poorer performance of the statistical tests on these sequences, as the persistence of these "inertia" effects can lead to subtle dependencies within the same state, making the process appear less Markovian than expected.

1: Continuous Ornstein Uhlenbeck process:

$$X_t = \theta * (\mu - X_{t-1}) + \sigma$$

Here the θ gives the mean reversion strength (in all simulations: 0.5). The results of the continuous process were discretized for analysis.

This faster tendency towards significance aligns well with the observed Markov property results for the *C. elegans* worm (Figure 5.18) and may provide insights into the type of Markov process emerging at the *cognitive level* of the worm.

stationary_property_test()

The *stationary_property_test()* successfully identified the time-inhomogeneity and, consequently, the likely absence of a stationary distribution in the Random Walk in most cases (Figures 5.6 and 5.7). Detecting time-inhomogeneous behavior is particularly important because the Random Walk, often considered a model for approximating certain natural processes [131]. Similarly, the *pseudo-continuous* time-inhomogeneous process, designed to simulate a primitive form of neuroplasticity by gradually altering transition probabilities between cognitive states, is also consistently rejected by the test.

Taking a look at the power analysis of the test, one can see that similarly to the *markov_property_test()*, this test also shows decreasing power for smaller effect sizes with higher state spaces and shorter sequences (Figures 5.13 and 5.14). However, the power curves show promising results for smaller state spaces (see Figures 5.9 to 5.12), by consistently rejecting the simulated *discrete* and *pseudo-continuous* time-inhomogeneous sequences with length of 2000 or more for moderate to high effect sizes (*discrete* case: 5 changes $\epsilon \geq 0.04$ and *pseudo-continuous* case: changes ≥ 200) with a power of more than 80%. The more consistent performance of the test with longer sequences is further highlighted by the decrease in variance of effect sizes for all simulated sequences. This decrease is quantified by the linear regressions with negative slopes and R^2 values mostly above 0.80. While such R^2 values are easy to achieve with only 5 points per model, the consistency of the results reveals a pattern: longer sequences provide more stable results in both tests. The most important issue with this newly proposed test is that since the H_0 is not percisely stating time-homogeneity in combination with the problem of simulating an according effect size, the Type I error can reach levels of up to 40%. Even tough shorter sequences generally keep it within the preferred 5-10% range, this gets much worse with longer sequences and small state spaces.

This is why further power analysis was done for a smaller α of 0.005 Supplementary Figures 7.6 to 7.11. Adopting this more conservative threshold for the p-values provided by the *stationary_property_test()* yields a less sensitive but more robust detection of time-inhomogeneous sequences, achieving a Type I error rate closer to the preferred 5%. Sequences of 8000 units with 10 states demonstrate the most robustness, achieving a Type I error rate near 5% while detecting all tested effect sizes with over 80% power. Similarly, sequences of 4000 units also show promising results, though with reduced power for weaker effect sizes (see Supplementary Figures 7.8 and 7.11). In general, the high percentage of Type I errors for sequences with no effect result in a less robust test. As the Type I error should harmonize with the expected 5% dictated by the α 0.05 one might rethink the H_0 of the test or the testing procedure needs to be reworked. Important to note here is that a switch from the *State-Balanced splitting* to a purely chronological splitting of chunks did not improve this problem. The numerous hyper-parameters influencing the results of this test — such as chunk number, sequence length, and

state space size — may limit its practical utility. Additionally, the KS- and T-tests are sensitive to significant differences in test group sizes, making the number of simulated sequences another critical hyperparameter affecting the outcomes. If the number of simulations for generating the test statistic (*sim_s*)is set too high, it can lead to an overabundance of significant results. To avoid this, it is recommended to use lower values based not exeding rule of thumb:

$$\text{sim\_s} <= 1 + 1000 * e^{-0.0007 * \text{sequence length}}$$

While lower values in the range of 25–100 are generally sufficient for most cases, this guideline ensures the reliability of the test outcomes. However, the test can still help researcher detect sequences that strongly violate time-inhomogeneity as the Type II error for strong effect sizes was consistently below 10 or 20% for sequences of length 4000 or longer Figures 5.9 to 5.14.

Furthermore, to assess the test's ability to self-determine chunk numbers, Figures 5.6 and 5.8 and Supplementary Figure 7.2 were examined. Results were similar for the self-determined chunks in Supplementary Figure 7.2 and fixed chunk numbers Figure 5.6. The most notable differences emerged in the Ornstein-Uhlenbeck process and the discretized Random Walk when changing from a fixed amount of chunks to self determined chunks. For the Random Walk, time-inhomogeneity was only consistently detected in state spaces 2 to 8 for 3000 long sequences (see Supplementary Figure 7.2) with self determined chunks, while the fixed low number of chunks was better able to detect the time-inhomogeneous nature of the process (2 to 10 with 5 chunks). For the Ornstein-Uhlenbeck process on the other hand, p-values across all state spaces were more stable when self-determining number of chunks, reflecting its stationarity. The more consistent performance for higher state spaces is likely due to the self-determined chunk algorithm's lower number of chunks with more states, which can be observed in p-value results for the Ornstein-Uhlenbeck process in Supplementary Figure 7.2. This is not a given but it will likely pick more chunks with fewer states and lower amounts of chunks for big state spaces which results in more accurate distinctions between time-inhomogeneous and time-homogeneous processes for bigger state spaces (see Figure 5.8 and Supplementary Figure 7.5). A too conservative approach of 2 chunks will lead to worse results for the Random Walk with more than 8 states (see Supplementary Figure 7.2), as the ability to differentiate time-homogeneous and inhomgeneous sequences worsens for very few chunks faster then with intermediate amounts - like the fixed 5 chunks in Figure 5.6 - when state spaces grow (see Figure 5.8). For the bigger state spaces there seems to be a sweet spot between 7-8 chunks for sequences of this length and origins (see Figure 5.8 and Supplementary Figure 7.5).

While there is room for improvement, the method produces practical results. The self determining parameter can be useful when using *cluster_BPT()* for many different state spaces as it automatically adjusts chunk number to state spaces (see Figure 7.1) and may provide more accurate results. This will however strongly depend on the dataset used and a more granular approach using *cluster_BPT_single()* should be preferred.

Interpretation

As discussed, the *markov_property_test()* can readily detect sequences originating from 2nd-order Markov processes (see Figures 5.3 to 5.5). However, higher orders become increasingly difficult to detect in larger state spaces, with 3rd-order Markov processes falling below the 80% power threshold at 10 states for sequences of length 4000 or shorter. This could pose challenges when attempting to confirm that a sequence originates from a 1st-order Markov process with limited neural activity recording lengths. The test did not reject time-inhomogeneous sequences (see Figure 5.1), which are assumed in *NC-MCM* to simplify empirical inference [96]. This weakness is compensated by the second test at least for the synthetic sequences (see Figure 5.6).

Longer sequences generally yield better results (see Figures 5.3, 5.9 and 5.10 and Supplementary Figure 7.4) and excessively large state spaces lead to inconclusive outcomes. The threshold at which state spaces become "too large" for effective testing depends on multiple other factors (system dynamics, recording frequency), but mainly and sequence length.

While the *stationary_property_test()* is less robust and its Type I error is influenced by many hyper-parameters, it appears to be more adept at categorizing time-inhomogeneous sequences and the time-homogeneous version of the Ornstein-Uhlenbeck process. In rejecting *discrete* and *pseudo-continuous* time-inhomogeneous processes the test scores consistently well for sequences of 3000 units (see Figures 5.1 and 5.6). With longer sequence lengths effecting both tests positively in correctly suggesting the Markovian nature of the sequence. This can be seen by the differences for 2nd- and 3rd- order processes in Figures 5.1 and 5.2 for the *markov_-property_test()* and for the time-inhomogeneous processes in Figures 5.6 and 5.7 for the *stationary_property_test()*.

The differences in the properties assessed and therefore different rejection patterns (see Figures 5.1 and 5.6) highlights the complementary nature of the two tests and underscores the importance of using multiple tests to enhance the likelihood of rejecting sequences that deviate from Markov properties or the assumptions of the framework. The less robust *stationary_property_test()*, should be used with more care as its Type I error can be fairly high, this is why it was chosen as an optional test when clustering cognitive state sequences with *cluster_BPT()* or *cluster_BPT_-single()*. One might also consider a lower significance level for detecting violations of the time-homogeneity assumption.

6.2 Performance

Regarding the performance of the toolbox, results seem promising. Apart from the *cluster_BPT()* method, which shows elevated run times (see Table 5.1), each method is straightforward without stretching analysis time unnecessary. In a real-world analysis, *cluster_BPT()* will only be run once to get insight into which amount of cognitive states are consistent with the Markov properties. In later steps of analysis the method *cluster_-BPT_single()* should be sufficient, if one wants to rerun the analysis steps. When cluster size has already been determined, one can run multiple

single clusterings in descending order. This way no new storage array will be created if the existent array is big enough and all data is stored and accessible.

```
data.cluster_BPT_single(nrep=30, nclusters=5, stationary=True)
data.cluster_BPT_single(nrep=30, nclusters=4, stationary=True)
data.cluster_BPT_single(nrep=30, nclusters=3, stationary=True)
```

The runtime durations for statistical tests indicated runtime complexities similar to what was expected (see Subsections 4.3.1 and 4.3.2). On one hand, for both cases the bounding terms ($O(n^3)$ for *markov_property_test()* and $O(n^2c^2)$ for *stationary_property_test()*) for the runtime complexity show only minuscule effects on the results (see Tables 5.2 to 5.5). This is the case because the number of states or the number of chunks are generally much lower than sequence length or even amount of simulations per test. On the other hand, the expected practical complexities, with a clear linear growth of runtime for longer sequences and more simulations. A linear regression predicting runtime from these two values can explain 87% of the runtime for the *markov_property_test()* and for the *stationary_property_test()* ($R^2 = 0.8721$; $R^2 = 0.8723$) (see Tables 5.4 and 5.5), with a slight increase in runtime also being observed for bigger state spaces (see Tables 5.2 and 5.3).

To be precise, as user have influence over these parameters by setting the values for *cluster_num* and *chunk_num* respectively the final computational complexity for both methods will be $O(m)$, as even the *simulations* (or *sim_m* and *sim_s*) can be set by the user. Although too few simulations will influence the results reliability, especially for the *markov_property_test()*.

In the *stationary_property_test()* the self-determining of chunks increases runtimes slightly for longer sequences Table 5.3. This is explained by the algorithm picking greater numbers of chunks if sequences are long enough. In both cases, the growth rate for sequence length is linear, with a doubling of length also doubling the runtime. This is consistent with the predicted computational complexity, where the sequence length occurs only as a linear term for both tests (see Subsections 4.3.1 and 4.3.2). While state space size is a third-order term in the complexity for *markov_property_test()* and a quadratic term in the *stationary_property_test()*, in real-life sequences the tests seem only to work reliably with state spaces on the lower side (see Figures 5.3 to 5.5 and 5.9 to 5.14), which will keep these terms from negatively impacting the runtime too much. Similarly, the quadratic term for the amount of chunks faces similar ignorable influence on runtime, as high chunk numbers will generally negatively impact the accuracy of the *stationary_property_test()* before the runtime will become a problem (see Supplementary Figure 7.5).

6.3 Use on Real-Life Datasets

To evaluate the applicability and robustness of the toolbox, its performance was tested on neural activity datasets. The application of the *NC-MCM* toolbox to the worm data yielded results consistent with those reported in the original *NC-MCM* paper (see Figures 5.16 to 5.18 and 5.22 to 5.24) [96]. Brain wide imaging data from *C. elegans* revealed a potential 1^{st}-order Markov process for worm 4, across lower cognitive state spaces

ranging from 2 to approximately 10 states. The generated diagrams, using 2 or 5 cognitive states (see Figures 5.22 and 5.23), provide valuable insights into cognitive state transitions and the broader processes underlying cognition in this model organism. For the second dataset, while the processes seemed to follow a time-homogeneity, the Markov property was violated for all cognitive state spaces, however, some visualizations still offered meaningful representations of the frameworks workflow (see Figures 5.27 and 5.30 to 5.32).

6.3.1 *Caenorhabditis elegans*

In Figure 5.22 (also replicated in Figure 6.1) the cognitive graph was plotted with two cognitive states, since this state space showed no violations against the Markov properties according to the tests (see Figure 5.18). In it two distinct *slow* behavioral events are evident: one associated with cognitive state 2 (*C2*) and the other, more prevalent, linked to cognitive state 1 (*C1*). Cognitive state 1 is predominantly characterized by the behavioral motif *slow*, followed by *rev2* (reversal 2) and *revsus* (sustained reversal), whereas state 2 comprises the motif *slow*, *rev1* to *revsus*. The *revsus* behavior within *C2* represents a pivotal decision point where the worm selects between initiating a *dt* (dorsal turn) to transition to *C1* or undertaking a *vt* (ventral turn), marking another decision point. These rapid directional changes, described as *pirouettes* [136, 137], have been linked to environmental conditions. Improved environmental quality reduces their frequency [137].

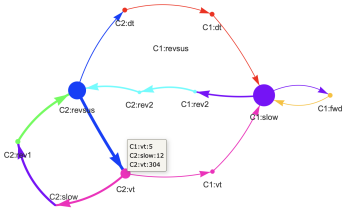


Figure 6.1: This plot is a copy of Figure 5.22 which displays a cognitive graph with 2 cognitive clusters for worm 4.

In Figure 5.23 (also replicated in Figure 6.2), the same motifs are discernible but in greater granularity. Higher numbers of cognitive states reveal otherwise hidden patterns in behavioral diagrams, such as the *C4:slow* node bypassing *C2* (formerly *C1* in Figure 6.1) entirely before transitioning back to *C3:revsus* (formerly *C2* in Figure 6.1) through *C4:rev2* and *C3:rev2*. Important to not here is that, while the 5 cognitive state sequences did not seem to violate the Markov property, it suggested sufficient evidence against the time-homogeneity assumption according to the results of the *stationary_property_test()* seen in Figure 5.18.

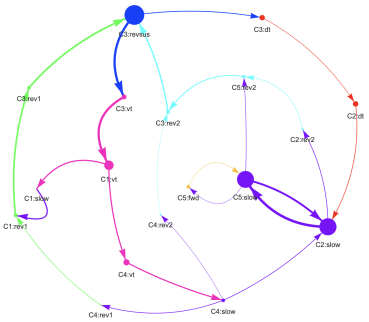


Figure 6.2: This plot is a copy of Figure 5.23 which displays a cognitive graph with 5 cognitive clusters for worm 4.

6.3.2 V1 Area of *Mus musculus*

The toolbox was also tested on calcium imaging data from the V1 area of a mouse brain. Unlike the *C. elegans* data, this dataset lacked inputs from multiple brain areas. Results from Figure 5.28 confirm violations against the Markov property. The resulting cognitive state diagram in Figures 5.31 and 5.32 fails to meet the tested assumptions, prohibiting causal reasoning on cognitive graphs. Disconnected sub-graphs, such as the *C1:moving backward* node, often indicate violations of the Markov property.

A significant issue in this analysis is the potential incompatibility between the recorded brain area — V1 — and the behavior labeled as a motor response. The V1 area has traditionally been associated with visual integration, however, recent studies have also shown that it incorporates hippocampal information and subjective spatial positioning [138, 18, 25]. This mismatch may hinder the assumed presence of a 1st-order Markov

process. However, subsequent additional analysis using behavioral labels based on spatial grid position within the Y-maze yielded similar results in terms of the *markov_property_test()*. Future work might consider using the cues presented during the trials [18] as behavioral labels, as these could more closely align with the processes occurring in the visual cortex of the mouse. This approach could possibly facilitate the identification of Markovian *cognitive level* sequences.

Despite these challenges, Figure 5.30 highlights the utility of *BunDLe-Net*, which effectively imposed a Markov process on the latent embedding and achieved an in-sample accuracy of 93.2%, outperforming multi-label logistic regression (in-sample accuracy = 82.6%). This visualization method is particularly effective for analyzing transitions between behaviors. This is particularly visible in the comparison plot for the *C. elegans* worm 4 (Figure 5.24), where the algorithm consistently correctly identified *revsus* behaviors, while it occasionally misclassified preceding or succeeding states as *revsus*.

This analysis focused on a single session from one mouse in the dataset provided by Tseng et al. [18]. The selected NWB file was 0.5 GB in size, and both downloading it from the DANDI Archive [48] and processing the data were computationally demanding on the machine used. While additional studies involving different mice and sessions could yield more conclusive results, expanding the scope to include these was beyond the current project's limitations.

6.3.3 Other Datasets

At present, the availability of open-source brain imaging datasets including meaningful behavioral labels is limited. While behavioral labels are increasingly accessible, the combination with neural activity data remains rare. Advances in whole-brain imaging technologies for more complex organisms promise exciting opportunities for applying frameworks like *NC-MCM*. Adopting FAIR (Findable, Accessible, Interoperable, Reusable) principles [47] will be essential for fostering collaboration and enabling researchers from diverse fields to unravel the complexities of neural and cognitive processes.

6.4 Toolbox choices & Usability

The toolbox' focus on usability by limiting classes and analysis steps, while not sacrificing customizability in the use of the *NC-MCM* framework was executed nicely. This is highlighted by the fact, that only two (in case of the *step_plot()*) or four (to visualize cognitive graphs) code lines are needed to access *NC-MCM* and its visualizations.

The interactive cognitive graph visualizations generated by the *pyvis* package allow researchers to manipulate nodes and edges, aiding in the interpretation of results. These visualizations are accessible directly through browsers without requiring specialized software. However, the *xml_file* parameter when creating theses cognitive graphs also allows users to generate XML or GML files, facilitating compatibility with external analysis tools like *Cytoscape* [139].

The method names are descriptive and after some time working with the toolbox the use of individual components becomes intuitive. While there is something to be said regarding the high coupling of the individual components, the toolbox still allows for easy integration of new tools, by storing data in a single container class (*Database*) and thereby being able to hold modularity high with future updates.

Accessing the *GitHub* repository and comprehensive documentation using *Sphinx* helps researchers to get started. The repository allows users to easily access the code base and add to it, improving upon the initial version.

6.4.1 Reasoning for *CustomEnsembleModel*

As mentioned in Subsection 4.2.4 the *CustomEnsembleModel* was set as a default for models fit the a *Database* instance, since the results provided by the higher dimensional probability space were more agreeable with Markovian behavior in *C. elegans*. This could be explained by the probability space (*yp_map*) having more dimensions (*behavioral probabilities*), which in turn helps differentiating singular behaviors during clustering. Increasing the amounts of dimensions indefinitely is clearly not the best solution, however when working with smaller dimensions, this is a option which helps make the clustered cognitive state sequences more align with the Markovian assumption. As the dimensions of the *CustomEnsembleModel* will grow exponentially with the amount of different behavior labels, the curse of dimensionality [140] will decrease the quality of the clustering with more behaviors. Where exactly this point lies should be investigated in future studies, where datasets of more different animals with more complex behaviors are available.

6.5 Conclusion

The package is a promising tool for researchers working on brain wide dynamics. While for now the toolbox has only been tested on single neuron percision data, certainly an application on much more widely available fMRI dataset could also bring interesting results. This is especially interesting as the fMRI datasets normally include whole brain imaging data. The toolbox' plots can be used to generate and test hypotheses on the *cognitive* level, by employing the groundbreaking *NC-MCM* framework, which allows for causal reasoning if Markovian behavior is given on the *cognitive level*. This constraint is facilitated by the two statistical tests implemented, while the *stationary_property_test()* is less robust it can still provide valuable data in rejecting time-inhomogeneous processes, such as a Random Walk, more reliably. The export of cognitive graphs in XML format allows cross platform research using other tools such as *Cytoscape*. Using *Python's* accessibility a wide range of researchers and early career scientists can be reached, while the *GitHub* repository will enable growth of the toolbox.

6.6 Acknowledgment

I want to thank Prof. Grosse-Wentrup M. for his assistance and advice, as well as for allowing me to create this toolbox. Furthermore, I want to thank Akshey K. for his assistance with questions relating *BunDLe-Net*.

Appendix 7

7.1 Abbreviations

- **PCA** = Principal Component Analysis
- **UMAP** = Uniform Manifold Approximation and Projection
- **T-SNE** = t-Distributed Stochastic Neighbor Embedding
- **NMF** = Non-Negative Matrix Factorization

7.2 Variables Used in Mathematical Equations

- D = Underlying distribution
- $\hat{D}$ = Sample from the underlying distribution
- d = Sample from the test
- n = Number of different (cognitive) states in a sequence
- m or T = Length of the sequence
- s = Simulated sequences
- R = Number of occurrences of the rarest state
- k = Dimensions for the custom model
- ϵ = A very small float number

7.3 Detailed Documentation Overview

7.3.1 Database Class

ncmcm → ncmcm_classes → Database.py

The `Database` class serves as a container for neuronal activity data and behavioral labels. It offers the following parameters:

Required Parameters

- **neuron_traces:** A list of lists or a `numpy` array containing deconvolved neuronal activity data.
- **behavior:** Behavioral labels provided as a `Python` list or a `numpy` array. If given as a list of strings, they are converted to integers and stored in the `behavioral_states` attribute.

Optional Parameters

- **neuron_names:** Names of neurons, relevant for species with determined mosaic development like *C. elegans* [141] or for single neuron analysis.
- **behavioral_states:** Custom names for the behaviors in the `behavior` array; autogenerated if left empty.

Table 7.1: A legend for the colors used to fill the time tables of Subsection 5.1.1.

Duration (Color)
0 - 2 s
2 - 3.5 s
3.5 - 5 s
5 - 6.5 s
6.5 - 8 s
8 - 9.5 s
9.5 - 15 s
15 - 20 s
20 - 30 s
30 - 45 s
45 - 70 s
70+ s

Table 7.2: Legend indicating text styles used for different objects.

Object Type and Text Style
Introduced nomenclature
Python libraries
Python classes
Abbreviations
Species names
Behaviors
Paper titles
Class methods
Folders
Files

- **fps:** Frame rate of the imaging, necessary for the bandpass filter of `BunDLe-Net` and real-time movies.
- **colors:** Specifies a color spectrum for behavioral labels (1, 2, or 3); defaults to equidistant colors.
- **name:** Name of the container object, used in the titles of subsequent plots.

Main Analysis Methods

The `Database` class offers several methods to facilitate analysis within the *NC-MCM* framework. While the `step_plot()` method allows the user to directly access the step-by-step application of the framework, other methods allow the user more nuanced control over each individual analysis step:

- **fit_model***(base_model, ensemble=True, cv_folds=0)*:
  Fits a classification model to the neuronal activity data and behavioral labels. The model must support the `fit`, `predict`, and `predict_proba` methods. By default, an ensemble model is used (see Section 4.2.4). This method also supports cross-validation via the `cv_folds` parameter. The method will also generate the behavioral probabilities needed late (`yp_map`).
- **cluster_BPT***(nrep=200, cluster_num=20, sim_m=200, sim_s=100, chunks=None, clustering='kmeans', kmeans_init='auto', plot_markov=True, stationary=False, verbose=1)*:
  Iteratively clusters the time-series of behavioral probabilities from the `yp_map` into cognitive states. The sequence created by the clustering algorithm (default: k-means) is evaluated for non-Markovian behavior (see Section 4.3.1). The sequence with the least signs of non-Markovian behavior is used for later visualizations.
- **cluster_BPT_single***(cluster_num, nrep=200, sim_m=200, sim_s=100, chunks=None, clustering='kmeans', kmeans_init='auto', stationary=False, verbose=1)*:
  A streamlined version of `cluster_BPT()`, focusing on a single cluster size to reduce computational complexity. This method takes the same parameters as `cluster_BPT()`, except for `max_clusters`, which is replaced by `nclusters`.
- **step_plot***(clusters=5, nrep=10, sim_m=200, sim_s=100, save=False, show=True, png_name=None)*:
  Generates a four-panel plot using outputs from `cluster_BPT_single()`. The panels show behavioral labels and cognitive states plotted against neuronal data and `yp_map` points, effectively demonstrating the *NC-MCM* framework.
- **behavioral_state_diagram***(cog_stat_num=3, threshold=None, offset=2.5, adj_matrix=False, show=True, save=False, interactive=False, xml_file=False, physics=None, weights_hist=False, bins=15, clustering_rep=None)*:
  Creates cognitive graphs that illustrate cognitive states categorized into behavior nodes and their transitions. The `interactive` parameter enables the creation of an HTML file with interactive

graph features, while *xml_file* allows to specify the generation of a XML-file. Further features include the adjacency matrix which can be displayed by setting `adj_matrix` to `True` and the `weight_hist` parameter that allows users to see a histogram of edge strengths. The latter can be used to decide on a `threshold` at which to stop displaying thinner edges. Lastly, the `offset` parameter fixes the distance between individual cognitive clusters for the simple plot, while the `physics` parameter allows the user to add custom physics for the interactive graph.

- **plotting_behavioral_neuronal***(vmin=0, vmax=2)*:
  Produces a plot with neuronal activation in a heat map and behavior over time, useful for initial data exploration. The heatmap's color scale can be adjusted with `vmin` and `vmax`.

- **exclude_neurons***(exclude_neurons)*:
  Excludes specified neurons from the analysis, useful when certain neurons are used for labeling.

Using a `Database` object the user can create a second object called the `Visualizer` using methods from the `visualizer_creation.py` script:

- **create_bundle_visualizer***(database, epochs=2000, window=15, use_predictor=True, l_dim=3)*:
  Creates a `Visualizer` instance using a `BunDLe-Net` for three-dimensional visualizations. The parameters `l_dim` and `use_predictor` are generally not used, while with `epochs` the training time can be adjusted and the `window` parameter influences the number of past states influencing the prediction.

- **load_bundle_visualizer***(database, weights_path, l_dim=3, window=15, use_predictor=True)*:
  Similar to `create_visualizer()`, but loads a previously saved `BunDLe-Net` model from its saved weights (located using the `weights_path`).

7.3.2 Visualizer Class

ncmcm → ncmcm_classes → Visualizer.py

The `Visualizer` class handles all three-dimensional visualizations. It requires a `Database` object as input and offers several optional parameters for customizing visualizations.

Required Parameter

- **data:** The `Database` object containing the data to be visualized.

Optional Parameters

- **mapping:** Specifies the dimensionality reduction technique for mapping neuronal activity data to three dimensions.

- **transform:** A boolean indicating whether to transform the neuronal activity data using the mapping. Default is `True`.

Visualization Methods

- **plot_mapping***(show_legend=False, grid_off=True, quivers=False, show=True, draw=True)*:
  Plots neuronal activity data in three-dimensional space using the previously specified mapping. The plot can be customized with options like `show_legend`, `grid_off`, and `quivers`.

- **change_mapping***(new_mapping)*:
  Allows replacing the current mapping with a new one.

- **attach_bundle***(l_dim=3, epochs=2000, window=15, train=True, use_predictor=True)*:
  Creates and trains a new `BunDLe-Net` using the `create_bundle_visualizer` method described above. It will replace current mappings and is used for future three-dimensional plots as well as prediction of labels if `use_predictor` is `True`.

- **save_weights***(path)*:
  If a `BunDLe-Net` is attached to the `Visualizer`, the weights are saved at the location defined by `path`.

- **plot_loss***()*:
  Displays the training loss over epochs after training a `BunDLe-Net` model on data of the `Database` instance.

- **make_movie***(interval=None, save=False, show_legend=False, grid_off=True, quivers=False, draw=True)*:
  Creates an animated GIF of the mapping, with options for saving, setting the frame interval, and drawing all points at the start.

- **save_gif***(name, bitrate=1800, dpi=144)*:
  If a GIF has been created previously, this allows the user to directly access the resolution (`dpi`) and framerate (`bitrate`) of the movie and save it.

- **use_mapping_as_input***()*:
  Returns a new `Database` instance using the transformed points as input for `neuron_traces`.

- **make_comparison***(show_legend=True, quivers=True)*:
  Visualizes model in-sample predictions against true behaviors, showing correctly and incorrectly predicted behaviors in a subplot.

7.4 Supplementary Tables

Table 7.3: This table lists the exact rejection counts for the *markov_property_test()* at a significance level of $\alpha = 0.05$, based on 100 simulated sequences of 5000 units each. The sequences were generated from processes including 1st-, 2nd-, and 3rd-order Markov processes, a random process, an Ornstein-Uhlenbeck process, a *discrete* and *pseudo-continuous* non-stationary process, and a Random Walk. Results are visualized in Figure 5.2.

3 States	HO kept	H_0 rejected
1st order M.	94	6
2nd order M.	0	100
3rd order M.	0	100
Random	95	5
OU	96	4
Discrete non-stat.	0	100
Pseudo-Cont. non-stat.	45	55
Random Walk	32	68

5 States	HO kept	H_0 rejected
1st order M.	97	3
2nd order M.	0	100
3rd order M.	0	100
Random	94	6
OU	87	13
Discrete non-stat.	0	100
Pseudo-Cont. non-stat.	79	21
Random Walk	99	1

10 States	HO kept	H_0 rejected
1st order M.	97	3
2nd order M.	0	100
3rd order M.	11	89
Random	98	2
OU	55	45
Discrete non-stat.	83	17
Pseudo-Cont. non-stat.	61	39
Random Walk	100	0

15 States	HO kept	H_0 rejected
1st order M.	99	1
2nd order M.	0	100
3rd order M.	98	2
Random	100	0
OU	4	96
Discrete non-stat.	100	0
Pseudo-Cont. non-stat.	92	8
Random Walk	100	0

Table 7.4: The table lists the exact rejection counts for the KS-Test in the *stationary_property_test()* at a significance level of $\alpha = 0.05$, based on 100 simulated sequences of 5000 units each. The amount of chunks was set to 5 and the *simulations* to 50. Sequences were generated from processes including 1st-, 2nd-, and 3rd- order Markov processes, a random process, an Ornstein-Uhlenbeck process, a *discrete* and *pseudo-continuous* non-stationary process and a discretized Random Walk. Results are visualized in Figure 5.7.

3 States	HO kept	H_0 rejected
1st order M.	87	13
2nd order M.	82	18
3rd order M.	84	16
Random	86	14
OU	97	3
Discrete non-stat.	0	100
Pseudo-Cont. non-stat.	3	97
Random Walk	2	98

5 States	HO kept	H_0 rejected
1st order M.	85	15
2nd order M.	87	13
3rd order M.	85	15
Random	90	10
OU	93	7
Discrete non-stat.	0	100
Pseudo-Cont. non-stat.	2	98
Random Walk	4	96

10 States	HO kept	H_0 rejected
1st order M.	90	10
2nd order M.	83	17
3rd order M.	86	14
Random	87	13
OU	81	19
Discrete non-stat.	0	100
Pseudo-Cont. non-stat.	0	100
Random Walk	15	85

15 States	HO kept	H_0 rejected
1st order M.	90	10
2nd order M.	85	15
3rd order M.	87	13
Random	81	19
OU	58	42
Discrete non-stat.	0	100
Pseudo-Cont. non-stat.	3	97
Random Walk	30	70

7.5 Supplementary Figures

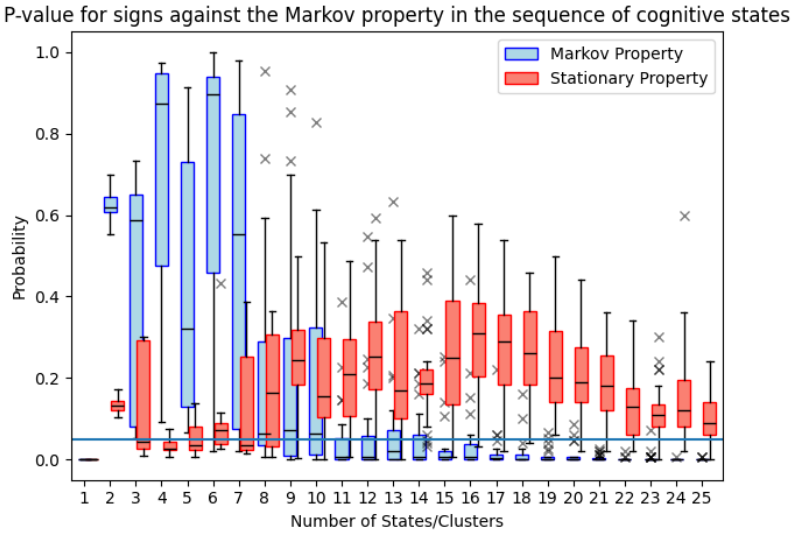


Figure 7.1: Results for sequence through cognitive state space of the test for the Stationary property are in red with self-determined chunks. The Markov property result are in blue. The amount of cognitive states is listed on the x-axis, while p-values for both tests are indicated by y-values. Each boxplot indicates results for 30 repetitions of clustering and subsequent testing. The *cluster_BPT()* method created this plot for worm 4.

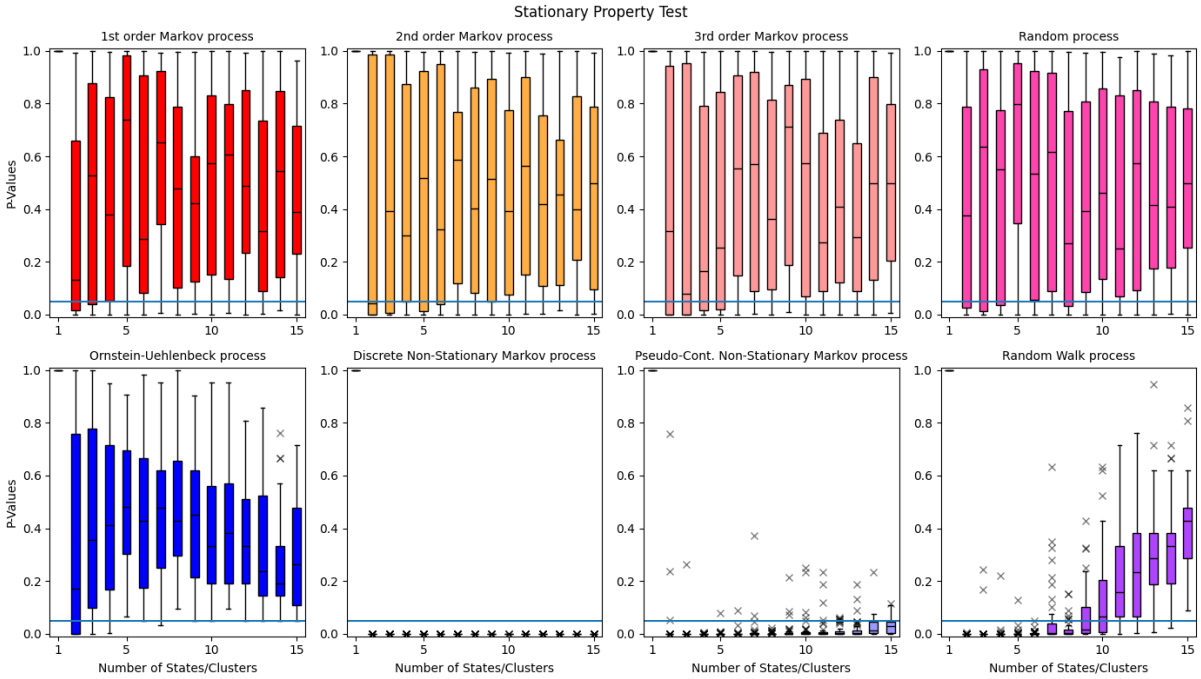


Figure 7.2: This plot shows results of the *stationary_property_test()* method with self-determined chunks for simulated sequences of length 3000 with p-values on the Y-axis and number of states on the X-axis. The plot is created from 50 of each 1st order Markov (*A*), 2nd order Markov (*B*), random (*C*) and 1st order non-stationary Markov sequences (*D*). Below 0.05 p-values would indicate that the sequence is following a stationary process. Each boxplot visualizes results from 50 sequences and the parameter *simulations* was fixed at 50 for all runs.

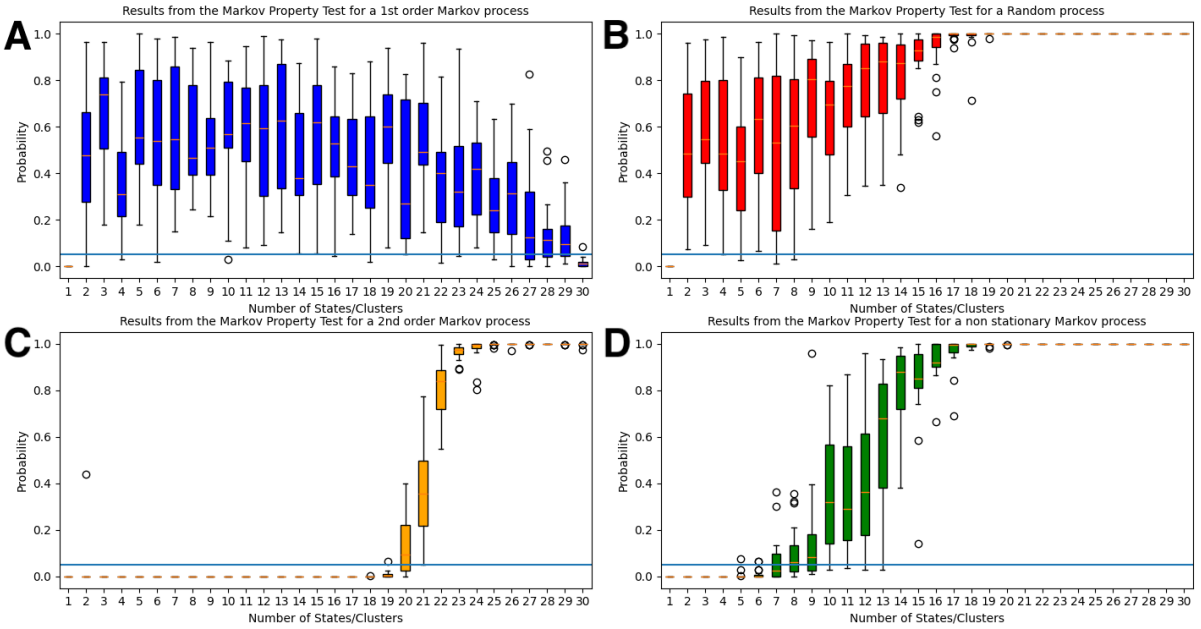


Figure 7.3: This plot shows results of the test for the Markov property of a 1st- and 2nd- order Markov process, a random process as well as a *discrete* non-stationary process. All sequences were 5000 units long and the amount of different states are listed on the X-axis, while p-values are indicated by the Y-axis. Each boxplot visualizes results from 50 sequences and the parameter *simulations* was fixed at 300 for all runs.

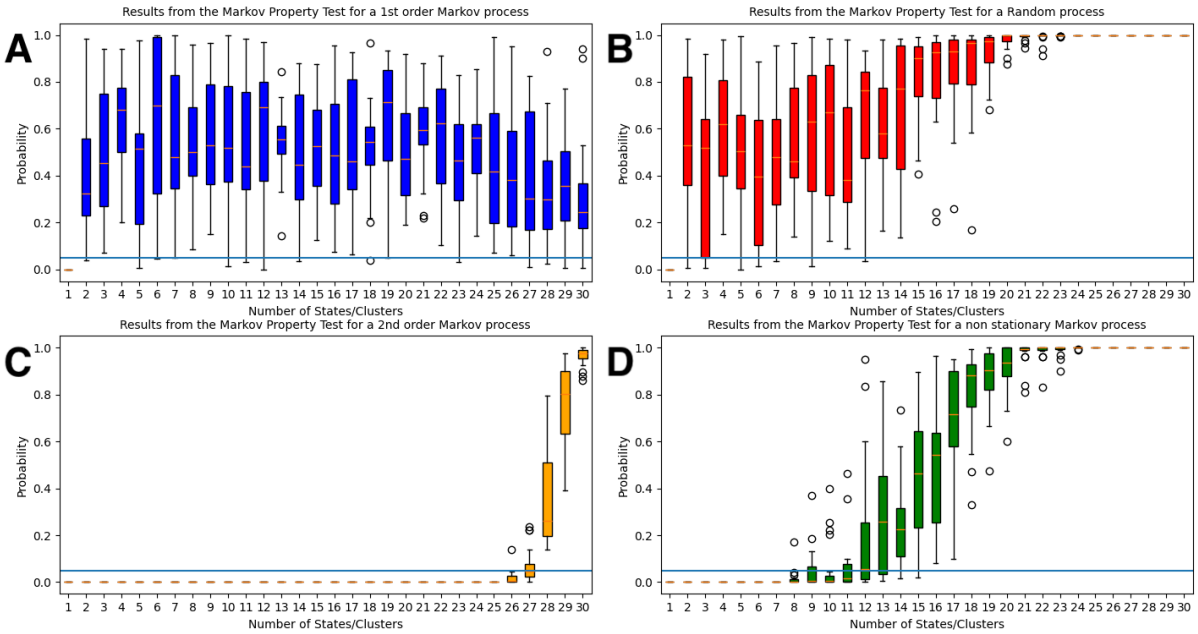


Figure 7.4: Results of the test for the Markov property of a random (*B*), 1st order (*A*), 2nd order (*C*) and non-stationary process (*D*). The sequences were 10000 units long and the amount of different states are listed on the x-axis, while p-values are indicated by the y-axis.

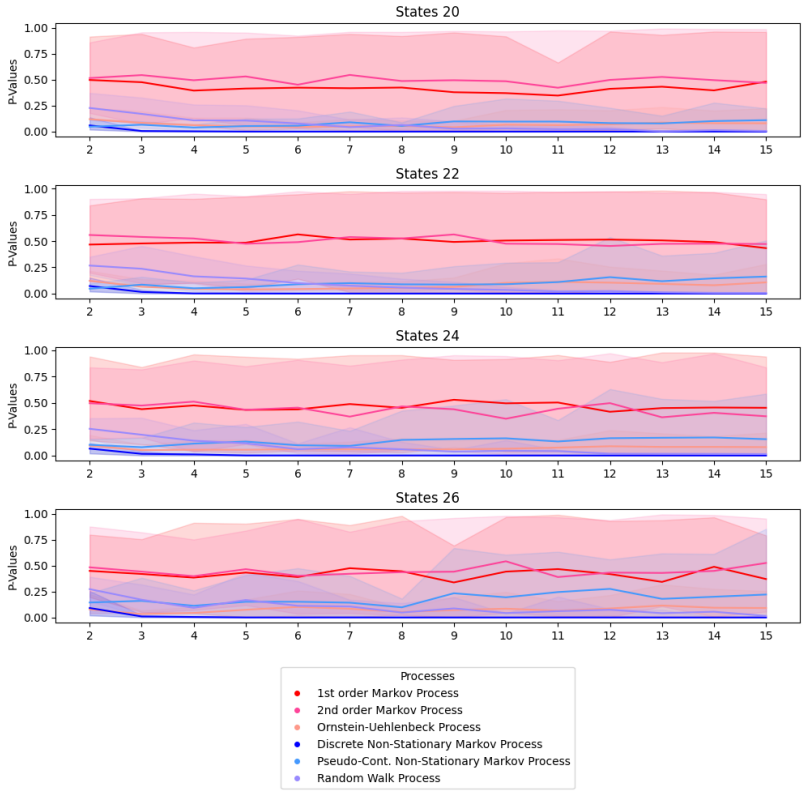


Figure 7.5: Results of the *stationary_property_test()* for 50 simulated sequences each of a 1st-, 2nd- order Markov process, a Ornstein-Uhlenbeck process, a *discrete* and *pseudo-continuous* non-stationary process as well as a discretized Random Walk. All sequences were of length 3000. Different subplots were made for 18, 20, 22, & 20 states. The amount of chunks used in the test is displayed on the X-axis, while p-values are displayed on the Y-axis. A 75% confidence interval is given for all results.

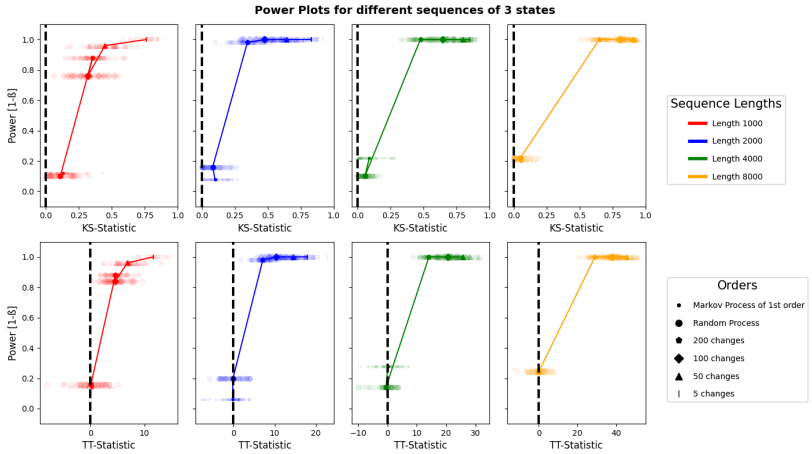


Figure 7.6: Power curve for the *stationary_property_test()* method for sequences of different lengths with 3 states and $\alpha = 0.005$ for the KS- or t- Tests, using 5 directed changes in the transition probabilities with different amplitudes (epsilon) as effect sizes. Effect sizes are listed on the x-axis, while the power $(1 - \beta)$ or Type I error probability is indicated by the y-axis.

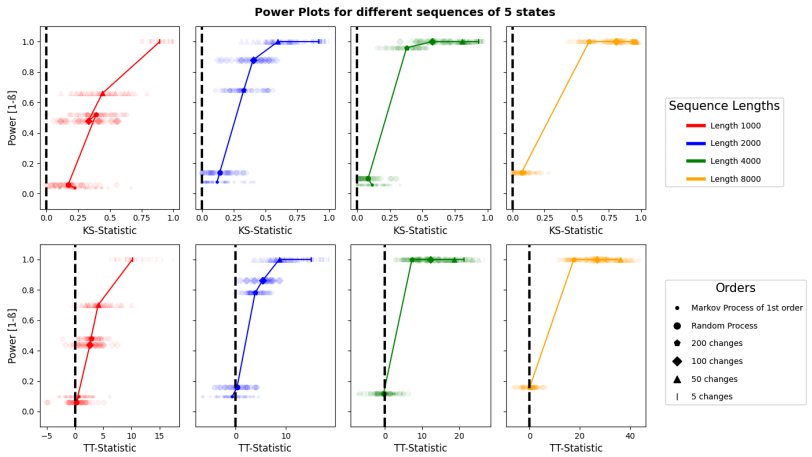


Figure 7.7: Power curve for the *stationary_property_test()* method for sequences of different lengths with 5 states and $\alpha = 0.005$ for the KS- or t- Tests, using 5 directed changes in the transition probabilities with different amplitudes (epsilon) as effect sizes. Effect sizes are listed on the x-axis, while the power $(1 - \beta)$ or Type I error probability is indicated by the y-axis.

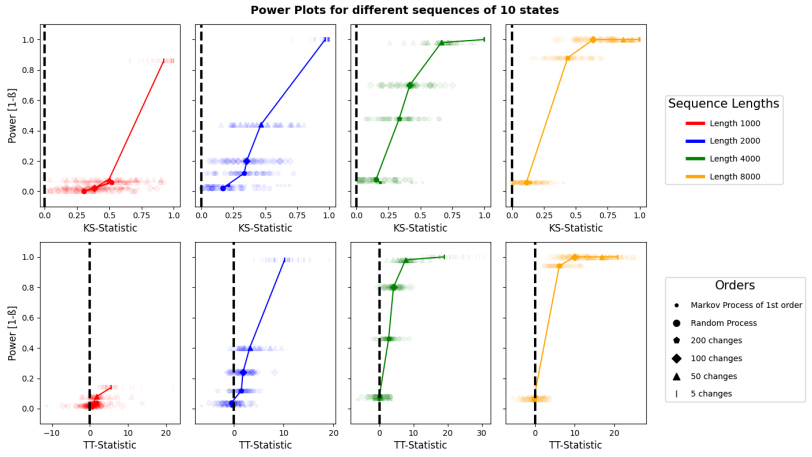


Figure 7.8: Power curve for the *stationary_property_test()* method for sequences of different lengths with 10 states and $\alpha = 0.005$ for the KS- or t- Tests, using 5 directed changes in the transition probabilities with different amplitudes (epsilon) as effect sizes. Effect sizes are listed on the x-axis, while the power $(1 - \beta)$ or Type I error probability is indicated by the y-axis.

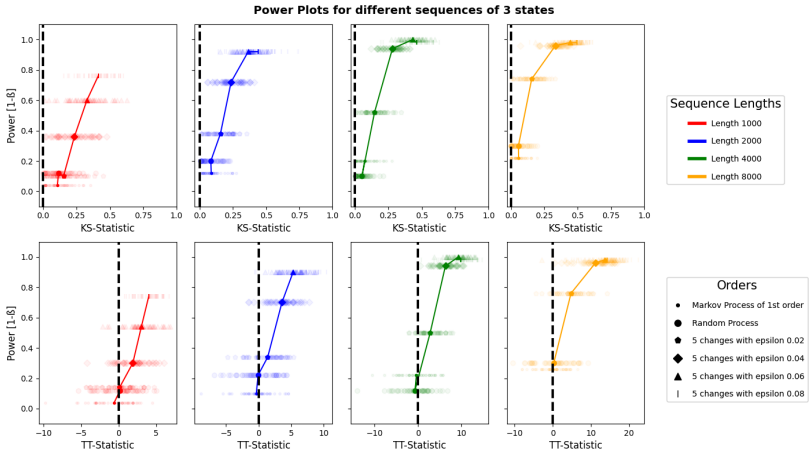


Figure 7.9: Power curve for the *stationary_property_test()* method for sequences of different lengths with 3 states and $\alpha = 0.005$ for the KS- or t- Tests, using 5 directed perturbations in the transition probabilities with different amplitudes (epsilon) as effect sizes. Effect sizes are listed on the X-axis, while the power $(1 - \beta)$ or Type I error probability is indicated by the Y-axis.

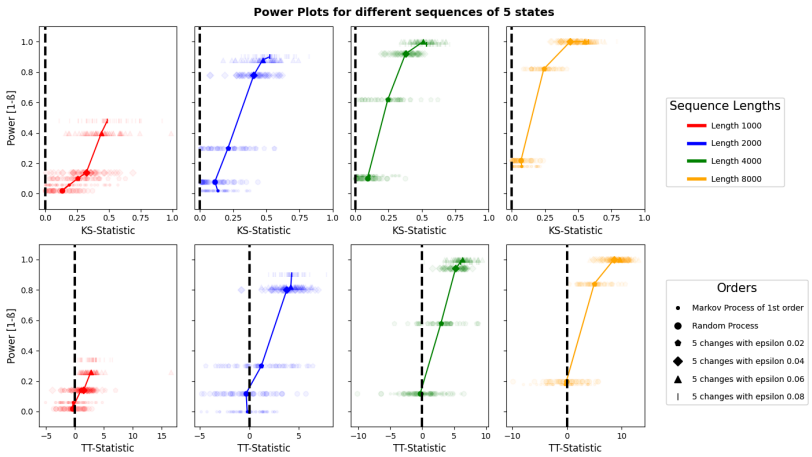


Figure 7.10: Power curve for the *stationary_property_test()* method for sequences of different lengths with 5 states and $\alpha = 0.005$ for the KS- or t- Tests, using 5 directed perturbations in the transition probabilities with different amplitudes (epsilon) as effect sizes. Effect sizes are listed on the X-axis, while the power $(1 - \beta)$ or Type I error probability is indicated by the Y-axis.

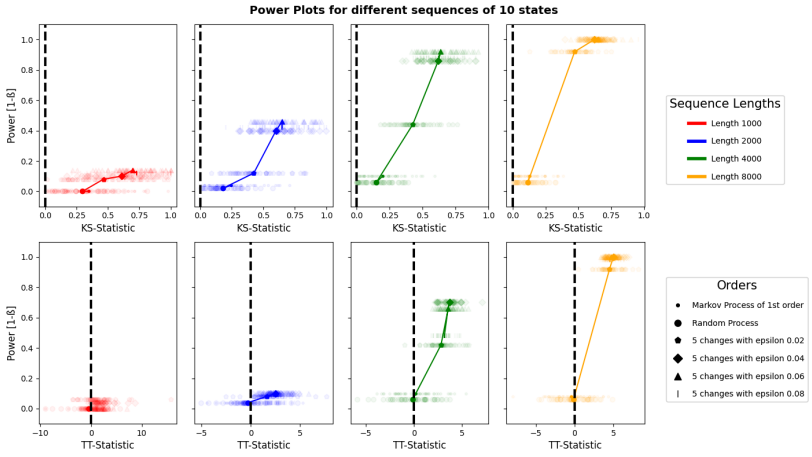


Figure 7.11: Power curve for the *stationary_property_test()* method for sequences of different lengths with 10 states and $\alpha = 0.005$ for the KS- or t- Tests, using 5 directed perturbations in the transition probabilities with different amplitudes (epsilon) as effect sizes. Effect sizes are listed on the X-axis, while the power $(1 - \beta)$ or Type I error probability is indicated by the Y-axis.

Bibliography

[1] Sara Larivière, Şeyma Bayrak, Reinder Vos de Wael, Oualid Benkarim, Peer Herholz, Raul Rodriguez-Cruces, Casey Paquola, Seok-Jun Hong, Bratislav Misic, Alan C. Evans, Sofie L. Valk, and Boris C. Bernhardt. Brainstat: A toolbox for brain-wide statistics and multimodal feature associations. NeuroImage, 266:119807, 2023.

[2] Reinder Vos de Wael, Oualid Benkarim, Casey Paquola, Sara Lariviere, Jessica Royer, Shahin Tavakol, Ting Xu, Seok-Jun Hong, Georg Langs, Sofie Valk, Bratislav Misic, Michael Milham, Daniel Margulies, Jonathan Smallwood, and Boris C. Bernhardt. BrainSpace: a toolbox for the analysis of macroscale gradients in neuroimaging and connectomics datasets. Communications Biology, 3(1):103, March 2020.

[3] Santiago Ramón y Cajal. Histologie du système nerveux de l'homme & des vertébrés, volume v. 1. Maloine, Paris, 1909. Pages: 1-1012.

[4] Alexander Borst. Fly visual course control: behaviour, algorithms and circuits. Nature Reviews Neuroscience, 15(9):590–599, September 2014. Number: 9 Publisher: Nature Publishing Group.

[5] Mriganka Sur and Catherine A. Leamey. Development and plasticity of cortical areas and networks. Nature Reviews Neuroscience, 2(4):251–262, April 2001.

[6] Ramon y Cajal. Estudios sobre la cortexa cerebral humana. Corteza visual. Rev. Trim. Microgr., 4:1, 1899.

[7] May-Britt Moser, David C. Rowland, and Edvard I. Moser. Place Cells, Grid Cells, and Memory. Cold Spring Harbor Perspectives in Biology, 7(2):a021808, February 2015. Company: Cold Spring Harbor Laboratory Press Distributor: Cold Spring Harbor Laboratory Press Institution: Cold Spring Harbor Laboratory Press Label: Cold Spring Harbor Laboratory Press Publisher: Cold Spring Harbor Lab.

[8] J. O'Keefe and J. Dostrovsky. The hippocampus as a spatial map. Preliminary evidence from unit activity in the freely-moving rat. Brain Research, 34(1):171–175, November 1971.

[9] Laure Buhry, Amir H. Azizi, and Sen Cheng. Reactivation, Replay, and Preplay: How It Might All Fit Together. Neural Plasticity, 2011:203462, 2011.

[10] C Pavlides and J Winson. Influences of hippocampal place cell firing in the awake state on the activity of these cells during subsequent sleep episodes. The Journal of Neuroscience, 9(8):2907–2918, August 1989.

[11] Miriam S. Nokia, Markku Penttonen, and Jan Wikgren. Hippocampal Ripple-Contingent Training Accelerates Trace Eyeblink Conditioning and Retards Extinction in Rabbits. The Journal of Neuroscience, 30(34):11486–11492, August 2010.

[12] Aya Ben-Yakov, Yadin Dudai, and Mark R. Mayford. Memory Retrieval in Mice and Men. Cold Spring Harbor Perspectives in Biology, 7(12):a021790, December 2015.

[13] William E. Skaggs, Bruce L. McNaughton, Michele Permenter, Matthew Archibeque, Julie Vogt, David G. Amaral, and Carol A. Barnes. EEG Sharp Waves and Sparse Ensemble Unit Activity in the Macaque Hippocampus. Journal of Neurophysiology, 98(2):898–910, August 2007. Publisher: American Physiological Society.

[14] David Rose. Some Reflections on (or by?) Grandmother Cells. Perception, 25(8):881–886, August 1996. Publisher: SAGE Publications Ltd STM.

[15] Gabriel Kreiman, Christof Koch, and Itzhak Fried. Category-specific visual responses of single neurons in the human medial temporal lobe. Nature Neuroscience, 3(9):946–953, September 2000. Number: 9 Publisher: Nature Publishing Group.

[16] Charles G. Gross. Coding for visual categories in the human brain. Nature Neuroscience, 3(9):855–856, September 2000. Number: 9 Publisher: Nature Publishing Group.

[17] Ann-Sophie Barwich. The Value of Failure in Science: The Story of Grandmother Cells in Neuroscience. Frontiers in Neuroscience, 13, 2019.

[18] Shih-Yi Tseng, Selmaan N. Chettih, Charlotte Arlt, Roberto Barroso-Luque, and Christopher D. Harvey. Shared and specialized coding across posterior cortical areas for dynamic navigation decisions. Neuron, 110(15):2484–2502.e16, August 2022.

[19] Carsen Stringer, Marius Pachitariu, Nicholas Steinmetz, Charu Bai Reddy, Matteo Carandini, and Kenneth D. Harris. Spontaneous Behaviors Drive Multidimensional, Brain-wide Activity. Science (New York, N.Y.), 364(6437):255, April 2019.

[20] Nicholas A. Steinmetz, Peter Zatka-Haas, Matteo Carandini, and Kenneth D. Harris. Distributed coding of choice, action, and engagement across the mouse brain. Nature, 576(7786):266–273, December 2019.

[21] Joshua H. Siegle, Xiaoxuan Jia, Séverine Durand, Sam Gale, Corbett Bennett, Nile Graddis, Greggory Heller, Tamina K. Ramirez, Hannah Choi, Jennifer A. Luviano, Peter A. Groblewski, Ruweida Ahmed, Anton Arkhipov, Amy Bernard, Yazan N. Billeh, Dillan Brown, Michael A. Buice, Nicolas Cain, Shiella Caldejon, Linzy Casal, Andrew Cho, Maggie Chvilicek, Timothy C. Cox, Kael Dai, Daniel J. Denman, Saskia E. J. de Vries, Roald Dietzman, Luke Esposito, Colin Farrell, David Feng, John Galbraith, Marina Garrett, Emily C. Gelfand, Nicole Hancock, Julie A. Harris, Robert Howard, Brian Hu, Ross Hytnen, Ramakrishnan Iyer, Erika Jessett, Katelyn Johnson, India Kato, Justin Kiggins, Sophie Lambert, Jerome Lecoq, Peter Ledochowitsch, Jung Hoon Lee, Arielle Leon, Yang Li, Elizabeth Liang, Fuhui Long, Kyla Mace, Jose Melchior, Daniel Millman, Tyler Mollenkopf, Chelsea Nayan, Lydia Ng, Kiet Ngo, Thuyahn Nguyen, Philip R. Nicovich, Kat North, Gabriel Koch Ocker, Doug Ollerenshaw, Michael Oliver, Marius Pachitariu, Jed Perkins, Melissa Reding, David Reid, Miranda Robertson, Kara Ronellenfitch, Sam Seid, Cliff Slaughterbeck, Michelle Stoecklin, David Sullivan, Ben Sutton, Jackie Swapp, Carol Thompson, Kristen Turner, Wayne Wakeman, Jennifer D. Whitesell, Derric Williams, Ali Williford, Rob Young, Hongkui Zeng, Sarah Naylor, John W. Phillips, R. Clay Reid, Stefan Mihalas, Shawn R. Olsen, and Christof Koch. Survey of spiking in the mouse visual system reveals functional hierarchy. Nature, 592(7852):86–92, April 2021.

[22] Matthias Minderer, Kristen D. Brown, and Christopher D. Harvey. The Spatial Structure of Neural Encoding in Mouse Posterior Cortex during Navigation. Neuron, 102(1):232–248.e11, April 2019.

[23] Georg B. Keller, Tobias Bonhoeffer, and Mark Hübener. Sensorimotor Mismatch Signals in Primary Visual Cortex of the Behaving Mouse. Neuron, 74(5):809–815, June 2012.

[24] Marshall G Shuler and Mark F Bear. Reward Timing in the Primary Visual Cortex. Science, 311, 2006.

[25] Andrea Pavan, Filippo Ghin, Rita Donato, Gianluca Campana, and George Mather. The neural basis of form and form-motion integration from static and dynamic translational Glass patterns: A rTMS investigation. NeuroImage, 157:555–560, August 2017.

[26] William E. Allen, Michael Z. Chen, Nandini Pichamoorthy, Rebecca H. Tien, Marius Pachitariu, Liqun Luo, and Karl Deisseroth. Thirst regulates motivated behavior through modulation of brainwide neural population dynamics. Science (New York, N.Y.), 364(6437):253, April 2019.

[27] William E. Allen, Isaac V. Kauvar, Michael Z. Chen, Ethan B. Richman, Samuel J. Yang, Ken Chan, Viviana Gradinaru, Benjamin E. Deverman, Liqun Luo, and Karl Deisseroth. Global Representations of Goal-Directed Behavior in Distinct Cell Types of Mouse Neocortex. Neuron, 94(4):891–907.e6, May 2017.

[28] Apostolos P Georgopoulos, JOSEPH T LuRITo, Michael Petrides, Andrew B Schwartz, and Joe T Massey. Mental Rotation of the Neuronal Populaton Vector. Science, 243, 1989.

[29] Joaquín González, Matias Cavelli, Alejandra Mondino, Nicolás Rubido, BL Tort, and Pablo Torterolo. Communication through coherence by means of cross-frequency coupling. Neuroscience, 2020.

[30] Ryan T. Canolty and Robert T. Knight. The functional role of cross-frequency coupling. Trends in Cognitive Sciences, 14(11):506–515, November 2010.

[31] Isaac V. Kauvar, Timothy A. Machado, Elle Yuen, John Kochalka, Minseung Choi, William E. Allen, Gordon Wetzstein, and Karl Deisseroth. Cortical Observation by Synchronous Multifocal Optical Sampling Reveals Widespread Population Encoding of Actions. Neuron, 107(2):351–367.e19, July 2020.

[32] Simon Musall, Matthew T. Kaufman, Ashley L. Juavinett, Steven Gluf, and Anne K. Churchland. Single-trial neural dynamics are dominated by richly varied movements. Nature neuroscience, 22(10):1677–1686, October 2019.

[33] Tom M. Mitchell, Svetlana V. Shinkareva, Andrew Carlson, Kai-Min Chang, Vicente L. Malave, Robert A. Mason, and Marcel Adam Just. Predicting Human Brain Activity Associated with the Meanings of Nouns. Science, 320(5880):1191–1195, May 2008.

[34] Mamoon Rashid, Harjeet Singh, and Vishal Goyal. The use of machine learning and deep learning algorithms in functional magnetic resonance imaging—A systematic review. Expert Systems, 37(6):e12644, 2020. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/exsy.12644.

[35] Ashish Gupta, Murat Ayhan, and Anthony Maida. Natural image bases to represent neuroimaging data. In Sanjoy Dasgupta and David McAllester, editors, Proceedings of the 30th International Conference on Machine Learning, volume 28 of Proceedings of Machine Learning Research, pages 987–994, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[36] Heung-Il Suk and Dinggang Shen. Deep learning-based feature representation for ad/mci classification. In Medical Image Computing and Computer-Assisted Intervention–MICCAI 2013: 16th International Conference, Nagoya, Japan, September 22-26, 2013, Proceedings, Part II 16, pages 583–590. Springer, 2013.

[37] Ye Ji, Hong-Bo Liu, Xiu-Kun Wang, and Yi-Yuan Tang. Cognitive states classification from fmri data using support vector machines. In Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826), volume 5, pages 2919–2923 vol.5, Aug 2004.

[38] Meenakshi Khosla, Keith Jamison, Amy Kuceyeski, and Mert R. Sabuncu. Ensemble learning with 3d convolutional neural networks for functional connectome-based prediction. NeuroImage, 199:651–662, 2019.

[39] Heung-Il Suk, Chong-Yaw Wee, Seong-Whan Lee, and Dinggang Shen. State-space model with deep learning for functional dynamics estimation in resting-state fmri. NeuroImage, 129:292–307, 2016.

[40] Kenneth A. Norman, Sean M. Polyn, Greg J. Detre, and James V. Haxby. Beyond mind-reading: multi-voxel pattern analysis of fMRI data. Trends in Cognitive Sciences, 10(9):424–430, September 2006.

[41] Jerry Tang, Amanda LeBel, Shailee Jain, and Alexander G. Huth. Semantic reconstruction of continuous language from non-invasive brain recordings. Nature Neuroscience, 26(5):858–866, May 2023.

[42] James S. Gao, Alexander G. Huth, Mark D. Lescroart, and Jack L. Gallant. Pycortex: an interactive surface visualizer for fMRI. Frontiers in Neuroinformatics, 9, September 2015. Publisher: Frontiers.

[43] Elon Musk and Neuralink. An Integrated Brain-Machine Interface Platform With Thousands of Channels. Journal of Medical Internet Research, 21(10):e16194, October 2019. Company: Journal of Medical Internet Research Distributor: Journal of Medical Internet Research Institution: Journal of Medical Internet Research Label: Journal of Medical Internet Research Publisher: JMIR Publications Inc., Toronto, Canada.

[44] James J. Jun, Nicholas A. Steinmetz, Joshua H. Siegle, Daniel J. Denman, Marius Bauza, Brian Barbarits, Albert K. Lee, Costas A. Anastassiou, Alexandru Andrei, Çağatay Aydın, Mladen Barbic, Timothy J. Blanche, Vincent Bonin, João Couto, Barundeb Dutta, Sergey L. Gratiy, Diego A. Gutnisky, Michael Häusser, Bill Karsh, Peter Ledochowitsch, Carolina Mora Lopez, Catalin Mitelut, Silke Musa, Michael Okun, Marius Pachitariu, Jan Putzeys, P. Dylan Rich, Cyrille Rossant, Wei-lung Sun, Karel Svoboda,

Matteo Carandini, Kenneth D. Harris, Christof Koch, John O'Keefe, and Timothy D. Harris. Fully Integrated Silicon Probes for High-Density Recording of Neural Activity. Nature, 551(7679):232–236, November 2017.

[45] Nicholas A Steinmetz, Christof Koch, Kenneth D Harris, and Matteo Carandini. Challenges and opportunities for large-scale electrophysiology with Neuropixels probes. Current Opinion in Neurobiology, 50:92–100, June 2018.

[46] Sébastien Wolf, Willy Supatto, Georges Debrégeas, Pierre Mahou, Sergei G Kruglik, Jean-Marc Sintes, Emmanuel Beaurepaire, and Raphaël Candelier. Whole-brain functional imaging with two-photon light-sheet microscopy. Nature Methods, 12(5):379–380, May 2015.

[47] Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J. G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A. C. 't Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. The FAIR Guiding Principles for scientific data management and stewardship. Scientific Data, 3(1):160018, March 2016. Publisher: Nature Publishing Group.

[48] Oliver Rübel, Andrew Tritt, Ryan Ly, Benjamin K Dichter, Satrajit Ghosh, Lawrence Niu, Pamela Baker, Ivan Soltesz, Lydia Ng, Karel Svoboda, Loren Frank, and Kristofer E Bouchard. The neurodata without borders ecosystem for neurophysiological data science. eLife, 11:e78362, oct 2022.

[49] Alberto Redolfi, Damiano Archetti, Silvia De Francesco, Claudio Crema, Fabrizio Tagliavini, Raffaele Lodi, Roberta Ghidoni, Claudia A. M. Gandini Wheeler-Kingshott, Daniel C. Alexander, and Egidio D'Angelo. Italian, European, and international neuroinformatics efforts: An overview. European Journal of Neuroscience, 57(12):2017–2039, 2023. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/ejn.15854.

[50] Yannick Schwartz, Alexis Barbot, Benjamin Thyreau, Vincent Frouin, Gaël Varoquaux, Aditya Siram, Daniel S. Marcus, and Jean-Baptiste Poline. Frontiers - pyxnat: Xnat in python, 2012.

[51] Steffen Schneider, Jin Hwa Lee, and Mackenzie Weygandt Mathis. Learnable latent embeddings for joint behavioural and neural analysis. Nature, 617(7960):360–368, May 2023. Number: 7960 Publisher: Nature Publishing Group.

[52] Rajesh P. N. Rao and Dana H. Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. Nature Neuroscience, 2(1):79–87, January 1999.

[53] Mayank Kabra, Alice A Robie, Marta Rivera-Alba, Steven Branson, and Kristin Branson. JAABA: interactive machine learning for automatic annotation of animal behavior. Nature Methods, 10(1):64–67, January 2013.

[54] Saima Rathore, Mohamad Habes, Muhammad Aksam Iftikhar, Amanda Shacklett, and Christos Davatzikos. A review on neuroimaging-based classification studies and associated feature extraction methods for alzheimer's disease and its prodromal stages. NeuroImage, 155:530–548, 2017.

[55] Angela Serra, Paola Galdi, and Roberto Tagliaferri. Machine learning for bioinformatics and neuroimaging. WIREs Data Mining and Knowledge Discovery, 8(5):e1248, 2018. e1248 DMKD-00309.R1.

[56] Joshua I. Glaser, Ari S. Benjamin, Roozbeh Farhoodi, and Konrad P. Kording. The roles of supervised machine learning in systems neuroscience. Progress in Neurobiology, 175:126–137, 2019.

[57] Cynthia A. Chestek, Aaron P. Batista, Gopal Santhanam, Byron M. Yu, Afsheen Afshar, John P. Cunningham, Vikash Gilja, Stephen I. Ryu, Mark M. Churchland, and Krishna V. Shenoy. Single-neuron stability during repeated reaching in macaque premotor cortex. Journal of Neuroscience, 27(40):10742–10750, 2007.

[58] Mohammad R. Arbabshirani, Sergey Plis, Jing Sui, and Vince D. Calhoun. Single subject prediction of brain disorders in neuroimaging: Promises and pitfalls. NeuroImage, 145:137–165, 2017. Individual Subject Prediction.

[59] Sandra Vieira, Walter H.L. Pinaya, and Andrea Mechelli. Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications. Neuroscience & Biobehavioral Reviews, 74:58–75, 2017.

[60] Saman Sarraf and Ghassem Tofighi. Deep learning-based pipeline to recognize alzheimer's disease using fmri data. In 2016 Future Technologies Conference (FTC), pages 816–820, 2016.

[61] Harold C. Burger, Christian J. Schuler, and Stefan Harmeling. Image denoising: Can plain neural networks compete with bm3d? In 2012 IEEE Conference on Computer Vision and Pattern Recognition, pages 2392–2399, 2012.

[62] Junyuan Xie, Linli Xu, and Enhong Chen. Image denoising and inpainting with deep neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, Advances in Neural Information Processing Systems, volume 25. Curran Associates, Inc., 2012.

[63] Li Xu, Jimmy SJ Ren, Ce Liu, and Jiaya Jia. Deep convolutional neural network for image deconvolution. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, Advances in Neural Information Processing Systems, volume 27. Curran Associates, Inc., 2014.

[64] Heung-Il Suk, Seong-Whan Lee, Dinggang Shen, and The Alzheimer's Disease Neuroimaging Initiative. Latent feature representation with stacked auto-encoder for AD/MCI diagnosis. Brain Structure and Function, 220(2):841–859, March 2015.

[65] Silvia Basaia, Federica Agosta, Luca Wagner, Elisa Canu, Giuseppe Magnani, Roberto Santangelo, and Massimo Filippi. Automated classification of alzheimer's disease and mild cognitive impairment using a single mri and deep neural networks. NeuroImage: Clinical, 21:101645, 2019.

[66] Sachin S. Talathi. Deep Recurrent Neural Networks for seizure detection and early seizure detection systems, June 2017. arXiv:1706.03283 [cs, q-bio].

[67] Vivek Prakash Nigam and Daniel Graupe. A neural-network-based detection of epilepsy. Neurological Research, 26(1):55–60, 2004. PMID: 14977058.

[68] Benjamin H. Brinkmann, Joost Wagenaar, Drew Abbot, Phillip Adkins, Simone C. Bosshard, Min Chen, Quang M. Tieng, Jialune He, F. J. Muñoz-Almaraz, Paloma Botella-Rocamora, Juan Pardo, Francisco Zamora-Martinez, Michael Hills, Wei Wu, Iryna Korshunova, Will Cukierski, Charles Vite, Edward E. Patterson, Brian Litt, and Gregory A. Worrell. Crowdsourcing reproducible seizure forecasting in human and canine epilepsy. Brain, 139(6):1713–1722, March 2016.

[69] David Sussillo, Paul Nuyujukian, Joline M Fan, Jonathan C Kao, Sergey D Stavisky, Stephen Ryu, and Krishna Shenoy. A recurrent neural network for closed-loop intracortical brain–machine interface decoders. Journal of Neural Engineering, 9(2):026027, mar 2012.

[70] David Sussillo, Sergey D. Stavisky, Jonathan C. Kao, Stephen I. Ryu, and Krishna V. Shenoy. Making brain–machine interfaces robust to future neural variability. Nature Communications, 7(1):13749, December 2016.

[71] Venkatesh Elango. Sequence Learning for Brain Computer Interfaces - ProQuest.

[72] Pouya Bashivan, Irina Rish, Mohammed Yeasin, and Noel Codella. Learning representations from eeg with deep recurrent-convolutional neural networks, 2016.

[73] Saul Kato, Harris S. Kaplan, Tina Schrödel, Susanne Skora, Theodore H. Lindsay, Eviatar Yemini, Shawn Lockery, and Manuel Zimmer. Global Brain Dynamics Embed the Motor Command Sequence of Caenorhabditis elegans. Cell, 163(3):656–669, October 2015.

[74] Philipp Weidel, Renato Duarte, and Abigail Morrison. Unsupervised Learning and Clustered Connectivity Enhance Reinforcement Learning in Spiking Neural Networks. Frontiers in Computational Neuroscience, 15, March 2021. Publisher: Frontiers.

[75] Marius Pachitariu, Carsen Stringer, and Kenneth D. Harris. Robustness of spike deconvolution for neuronal calcium imaging. Journal of Neuroscience, 38(37):7976–7985, 2018.

[76] Peter Rupprecht, Stefano Carta, Adrian Hoffmann, Mayumi Echizen, Antonin Blot, Alex C. Kwan, Yang Dan, Sonja B. Hofer, Kazuo Kitamura, Fritjof Helmchen, and Rainer W. Friedrich. A database and deep learning toolbox for noise-optimized, generalized spike inference from calcium imaging. Nature Neuroscience, 24(9):1324–1337, September 2021.

[77] Henry Lütcke, Felipe Gerhard, Friedemann Zenke, Wulfram Gerstner, and Fritjof Helmchen. Inference of neuronal network spike dynamics and topology from calcium imaging data. Frontiers in Neural Circuits, 7, December 2013. Publisher: Frontiers.

[78] Thomas Deneux, Attila Kaszas, Gergely Szalay, Gergely Katona, Tamás Lakner, Amiram Grinvald, Balázs Rózsa, and Ivo Vanzetta. Accurate spike estimation from noisy calcium signals for ultrafast three-dimensional imaging of large neuronal populations in vivo. Nature Communications, 7(1):12190, July 2016.

[79] Daniel A. Cantu, Bo Wang, Michael W. Gongwer, Cynthia X. He, Anubhuti Goel, Anand Suresh, Nazim Kourdougli, Erica D. Arroyo, William Zeiger, and Carlos Portera-Cailliau. EZcalcium: Open-Source Toolbox for Analysis of Calcium Imaging Data. Frontiers in Neural Circuits, 14, May 2020. Publisher: Frontiers.

[80] Zhou Zhi, Kuo Hsien-Chi, Peng Hanchuan, and Long Fuhui. DeepNeuron: an open deep learning toolbox for neuron tracing - PMC, 2018.

[81] Sebastián A. Romano, Verónica Pérez-Schuster, Adrien Jouary, Jonathan Boulanger-Weill, Alessia Candeo, Thomas Pietri, and Germán Sumbre. An integrated calcium imaging processing toolbox for the analysis of neuronal population dynamics. PLOS Computational Biology, 13(6):1–23, 06 2017.

[82] Mikhail A. Lebedev and Miguel A.L. Nicolelis. Brain–machine interfaces: past, present and future. Trends in Neurosciences, 29(9):536–546, September 2006. Publisher: Elsevier.

[83] Daniel S. Marcus, Timothy R. Olsen, Mohana Ramaratnam, and Randy L. Buckner. The extensible neuroimaging archive toolkit. Neuroinformatics, 5(1):11–33, March 2007.

[84] Yannick Schwartz, Alexis Barbot, Benjamin Thyreau, Vincent Frouin, Gael Varoquaux, Aditya Siram, Daniel Marcus, and Jean-Baptiste Poline. PyXNAT: XNAT in Python. Frontiers in Neuroinformatics, 6, May 2012. Publisher: Frontiers.

[85] Paul A. Harris, Robert Taylor, Robert Thielke, Jonathon Payne, Nathaniel Gonzalez, and Jose G. Conde. Research electronic data capture (REDCap)—A metadata-driven methodology and workflow process for providing translational research informatics support. Journal of Biomedical Informatics, 42(2):377–381, April 2009.

[86] Tsai-Wen Chen, Trevor J. Wardill, Yi Sun, Stefan R. Pulver, Sabine L. Renninger, Amy Baohan, Eric R. Schreiter, Rex A. Kerr, Michael B. Orger, Vivek Jayaraman, Loren L. Looger, Karel Svoboda, and Douglas S. Kim. Ultrasensitive fluorescent proteins for imaging neuronal activity. Nature, 499(7458):295–300, July 2013.

[87] Rufus Mitchell-Heggs, Seigfred Prado, Giuseppe P. Gava, Mary Ann Go, and Simon R. Schultz. Neural manifold analysis of brain circuit dynamics in health and disease. Journal of Computational Neuroscience, 51(1):1–21, February 2023.

[88] Mohsen Bahrami, Robert G. Lyday, Ramon Casanova, Jonathan H. Burdette, Sean L. Simpson, and Paul J. Laurienti. Using Low-Dimensional Manifolds to Map Relationships Between Dynamic Brain Networks. Frontiers in Human Neuroscience, 13, 2019.

[89] Mary Ann Go, Jake Rogers, Giuseppe P. Gava, Catherine E. Davey, Seigfred Prado, Yu Liu, and Simon R. Schultz. Place Cells in Head-Fixed Mice Navigating a Floating Real-World Environment. Frontiers in Cellular Neuroscience, 15, 2021.

[90] A. N. Gorban and I. Y. Tyukin. Blessing of dimensionality: mathematical foundations of the statistical physics of data. Philosophical transactions. Series A, Mathematical, physical, and engineering sciences, 376(2118):20170237, April 2018.

[91] Daniel R. Strebe. Mercator projection. Wikipedia, 2011.

[92] Gaël Varoquaux, Yannick Schwartz, Russell A. Poldrack, Baptiste Gauthier, Danilo Bzdok, Jean-Baptiste Poline, and Bertrand Thirion. Atlases of cognition with large-scale human brain mapping. PLOS Computational Biology, 14(11):e1006565, November 2018. Publisher: Public Library of Science.

[93] Russell A. Poldrack and Tal Yarkoni. From brain maps to cognitive ontologies: Informatics and the search for mental structure. Annual Review of Psychology, 67(Volume 67, 2016):587–612, 2016.

[94] Guo Jiahui, Ma Feilong, Samuel A Nastase, James V Haxby, and M Ida Gobbini. Cross-movie prediction of individualized functional topography. eLife, 12:e86037, nov 2023.

[95] Keith B. G. Dear. SurfStat.australia: a Statistics Textbook for the Web. International Journal of Innovation in Science and Mathematics Education, 3(1), 1999.

[96] Moritz Grosse-Wentrup, Akshey Kumar, Anja Meunier, and Manuel Zimmer. Neuro-cognitive multilevel causal modeling: A framework that bridges the explanatory gap between neuronal activity and cognition. PLOS Computational Biology, 20(12):e1012674, December 2024. Publisher: Public Library of Science.

[97] David Danks and Isaac Davis. Causal inference in cognitive neuroscience. WIREs Cognitive Science, 14(5):e1650, 2023.

[98] Robert A. Jarrow, David Lando, and Stuart M. Turnbull. A Markov Model for the Term Structure of Credit Risk Spreads. The Review of Financial Studies, 10(2):481–523, April 1997.

[99] Krogh A. Sonnhammer E. L., von Heijne G. A hidden markov model for predicting transmembrane helices in protein sequences. Proc Int Conf Intell Syst Mol Biol, 1998.

[100] Marc A. Suchard, Robert E. Weiss, and Janet S. Sinsheimer. Bayesian Selection of Continuous-Time Markov Chain Evolutionary Models. Molecular Biology and Evolution, 18(6):1001–1013, June 2001.

[101] Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, pages 252–259, 2003.

[102] Alexandru Hening and Yao Li. Stationary distributions of persistent ecological systems. Journal of Mathematical Biology, 82(7):64, May 2021.

[103] Bapi Surampudi, Krishna Miyapuram, and V. S. Chandrasekhar Pammi. A multi-disciplinary approach to the investigation of aspects of serial order in cognition. 2002. Paper Presented at the National Seminar on Cognitive Science.

[104] Akshey Kumar, Aditya Gilra, Mauricio Gonzalez-Soto, Anja Meunier, and Moritz Grosse-Wentrup. BunDLe-Net: Neuronal Manifold Learning Meets Behaviour, August 2023.

[105] Betony Adams and Francesco Petruccione. Quantum effects in the brain: A review, October 2019. arXiv:1910.08423 [quant-ph, q-bio].

[106] Paul W. Glimcher. Indeterminacy in Brain and Behavior. Annual Review of Psychology, 56(1):25–56, February 2005.

[107] Robert Maurice Sapolsky. Determined: A Science of Life Without Free Will. Penguin Press, New York, 2023.

[108] George F. R. Ellis. Physics, Determinism, and the Brain, September 2020. arXiv:2008.12674 [physics].

[109] Burrhus Frederic Skinner. The Behavior of Organisms: An Experimental Analysis. B. F. Skinner Foundation, Cambridge, 1938.

[110] Yunzhe Liu, Raymond J. Dolan, Zeb Kurth-Nelson, and Timothy E. J. Behrens. Human Replay Spontaneously Reorganizes Experience. Cell, 178(3):640–652.e14, July 2019. Publisher: Elsevier.

[111] Cameron Higgins, Yunzhe Liu, Diego Vidaurre, Zeb Kurth-Nelson, Ray Dolan, Timothy Behrens, and Mark Woolrich. Replay bursts in humans coincide with activation of the default mode and parietal alpha networks. Neuron, 109(5):882–893.e7, March 2021. Publisher: Elsevier.

[112] Vincent Castellucci, Harold Pinsker, Irving Kupfermann, and Eric R. Kandel. Neuronal Mechanisms of Habituation and Dishabituation of the Gill-Withdrawal Reflex in Aplysia. Science, 167(3926):1745–1748, March 1970. Publisher: American Association for the Advancement of Science.

[113] Yaniv Sagi, Ido Tavor, Shir Hofstetter, Shimrit Tzur-Moryosef, Tamar Blumenfeld-Katzir, and Yaniv Assaf. Learning in the Fast Lane: New Insights into Neuroplasticity. Neuron, 73(6):1195–1203, March 2012.

[114] Yingjie Du, Yunfeng Li, Xiangting Zhao, Yishan Yao, Bin Wang, Liming Zhang, Guyan Wang, and Jing Ni. Psilocybin facilitates fear extinction in mice by promoting hippocampal neuroplasticity. Chinese Medical Journal, 136(24):2983–2992, 2023.

[115] Jared Kopelman, Timothy A. Keller, Benjamin Panny, Angela Griffo, Michelle Degutis, Crystal Spotts, Nicolas Cruz, Elizabeth Bell, Kevin Do-Nguyen, Meredith L. Wallace, Sanjay J. Mathew, Robert H. Howland, and Rebecca B. Price. Rapid neuroplasticity changes and response to intravenous ketamine: a randomized controlled trial in treatment-resistant depression. Translational Psychiatry, 13(1):159, May 2023.

[116] Python Software Foundation. 3. Data model. https://docs.python.org/3/reference/datamodel.html.

[117] Dominique Makowski, Tam Pham, Zen J. Lau, Jan C. Brammer, François Lespinasse, Hung Pham, Christopher Schölzel, and S. H. Annabel Chen. NeuroKit2: A Python toolbox for neurophysiological signal processing. Behavior Research Methods, 53(4):1689–1696, August 2021.

[118] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. Nature, 585(7825):357–362, September 2020. Publisher: Nature Publishing Group.

[119] Stefan van der Walt, S. Chris Colbert, and Gael Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation. Computing in Science & Engineering, 13(2):22–30, March 2011. Conference Name: Computing in Science & Engineering.

[120] John D. Hunter. Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering, 9(03):90–95, May 2007. Publisher: IEEE Computer Society.

[121] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, and David Cournapeau. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2011.

[122] Aric Hagberg, Pieter J. Swart, and Daniel A. Schult. Exploring network structure, dynamics, and function using NetworkX. Technical Report LA-UR-08-05495; LA-UR-08-5495, Los Alamos National Laboratory (LANL), Los Alamos, NM (United States), January 2008.

[123] Giancarlo Perrone, Jose Unpingco, and Haw minn Lu. Network visualizations with pyvis and visjs, 2020.

[124] Martín Abadi. TensorFlow: learning functions at scale. In Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, page 1, New York, NY, USA, September 2016. Association for Computing Machinery.

[125] Pei-Lee Teh Hongyi Sun, Waileung Ha and Jianglin Huang. A case study on implementing modularity in software development. Journal of Computer Information Systems, 57:130–138, 2017.

[126] Kevin J. Sullivan, William G. Griswold, Yuanfang Cai, and Ben Hallen. The structure and value of modularity in software design. SIGSOFT Softw. Eng. Notes, 26(5):99–108, sep 2001.

[127] Jacob Cohen. Statistical power analysis for the behavioral sciences. routledge, 2013.

[128] Joachim Hartung, Guido Knapp, and Bimal K Sinha. Statistical meta-analysis with applications. John Wiley & Sons, 2011.

[129] Johannes Friedrich, Pengcheng Zhou, and Liam Paninski. Fast online deconvolution of calcium imaging data. PLOS Computational Biology, 13(3):e1005423, March 2017. Publisher: Public Library of Science.

[130] Frederic Bartumeus, M G E da Luz, Gandhimohan M Viswanathan, and Jordi Catalan. Animal search strategies: a quantitative random-walk analysis. Ecology, 86(11):3078–3087, 2005.

[131] Kris S Chaisanguanthum, Helen H Shen, and Philip N Sabes. Motor variability arises from a slow random walk in neural state. Journal of Neuroscience, 34(36):12071–12080, 2014.

[132] Richard A Heath. A general nonstationary diffusion model for two-choice decision-making. Mathematical Social Sciences, 23(3):283–309, 1992.

[133] Richard A Heath. The ornstein-uhlenbeck model for decision time in cognitive tasks: An example of control of nonlinear network dynamics. Psychological Research, 63(2):183–191, 2000.

[134] Aidan Lyon. Why are normal distributions normal? The British Journal for the Philosophy of Science, 2014.

[135] Joseph L Doob. The brownian movement and stochastic equations. Annals of Mathematics, 43(2):351–369, 1942.

[136] Jesse M. Gray, Joseph J. Hill, and Cornelia I. Bargmann. A circuit for navigation in Caenorhabditis elegans. Proceedings of the National Academy of Sciences, 102(9):3184–3191, March 2005. Publisher: Proceedings of the National Academy of Sciences.

[137] Jonathan T. Pierce-Shimomura, Thomas M. Morse, and Shawn R. Lockery. The Fundamental Role of Pirouettes in *Caenorhabditis elegans* Chemotaxis. The Journal of Neuroscience, 19(21):9557–9569, November 1999.

[138] Aman B. Saleem, E. Mika Diamanti, Julien Fournier, Kenneth D. Harris, and Matteo Carandini. Coherent encoding of subjective spatial position in visual cortex and hippocampus. Nature, 562(7725):124–127, October 2018.

[139] Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. Genome research, 13(11):2498–2504, 2003.

[140] Mario Köppen. The curse of dimensionality. In 5th online world conference on soft computing in industrial applications (WSC5), volume 1, pages 4–8, 2000.

[141] Robert K. Herman. Chapter 6 mosaic analysis. In Henry F. Epstein and Diane C. Shakes, editors, Cuenorhubditis elegans: Modern Biologcal Analysis of an Organism, volume 48 of Methods in Cell Biology, pages 123–146. Academic Press, 1995.