National University of Colombia

Physics Department

Topic:

Cellular Automata Modeling of Neuronal Networks

A study of multiple network topologies on a Brain activity patterns simulation inspired by action potential and neural connectograms

Submitted by:

Maria Fernanda Piracoca Mora

ID: 1032499832

mfpiracocam@unal.edu.co

Advisor: José Daniel Muñoz Castaño

Bogotá D.C., June 26, 2023

Study hard what interests you the most in the most undisciplined, irreverent and original manner possible.

Richard Feynmann
"As long as our brain is a mystery, the universe, the reflection of the structure of the brain will also be a mystery.".

Santiago Ramón y Cajal

Acknowledgments

I would like to express my deepest gratitude to my advisor, José Daniel Muñoz, for his unwavering support, patience, and guidance throughout this work. His exceptional knowledge, understanding, and constant encouragement have been instrumental in shaping my growth as a physicist. I am truly grateful for the opportunity to learn and create under his mentorship, and he continues to inspire me in my journey towards becoming the scientist I aspire to be.

I am also indebted to my parents for their relentless support and belief in me. Their constant encouragement and belief in my abilities have been a driving force behind my accomplishments. I would also like to extend my appreciation to my kitten, Frida, whose playful presence and unconditional love have brought joy and comfort during challenging times.

I am immensely grateful to my friends, including Geiye Velasquez, Nicolás Claro, and numerous other remarkable individuals I have had the pleasure of meeting throughout my academic journey. Their constant support, uplifting spirits, and timely pushes have been invaluable in helping me overcome obstacles and stay motivated.

Special thanks go to Diego Veloza, whose companionship during sleepless nights and shared knowledge have been vital in pushing me forward. Your collaboration and dedication have been truly invaluable.

To all those who have contributed to my journey, whether mentioned here or not, please accept my heartfelt appreciation for your support, encouragement, and belief in me. Your contributions have played an integral role in my personal and academic growth, and I am grateful for the privilege of having you by my side.

Abstract

The human brain and nervous system exhibit a complex structure and intricate functionality, where individual neuron function and network connectivity play fundamental roles, and computational cellular automata models are valuable tools for understanding these interactions and mechanisms. The present work aims to implement a recent model by Ali Khaleghi et. al. [1] to investigate the effect of the network geometry on the synthesized ECG signal. For this purpose, the model has implemented on three network geometries: random, small-world and local, but non uniform. Our results show that the three geometries are able to generate activity signals with spectral contents, Recurrence Qualification analysis (RQA), fractal measurements and multiscale entropies similar to those of real ECGs, with the small world one showing the best results, highlighting the importance of combining short and long range interactions. Our results suggest that increasing the fraction of inhibitory cells reduces the high frequency contents of the signal and the intensity of long range correlations in the Recurrence Qualification maps, but still close to the experimental values. The work constitutes a valuable contribution to the understanding of the neural networks' behavior.

Keywords: Cellular automata (CA), action potential, Small world network, Watts-Strogatz Model, Electroencephalogram (EEG))

Resumen

El cerebro humano y el sistema nervioso en general exhiben una estructura compleja y una funcionalidad intrincada, en las cuales el funcionamiento individual de cada neurona y la conectividad de la red juegan papeles esenciales. Por su parte, los modelos computacionales de autómata celular son herramientas valiosas para comprender esas interacciones y mecanismos. este trabajo busca implementar un modelo reciente propuesto por Ali Khaleghi et. al. [1] para investigar el efecto de la geometría de la red sobre la señal ECG que se sintetiza. Con este fin, el modelo se ha implementado sobre tres geometrías de red: aleatoria, small-world y local, pero no uniforme. Nuestros resultados muestran que las tres geometrías son capaces de generar señales de actividad con contenidos espectrales, mapas de recurrencia, dimensiones fractales y entropía multiescala similares a las de las señales experimentales, siendo las redes Small-World las que presentan lso mejores resultados, lo que resalta la importancia de combinar interacciones de corto y largo alcance. Nuestros resultados sugieren que incrementar la fracción de células inhibitorias reduce los contenidos de altas frecuencias y la intensidad de las correlaciones temporales de largo alcance, aunque aún cerca de los rangos de los valores experimentales. El trabajo constituye una contribución valiosa para la comprensión del comportamiento de las redes de neuronas.

Palabras clave: Automatas celulares, potencial de acción, redes de pequeño mundo, Modelo de Watts- Strogatz , Electreoenefalograma.

Contents

List of Figures

List of Tables

Listings

Abbreviations

CA Cellular automata

EEG Electroencephalogram

PSD Power spectral density

RP Recurrence plot

RQA Recurrence quantification analysis

1 Introduction

The human brain and nervous system possess an inherent allure as a captivating biological system, primarily attributed to their highly complex structure and intricate functionality [1]. Comprised of billions of interconnected neurons and elaborate networks, the brain exhibits an extraordinary level of organization and sophistication. It derives its properties from the intricate interconnections formed among 100 billions of neurons within elaborate networks [4]. Brain activity is encoded through activity patterns of action potentials within extensive neural networks [5]. Action potentials are the primary means of information transmission between neurons. Those electrical impulses exhibit great consistency in terms of their shape and duration; therefore, the encoding of information relies on the temporal sequence of action potential generation rather than variations in their individual characteristics [6].

The initial pursuit of understanding the brain involved conducting purely experimental investigations that generated vast quantities of highly specialized and disconnected data. To achieve analytical tractability and integrate the knowledge gained from this experimental phase, simple mathematical models and new concepts were developed. Those models and concepts aimed to simplify experimental observations and apply simplified versions of methods from physics and mathematics; thereby facilitating a deeper understanding of the brain [7]. Hodgkin and Huxley pioneered the development of a digital simulation that modeled a fundamental physiological property of neurons. Their work involved integrating their experimental findings on the flow of electric current across the surface membrane of an axon with a mathematical representation inspired by a 19th-century model used for signaling through submarine telegraph cables [8]. Theoretical approaches offer a broader perspective by providing a comprehensive understanding of model behavior across all possible parameter settings. Nevertheless, the availability of analytical solutions is limited to relatively simple models. Both experimental and theoretical approaches face challenges posed by the vast scale and complexity of the brain, which can hinder their progress in uncovering its intricacies [7]. Consequently, the field of neuroscience has become fragmented, with different research laboratories focusing on diverse aspects such as genes and molecules, single-cell electrophysiology, multineuron recordings, cognitive neuroscience, and psychophysics, among others [8]

One of the primary objectives in computational neuroscience is to connect and reconcile the distinct levels of description through the use of simulation methods and mathematical principles, starting from small-scale phenomena, such as studying the properties of ion channels, and extrapolating their findings to understand larger-scale behaviors or beginning with complex processes, such as studying working memory, and dissecting its components to unravel the underlying mechanisms [8]. By employing computational simulations and developing mathematical frameworks, computational neuroscience plays a pivotal role in integrating experimental findings and theoretical concepts. It also play a crucial role in advancing, confirming, and validating hypotheses [1], pursuing our understanding of the brain's complex dynamics. The importance of computational neuroscience is best represented by this quote from *A Brief History of Simulation Neuroscience* [7]: "The deep meaning of simulation neuroscience consists in reconstructing and simulating the brain from the most fundamental principles we can isolate to understand and link the multiple layers that form ourselves, from molecules and cells to brain function and behavior, to give meaning and life to data and theories."

Computational models used in the field of neuroscience can be classified based on the scales of the neurophysiological activity they encompass. Microscopic models operate at the level of single cells and micro-circuits, providing detailed simulations of individual neurons and their interactions within small networks. Mesoscopic models focus on neural masses and neural fields, aiming to understand the collective behavior of groups of neurons or brain regions. These models describe the average activity and dynamics of populations of neurons, using statistical methods and field equations to represent spatiotemporal patterns of neural activity. Lastly, macroscopic models consider the broader scale by incorporating the connectome and white matter connectivity of the brain. These models study large-scale brain dynamics and network-level phenomena, using graph theory and network analysis techniques to explore information flow and functional connectivity [9].

Connectograms, which are circular arrays of neurons, provide a visual representation of the connections and other characteristics of the studied system. These connectograms offer insights into the presence of long-distance connections in addition to the expected local connections within a regular structure. In fact, some studies have reported the observation of Small-World networks in neuronal structures. Initially, the term "connectome" referred to a comprehensive description of the structural network of brain elements and their connections. It encompassed the entire set of neuronal connections in the brain. However, with the growing understanding that neural connections are closely associated with brain functions, the concept of the connectome has evolved. It now includes not only the structural aspects but also the functional relationships between different types of neurons and their connections with other cells within a specific neural region or the entire brain [7].

EEG is a non invasive common technique to get a recording of the neural firing [10]. EEG is one of the most complex sets of biomedical signals. It does not directly capture the action potentials of individual neurons. Instead, EEG records the electrical activity of the brain through electrodes placed on the scalp, providing a macroscopic view of neural dynamics. Nevertheless, certain observations suggest a potential association between high-frequency oscillations detected in EEG signals and the spiking activity of neurons[9]. Its analysis can be approached from both qualitative and quantitative perspectives. Whereas qualitative observations provides valuable insights into the dynamics of the EEG, quantitative analysis involves employing a range of statistical methods to uncover specific characteristics and features within the signal. Those quantitative methods allow the identification and interpretation of intricate patterns, statistical properties, and other relevant parameters that contribute to a comprehensive understanding of the signal [11].

Spectral analysis is the basic tool for studying an ECG signal, and dividing the signal in frequency bands (delta, theta, alpha, sigma, beta, gamma) is one of the primary tasks when studying an ECG. Nevertheless, Spectral analysis can be susceptible to imprecision, inconsistent findings have been observed among studies utilizing this method. These inconsistencies stem from variations in experimental factors, including the placement of reference and recording electrodes, preprocessing approaches, and methodological parameters such as epoch length, windowing functions, and frequency resolution. It is worth noting that many studies often neglect to report these parameters, leaving new researchers in the field to speculate or rely on default values provided by analysis software, without substantial background knowledge or concrete guidelines. [11]. Thus, in addi-

tion to spectral analysis, various other conventional techniques for Electroencephalogram (EEG) analysis, like recurrence quantification (RQA) fractal and entropy analysis are also examined.

Cellular automata (CA) offer a first playground for studying how the connectogram determines the behavior of a set of networks. in a cellular automaton, each cell is connected to neighboring cells - often enriched by long-distance connections -, and every cell determines its next state from the states of the cells connected to it by following simple evolution rules. All cells evolve in parallel, simulating the global behavior of the system. One of the early explorations of cellular automata in neural simulations can be traced back to 1970s. In 1970, Mortimer Arthur introduced a CA model for simulating a mammalian cerebellar cortex [12]. He studied which parameters (synaptic time constants, spatial distributions of axonal plexuses, relative strengths of synaptic connections) are most important in determining the form and time course of the cortical response providing a transition and output functions. Though rudimentary, this work laid the foundation for employing cellular automata for EEG simulations. Building upon earlier research, John J. Wright, in 1999, published a more comprehensive study, entitled "Simulation of EEG: dynamic changes in synaptic efficacy, cerebral rhythms, and dissipative and generative activity in cortex" [13]. Here, Wright employed a cellular automaton as a network of discrete elements with the aim of generating EEG-like patterns. His model displayed cortical interactions and proved to be instrumental in advancing the use of CA in EEG simulations. In 2009, L. Acedo published a paper titled "A cellular automaton model for collective neural dynamics" [14] This research work was significant, as it used cellular automata to model EEG signals with a focus on investigating simulated EEG signals' properties, making it a significant contribution to understanding brain wave patterns.

The study of CAs for EEG simulations saw a more refined approach in 2016, when Robert Kozma and Walter J Freeman published "Cognitive phase transitions in the cerebral cortex-enhancing the neuron doctrine by modeling neural fields" [15] This study used a conductance-based CA model to generate EEG-like patterns and understand oscillations in the human brain. Those and similar studies have collectively contributed to the growing interest and development of cellular automata-based models for simulating electroencephalograms. As the field continues to evolve, one can anticipate further advancements and refinements in CA-based EEG simulation techniques. In the article entitled "A neuronal population model based on cellular automata to simulate the electrical waves of the brain" by A. Khaleghi et al. [1], a novel Cellular Automaton (CA) model was introduced. This model captures the interconnectedness of a neural population and incorporates an evolution rule inspired by neural action potentials. By generating time series of neural activity and recurrence plots, the proposed CA model demonstrates comparable results to those observed in real EEG data.

The general objective of this work is to use the rules described in the article by Khalegui to build the CA and explore how a change in the topology of the network (random, small-world or local) affects the behavior of the resulting time series. We aim to compare the results with real data of an EEG, determine the changes of the performance of the algorithm when the topology changes and analyze which are the key components to make the network to work properly. In our study, a one dimensional periodic array is connected as a random network, a small-world network using the Watts-Strogatz algorithm and a network with local connections of variable size to each cell are built and analyzed. Parameters in both the construction of the network and the rules of evolution

are varied as to obtain the general original behavior and then each of this networks is compared to a set of real EEG data by using Spectral analysis, recurrence quantification recurrence quantification (RQA) fractal and entropy analysis (RQA) and fractal and entropy measurements. In the subsequent chapters, this work provides a comprehensive overview of Cellular Automata CA in the context of EEG simulations (Section 2). The foundational CA model and the accompanying methods for analysis are extensively elucidated (Section 3). The subsequent chapter delves into the step-by-step implementation of the networks and presents the corresponding code (Section 4). The obtained results are presented in detail (Section 5), followed by a discussion of these results, future research directions, and potential conclusions (Section 6). In this way, we expected to achieve the double purpose of learning both from the analysis of complex networks and from the study of networks of brain neurons, and contribute with a work original to the subject.

2 Modeling neural activity with celullar automata

Several studies have been accomplished to simulate neural activity using cellular automata. Hereby we present some of the most relevant ones.

2.1 Simulation of Healthy and Epileptiform Brain Activity Using Cellular Automata

According to the study conducted by Tsoutsouras et. al. titled "Simulation of Healthy and Epileptiform Brain Activity Using Cellular Automata" [2] the main objective was to model the mechanisms that lead to the loss of neurons in limbic brain regions, including the hippocampus, resulting in epileptic disorders. The study also aimed to explore how this disorder can spread to healthy neuronal networks without any apparent pathology, leading to generalized epilepsy. The CA model presented in this study allows for faithful simulation of brain activity during the transition from health to epilepsy.

To execute the simulation, the following steps were taken. Firstly, the number of cells, denoted as N, was determined to create two-dimensional grids of neuronal cells with dimensions $N \times N$. The initial conditions of the neuronal cell networks were then established using a cellular automaton CA-based pseudorandom generator. Next, a period of operation in a healthy state was defined, during which each neuron-cell could have an active number of synapses ranging from No to Na, with $Na \leq N$. Additionally, the duration of neuronal loss in one of the two neuronal networks was specified. During this period, epileptic neuron-cells were allowed to have an active number of synapses ranging from No to Nb, with $Nb \leq Na \leq N$. A set of CA rules was established for each neuron-cell as follows:

- **Rule 1:** Neuron-cells can exist in state '1' during time t_{on} and in state '0' during time t_{off}, with $t_{\text{on}} < t_{\text{off}}$. For successive activations in a normal situation, the time toff should be at least equal to the time toff of the previous activation.
- **Rule 2:** If the activation value surpasses a set threshold and t > toff, the neuron-cell is activated individually.
- **Rule 3:** If the sum of inputs received by neurons exceeds a threshold value Vth and t > toff, the neuron-cell is activated.
- **Rule 4:** Neuron-cells mostly return to state '0' during time ton.
- **Rule 5:** If the sum of entries exceeds a value Vmax greater than Vth, the neuron-cell transitions to state '1,' or remains in its current state if the timing of activation is greater or less than time toff.

Through the analysis of the CA simulated data and the real EEG data, the efficiency of the CA algorithm in simulating brain activity during both healthy and seizure states was evident. The CA modeling in this study revealed the nonlinear nature of distributed brain dynamics, with high-dimensional stochastic behavior observed during the healthy state and low-dimensional chaotic behavior during the epilepsy state. These findings suggest

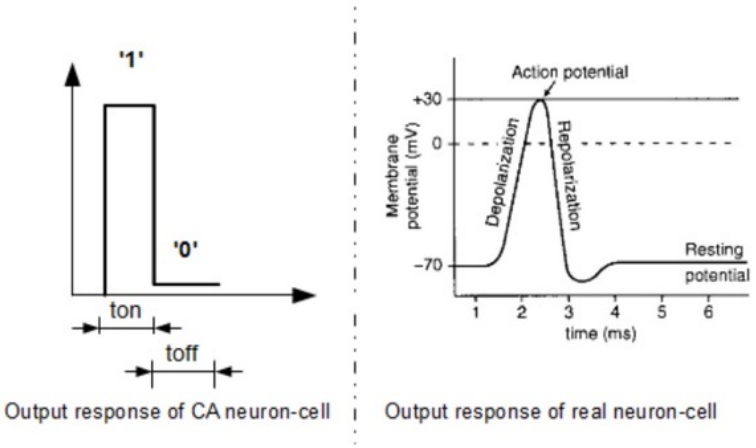


Figure 1: Output response of CA cell to the left and the output response of a real neuron to the right. [2].

the presence of spatio-temporal chaotic dynamics in the brain, while the development of epilepsy can be classified as a non-equilibrium phase transition process, commonly observed in nonlinear dynamical and spatially distributed systems.

The dynamics of the brain exhibits critical states or fixed points characterized by high dimensionality (SOC - self-organzed criticality - states) during healthy states and low-dimensional chaotic states during seizure states. According to the general theory of nonlinear distributed dynamics, critical states with high-dimensional characteristics correspond to second-order far-from-equilibrium phase transitions, whereas low-dimensional chaotic behavior aligns with first-order non-equilibrium phase transition states. The development of seizures in brain activity involves stochastic synchronization of distributed dynamics driven by a noise component, leading the system from one fixed point to another and causing a global phase transition.

The presented results open the possibility of including brain activity in the universality class of directed percolation and anomalous diffusion phenomena, as suggested by previous studies.

2.2 Firing patterns in a random network cellular automata model of the brain

In their work entitled "Firing patterns in a random network cellular automata model of the brain," Acedo et. al. [16] address the challenge of simulating the behavior of complex brain networks due to the large number of neurons and connections involved. They present a cellular automata network model of the brain that incorporates excitatory and inhibitory neurons and aims to classify normal and abnormal patterns of activity, along with their stability and parameter ranges. The simulation of this model is performed using a distributed computing environment based on the BOINC platform, enabling the study of a network comprising up to one million sites with an average of three hundred connections per neuron.

The algorithm for simulating the Cellular Automata Random Network model involves sev-

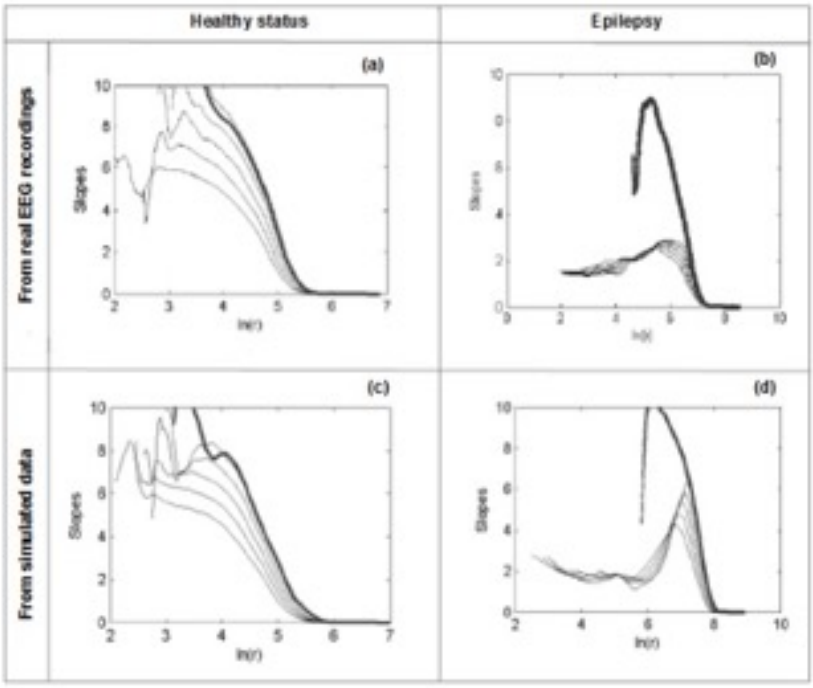


Figure 2: Output response of CA cell to the left and the output response of a real neuron to the right. [2].

eral steps. First, a random network with N sites and an average degree k is generated by randomly assigning $Nk/2$ bonds to pairs of sites. The network properties are statistically consistent across different realizations. A fraction of the sites is designated as inhibitory neurons, while the rest are labeled as excitatory ones. The network is initialized with a subset of neurons in the firing state, while the remaining neurons are in the resting state.

During each time step, the resting neurons in the network are examined. If their neighboring neurons are excitatory and firing, the resting neuron may transit to the firing state with some probability. Similarly, if a neighboring neuron is inhibitory and firing, the state of the resting neuron may change back to the resting state with another probability if it has been stimulated by an excitatory firing neuron in its neighborhood during that step. Firing neurons can become refractory with some defined probability per step, and the presence of inhibitory firing neurons in their neighborhood may increase the probability of transitioning to the refractory state. Refractory neurons can change back to the resting state with some probability per time step. The activity of the network is quantified by counting the number of excitatory and inhibitory firing neurons at each time step.

In their simulations, Acedo et al. used a stochastic neural network of one million neurons. Through the distributed computing system, they conducted a substantial number of tests, aiming to identify collective behaviors in the network that correspond to abnormal electroencephalogram (EEG) patterns observed in epileptic seizures. The simulations successfully generated hypersynchronous oscillations similar to those found in the EEG of epileptic patients during the ictal state. The complexity level of the network, with $N = 106$ and $k = 300$, was found to be the minimum requirement for the emergence of self-sustained oscillatory patterns. This level of complexity is comparable to that observed in insects and suggests a potential explanation for their success and diverse behavioral repertoire.

To enhance the realism of the model, neurons were classified as either excitatory or inhibitory, each with specific probabilities of excitation and inhibition. The study highlights the rarity of self-sustained periodic oscillations of large amplitude in the brain, as they require specific parameter combinations. Controlling those parameters could contribute to preventing abnormal excitatory activity patterns. By comparing real EEG signals with the model-generated signals, the excitation and inhibition parameters could potentially be used to classify different stages of epileptic seizures, including pre-ictal, ictal, and interictal stages. Additionally, the model can facilitate the investigation of the effects of antiepileptic drugs by simulating small changes in the parameters, aiding in the assessment of drug effectiveness.

The authors emphasize that the model's signals can be compared with real EEG data to study the statistical properties and analogies between the model and different stages of epileptic seizures, such as ictal and pre-ictal stages. Moreover, they mention that other mathematical and computational approaches are being explored in the field of epilepsy research, and the understanding and modeling of epilepsy mechanisms remain active areas of investigation.

The authors conclude by mentioning that their stochastic cellular automata random network model has exhibited both low-amplitude noisy patterns and oscillatory behaviors with high amplitudes, comparable to epileptiform activity in real brains. Further research can be conducted to explore the statistical properties of the model-predicted signals and their correspondence to different stages of epileptic seizures. Additionally, the model can be extended by replacing the cellular automata with more sophisticated neuron models, such as Hodgkin–Huxley or FitzHugh–Nagumo, to investigate the emergence of large-scale oscillations. The ongoing work aims to refine and expand the understanding of brain dynamics through computational modeling approaches.

2.3 A model of the electrocortical effects of general anesthesia

In their work entitled "A model of the electrocortical effects of general anesthesia," Sleigh and Galletly [17] investigate the impact of general anesthetic agents on the loss of consciousness by studying a two-dimensional cellular automaton (CA) computer model. The model consists of an 80×80 lattice representing cortical elements with varying connectivity to their neighboring elements. The behavior of the CA is examined as the connectivity between cellular elements is altered to simulate the reduction in synaptic efficiency induced by general anesthetics.

The authors observed that, when the cellular automaton elements have a high probability of connectivity, the simulated electroencephalogram (EEG) signal exhibits high-frequency predominance and low amplitudes, resembling the desynchronized pattern observed in an awake individual. As the connectivity is decreased, the median frequency of the simulated EEG decreases and the amplitude increases, resembling the patterns seen in anesthetized patients. Based on their model, the authors propose that the human central nervous system operates above a critical threshold of synaptic efficiency in the conscious resting state. Increased synaptic efficiency is associated with awareness, while anesthesia involves a sharp reduction in cortical information transfer due to a decreased synaptic efficiency.

To simulate the CA lattice, a computer program was developed using the C language.

Each unit in the lattice communicates with its four neighbors through a bond linkage probability (Pb), representing the probability of passing information or activating neighboring units. The output of the system is measured as the number of active units. The model used in this study is analogous to the well-known "forest fire" model used in various fields, such as describing activity in random resistor networks, polymer bonds, and epidemics.

The simulation involves the application of an excitatory depolarizing input signal to a random element on one side of the lattice at regular intervals of one time step. In each subsequent time step, neighboring elements become depolarized based on the predefined probability Pb. Depolarized sites then enter a refractory state for one time step. This process generates a wave of activity propagating through the lattice, with the time taken for the wave to traverse the lattice referred to as the percolation time. A short percolation time indicates efficient information transfer, while a long percolation time suggests poor information transfer. After reaching a steady state, the voltage within the lattice is recorded over time, and the activity is examined for various Pb values. In a separate simulation, the excitatory input signal is applied only when the preceding wave has dissipated or reached the opposite side of the lattice. The percolation times are calculated, and the efficiency of information transfer across the lattice is estimated using the percentage of percolation times below a specific threshold, termed the "percolation time index" (PTI).

The assumptions underlying the model's relevance to the effects of anesthesia on cortical function are as follows: (a) the cerebral cortex consists of functional units approximately 0.1 mm^3 in size, with each unit representing a neuronal subsystem in the lattice; (b) the EEG signal voltage results from the summation of postsynaptic potentials, which is reflected in the model as the total number of active or depolarized units in the lattice at a given time; (c) anesthesia primarily disrupts synaptic efficiency, which is simulated by reducing the probability (Pb) of each unit depolarizing its neighbor. By adjusting the Pb values, the hypothetical effect of a drug altering synaptic efficiency can be examined.

In summary, Sleigh and Galletly employ a cellular automaton model to investigate the electrocortical effects of general anesthesia. Through simulations with varying connectivity and synaptic efficiency, the authors observe changes in the simulated EEG signals, demonstrating how reduced synaptic efficiency can lead to the loss of consciousness. The model provides insights into the mechanisms underlying anesthesia-induced alterations in cortical function and information transfer.

2.4 A Density-Based Cellular Automaton Model for Studying the Clustering of Noninvasive Cells

In their study titled "A Density-Based Cellular Automaton Model for Studying the Clustering of Noninvasive Cells," Bonnet et. al. [3] explore the formation of clusters by noninvasive cells and investigate whether such clustering can be attributed to a global attractive potential. They compare indices quantifying the persistence of cell migration in experimental conditions with those computed from simulations using a density-based cellular automaton. The authors conclude that the hypothesis of an attractive potential must be rejected.

The simulation system used in the study consists of a two-dimensional (2-D) square grid,

either 500×500 or optionally 1000×1000 units in size. The grid is initialized with a few hundred cells (typically 200 cells) randomly seeded on the grid. A potential field is computed based on the local density of cells, using a kernel-based Parzen estimator. The density of cells at a position X is determined by considering the positions of neighboring cells Xn, following the formula

$$P(X) = \sum ker(\|X - X_n\|) \quad ,$$

where ker($\cdot$) represents a normalized kernel function. In this study, a Gaussian kernel is implemented with a standard deviation equal to one eighth of the grid size.

Through a comparative analysis of simulations and experiments, the authors distinguish between two types of models: a global attraction model, where all cells attract each other, and a no-global-attraction model, where cells do not exhibit mutual attraction. The former model involves cells moving in a potential field towards regions of higher potential, computed based on the local cell density. The latter model represents cells undergoing random walks. To evaluate the presence of an attraction/repulsion potential resulting from the local cell density, the authors introduce a new parameter.

Once the potential field is computed, cell migration occurs based on the assumed model. For the density-based attractive model, cells in the Moore neighborhood (eight neighbors) of each cell Xi are ranked according to their potential values. Cells are then allowed to move towards the direction corresponding to the highest potential value. Conversely, in a no-global-attraction model, cells are permitted to move in random directions.

The results of the study do not support a global (density-based) attraction model, indicating that universal attraction among cells is not a valid explanation for the observed formation of clusters in noninvasive cell lines. Instead, the authors suggest that the formation of clusters occurs through a random migration of cells until cells with agglomerative properties come into contact, initiating the formation of an aggregate. The existence and characteristics of these agglomerative properties, which are not universally shared by all cells in a population, require further experimental investigation.

In conclusion, Bonnet et al. investigate the clustering behavior of noninvasive cells using a density-based cellular automaton model. Their simulations provide evidence against a global attraction model based on the local cell density, highlighting the importance of considering alternative mechanisms for cluster formation. The study contributes to the understanding of cell migration dynamics and offers insights into the formation of cell aggregates in experimental settings.

This summary exemplarizes some of the research lines that have used Cellular Automaton models to simulate neural activity and organization. In the next chapter we will discuss the model by Khaleghi et. al., which will be the base of our work.

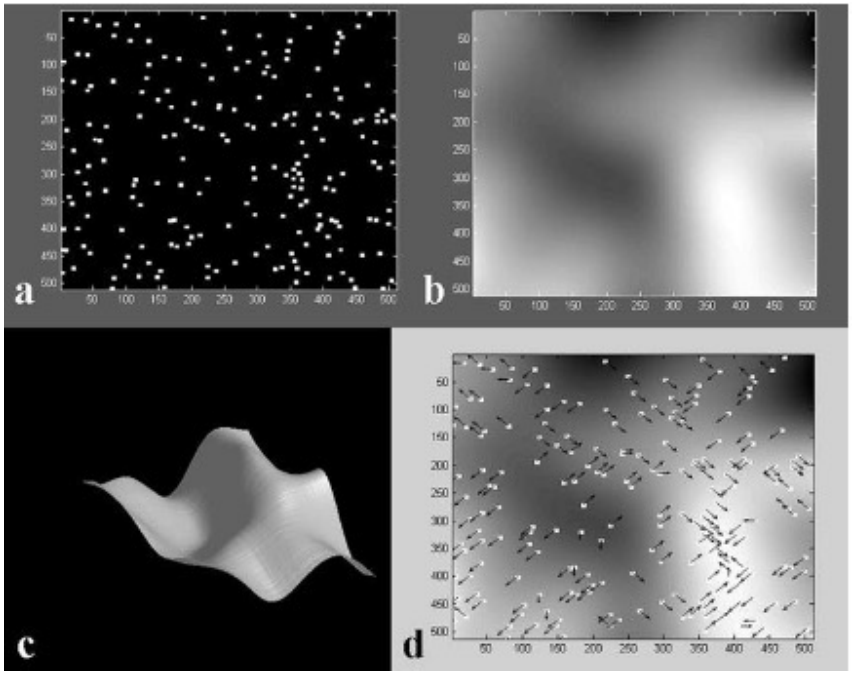


Figure 3: a) Simulation of a population of cells (hard-core model). (b) Estimation of the local density of cells. Note that despite the random seeding, the local density is not constant. (c) Three-dimensional representation of the estimated density. (d) Direction of migration of the different cells superimposed on the cell density. In this simulation, cells are assumed to move in the direction of highest cell density. [3].

3 Khaleghi, Ali, CA neuronal population model

The study conducted by Ali Khalegui et al [1] focuses on neural oscillations. These oscillations have been a subject of extensive research due to their significant importance in human behavior and functions. However, the lack of a comprehensive mathematical model for simulating brainwaves prompted the researchers to propose a CA model for a neuronal population.

The CA model takes into account the different states of an action potential at the cellular level to capture some crucial characteristics of neural networks. This includes considering various states of neuron activation and incorporating both excitatory and inhibitory synapses. By employing this computational model, the researchers were able to replicate different dynamics observed in real neuronal populations, ranging from fixed point and limit cycle behaviors to chaotic patterns. To evaluate the effectiveness of their CA network, the researchers performed qualitative comparisons between actual EEG data and CA simulations. The time series generated by the model exhibited high dimensional stochastic behavior, which closely corresponds to the behavior observed in a healthy brain.

Due to the comprehensive nature and reliable performance of this model, it will serve as the foundation for our study. Its ability to accurately capture the behavior of biological neuronal populations makes it an invaluable tool for investigating neural dynamics and significantly advancing our understanding of brain activity.

3.1 Algorithm

The proposed CA, as described in the article, consists of the following components:

- A two-dimensional NxN square lattice
- A randomly connected neighborhood, which represents the synapses and exhibits varying sizes for each cell. The number of synapses per cell can range betweenN_{min} and N_{max} with $N_{max} < N$
- An evolution rule and a set off states inspired on the neural action potentials.

The initial conditions for the cells are selected with a random number generator based on different phases of the action potential, and their connections are described as short or long range based on an uniform random distribution, as described in figure

4(a)

The possible states of a neuron in the CA system are:

$$S = \begin{cases} \textit{resting state} & \textit{if} \quad AP = 0 \\ \textit{firing state} & \textit{if} \quad 1 \leq AP \leq 4 \\ \textit{hyperpolarization state} & \textit{if} \quad AP = 5 \\ \textit{refractory state} & \textit{if} \quad 6 \leq AP \leq 10 \end{cases}$$

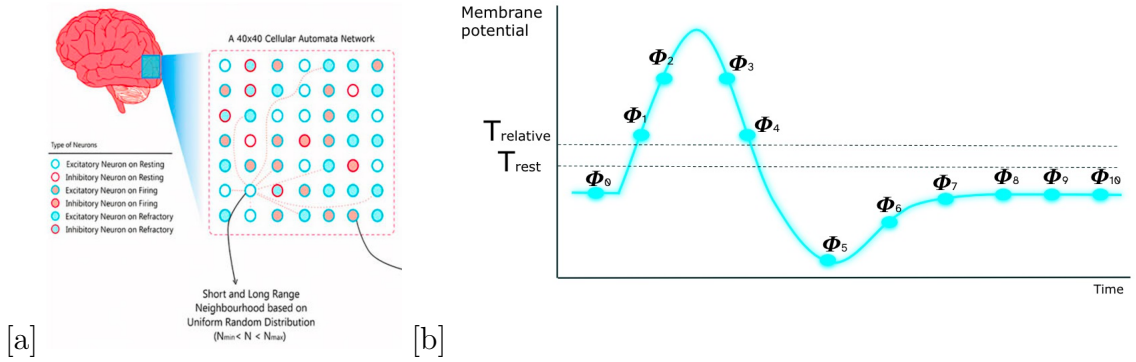


Figure 4: (a)Representation of the neighborhood configurations utilized in the article, highlighting the specific spatial relationships and connectivity between neighboring cells. [1].(b)Cycle of the membrane potential of a single neuro,n as defined by the rules of the algorithm .

In Figure 4(b), a visual depiction is presented to illustrate the correspondence between the evolution of the action potential of a neuron, denoted as ϕ_{AP} and AP.

The evolution rules fore each cell depend on the current state of the cell and are as follows:

- In the case of a resting neuron, its activation occurs when the cumulative inputs from neighboring cells' synapses surpass a predetermined resting threshold, denoted as Trest.

$$AP(t+1) = \begin{cases} AP(t)+1 & if \quad C_e - C_i - \alpha C_h \geq T_{rest} \\ AP(t) & if \quad C_e - C_i - \alpha C_h > T_{rest} \end{cases}$$

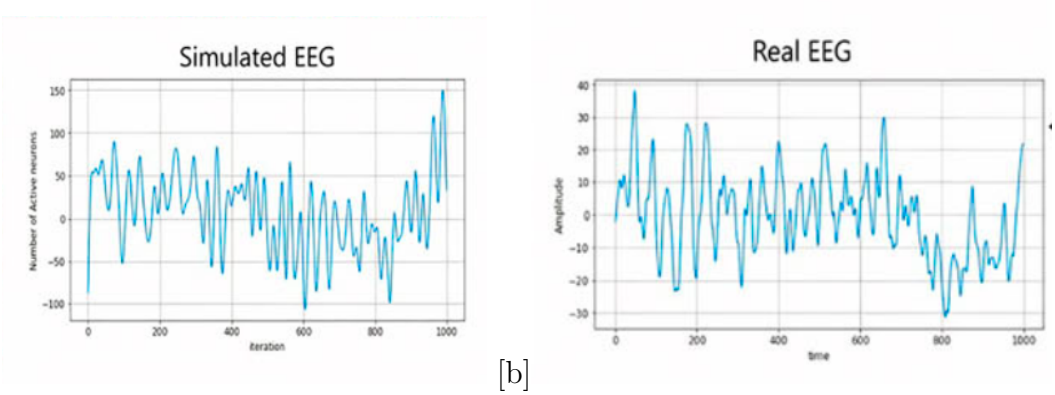
- The firing state and hyperpolarization state, impose restrictions on the responsiveness of neurons to external stimuli, however after activation, a neuron must pass firing, hyperpolarization and refractory states and then return to the resting state. Therefore for this states in each time step the action potential advances to the next value with out being affected by it's neighbors.

$$AP(t+1) = AP(t) + 1$$

- During the refractory period, a neuron has the potential to be reactivated if the level of stimulation from its neighboring cells surpasses a threshold referred to as $T_{relative}$, where $T_{relative}$ is greater than T_{rest}. This distinct state, known as the refractory state, signifies the temporary period during which the neuron exhibits a heightened responsiveness to external stimuli, allowing for subsequent activation under specific conditions.

$$AP(t+1) = \begin{cases} 1 & if \quad C_e - C_i - \alpha C_h \geq T_{relative} \\ AP(t)+1 & if \quad C_e - C_i - \alpha C_h < T_{relative} \ and \ AP(t) < 10 \\ 0 & if \quad C_e - C_i - \alpha C_h < T_{relative} \ and \ AP(t) = 10 \end{cases}$$

Hereby,



[a] [b]

Figure 5: Comparison of a real and simulated EEG[1] .

- C_e Number of excitatory neighbors neurons
- C_i Number of inhibitory neighbor neurons
- C_h Number neighbor neurons in hyperpolarization state
- α Hyperpolarization coefficient.

The output of the system is obtained by extracting time series data, representing the cumulative effect contributed by each individual neuron. This summation is calculated using the values assigned to each specified phase of the action potential, denoted as W(AP), as follows:

$$y(t) = \sum_{n=0}^{N} W(AP_n) \quad .$$

The resulting time series provides a comprehensive representation of the combined influence exerted by the neurons within the system.

The time series so obtained undergoes a Butterworth filter to address experimental considerations associated with EEG recording, including the impact of factors such as skin, electrodes, and the skin-electrode interface. The Butterworth filter used in this study has a low cut-off frequency of 1 Hz and a high cut-off frequency of 50 Hz [1]. By applying this filter, the undesired frequency components outside the specified range are effectively attenuated, ensuring the signal primarily contains relevant EEG activity. A visual representation of the simulated series after applying the Butterworth filter and a real EEG signal can be observed in Figure 5 .

3.2 Mathematical analyses for model validation

In order to compare the real and simulated time series, a variety of techniques commonly employed in neuroscience for EEG analysis were employed. These techniques were selected to provide comprehensive insights into the underlying dynamics and characteristics of the time series data. By applying these established methods, it becomes possible to discern similarities, differences, and patterns in both the real and simulated EEG signals, facilitating a thorough examination and comparison of their respective properties.

3.2.1 Spectral Analysis

Spectral analysis is a technique employed in signal processing to examine the frequency components of a given signal [18]. It can be used for the detection of periodic signals that may be hidden within noise, allowing to get information that is not as evident in the time domain [19].

The application of spectral analysis techniques to EEG signals has revealed significant relationships between frequency components and specific brain activities. Studies have reported associations between EEG spectral features and human behavior, cognitive states, and mental illnesses. By decomposing EEG signals into frequency components, distinct patterns and frequency bands related to different brain activities can be identified. [20; 21; 22; 23; 24].

There are several spectral analysis methods, including techniques such as the Fast Fourier Transform (FFT), autoregressive-based spectral analysis, trigonometric regressive spectral analysis, and wavelet transform. Those methods offer distinct approaches for analyzing and decomposing signals in the frequency domain [25].

In this case to understand the frequency components of the signal the PSD, which represents the power distribution across different frequencies, we employed the Welch method, desribed in the next section.

The Welch method

Welch's method is a widely used PSD estimation method based on the concept of using periodogram spectrum estimates. It involves dividing the signal into overlapping segments, computing a modified periodogram for each segment and averaging the periodograms to obtain the PSD estimate [26]. The Welch method is known for its improved accuracy and reduced variance, especially when dealing with signals that have high levels of noise. The steps to apply Welch's method are as follows [27]:

1. Take a data sequence of length N

$$x[0], x[1], .., x[N-1]$$

2. Divide the data into K segments of size M , with S points between segments, so that the first couple of segments would be written as

$$x[0], x[1], .., x[M-1]$$

$$x[S], x[S+1], .., x[M+S-1]$$

   In our case, the values we used were $M = 256$ and $S = 128$.

3. Perform a windowed discrete Fourier transform (DFT) for each segment,

$$X_k(\nu) = \sum_m x[m]w[m]e^{-j2\pi\nu m},$$

with $w[m]$ the window function and m the elements of the segment. The windowing function selected was the Hanning window, given by

$$w[n] = 0.5\left[1 - \cos\left(\frac{2\pi n}{N}\right)\right] = \sin^2\left(\frac{\pi n}{N}\right)$$

.

4. Calculate the modified periodogram value for each segment,

$$P_k(v) = \frac{1}{W}|X_k(v)|^2$$

with

$$W = \sum_{m=0}^{M} w^2[m]$$

5. Compute the average of the periodogram values to obtain the Welch's estimate of the power spectral density PSD,

$$S_x(v) = \frac{1}{K}\sum_{k=1}^{K} P_k(v)$$

After computing the PSD, the frequency components derived from the obtained time series are categorized into six distinct frequency bands. These bands are defined as delta (1-4 Hz), theta (4-8 Hz), alpha (8-12 Hz), sigma (12-16 Hz), beta (16-24 Hz), and gamma (24-30 Hz) [1]. This separation of frequencies enables a comprehensive analysis of the power distribution within different frequency ranges, allowing for a more detailed examination of the neural activity captured by the PSD. An example of an EEG and the resulting PSD is presented in Figure 6. the comparison of the power on each band found in the article is shown on Figure 7.

3.2.2 Recurrence quantification analysis (RQA)

A RP is a powerful tool in time series analysis. It is a graphical representation of a square matrix, where columns and rows represent a time point. For each moment in time the plot shows if the dynamical system roughly returns to the previous state [28].

$$\vec{x}(i) \approx \vec{x}(j) \quad .$$

Recurrence plots are generated by plotting the distances between each point in a time series against every other point within the same series. Let X be a time series of length T. Then, $x(t)$ would be the t-th point of the series. A distance function, denoted as d_X, is defined. Additionally, a threshold ϵ_X is established to determine the maximum allowable distance. Consequently, the recurrence matrix for x(t) is constructed as [29]

$$R_X(i,j,\epsilon_X) = \begin{cases} 1, & \textit{if} \quad d_X(x(i),x(j)) < \epsilon_X \\ \\ 0, & \textit{if} \qquad \textit{otherwise.} \end{cases}$$

The resultant plot visually represents the recurrence of patterns within the data, providing insights into the underlying dynamics of the system. By analyzing the recurrence

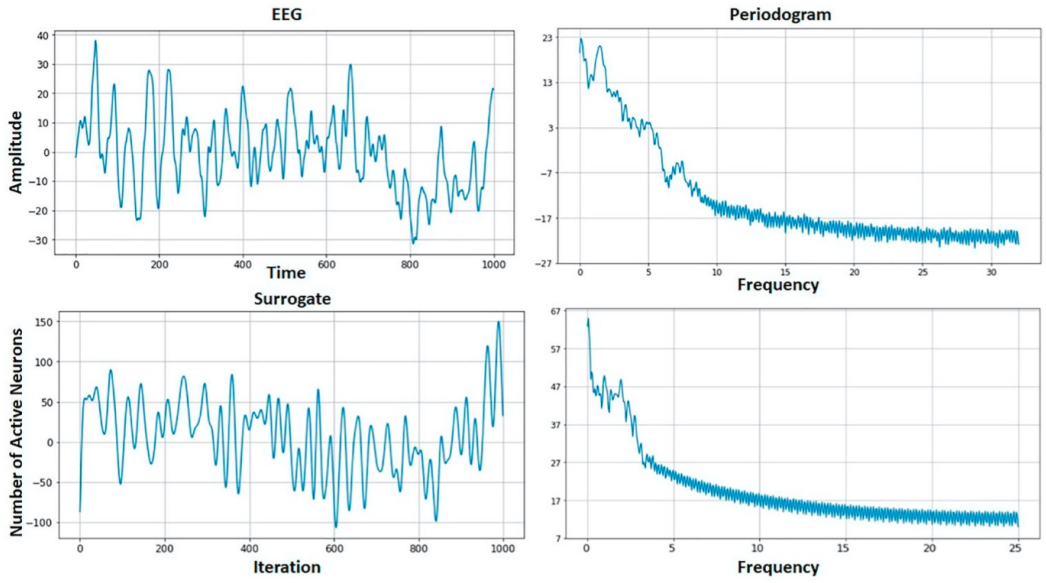


Figure 6: Comparison between real EEG data and simulated time series generated by the CA network, along with their corresponding periodograms. The top row of the figure illustrates the real EEG data, while the bottom row showcases the simulated time series [1].

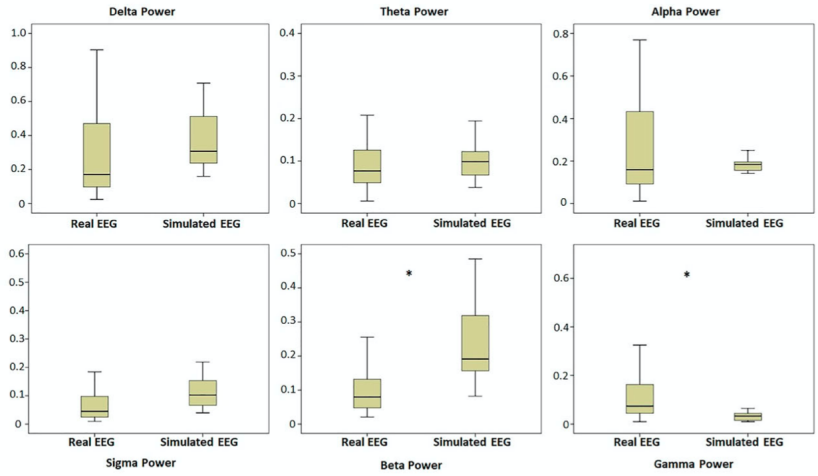


Figure 7: Box plots, of frequency components of the real and simulated EEG time series in the six frequency bands.[1].

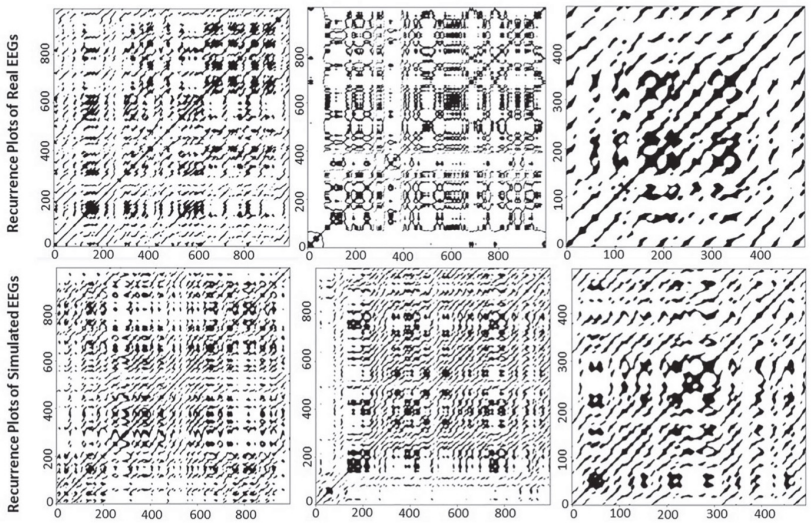


Figure 8: Recurrence plots of both real EEG data and simulated time series generated by the CA. The top row represents the recurrence plot of the real EEG data, while the bottom row corresponds to the recurrence plot of the simulated time series. [1].

plot, repetitive structures, intrinsic behaviors and temporal dependencies can be detected, enabling a deeper understanding of the system's complex dynamics at play within the analyzed data [30]. Figure 8 showcases examples of recurrence plots generated from both real EEG data and simulated time series produced by the cellular automaton network.

RQA is used to quantify the structures and patterns that are found in a RP [29]. Several of the computed RQA are the following:

- **Averaged diagonal line length:** It is a metric that represents the average length of diagonal lines in a recurrence plot. It is calculated as the mean length of diagonal lines using the following formula [28]:

$$L = \frac{\sum_{l=l_{min}}^{N} lP(l)}{\sum_{l=l_{min}}^{N} P(l)} \quad .$$

  Here, the parameter N represents the total number of measured points in the phase space trajectory, and $P(l)$ is the histogram of diagonal lines of length l as defined in equation 1.

  A diagonal line of length l indicates that a segment of the trajectory remains in close proximity to another segment for l consecutive time steps. These diagonal lines capture the divergence or convergence of trajectory segments, providing insights into the dynamics of the system. The averaged diagonal line length can be interpreted as the average duration during which two segments of the trajectory remain close to each other. This measurement carries the interpretation of the mean prediction time [28].

- **Maximum diagonal line** represents the length of the longest diagonal line observed in the recurrence plot. It is defined as the maximum value among a set of diagonal line lengths [28],

$$L_{max} = max(l_{ii=1}^{N_l}) \quad ,$$

where $N_l = \sum_{l \geq l_{min}} P(l)$ is the number of whole diagonal lines. This metric is closely associated with the exponential divergence of the phase space trajectory. Specifically, when trajectory segments diverge rapidly, the resulting diagonal lines tend to be shorter in length [28].

- **Determinism** It is the measure obtained by comparing the number of recurrence points forming diagonal structures (with a minimum length of l min) to the total count of recurrence points [29],

$$DET = \frac{\sum_{l=l_{min}}^{N} lP(l)}{\sum_{l=1}^{N} lP(l)}.$$

  Deterministic processes exhibit distinct recurrence patterns characterized by longer diagonal lines, in contrast to both chaotic and stochastic processes. Chaotic and stochastic processes, on the other hand, tend to generate very short diagonals and numerous isolated recurrence points within the recurrence plots [1].

- **Trapping time** pertains to the average length of vertical lines observed in the recurrence plot. It is calculated using the following formula [28]:

$$TT = \frac{\sum_{v=v_{min}}^{N} vP(v)}{\sum_{v=v_{min}}^{N} P(v)} \quad .$$

  Here, P(v) denotes the histogram of vertical lines with length v, as defined in equation 2. The trapping time provides an estimation of the average duration that a system remains in a specific state or how long it is trapped within that state. The trapping time tells us on the stability of different states exhibited by the system.

- **Maximum vertical line** is a metric that indicates the length of the longest vertical line observed within the recurrence plot. It is determined as the maximum value among a collection of vertical line lengths [28],

$$V_{max} = max(v_{l_{l=1}}^{N_v}) \quad ,$$

  where N_v is the number of vertical lines.

- **Laminarity** refers to the ratio of recurrence points that contribute to the formation of vertical lines compared to the total number of recurrence points[28],

$$LAM = \frac{\sum_{v=v_{min}}^{N} vP(v)}{\sum_{v=1}^{N} vP(v)} \quad .$$

  LAM quantifies the occurrence of laminar states within the system without specifically addressing the duration of these laminar phases. A decrease in LAM suggests a higher prevalence of single recurrence points relative to vertical structures, indicating a reduction in the occurrence of laminar states within the recurrence plot [28].

The histograms of diagonal and vertical lines, denoted $P(l)$ and $P(v)$ respectively, are defined as follows [28; 29]:

$$P(l) = \sum_{i,j=1}^{N} (1 - R_{i-1,j-1})(1 - R_{i+1,j+1}) \prod_{k=0}^{l-1} Ri + k, j + k \quad , \tag{1}$$

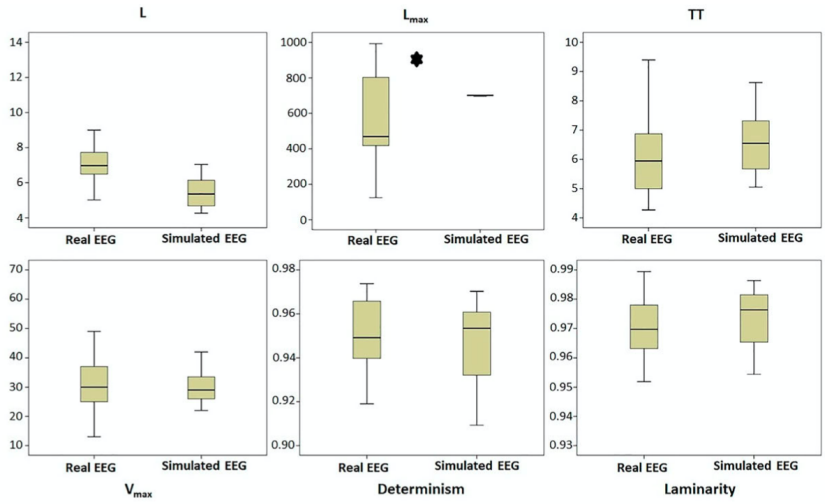


Figure 9: Box plots of six RQA features of real and simulated EEG time series.[1].

$$P(v) = \sum_{i,j=1}^{N} (1 - R_{i,j})(1 - R_{i,j+v}) \prod_{k=0}^{v-1} Ri, j + k \quad . \tag{2}$$

The comparative analysis of the obtained RQA measures reported in the original article [1] is illustrated in figure 9.

3.2.3 Fractal analysis

The fractal dimension, denoted as D, quantifies the intricate nature of a waveform, indicating its level of complexity [31] and serves as a statistical index used to characterize the complexity of fractal patterns. It quantifies the ratio of detail changes to scale changes, providing a measure of the intricate nature and self-similarity exhibited by these patterns [1].

Petrosian fractal dimension

The Petrosian fractal dimension serves as a simplified approximation of the fractal dimension. It simplifies the computation process by transforming the signal into a binary representation,

$$z_i = \begin{cases} 1, & if \quad x_i > \overline{y} \\ -1, & if \quad x_i \leq \overline{y}. \end{cases} \tag{3}$$

It estimates the fractal dimension based on the count of sign changes within the signal,

$$N_\Delta = \sum_{i=1}^{N-2} \left| \frac{z_{i+1} - z_i}{2} \right| \quad ;$$

therefore, the Petrosian fractal dimension is calculates as [32]

$$D_{Petrosian} = \frac{\log_{10} N}{\log_{10} N + \log_{10}(N/(N + 0.4N_\Delta))} \quad . \tag{4}$$

This approach provides a straightforward method for approximating the fractal dimension while reducing computational complexity .

Katz fractal dimension

The Katz fractal dimension, while computationally more demanding, offers a higher level of accuracy compared to the Petrosian fractal dimension [32]. The calculation of the Katz fractal dimension is performed as follows [31]:

$$D_{Katz} = \frac{\ln(N-1)}{\ln(N-1) - \ln(\frac{d}{L})} \quad . \tag{5}$$

Here, N represents the length of the sequence data, indicating the total number of data points, and L is the sum of distances between consecutive points in the data sequence,

$$L = \sum_{i=0}^{N-2} \sqrt{(y_{i+1} - y_i)^2 + (x_{i+1} - x_i)^2} \quad .$$

The parameter d refers to the diameter of the data sequence, representing the maximum distance between any two points within the sequence,

$$d = max\left(\sqrt{(x_i - x_1)^2 - (y_i - y_1)^2}\right) \quad .$$

Waveforms that are perfectly straight exhibit minimal dimensionality, with $D \approx 1.0$. Conversely, highly spiked waveforms possess maximal dimensionality, with $D \approx 1.5$ [31].

In general, the Petrosian method focuses on the identification of local extrema within a time series to discern its underlying fractal patterns. Conversely, the Katz method relies on analyzing the amplitude differences within the time series to compute the fractal dimension [1]. These distinct approaches offer complementary perspectives for characterizing the complexity and self-similarity exhibited by different types of data.

3.2.4 Entropy analysis

Entropy analysis is a statistical technique employed in the analysis of time series data, which offers a concise summary that takes into consideration the non-stationary characteristics of the data [33].

Entropy serves as an index that characterizes the randomness and complexity of dynamic systems, providing insight into the rate at which information is generated and developed within these systems [1].

Sample entropy

The sample entropy method is a modified version of the approximate entropy that reduces the self-matching bias and is independent of the length of the data, and also has relative consistency in different conditions [34]. Sample entropy is defined as [35]

$$SampEn(m, r, N) = -\ln \frac{A^m(r)}{B^m(r)} \quad .$$

In order to compute the sample entropy, several steps need to be followed. Firstly, a window length, denoted as m, and a tolerance value, represented as r, must be selected.

Vectors of length m are then constructed by considering all possible m consecutive points within the time series. The distance between each pair of vectors is calculated using either the Chebyshev distance or the Euclidean distance. The next step involves counting the number of vector pairs with a distance smaller than the chosen tolerance value, r [35].

Following this, the conditional probability is determined, representing the likelihood that two sequences, initially similar for m points, remain similar for $m+1$ points. $Bm(r)$ signifies the probability that two sequences of data points, each with a length of m, have a distance smaller than the tolerance value r. Similarly, $Am(r)$ defines the probability of similarity for two sequences of data points, each with a length of $m+1$. In this particular study, a window length of $m = 24 and a tolerance value of r = 0.2 \times$ SD4 were chosen, where SD denotes the standard deviation of the time series [1].

Permutation entropy

The permutation entropy of an information source is defined in a manner similar to Shannon's entropy, with the distinction that probabilities are associated with ordinal patterns. This measure offers an alternative approach to assess the complexity of a time series by transforming it into a symbolic sequence. In this representation, the original continuous time series is mapped onto a sequence of symbols. The permutation entropy captures the inherent complexity of the time series by considering the frequencies of different ordinal patterns within the symbolic sequence. The normalized permutation entropy is given by [36]

$$PE = -\frac{\sum_{i=1}^{n!} p_i \ln p_i}{\ln n!} \quad .$$

To compute the permutation entropy, a time series and a window length m are selected. Vectors of length m are constructed by considering all possible m consecutive points in the time series. The values in each vector are ranked to determine the ordinal pattern. The frequency of occurrence of each ordinal pattern is then counted. The relative frequencies of the possible patterns of the symbolic sequences, referred to as permutations and denoted as p_i , is calculated. Finally, the permutation entropy is obtained by taking the negative sum of the product of the probability of each ordinal pattern and its natural logarithm. The parameter n corresponds to the embedding dimension of the time series or the order of permutation [36].

Multiscale entropy

It is a technique that extends the sample entropy method to assess the complexity of signals by incorporating a temporal coarse-graining procedure with a scaling parameter (τ) and non-overlapping windows. [35; 37]. This procedure involves dividing the time series into segments denoted $y_j^{(\tau)}$where $1 \leq j \leq N/\tau$, with N representing the length of the time series [35], and is computed as

$$y_j^{(\tau)} = \sum_{i=(j-1)\tau+1}^{j\tau} x_i \quad .$$

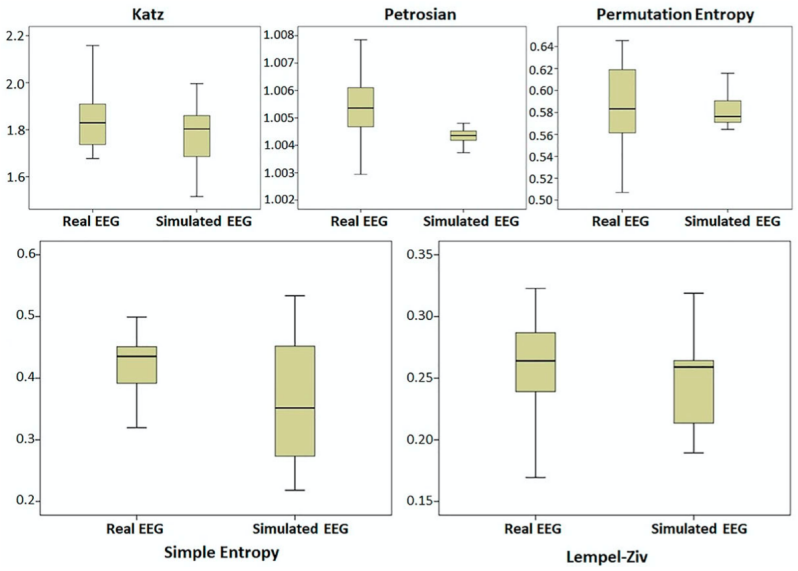


Figure 10: Box plots of fractal, entropy and Lempel-ziv features of real and simulated EEG time series[1].

At the first scale of the multiscale entropy analysis, the time series $y^{(1)}$ remains identical to the original time series. Subsequently, the time series is segmented according to a specific interval determined by the scaling parameter. The entropy is then calculated for each segment at different scales[1], the results of this analysis presented in the article of reference are shown in figure 11.

3.2.5 Lempel-Ziv complexity

The Lempel-Ziv complexity is a measure of repetitiveness in binary sequences and text. It is a complexity measure that specifically utilizes the recursive copy function, and it is closely related to Kolmogorov complexity. By applying the Lempel-Ziv complexity, one can quantify the level of uncertainty present in time series data and evaluate the diversity of patterns within a given signal [38].

First, a binary sequence, of length n, is often obtained through thresholding, commonly using the median value of the time series as described in equation 3, read from left to right. Let $C(n)$ denotes the count of distinct subsequences encountered during the stream processing of the binary sequence. The Lempel-Ziv measure is defined as [38]

$$LZ_n = -\frac{\ln(n) \cdot c(n)}{n} \quad . \tag{6}$$

The Lempel-Ziv complexity then measures the uniqueness and richness of subsequences in this binary representation.

Figure 11 illustrates the combined presentation of the fractal dimension, entropy, and Lempel-Ziv complexity measures for both the real and simulated series, as reported in the original article [1].

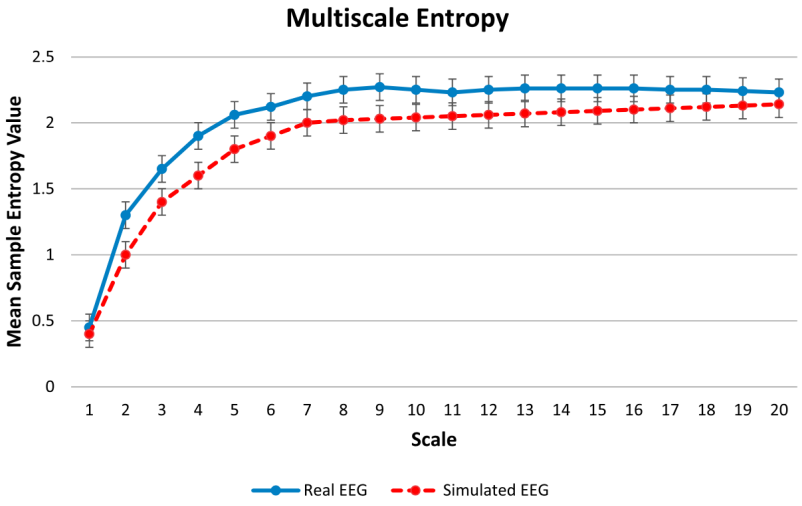


Figure 11: Mutltiscale entropy curves of real and simulated EEG data. .[1].

3.2.6 Statistical analysis

The Shapiro-Wilk test, a commonly employed statistics, is utilized to ascertain whether a given sample of data conforms to a normal distribution [39; 40]. This test involves computing a test statistic, W, which quantifies the extent of deviation from normality [40]. The test has demonstrated notable success and efficiency, particularly when applied to smaller sample sizes ($s \leq 50$) [41]. Its robustness and effectiveness make it a preferred choice for assessing the normality of data distributions in various research contexts.

The Shapiro-Wilk test was employed to assess the normality of the data, confirming its adherence to a normal distribution. Consequently, parametric tests were deemed appropriate for evaluating the statistical significance of differences between the extracted features from the real and simulated EEG signals. In light of this, the independent t-test was employed as the parametric sest to compare the means of the two groups. A significance criterion of $p < 0.05$ was adopted to determine statistical significance [1].

4 Our implementation

In this chapter we will describe how we implemented the model by Khalegui et. al. the code was developed by using object-oriented programming principles in the C++ programming language. Math anlyses were implemented in c++ and python

Although the model is described in the reference article [1] there are details and fundamental aspects that were not specified there, such as the construction algorithm for the network, the threshold levels for the neuron's firing, etc. The chapter describes in detailed how we did it, as a reference document for future researchers interested in this area.

4.1 Construction of the networks

The arrangement of neurons followed a one-dimensional periodic array of 1600 spaces to match the square lattice on the original article of 40×40. Each cell within this array was assigned three crucial characteristics, which played a fundamental role in the simulation model.

- **Immutable Vector:** This vector determined whether each cell in the network was excitatory or inhibitory. The characteristic of each cell was randomly assigned by a random number between 0 and 1. If the generated number exceeded a predefined threshold, the cell was considered excitatory; otherwise, it was inhibitory.
- **Connections (Synapses):** The number of connections and the list of connections were crucial in determining the influence of neurons on each other throughout the simulation. This aspect varies among the different networks studied. The number of connections for each neuron was determined by a function that randomly selected a predetermined number of connections meaning that a total number of connections was set for the whole array and this function randomly decided how many of this connections would correspond to each cell in the array. The way the connections were selected determined the type of network random, small world or local.
- **Cycle Tracking:** A vector was employed to track the phase of the membrane potential for each cell at every step of the simulation. This information helps to monitor the progress of the network dynamics. This vector was initialized using a random number generator, which produced values between 0 and 1. Based on the generated number, the membrane potential was assigned to achieve a distribution of neuron states with the following percentages: resting (20%), firing (40%), refractory (35%), and hyperpolarization (5%)

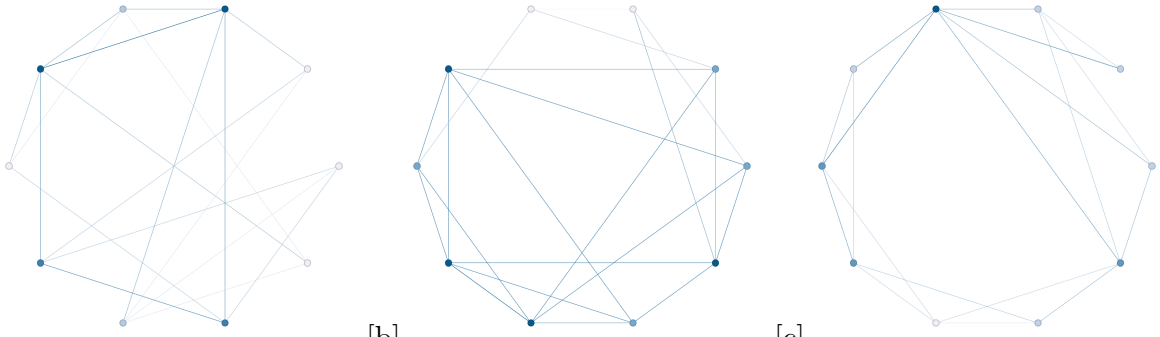
Among these characteristics, only the vector responsible for storing the phase of each neuron's cycle could change at each simulation step. This enabled the tracking of dynamic changes in the network's behavior.

Those components collectively formed the basis of the simulation model, enabling the exploration and analysis of the different network architectures and their effects on the neuronal behavior. The following network architectures were considered:

- **Random:** This architecture consists of N labeled nodes, representing cells or neurons, connected by L randomly placed links [42], symbolizing connections or synapses. An example of a random network with 10 nodes is illustratedFig. 12(a)
- **Small-world:** This type of network exhibited a characteristic where the average path length or diameter showed a logarithmic dependence on the system size. The Watts-Strogatz Model was employed to construct this network architecture. Initially, a ring of nodes was created, with each node connected to its immediate and next neighbors. Subsequently, with a probability of p, each link was rewired to a randomly selected node [42]. An example of a small-world network is depicted in Fig. 12(b)
- **Local:** This architecture emphasized the connection of each node to its closest neighbors. In this case, the number of connected links varied for each node. An example of a local network can be observed inFig. 12(c)

Figure 12 illustrates the representation of three types of networks: random, small-world, and local. Similar to the original code, all three networks have the same number of links and nodes. However, the distinguishing factor lies in the manner in which the nodes are interconnected, resulting in distinct network properties for each type.

These network architectures were instrumental in studying the behavior and dynamics of neuronal systems. By analyzing their characteristics, insights into the impact of network structure on neuronal processes and interactions could be gained.



[a] [b] [c]

Figure 12: (a) A 10 node randomly connected network (b) A 10 node small world connected network (c) A 10 node non uniform locally connected network

4.2 Rules of evolution

Several functions made possible the evolution of the system after the initial point:

- A function that had access to the list of connections for each cell and checks the number of active connected neurons, both excitatory and inhibitory, the number of connected neurons that were in hiperpolarization state, and returns the result of the evolution equation.

Network	T_{rest}	$T_{relative}$
Random	3	5
Small-world	3	5
Local	1.4	2.1

Table 1: Comparison of activation thresholds implemented in the studied networks.

- A function that takes the result of the previous one and, depending of the state of the cell is is being applied, changes its state according to the rules of evolution described in section 3.1.
- A function that assigns a potential to each cell depending on its current state.
- a function that adds the potential of all cells, and gives this result as the output of the program.

Lastly, a loop repeats the evolution steps a number of times defined at the begining of the simulation.

For each network type different thresholds were used. A summary of them is shown in Tabl. 1

4.3 Mathematical analysis

Preprocessing of real data

For the purpose of this study, the data utilized was obtained from Physionet [43]. Specifically, the "EEG During Mental Arithmetic Tasks" dataset was selected, which consists of recordings from healthy adults in both the resting state and during arithmetic tasks [44]. The data is provided in the European Data Format (.edf), a widely used file format for the storage and exchange of medical time series data [45]. Given that each EEG recording encompassed over 90,000 data points, a signal processing approach was employed to shorten the data using the code in Appendix A.

The dataset used in this study had already undergone preprocessing prior to its acquisition. The preprocessing steps included the application of a notch filter at 50 Hz and a high-pass filter with a cut-off frequency of 30 Hz. Thus, the dataset utilized in this research had already been filtered, ensuring its suitability for subsequent analyses.

Preprocessing of simulated data

The Butterworth filter, as recommended in the reference article, was implemented in C++ using the DSP IIR Realtime C++ filter library The code used for the implementation can be found in Appendix C. The library provides the necessary tools to effectively apply the Butterworth filter to process the signals as described in the base article.

Spectral Analysis

The PSD was calculated using the SigPack library. However, a modification was made to the library to replace the default Hamming window with the Hanning window during the calculation of the PSD. . The code used for the analysis and provided in Appendix D incorporates the definition of a time step to address the fact that the time series contains more than one point per second. This adjustment ensures proper handling of the data and maintains the temporal integrity during the analysis. Additionally, it is important to note that the calculated PSD values obtained from the library are converted to decibels (dB) for better representation and interpretation of the results. This transformation aids in the visualization and understanding of the signal's frequency content on a logarithmic scale.

RQA

The process of generating recurrence plots was facilitated by employing the recurrence_cpp library. This library provides the necessary functions and allows for a straightforward implementation by specifying key parameters such as the embedding dimension, delay, threshold, and the type of norm to be used. In this study, the following parameter values were used: an embedding dimension of 2, a delay of 1 and a threshold set to 0.1 times the standard deviation (SD) of the data. The calculation of the standard deviation was integrated into the code for ease of use.

For computing the measures associated with Recurrence Quantification Analysis (RQA), only few more steps were required. These steps primarily involved to define the minimum length of diagonal lines, as well as vertical white and black lines and the calculation of various measurement parameters such as (L, L_{MAX}, V_{MAX}, DET, LAM, TT).

To ensure clarity in the resulting recurrence plot of the real values, the parameters of the code were adjusted accordingly. Subsequently, the same parameter settings were employed to generate recurrence plots for the simulated data.

The complete code utilized for these procedures can be found in Appendix E.

Fractal Analysis

A single program was utilized to calculate both the Petrosian and Katz fractal dimensions. This program encompasses a function that computes the Euclidean distance. Initially, the program generated a binary series by assigning a value of 1 to points above the series average and -1 to points below it. The number of sign changes within this binary series was then counted. Additionally, the program computed the Euclidean distances between consecutive points to determine the signal length and identify the maximum distance between points. Subsequently, the program applied the equations 4 and 5, to obtain the respective Petrosian and Katz fractal dimension values. The complete code can be found in Appendix F.

Entropy analysis

The sample entropy, permutation entropy, and multiscale entropy were calculated using the Python library pyentrp The implementation code can be found in Appendix G For the sample entropy calculation, a tolerance of $0.2 \times SD$ (standard deviation) was used, while for the multiscale entropy, a tolerance of $0.1 \times SD$ was applied.

Lempel-Ziv complexity

The binary sequence is created by checking the elements of the series. Each element is represented by a 1 if the original value was less than the average, and by 0 if it was greater than or equal to that average.

The algorithm operates on the binary sequence and counts the number of distinct patterns, or "words", in the sequence. First it initializes the variables C (complexity count) to 1, u (length of current prefix) to 1, v (length of current component for the current pointer) to 1, and $vmax$ (final length used for the current component, largest among all possible pointers) to v. Then, it enters in a while loop that continues iterating as long as the sum of u and v is less than or equal to the length of the sequence n.

Inside the loop: If the v-th bit of the sequence starting from the position i ($Data[i+v]$) matches the v-th bit from the position u ($Data[u+v]$), it increments v. This means we are extending the current pattern because we have found a match in the sequence. If they do not match, the value of $vmax$ is updated to the maximum of v and $vmax$. Then, i is incremented. Now, if i equals u (indicating that all potential patterns starting from the beginning of the sequence up to the position u have been considered), increment C (a new unique pattern is found), set u to $u + vmax$ (start a new pattern), and reset v, i, and $vmax$ to 1. If i is not equal to u, reset v to 1 to look for a new pattern starting from the next position. The loop ends when $u + v$ is larger than n (indicating we have gone through the entire sequence). If $v \neq 1$ at this point, it means that there is a unique pattern from the last update of u to the end of the sequence, and C is incremented one final time. By the end of the algorithm, C holds the Lempel-Ziv complexity count of the sequence, representing the number of distinct patterns in the binary sequence.

Finally, the function calculates the LZC using Eq. (6), where n is the length of the sequence and C is the complexity count. The complexity count and LZC are printed to the console. The full implementation is found on Appendix H.

4.4 Post-processing

Due to the large amounts of data, the post-processing of each real and simulated EEG was done using a Python script that runs each C++-compiled mathematical analysis code and also generates related plots.

Initially, simulations are generated through a Python loop that executes the C++ simulations with varying seed values, as seen in Appendix I-9. The Butterworth filter is applied to each simulation using the same methodology.

Given each simulated EEG, both simulated and real data are then post-processed using the Python functions shown in Appendix I-10. The Power Spectral Density is calculated using the pwelch C++ implementation described in D-4.Additionally, the periodogram is divided into distinct frequency bands, such as Delta (1 - 4 Hz), Theta (4 - 8 Hz), Alpha (8 - 12 Hz), Sigma (12 - 16 Hz), Beta (16 - 24), and Gamma (24 - 30 Hz) bands. Recurrence plots are generated using the procedure outlined inE-5

Finally, the Entropy analysis G-7, Fractal analysis F-6, and Lempel-Ziv analysis H-8 are implemented for both the real and simulated EEG data. The results of each mathematical analysis are stored in Python Numpy objects for further examination.

5 Findings and Analysis

In the study, three distinct network types were employed: a randomly connected network, a small-world network, and a non-uniform local network, as described in Sect. 4.1. For each network type, a series of simulations were conducted using 30 different seeds for the neural connection generation function. The outcomes of these simulations served as the basis for subsequent mathematical analyses.

This section presents the findings obtained from the simulations and the analysis programs employed.

5.1 Randomly connected network

Effect of node degree in the network

One of the key findings of this study is that when the topology of the network is altered, it ASKS FOR adjustments in other system parameters, such as T_{rest} and $T_{relative}$. In the original study, the network establishes connections based on a uniform random distribution, with values of 8 and 12 for T_{rest} and $T_{relative}$ respectively. In our implemented network, however, we used connections with a Gaussian distribution, and the proportion of inhibitory cells was 50%, much higher than the 20% used by Khaleghi et. al. As a result, the original parameters of 8 and 12 maintained the random network at rest; therefore, the values for T_{rest} and $T_{relative}$ were set to 3 and 5, respectively, to obtain a proper signal.

Spectral analysis

Considering that the power of a signal represents the proportion of frequency content it contains, our analysis reveals notable differences in the frequency distribution between the randomly connected network under investigation and the experiments. Specifically, in the simulated network with the selected parameters, the components associated with high frequencies appear to be significantly diminished compared to the same components in the experimental ECG. An example of a simulated and a real EEG and PSD is depicted in Fig. 13. Nevertheless, we observe that the delta, theta, and alpha frequency bands are well-represented in the frequency space, exhibiting similar amplitudes in both the simulated and real signals. This comparison of the amplitudes can be observed in Fig. 14. This observation suggests that the inclusion of a larger percentage of inhibitory neurons in the network has a substantial impact on the reduction of high frequency components in the signal.

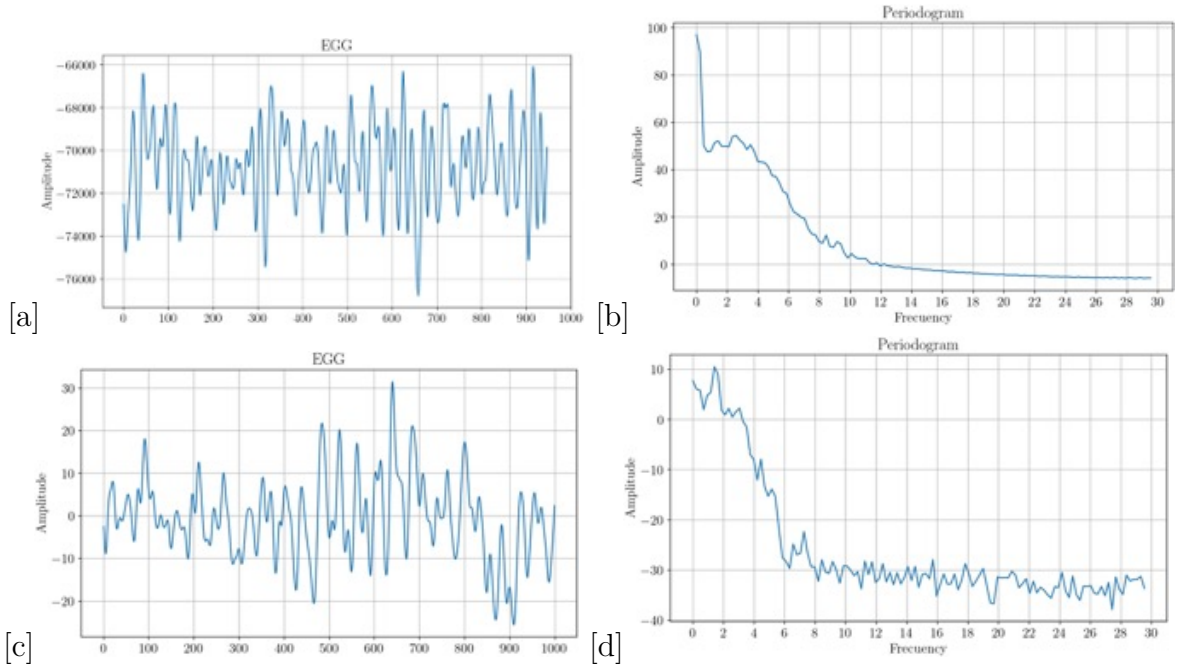


Figure 13: (a) Results of the simulated time series for a randomly connected network after passing the signal through the butterworth filter (EEG of the random network simulation) (b) PSD of the simulated series (c) Real EEG for comparison (d) PSD of a real EEG

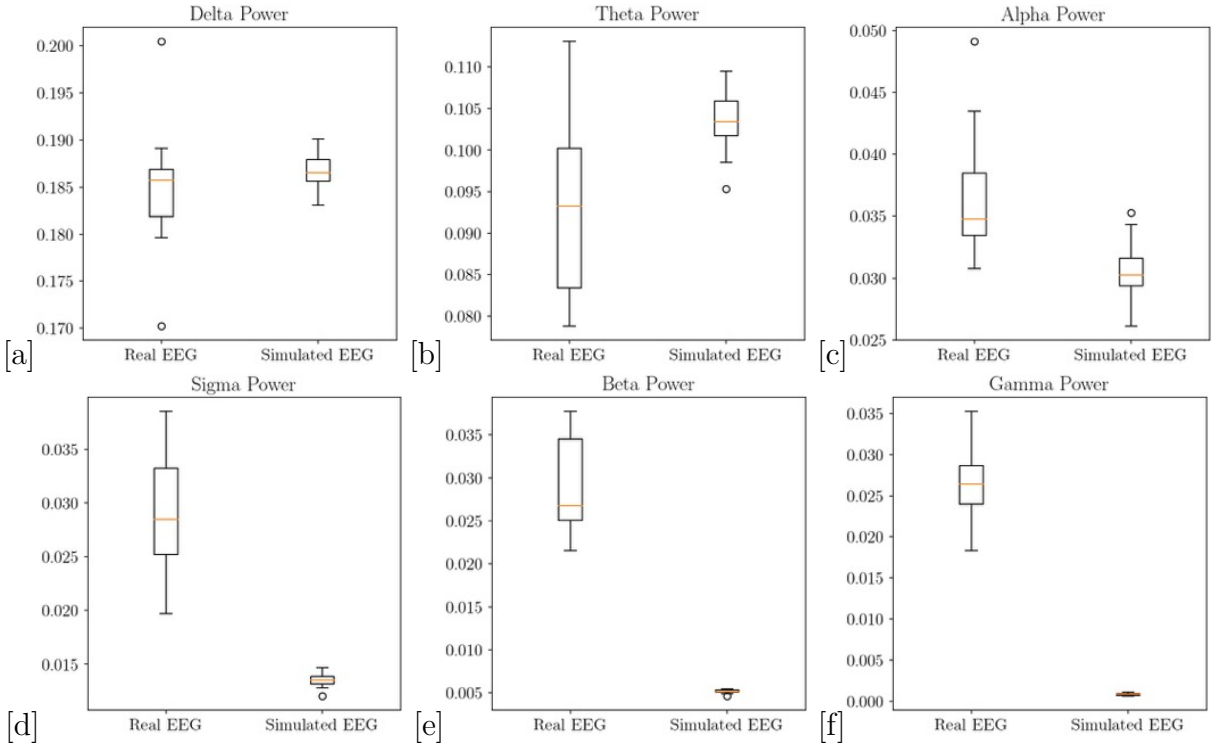


Figure 14: Comparison of the amplitude for each frequency component of the random network

RQA

During the analysis of the network, a noteworthy observation was made regarding the simulation results in comparison to the electroencephalogram (EEG) signals recorded during arithmetic tasks and resting states. Specifically, it was evident that the simulated results exhibited a greater similarity to the EEG patterns observed during arithmetic operations rather than those recorded during periods of rest. This distinction is visually demonstrated in Figure 15.

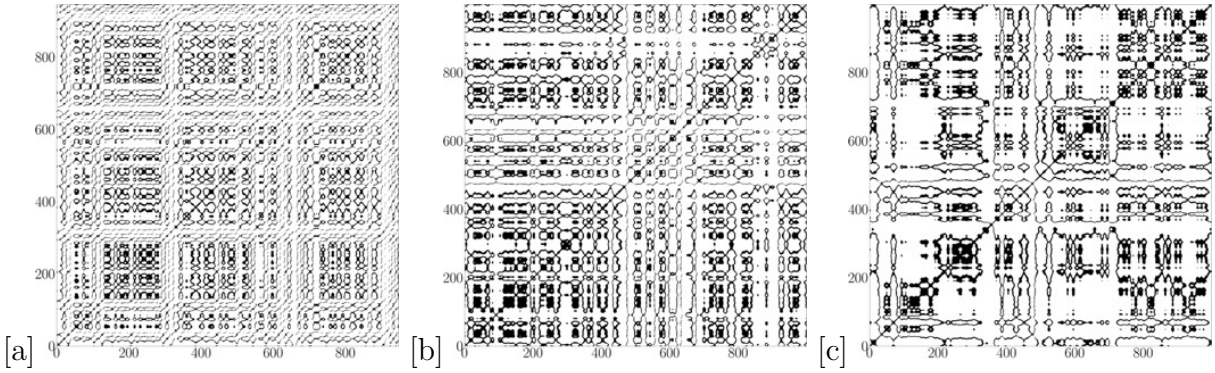


Figure 15: (a) RP of the simulated time series for a randomly connected network (b) RP of a subject mentally performing arithmetic operations (c) RP of the same subject in a resting state

This finding suggests that the cellular automaton network is better suited to capture the neural activity associated with cognitive processes such as arithmetic operations. Furthermore, it highlights the network's potential in modeling and simulating specific brain states or activities, potentially aiding in the understanding and analysis of complex cognitive functions. Consistent with the methodology employed in the original article, our analyses and comparisons encompassed datasets obtained from individuals in both resting and arithmetic operation states. However, as it was done in the base article, the analyses and comparisons were made using data of both people resting and doing arithmetic operations which yielded the results in figure 16.

Six measures were computed to analyze the recurrence plot (RP). The results reveal distinct patterns between the simulated and real EEG data, indicating divergent behaviors. Specifically, the simulated EEG demonstrates a considerably faster divergence rate compared to the real data, suggesting a more pronounced chaotic behavior in the simulated signals.

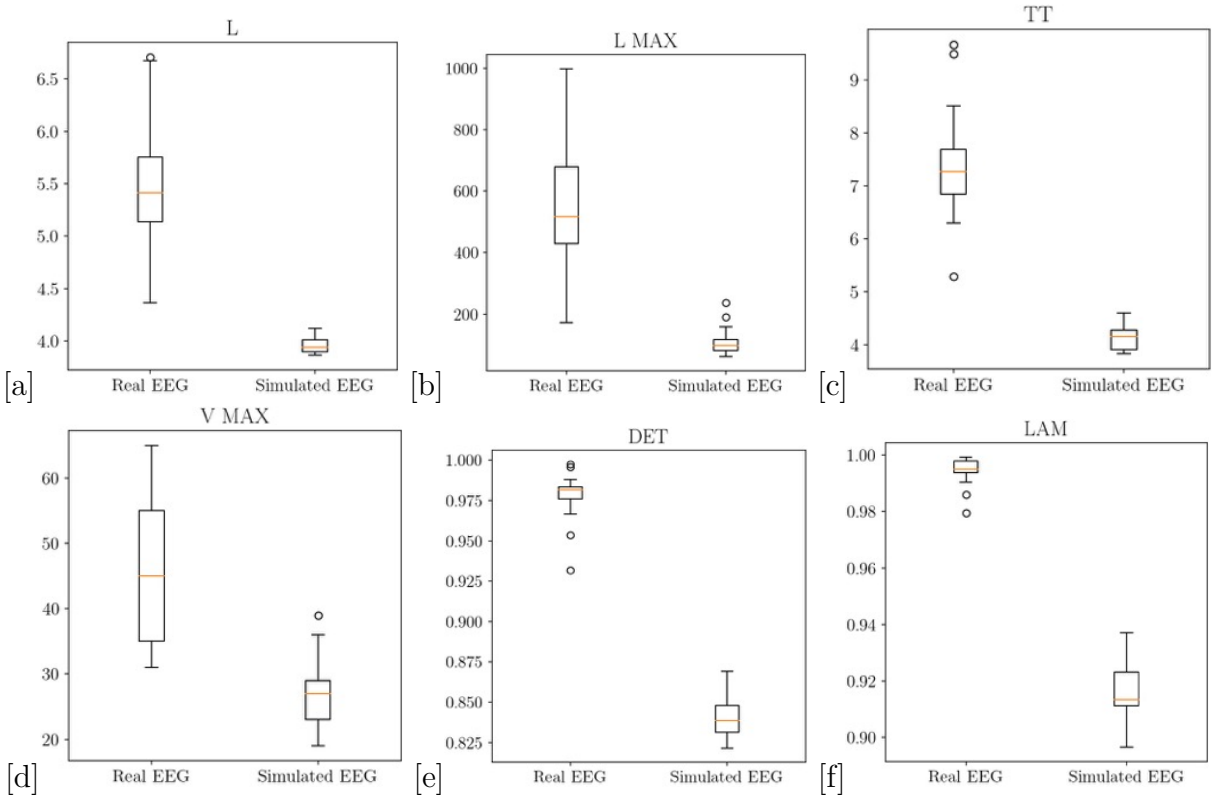


Figure 16: Box plots comparison of six RQA features of real and simulated EEG time series

Fractal analysis

The faster divergence and more chaotic behavior deduced from the RQA analyses in the previous section is confirmed when the complexity measures between the real and simulated EEG series are compared Figure 17 . The simulated series exhibits a greater complexity and more unique sequences. this would mean more rapid changes, sharp peaks, or unpredictable fluctuations in the simulated signal.

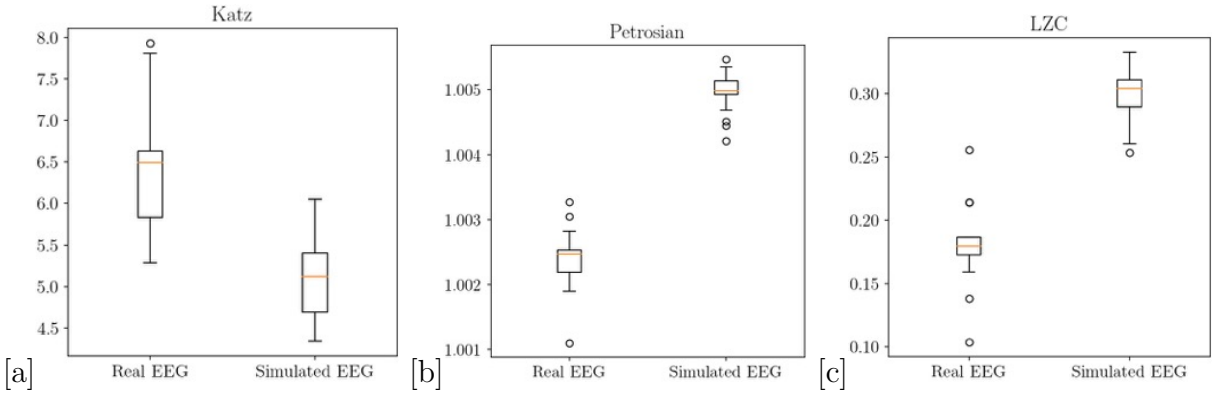


Figure 17: Complexity measures comparison of real EEG and randomly connected simulated network

Entropy analysis

The entropy measures of the EEG signal further support the findings from the analysis of recurrence maps, indicating a higher level of congruence with the EEG patterns observed during arithmetic tasks. Higher entropy values are indicative of more active cognitive states and heightened responsiveness to stimuli. The comparison of entropy values between real and simulated EEG signals is depicted in Figures Fig. 18 and Fig. 19, providing visual evidence of the differences in entropy measures between the two datasets.

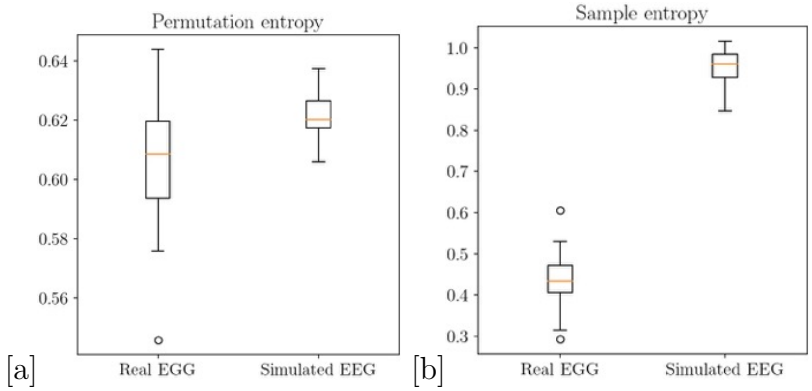


Figure 18: Entropy measures for a simulated random Network and real data

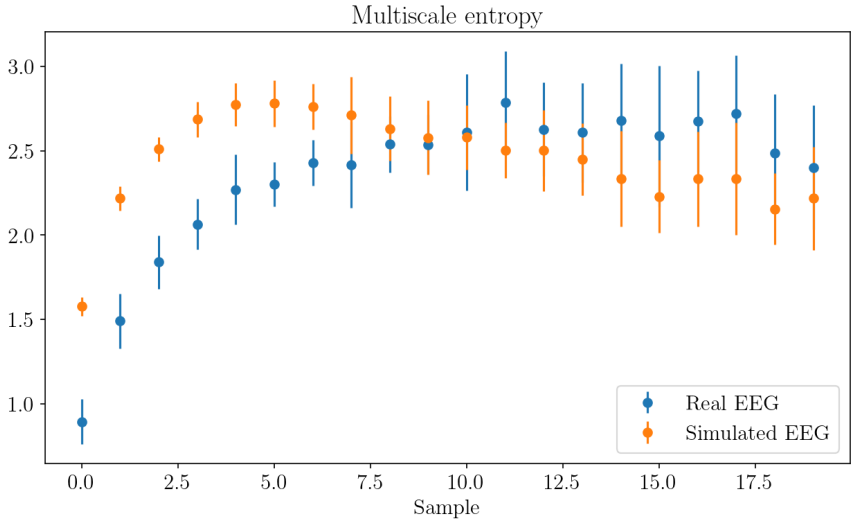


Figure 19: Multiscale entropy for real and random network simulated EEG

5.2 Small world network

Spectral analysis

The small-world network exhibits high concordance with the experimental ECGs the delta and theta frequency bands. The comparison of amplitudes between simulated and real EEG signals is illustrated in Fig. 21. Additionally, an example showcasing both a simulated EEG and a real EEG, along with their corresponding PSD plots, is presented in Figure Fig. 20

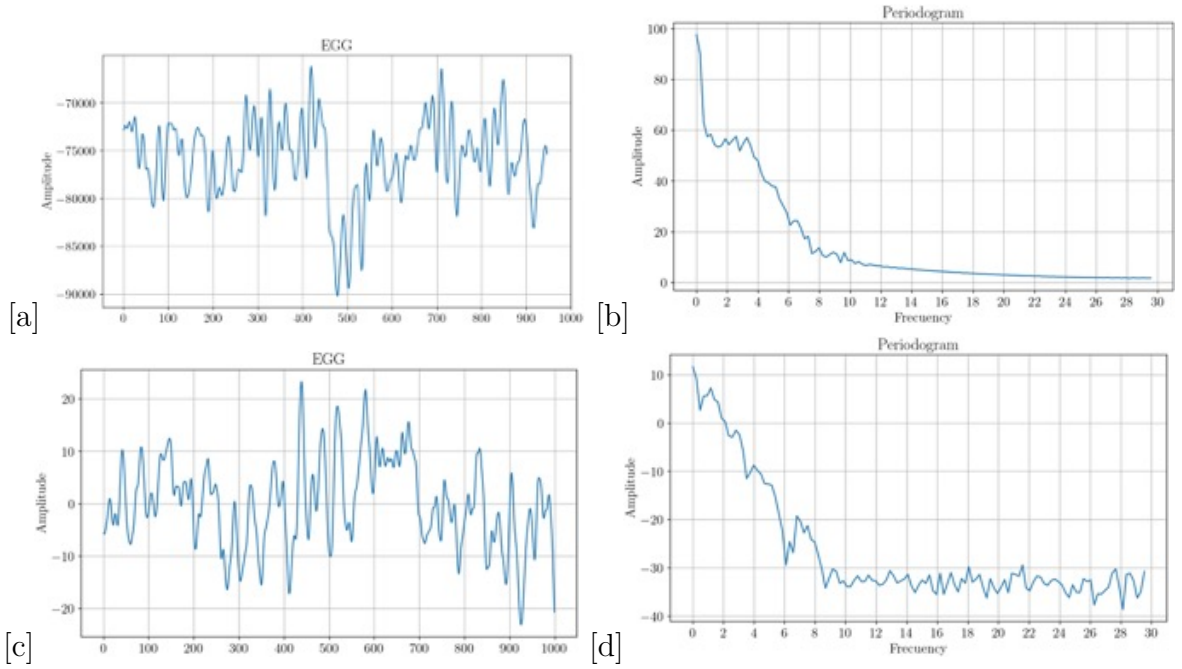


Figure 20: (a) Results of the simulated time series for a small-world network after passing the signal through the butterworth filter (EEG of the small-world network simulation) (b) PSD of the simulated series (c) Real EEG for comparison (d) PSD of a real EEG

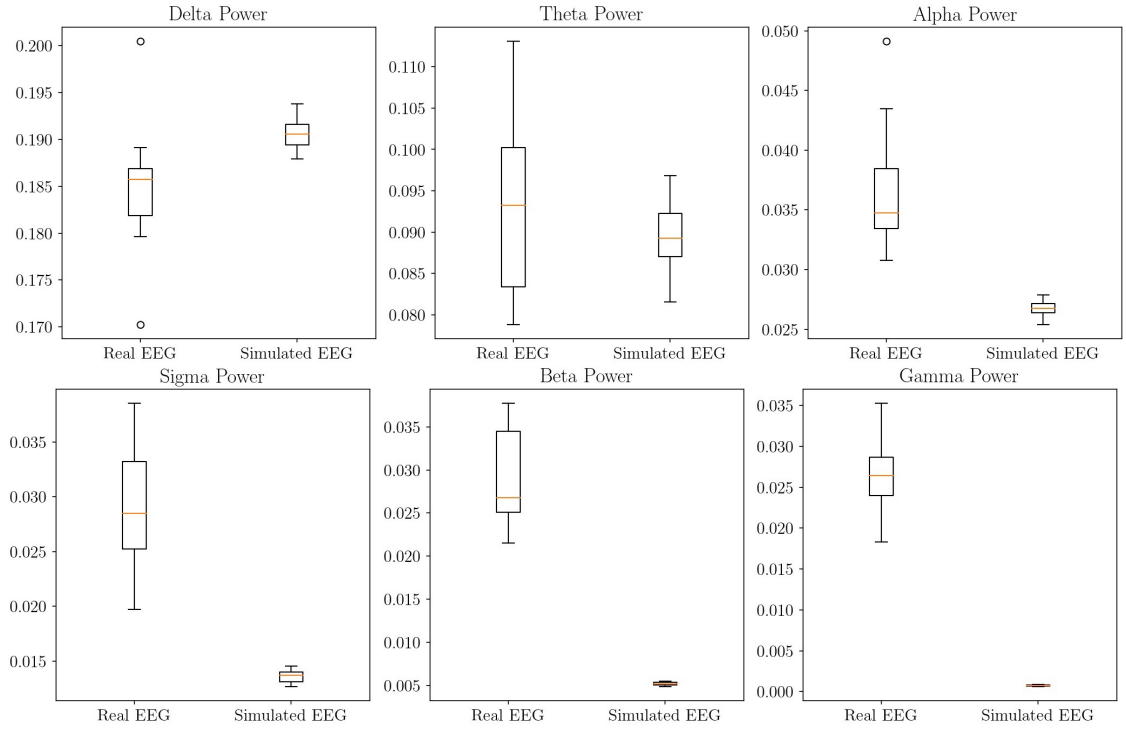


Figure 21: Comparison of the amplitude for each frequency component of the small world network

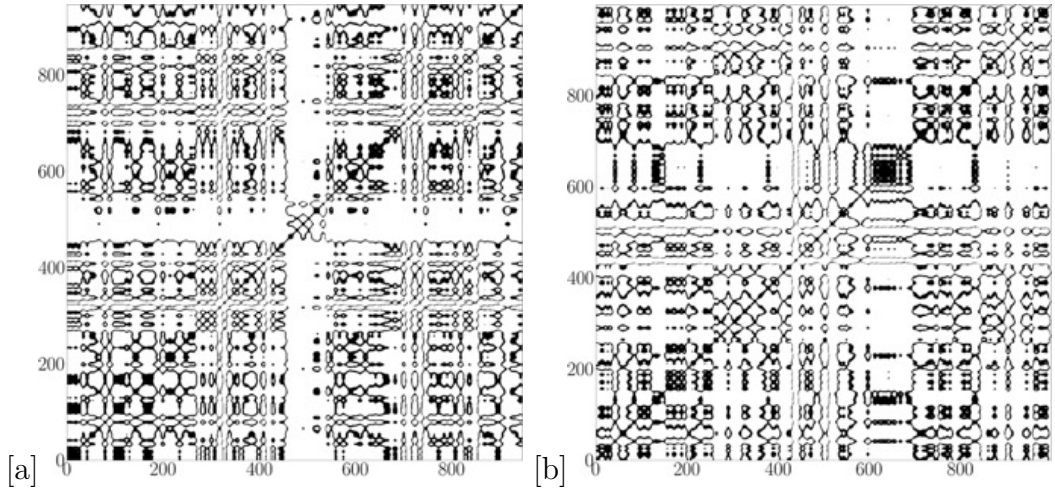


Figure 22: (a) RP of the simulated small network (b) RP of a real EEG

RQA

The measures of RQA for a small-world network exhibit a higher dispersion of values; however, the tendencies and concentration of values are consistent with the RQA measures observed for real data. This observation suggests that the behavior of small-world networks resembles better that of real data thatn the random ones. Fig. 23 shows the comparison between the results for the real and simulated data, and an example of an RP for each one shown in Fig. 22

Fractal analysis

The fractal analysis measures applied to the small-world network exhibit similar scales to the real data. Although their values are not identical, they display a comparable behavior, with overlapping valueintervals and similar dispersions, as depicted in Fig. 24. Those observed similarities highlight the capability of the small-world network to effectively simulate the characteristics of the real data.

Entropy analysis

In the case of the small-world network, the entropy measures exhibit similar scales and central values than the experimental ECGs. This is particularly evident in the permutation and multiscale entropy measures, as illustrated in Fig. 25 and Fig. 26 respectively. The multiscale entropy, in particular, shows a greater degree of similarity when calculated for longer sections of the signal. This observation suggests that in larger scales the simulation exhibits a significantly similar behavior to the real data.

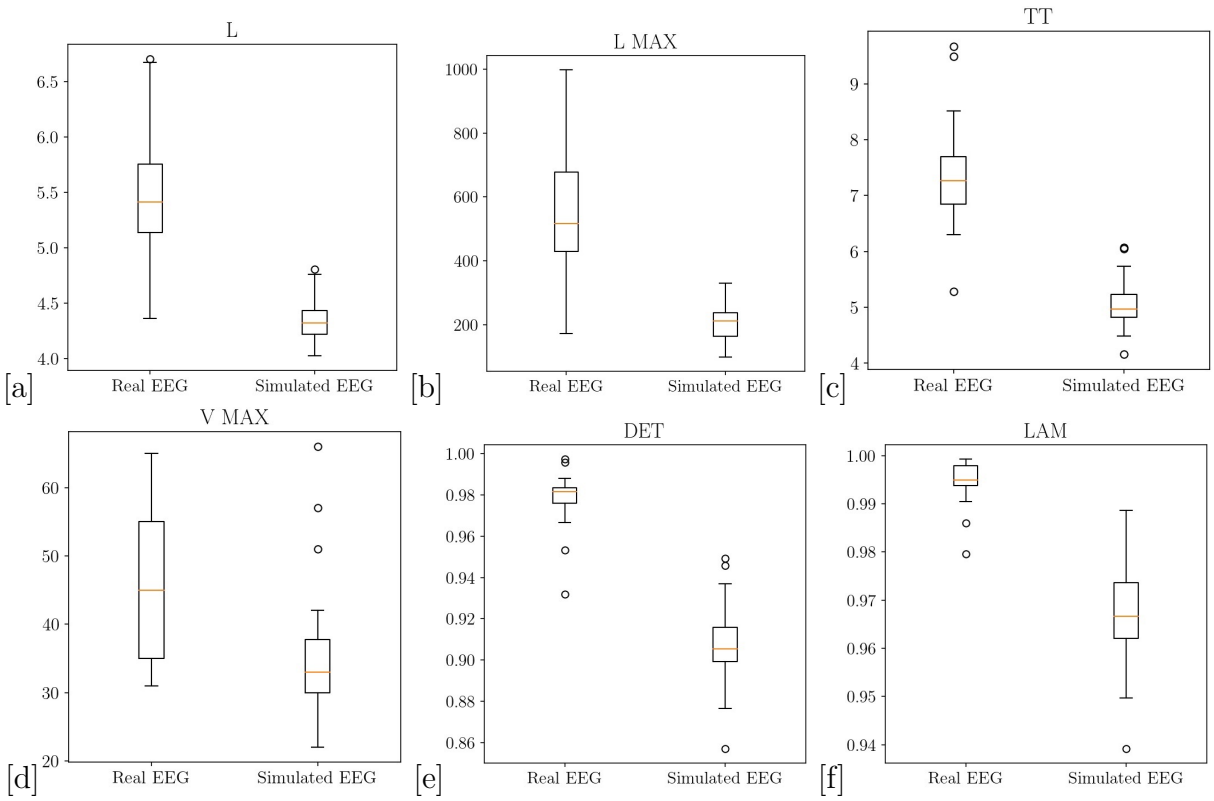


Figure 23: Box plots comparison of six RQA features of real and simulated small-world network EEG time series

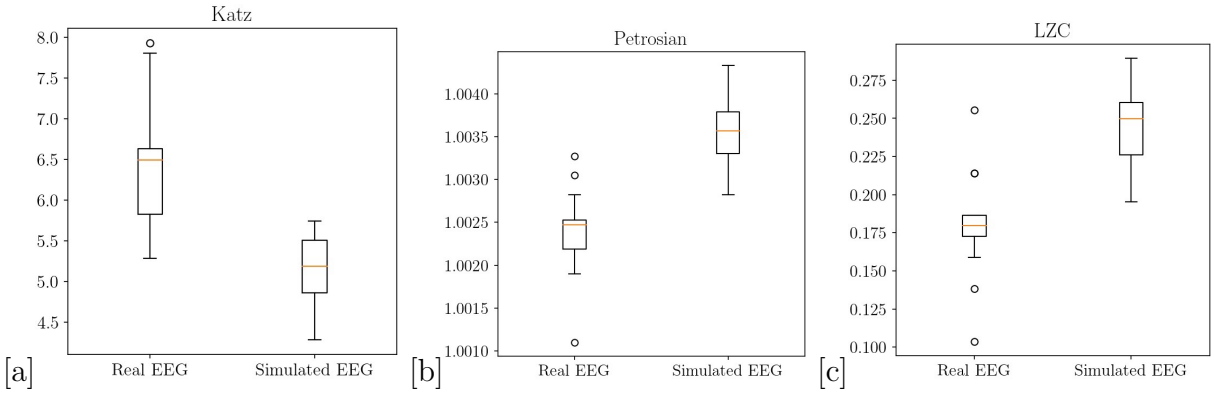


Figure 24: Complexity measures comparison of real EEG and Small world network generated EEG

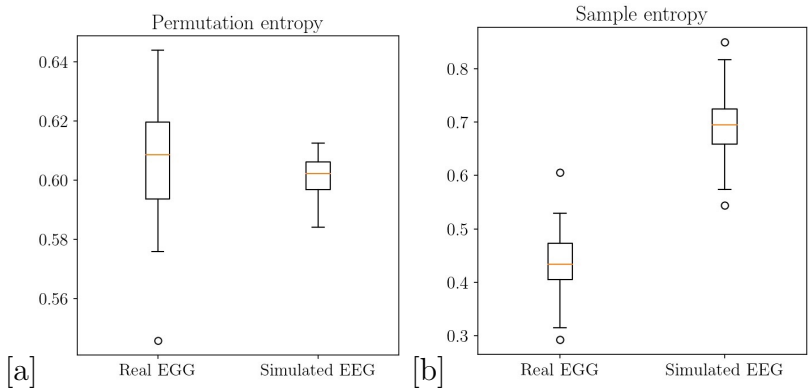


Figure 25: Entropy measures for a simulated small world network and real data

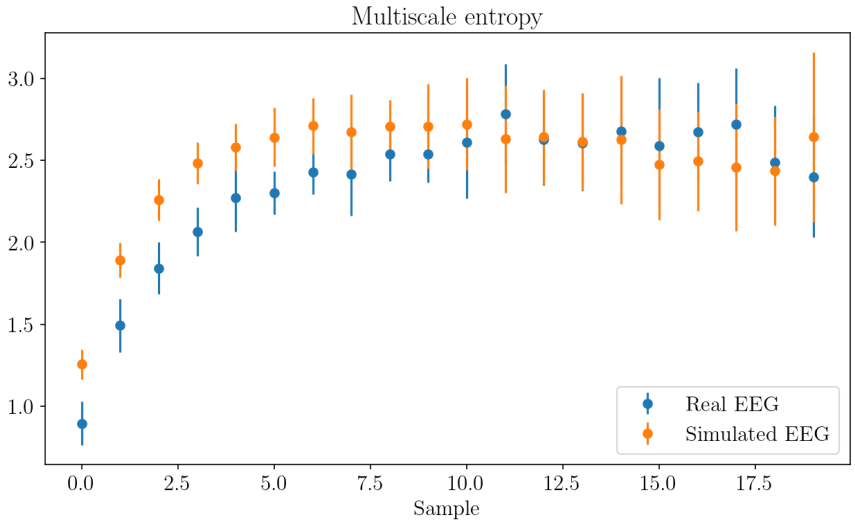


Figure 26: Multiscale entropy for real and small-world network simulated EEG

5.3 Local non uniform network

Effect of node degree in the network

During the construction of the networks investigated in this study, a lattice of cells was created where each cell was connected to the same number of first, second, etc. neighboring cells. However, it was observed that, after some time, those networks exhibited only two distinct stationary behaviors, regardless of other parameters: Some networks converges to a constant value whereas other ones oscillate at a constant frequency.

To address this challenge, a solution was implemented by assigning a different number of closed (first, second, etc.) neighbors to each cell within the array. This modification led to the emergence of three distinct behavioral patterns: Whereas some networks show the two stationary behaviors we found for regular grids, some others exhibited a chaotic behavior more similar to the experimental signals.Figure 27 illustrates those three different results obtained from the same program but with different seed values.

For the subsequent analyses, only the simulations showing chaotic characteristics were selected.

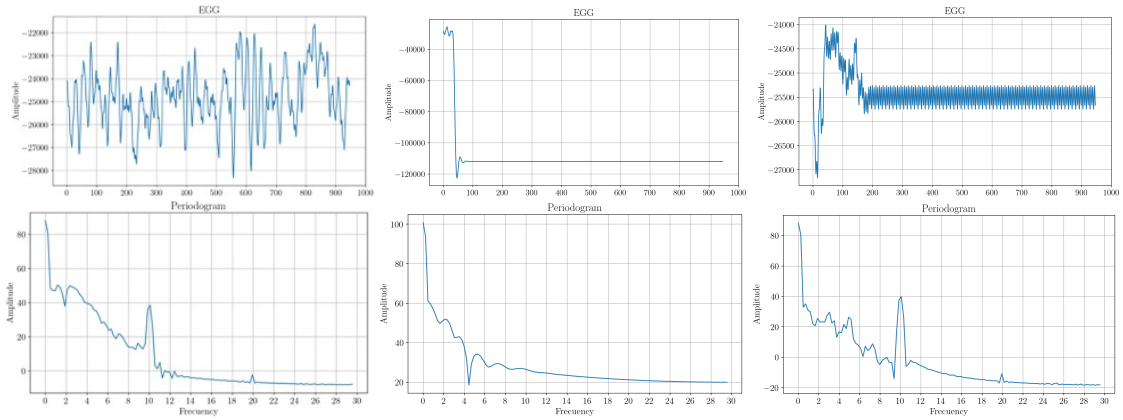


Figure 27: Results of multiple simulations, the top row shows the simulated EEG and bottom row the corresponding periodogram.

Spectral analysis

Within the locally connected network, a filtering process was employed to exclude simulations exhibiting constant or periodic time series. This filtering step effectively reduced the dispersion of amplitudes; nevertheless, the obtained values did not align with those observed in real EEG data. Notably, certain frequency bands exhibited higher presence than anticipated, while others displayed lower presence. Consequently, the simulated frequencies did not match those typically observed in real EEG recordings. A detailed comparison between the power spectral density PSD of real and simulated data is depicted in Fig. 28.

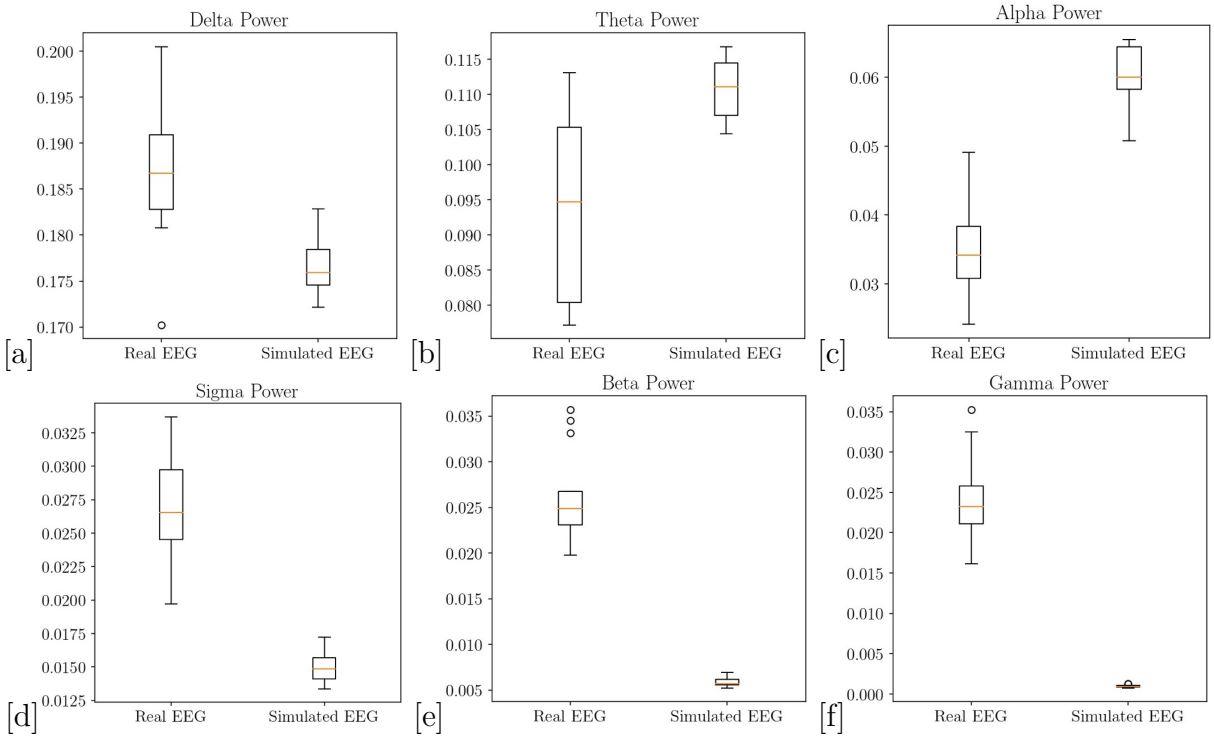


Figure 28: Comparison of the amplitude for each frequency component of the local network

RQA

The analysis using RQA revealed that, overall, the simulated EEG data exhibited trajectory segments that diverged more rapidly compared to the real EEG data. This observation was particularly evident when considering the lowest values of L and l_{max}. Despite the distinct frequency distribution between the simulated and real EEG, the RQA measures showed similarities with those calculated for random and small-world networks. The specific RQA measures are presented in Fig. 30

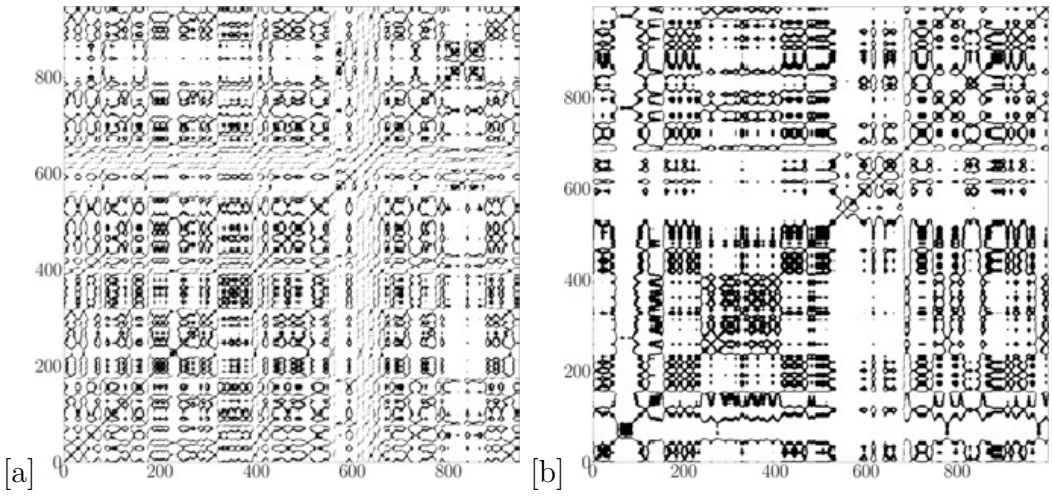


Figure 29: (a) RP of the simulated small network (b) RP of a real EEG

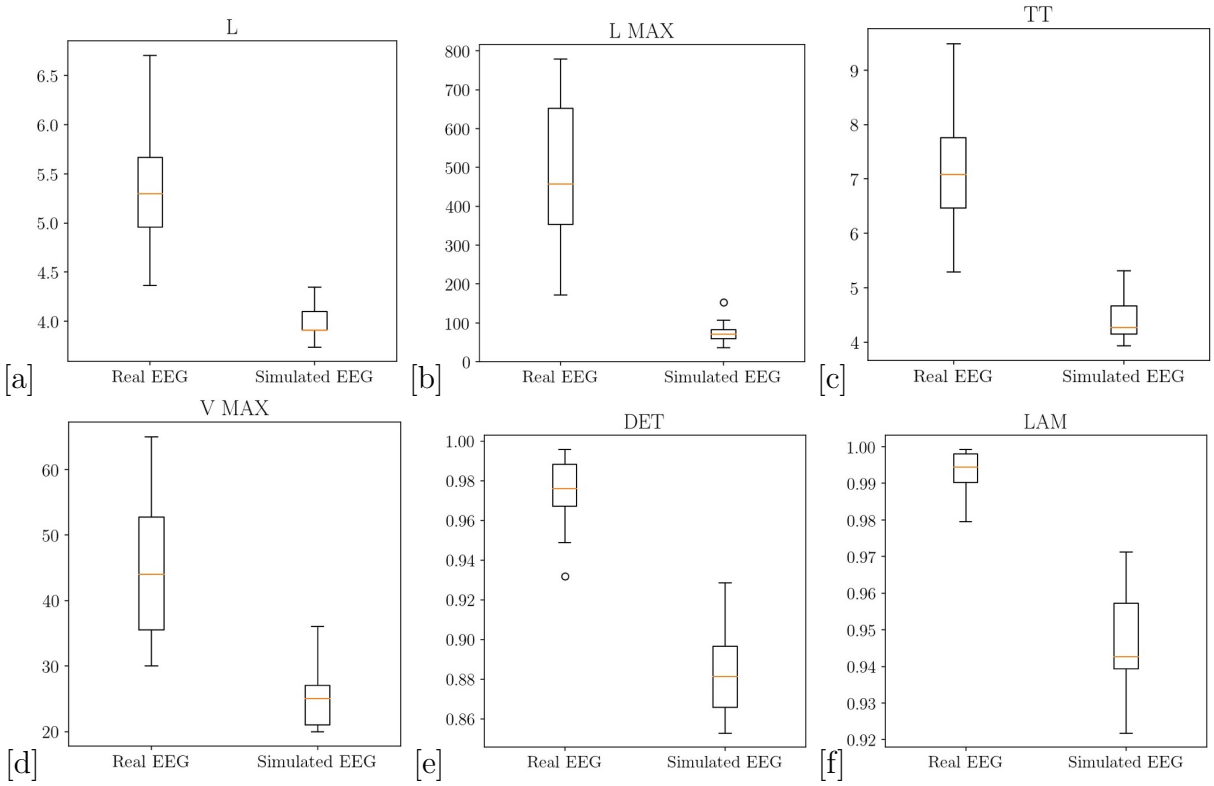


Figure 30: Box plots comparison of six RQA features of real and local network simulated EEG time series

Fractal analysis

In addition to the RQA measures, the Petrosian and fractal dimensions for the local network exhibit similar behavior to the dimensions measured for random and small-world networks. Those findings confirm that, despite the differences in frequency distribution, the three networks kinds generate signals with similar levels of complexity.

Entropy analysis

Permutation and sample entropy analyses reveal that the simulated signal possesses a more intricate and unpredictable structure compared to a real EEG signal. It exhibits a greater diversity of patterns and a higher degree of randomness in the arrangement of its values. Nevertheless, it is important to note that while the simulated signal demonstrates a high degree of local irregularity or complexity, its overall global complexity is relatively lower. This can be observed in the similar values of multiscale entropy to the real signal towards the end of the graph, while higher values are observed for small sizes.

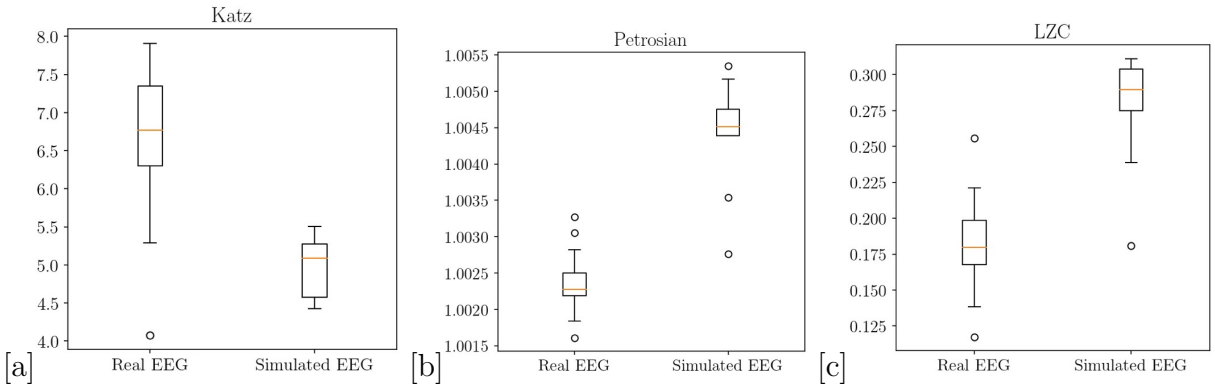


Figure 31: Complexity measures comparison of real EEG and simulated local network EEG

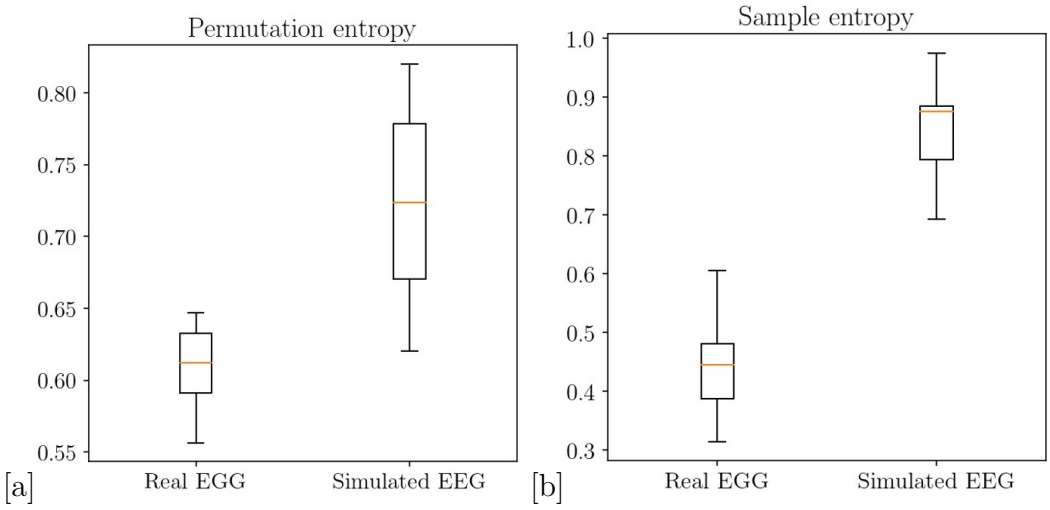


Figure 32: Entropy measures for a simulated random Network and real data

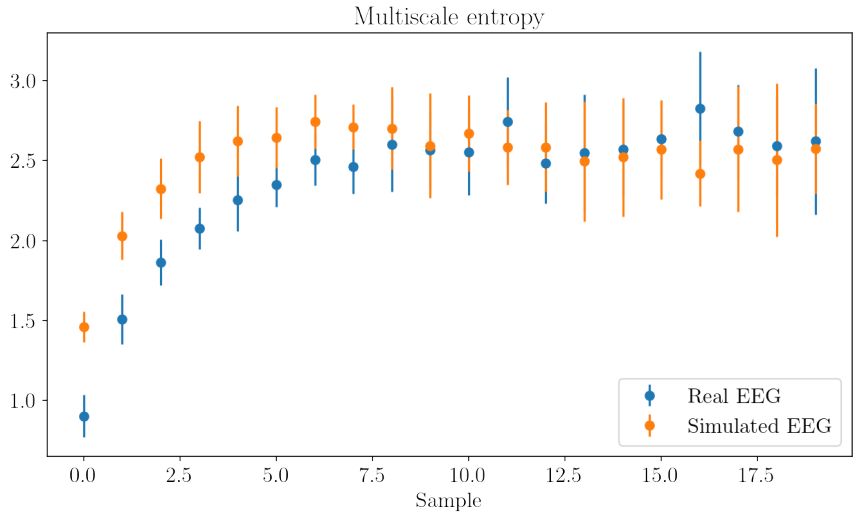


Figure 33: Multiscale entropy for real and local network simulated EEG.

5.4 Comparison between networks

Spectral analysis

In Fig. 34, the comparison of frequency components among the different networks reveals that, in general, all three networks struggle to replicate the results for high-frequency bands. However, when focusing on the lower end of the spectrum, the small-world network exhibits values that are more closer or even fall within the error bars of the real data. This finding indicates that the frequency content of the time series generated by the small-world network closely resembles that of the real data, suggesting a better representation of lower-frequency components compared to the other networks.

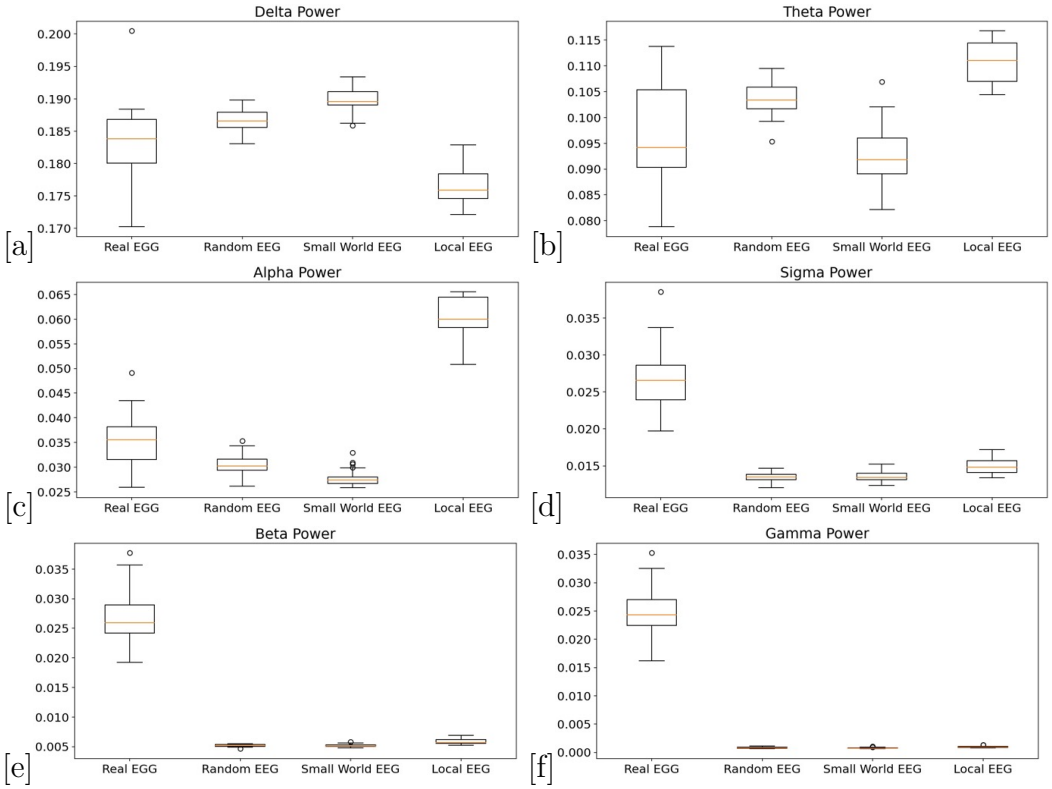


Figure 34: Comparison of the amplitude for each frequency component of real vs simulated EEGs

RQA

The analysis of RQA measures reveals that, overall, all simulated networks exhibit slightly more chaotic behaviors. Nevertheless, among the different networks, the small-world one stands out by showing the most similar values to the real ECGs signals. This finding is depicted in Fig. 35, which presents the comparisons of RQA measures for the networks. The small-world network consistently demonstrates a closer resemblance to the patterns and dynamics observed in the real data.

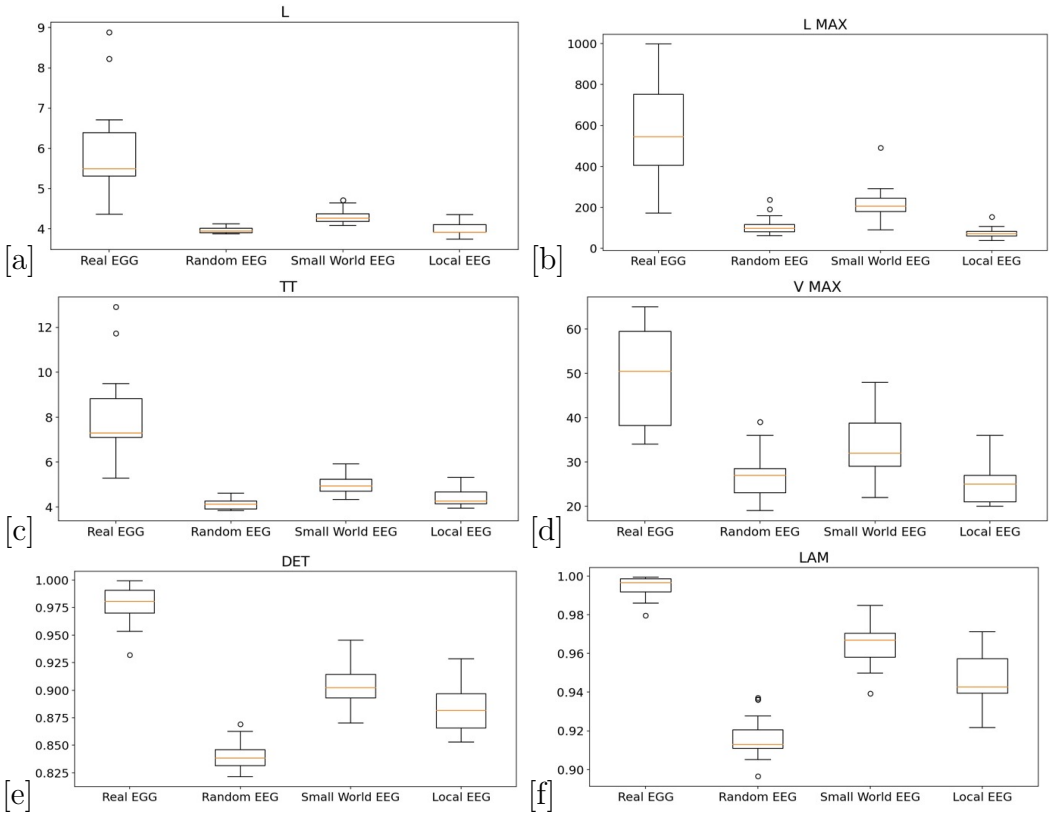


Figure 35: Box plots comparison of six RQA features of real and local network simulated EEG time series

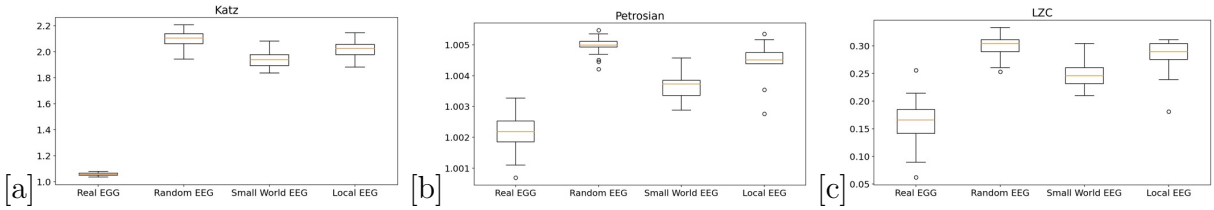


Figure 36: Complexity measures comparison of real EEG and simulated local network EEG

Fractal analysis

The analysis of complexity measures indicates that the simulated EEG exhibits higher complexity compared to the real data. However, it is noteworthy that the small-world network consistently demonstrates the closest resemblance to the real data, as observed in various complexity measures. Despite the overall higher complexity of the simulated EEG, the small-world network stands out as the network configuration that closely captures important aspects of complexity found in real data. This observation is illustrated in Fig. 36

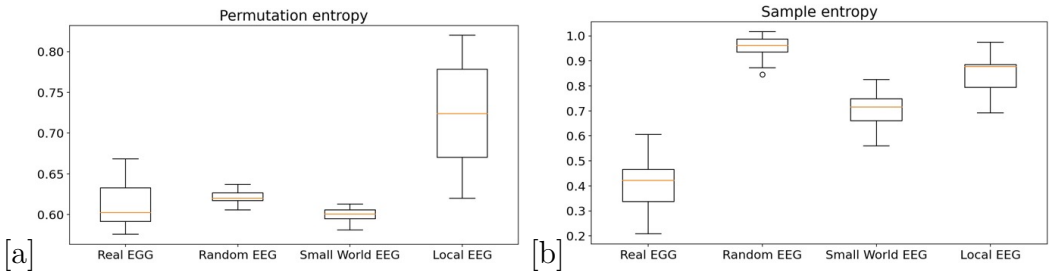


Figure 37: Entropy measures for simulated and real data

Entropy analysis

In the entropy analysis, all three networks show different patterns. The sample entropy of the real data consistently exhibits lower values compared to that of simulated networks. Among the simulated networks, the sample entropy of the small world network closely resembles the pattern observed in the real data. Indeed, in the case of permutation entropy, only the small world and random networks exhibit values within the range of real permutation entropy.

Further analysis using multiscale entropy reveals that the small world network exhibits the most similar results in terms of both local and overall complexity. Those findings are visually represented in Fig. 37 and Fig. 38.

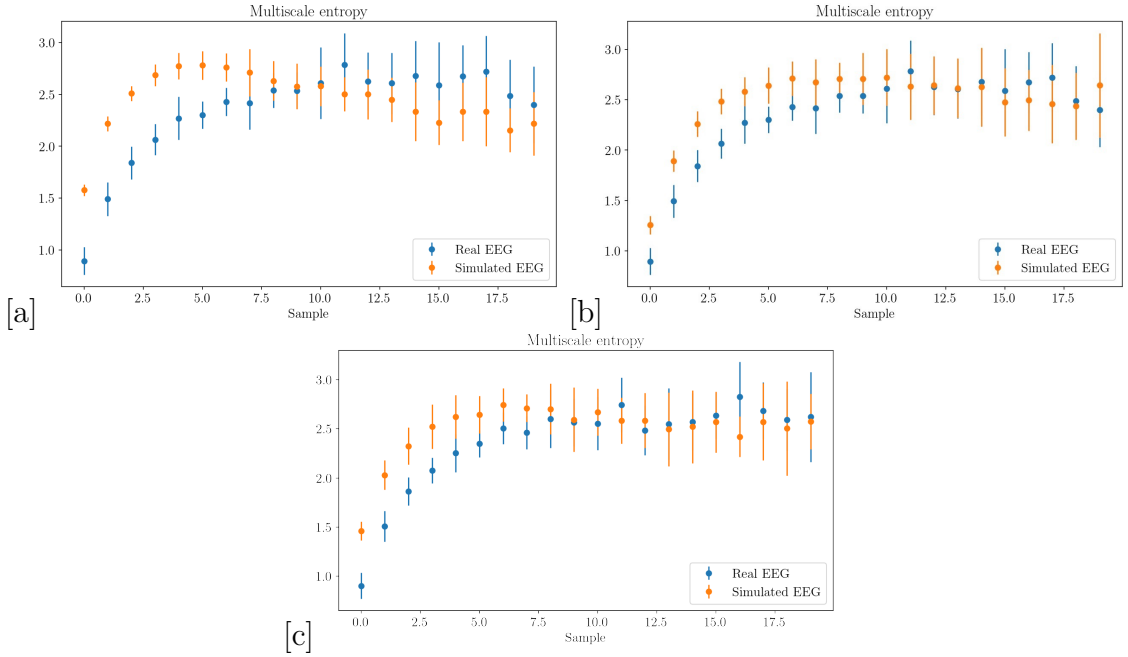


Figure 38: (a) multiscale entropy for a Random network (b) multiscale entropy for a small world network (c) multiscale entropy comparison for local and

This comparison among networks against the real ECGs datafiles finishes our results and analysis.

6 Conclusions

The present work implements on three different network geometries the cellular automaton model proposed by Khaleghi et al. [1] for synthesizing an EEG signal and compares them against experimental measurements. The three network geometries we studied were: random networks with nodes with Gaussian distributed degrees, small-world network generated using the Watts-Strogatz algorithm and local neighborhood networks where each cell has a different number of neighbors. the number of inhibitory cells was increased to 50%, and the threshold levels T_{rest} and $T_{relative}$ were set to obtain a chaotic signal in the three network kinds, resembling that of experimental EEGs. Next, we implemented the analysis tools outlined in Khaleghi et. al.'s paper. Those tools primarily involved developing C++ code and utilizing pre-existing libraries specifically designed for measures such as Spectral Analysis, Recurrence Quantification Analysis (RQA), fractal measurements and entropy calculations. Those results are then employed to compare among networks and see which geometries better resembles the experimental signals.

The study successfully determined the key parameters required for constructing and analyzing each network, including tolerances, embedding dimensions, delay degrees, and electric potential finding different parameters for random, small-world and local networks. Through a systematic search, we determined that one of the thresholds T_{rest} or $T_{relative}$ could have high values while ensuring that the other threshold remained low, even below 2. Adherence to the guiding principles of these thresholds, which reflect biological behaviors where a neuron can be excited by a slightly higher potential after firing, led to the selection of specific values of 3 and 5 for T_{rest} and $T_{relative}$ respectively to be used in our CA models.

The results of our research reveal that the chosen random, small-world, and local networks successfully replicate a chaotic behavior resembling real electroencephalogram (EEG) data. The random and small-world networks consistently exhibit this desired pattern when appropriate parameters are chosen. However, it was observed that the behavior of the local networks is highly sensitive to the distribution of node degrees. Even minor variations in the random seed used to generate connections can lead to stationary behaviors, such as periodic or constant values, rather than the intended chaotic dynamics.

The spectral analysis of the simulated signals show that all three geometries show similar power contents as the real EEG signals for low frequency bands (delta, theta and alpha), bur lower power content at high frequency ones (sigma, beta and gamma) than the Khaleghi et al. results and the real ac EEGs. This finding suggests that a high fraction of inhibitory cells diminishes high frequencies but not low ones. In addition, from the three geometries the small-world network were the ones that better approximated the power contents at low frequency bands.

By performing a Recurrence Quantification analysis (RQA), we obtained that our simulated signals better resembles the RQA maps of EEG recorded when the individuals were performing arithmetic operations. When the several quantities obtained from RQA (average diagonal line length L, Maximum diagonal line L_{max}, determinism, Trapping time TT, Maximum vertical line V_{max}, and laminarity) were computed, we obtained that all three network geometries exhibit similar values, a littule bit lower thatn the ones for real ECGs. In contrast, complexity measurements like Katz and Petrosian fractal dimensions and Lempel-Ziv complexity show to be also similar for the three network geometries, but

higher than the ones for the experimental signals. In contrast, the results by Khaleghi et al. show similar ranges to the experimental ones, except by the Petrosian fractal dimension. Those results could also be related to the difference in the fraction of inhibitory signals, and this would be an interesting issue of future research.

When looking at entropy measurements, sample entropy for random and small-world networks shows to be similar to the experimental results, but the range obtained for local networks is higher. Permutation entropy shows higher than the experimental data for all three geometries. In contrast, multiscale entropy shows to be quite similar to experimental results for small-world and local geometries at all scales, but random networks show higher values at small scales.

In general, small-world networks are the ones that closer resembles the characteristics of real ECG signals, suggesting that the small world network is better suited to replicate the authentic behavior of the system. Nevertheless, random networks exhibit a behavior similar to that observed in active mental states. This finding suggests that the presence of more long-distance connections, as opposed to predominantly local connections, can lead to heightened brain activity and increased complexity in the system. These findings provide insights into the role of network connectivity in generating high levels of brain activity and highlight the importance of considering the arrangement of connections when studying neural dynamics.

The observed change in behavior during the study of a local and random network highlights the importance of conducting a more extensive investigation into the influence of variable degrees per node in networks composed of nearest neighbors, as well as in random and small-world networks. Such a study would provide valuable insights into the unique characteristics and dynamics exhibited by those networks. By exploring the impact of varying node degrees on system behavior, a deeper understanding of the underlying mechanisms governing network dynamics could be achieved.

The interplay between the threshold values, t_{rest} and $t_{relative}$, plays a crucial role in determining the evolutionary dynamics of the system. It governs whether the system will continue to evolve or become stuck in a stagnant state. Exploring the boundaries of these threshold combinations, where the system transitions from evolution to stagnation, presents an intriguing avenue for investigation. Additionally, studying the diverse behaviors exhibited by the system under various choices of t_{rest} and $t_{relative}$ opens up exciting possibilities for further research. Unraveling the intricate relationship between threshold values and system behavior can deepen our understanding of complex dynamics in the context of this study.

During the implementation of the Watts-Strogatz model in this study, we used a single probability value, denoted as p, for rewiring links to randomly chosen nodes. It is important to note that the choice of this value was determined by the study's scope and objectives. Nevertheless, it is worth mentioning that a comprehensive analysis of the impact of different p values on the network's behavior remains an open area of study. Exploring the influence of p in greater depth holds significant potential for uncovering valuable insights into the behavior and dynamics of the network under varying rewiring conditions. Therefore, further research dedicated to investigating the effects of different p values is recommended to advance our understanding of complex network phenomena.

The contribution of this work to the area lies in its identification of significant findings

related to network behavior and connectivity. The study highlights the strong association between node degree and network behavior, showcasing diverse outcomes even with similar number of connections. The possibilities of each network to replicate real data is shown, suggesting that increased long-distance connections contribute to heightened brain activity and complexity. In addition, interpretations for the analysis measures were provided, serving as a comprehensive guide for researchers interested in conducting further investigations in this field. The study also provides valuable insights by determining key parameters for network construction and analysis and offering interpretations of analysis measures, serving as a comprehensive guide for future research in this field.

References

[1] Ali Khaleghi, Mohammad Reza Mohammadi, Kian Shahi, and Ali Motie Nasrabadi. A neuronal population model based on cellular automata to simulate the electrical waves of the brain. *Waves in Random and Complex Media*, pages 1–20, 2021.

[2] Vassilios Tsoutsouras, Georgios Ch Sirakoulis, Georgios P Pavlos, and Aggelos C Iliopoulos. Simulation of healthy and epileptiform brain activity using cellular automata. *International Journal of Bifurcation and Chaos*, 22(09):1250229, 2012.

[3] Noël Bonnet, Manuela Matos, Myriam Polette, J-M Zahm, Béatrice Nawrocki-Raby, and Philippe Birembaut. A density-based cellular automaton model for studying the clustering of noninvasive cells. *IEEE transactions on biomedical engineering*, 51(7):1274–1276, 2004.

[4] A.T. Skjeltorp and A.V. Belushkin. *Dynamics of Complex Interconnected Systems: Networks and Bioprocesses*. NATO Science Series II: Mathematics, Physics and Chemistry. Springer Netherlands, 2006.

[5] Edmund T Rolls and Alessandro Treves. The neuronal encoding of information in the brain. *Progress in neurobiology*, 95(3):448–490, 2011.

[6] Gaute T Einevoll. Mathematical modeling of neural activity. In *Dynamics of Complex Interconnected Systems: Networks and Bioprocesses*, pages 127–145. Springer, 2006.

[7] Xue Fan and Henry Markram. A brief history of simulation neuroscience. *Frontiers in Neuroinformatics*, 13, 2019.

[8] Wulfram Gerstner, Henning Sprekeler, and Gustavo Deco. Theory and simulation in neuroscience. *science*, 338(6103):60–65, 2012.

[9] Katharina Glomb, Joana Cabral, Anna Cattani, Alberto Mazzoni, Ashish Raj, and Benedetta Franceschiello. Computational models in electroencephalography. *Brain topography*, pages 1–20, 2021.

[10] Monica Cusenza. Fractal analysis of the eeg and clinical applications. 2012.

[11] C.H. Im. *Computational EEG Analysis: Methods and Applications*. Biological and Medical Physics, Biomedical Engineering. Springer Nature Singapore, 2018.

[12] JAMES ARTHUR MORTIMER. *A cellular model for mammalian cerebellar cortex*. PhD thesis, University of Michigan, 1970.

[13] JJ Wright. Simulation of eeg: dynamic changes in synaptic efficacy, cerebral rhythms, and dissipative and generative activity in cortex. *Biological cybernetics*, 81(2):131–147, 1999.

[14] Luis Acedo. A cellular automaton model for collective neural dynamics. *Mathematical and Computer Modelling*, 50(5-6):717–725, 2009.

[15] Robert Kozma and Walter J Freeman. *Cognitive phase transitions in the cerebral cortex-enhancing the neuron doctrine by modeling neural fields*. Springer, 2016.

[16] L Acedo, Erasmia Lamprianidou, J-A Moraño, J Villanueva-Oller, and R-J Villanueva. Firing patterns in a random network cellular automata model of the brain. *Physica A: Statistical Mechanics and its Applications*, 435:111–119, 2015.

[17] JW Sleigh and DC Galletly. A model of the electrocortical effects of general anaesthesia. *British journal of anaesthesia*, 78(3):260–263, 1997.

[18] Gwilym M Jenkins. Spectral analysis and its applications. *Holden-Day, Inc., San Francisco, Card Nr. 67-13840*, 1968.

[19] David B. Preston. Spectral analysis and time series. *Technometrics*, 25:213–214, 1983.

[20] Wolfgang Klimesch. Eeg alpha and theta oscillations reflect cognitive and memory performance: A review and analysis. *Brain Research Reviews*, 29(2-3):169–195, 1999.

[21] Wolfgang Klimesch, Paul Sauseng, and Simon Hanslmayr. Eeg alpha oscillations: The inhibition-timing hypothesis. *Brain Research Reviews*, 53(1):63–88, 2007.

[22] Wolfgang Klimesch. Alpha-band oscillations, attention, and controlled access to stored information. *Trends in Cognitive Sciences*, 16(12):606–617, 2012.

[23] Andreas K. Engel, Pascal Fries, and Wolf Singer. Dynamic predictions: Oscillations and synchrony in top-down processing. *Nature Reviews Neuroscience*, 2(10):704–716, 2001.

[24] Erol Başar, Canan Başar-Eroglu, Sirel Karakaş, and Martin Schürmann. Are cognitive processes manifested in event-related gamma, alpha, theta and delta oscillations in the eeg? *Neuroscience Letters*, 1-4(29):145–148, 2000.

[25] Chris Chatfield. Some comments on spectral analysis in marketing. *Journal of Marketing Research*, 11:101 – 97, 1974.

[26] E THOMAS, Shajimon K John, and Susan Abe. Power spectral density computation using modified welch method. *IJSTE-International Journal of Science Technology & Engineering*, 2(4), 2015.

[27] OM Solomon Jr. Psd computations using welch's method [power spectral density (psd)]: Nasa sti. Technical report, Recon Technical Report. 1991. doi: 10.2172/5688766.

[28] Norbert Marwan, M. Carmen Romano, Marco Thiel, and Jürgen Kurths. Recurrence plots for the analysis of complex systems. *Physics Reports*, 438(5):237–329, 2007.

[29] N. Kasabov. *Springer Handbook of Bio-/Neuro-Informatics*. Springer Handbooks. Springer Berlin Heidelberg, 2013.

[30] Linhua Deng. Nonlinear interrelation of chaotic time series with wavelet transform and recurrence plot analyses. In *CiiT international journal of digital image processing*, 2014.

[31] Michael J. Katz. Fractals and the analysis of waveforms. *Computers in Biology and Medicine*, 18(3):145–156, 1988.

[32] N.V. Thakor. *Handbook of Neuroengineering*. Handbook of Neuroengineering. Springer Nature Singapore, 2023.

[33] Diego C Nascimento, Gabriela Depetri, Luiz H Stefano, Osvaldo Anacleto, Joao P Leite, Dylan J Edwards, Taiza EG Santos, and Francisco Louzada Neto. Entropy analysis of high-definition transcranial electric stimulation effects on eeg dynamics. *Brain sciences*, 9(8):208, 2019.

[34] Joshua S Richman and J Randall Moorman. Physiological time-series analysis using approximate entropy and sample entropy. *American journal of physiology-heart and circulatory physiology*, 2000.

[35] Jue Lu and Ze Wang. The systematic bias of entropy calculation in the multi-scale entropy algorithm. *Entropy*, 23, 2021.

[36] J. Amigó. *Permutation Complexity in Dynamical Systems: Ordinal Patterns, Permutation Entropy and All That*. Springer Series in Synergetics. Springer Berlin Heidelberg, 2010.

[37] Joaquín Araya-Arriagada, Sebastián Garay, Cristóbal Rojas, Claudia Duran-Aniotz, Adrián G Palacios, Max Chacón, and Leonel E Medina. Multiscale entropy analysis of retinal signals reveals reduced complexity in a mouse model of alzheimer's disease. *Scientific Reports*, 12(1):8900, 2022.

[38] Mehran Talebinejad, Georgios Tsoulfas, and Sam Musallam. Level-ziv complexity for analysis of neural spikes. *CMBES Proceedings*, 34, 2011.

[39] Fatih Yavuz Ilgin. Spectrum sensing algorithm based on shapiro wilk test. *NWSA Academic Journals*, 2022.

[40] Elizabeth González-Estrada and Waldenia Cosmes. Shapiro–wilk test for skew normal distributions based on data transformations. *Journal of Statistical Computation and Simulation*, 89:3258 – 3272, 2019.

[41] Abdellah Nabou, My driss Laanaoui, Mohammed Ouzzif, et al. Shapiro-wilk test to detect the routing attacks in manet. 2021.

[42] A.L.Ã.Ã. BarabÃ¡si and M.Ã. PÃ3sfai. *Network Science*. Cambridge University Press, 2016.

[43] Ary L. Goldberger, Luis A. N. Amaral, Leon Glass, Jeffrey M. Hausdorff, Plamen Ch. Ivanov, Roger G. Mark, Joseph E. Mietus, George B. Moody, Chung-Kang Peng, and H. Eugene Stanley. PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, June 2000.

[44] Igor Zyma, Sergii Tukaev, Ivan Seleznov, Ken Kiyono, Anton Popov, Mariia Chernykh, and Oleksii Shpenkov. Electroencephalograms during mental arithmetic task performance. *Data*, 4(1), 2019.

[45] Bob Kemp and Jesus Olivan. European data format 'plus' (edf+), an edf alike standard format for the exchange of physiological data. *Clinical Neurophysiology*, 114(9):1755–1761, 2003.

A

Preprocessing of the real data

Listing 1: Load real EEG .edf files

```python
import mne
def get_real_data(file, col_name = 'EEG F3'):
    data___ = mne.io.read_raw_edf(file)
    raw_data = data___.get_data()
    df_data_ = data___.to_data_frame()[:1000]
    return df_data_[col_name].to_numpy()
```

B

Random network implementation of base CA

Listing 2: Ali Khaleghi base model with a random network

```
1
2  #include <iostream>
3  #include<fstream>
4  #include <cmath>
5  #include "Random64.h"
6
7  using namespace std;
8
9  const int N = 40; // Grid size (number of neurons) NxN
10 const int N2 = N * N;
11 const int Q = 20; // Interacting partner (Synapse) can vary per cell
12 const int C = N2 * Q; // Number of connections
13
14 const double Trest=3;//rest threshold
15 const double Trelative=5;//refractory threshold
16 const double alpha = 0.1; // HYPERPOLARIZATION COEFFICIENT
17
18 class NeuralNetwork
19 {
20 private:
21   int *AP, *APNew; // Power vector
22   int **Connections_per_neuron; // Connections matrix
23   bool IE[N2]; // Cell type vector
24   int Con[C][2]; // Connections list
25   int connection_count[N2];
26
27 public:
28   NeuralNetwork(void); // Constructor
29   ~NeuralNetwork(void); // Destructor
30   void Initialize(Crandom &ran64); // Fill the matrix, assign initial
       potentials with random values between 0 and 10, and the cell type (
       inhibitory or excitatory)
31   void ConnectionGenerator(void); // Generate random connections
32   enum state{rest, firing, hyperpolarization, refractory}; // What are
       the states
33   state GetState(int ix); // Return the state of the neuron based on its
        potential
34   double Query(int neu); // Read the potentials of other neurons, count
35   double Potential(int ix); // Translate the states to a physical
       potential
```

```
36    double TemporalSeries(); // Sum all the potentials at time t
37    double ActiveNeurons(); // All active neurons at time t
38    void Evolve(); // Time evolution update
39
40    };
41
42  NeuralNetwork::NeuralNetwork(void)
43  {
44    // Create vectors
45    AP = new int[N2];
46    APNew = new int[N2];
47
48    // Initialize the base line of the connection_count vector
49    for (int i = 0; i < N2; i++)
50    {
51      connection_count[i] = 0;
52    }
53
54    // Create the connections list (Con)
55    ConnectionGenerator();
56
57    // Fill the connection_count vector by checking all possible
        connections
58    for (int ix = 0; ix < C; ix++)
59    {
60      connection_count[Con[ix][0]]++;
61      connection_count[Con[ix][1]]++;
62    }
63
64    // Take the results from connection_count and allocate memory to form
        lists for each neuron
65    Connections_per_neuron = new int*[N2];
66    for (int ix = 0; ix < N2; ix++)
67    {
68      Connections_per_neuron[ix] = new int[connection_count[ix]];
69    }
70  }
71
72  NeuralNetwork::~NeuralNetwork(void)
73  {
74    delete[] AP;
75    delete[] APNew;
76
77    for (int ix = 0; ix < N; ix++)
78    {
79      delete[] Connections_per_neuron[ix];
80    }
81    delete[] Connections_per_neuron;
82  }
83
84
85
86  void NeuronNetwork::Start(Crandom & ran64)
87  {
88    double pot,inhib;
89
90
91    for (int ix = 0; ix<N2; ix++)
92    {
```

```
93      pot=ran64.r();
94
95      //Nrest = 20%, Nfiring = 40%, Nrefractory = 35% and
            Nhyperpolarization = 5%.
96      if ( pot < 0.2) {AP[ix]=0;}
97      else if (0.2<= pot && pot < 0.3){AP[ix]=1;}
98      else if (0.3<= pot && pot < 0.4){AP[ix]=2;}
99      else if (0.4<= pot && pot < 0.5){AP[ix]=3;}
100     else if (0.5<= pot && pot < 0.6){AP[ix]=4;}
101     else if (0.6<= pot && pot < 0.65) {AP[ix]=5;}
102     else if (0.65<= pot && pot < 0.72){AP[ix]=6;}
103     else if (0.72<= pot && pot < 0.79){AP[ix]=7;}
104     else if (0.79<= pot && pot < 0.86){AP[ix]=8;}
105     else if (0.86<= pot && pot < 0.93){AP[ix]=9;}
106     else if (0.93 <= pot){AP[ix]=10;}
107
108     APNew[ix]=0;
109   }
110
111
112   for (int ix = 0; ix<N2; ix++)
113    {
114     inhib=ran64.r();//type of distribution (you can try to use a
            different distribution)
115
116     if (inhib<0.5)
117     {IE[ix]=true;}
118     else
119     {IE[ix]=false;}
120
121    }
122
123   //make a list for each neuron
124
125   int counter;
126
127   for(int neu=0; neu<N2; neu++)
128   {
129     counter = 0;
130     for(int conex = 0; conex < C; conex++)
131     {
132       if(Con[conex][0] == neu)
133       {
134         Conexiones_por_neurona[neu][counter] = Con[conex][1];
135         counter++;
136       }
137       else if(Con[conex][1] == neu)
138       {
139         Conexiones_por_neurona[neu][counter] = Con[conex][0];
140         counter++;
141       }
142     }
143
144   }
145 }
146
147 void NeuronNetwork::connections_generator(void)
148 {
149   //Crea la lista de conxiones (Con)
```

```
150
151   Crandom ranita(9);
152
153   int iran = 0;
154   int jran = 0;
155   int coni=0;
156
157   while (coni<C)
158   {
159     //Escoger una posible conexion
160     iran = (int)(ranita.r()*N2);
161     jran = (int)(ranita.r()*(N2-1));
162
163     if(jran>=iran) jran++;
164
165
166     bool NoEstabaAntes=true;
167
168     //revisar que no estuviera antes
169
170     for(int i=0;i<coni;i++)
171     {
172       if((Con[i][0]==iran && Con[i][1]==jran) || (Con[i][0]==jran && Con
              [i][1]==iran))
173       {NoEstabaAntes=false; break;}
174     }
175
176     //Crear una óconexin que no estaba
177
178     if (NoEstabaAntes)
179     {
180       //Si no estaba antes, adicionarla a la lista
181       Con[coni][0]=iran; Con[coni][1]=jran;
182       coni++;
183     }
184   }
185 }
186
187 RedNeuronas::estado RedNeuronas::cualestado(int ix)
188 {
189   estado S;
190   if (AP[ix]== 0) S=reposo;
191   else if (1<=AP[ix] && AP[ix]<=4) S=disparo;
192   else if (AP[ix]== 5) S=hiperpolarizacion;
193   else if (6<=AP[ix] && AP[ix]<=10) S=refractario;
194
195   return S;
196 }
197
198
199
200 double RedNeuronas::Informese(int neu)
201 {
202   estado Sconexion, Sveci;
203   double Ca, Ce, Ci, Ch;
204   Ca=Ce=Ci=Ch=0.0;  //contador para saber cuantos vecionos son Ce (
        neuronas excitatorias activas), cuantos son Ci (neuronas
        inhibitorias activas) y cuantos Ch
205
```

```
206   //Lea el S de sus vecinos
207
208     for (int conex = 0; conex < cuantas_conexiones[neu]; conex++)
209     {
210       Sconexion=cualestado(Conexiones_por_neurona[neu][conex]);
211       if (Sconexion==disparo)
212       {
213         if (IE[conex]==true) {Ci++;}
214         else {Ce++;}
215       }
216       else if (Sconexion==hiperpolarizacion){Ch++;}
217     }
218
219     for (int veci = 1; veci <= 2; veci++)
220     {
221       int vecid =(neu+veci)%N2;
222       int vecii=(neu+N2-veci)%N2;
223
224       Sveci=cualestado(Conexiones_por_neurona[neu][vecid]);
225       if (Sveci==disparo)
226       {
227         if (IE[vecid]==true) {Ci++;}
228         else {Ce++;}
229       }
230       else if (Sveci==hiperpolarizacion){Ch++;}
231
232       Sveci=cualestado(Conexiones_por_neurona[neu][vecii]);
233       if (Sveci==disparo)
234       {
235         if (IE[vecii]==true) {Ci++;}
236         else {Ce++;}
237       }
238       else if (Sveci==hiperpolarizacion){Ch++;}
239     }
240
241   //Calcule la regla de óactivacin
242   Ca=Ce-Ci-alpha*Ch;
243
244   return Ca;
245 }
246
247 double RedNeuronas::Potencial(int ix)
248 {
249   double pot = 0;
250
251   estado St = cualestado(ix);
252
253   if (St==reposo) {pot = -70;}
254   else if (St==disparo){pot = 40;}
255   else if (St==hiperpolarizacion){pot =-90;}
256   else if (St==refractario){pot = -75;}
257
258   return pot;
259
260 }
261
262
263 double RedNeuronas::SerieTemporal()
264 {
```

```
265   double y_t = 0;
266   int butter_order = 1;
267
268   for(int ix=0; ix<N2 ; ix++)
269   {
270      y_t += Potencial(ix);
271   }
272
273   return y_t;
274 }
275
276 double RedNeuronas::NeuronasActivas()
277 {
278   estado Sa;
279   double Ca =0;
280   for(int ix=0; ix<N2; ix++)
281   {
282     Sa=cualestado(ix);
283     if (Sa==disparo){Ca++;}
284   }
285
286   return Ca;
287 }
288
289 void RedNeuronas::Evolucione()
290 {
291
292   estado St;
293   double Ct=0;
294
295   int *Aux;//Auxiliar intercambio de matrices
296
297   for (int ix = 0; ix<N2; ix++)
298   {
299     St = cualestado(ix);
300
301       if (St == reposo)
302       {
303         Ct= Informese(ix);
304         if (Ct>= Trest)
305         //if(Informese(ix,iy)>=Trest)
306         APNew[ix]=1;
307
308         else
309         APNew[ix]=0;
310       }
311
312       else if (St == disparo || St == hiperpolarizacion)
313       APNew[ix]=AP[ix]+1; //evoluciona sin poder ser afectado por las
              demas neuronas
314
315       else if (St == refractario)
316       {
317         Ct= Informese(ix);
318         if(Ct<Trelative)
319         //if(Informese(ix,iy)<Trelative)
320         APNew[ix]=(AP[ix]+1)%11; //evoluciona sin ser afectado por las
              demas neuronas
321         else if (Ct>= Trelative)
```

```
322           //else if (Informese(ix,iy)>= Trelative)
323           APNew[ix]=1; //Pasa del estado refractario a activarse otra vez
                  (estado de disparo)
324         }
325
326     }
327
328     Aux=AP; AP=APNew;APNew=Aux;//intercambio las matrices
329
330 }
331
332     //------------------- Global Functions -------------------
333
334
335 int main(void)
336 {
337     RedNeuronas Red;//Declare la red
338     Crandom ran64(1);//semilla para resultados constantes
339     int t=0,tmax=1000;
340
341     Red.Inicie(ran64);
342
343   for(t=0;t<tmax;t++)
344     {
345       Red.Evolucione();
346       //cout<<Red.NeuronasActivas()<<endl;
347       if (t>30)
348       {
349         cout<<Red.SerieTemporal()<<endl;
350       }
351     }
352
353     return 0;
354 }
```

C
Butterworth filter

Listing 3: Implementation of the butterworth filter

```
1  #include <iostream>
2  #include<fstream>
3  #include <cmath>
4  #include <string>
5  #include <vector>
6  #include "Iir.h"
7
8
9  using namespace std;
10
11
12 //It will iterate through all the lines in file and put them in given
       vector
13 bool getFileContent(string fileName, vector<string> & vecOfStrs)
14 {
15     // Open the File
16     ifstream in(fileName.c_str());
```

```
17      // Check if object is valid
18      if(!in)
19      {
20          cerr << "Cannot open the File : "<<fileName<<endl;
21          return false;
22      }
23      string str;
24      // Read the next line from File untill it reaches the end.
25      while (getline(in, str))
26      {
27          // Line contains string of length > 0 then save it in vector
28          if(str.size() > 0)
29              vecOfStrs.push_back(str);
30      }
31      //Close The File
32      in.close();
33      return true;
34  }
35
36  int main()
37  {
38      // Create a band-pass fourth-order Butterworth filter with a low cut
            -off frequency of 1 Hz and a high cut-off frequency of 50 Hz
39      const int order = 4; // 4th order (=2 biquads)
40      Iir::Butterworth::BandPass<order> filter;
41      const float samplingrate = 500; // Hz
42      const float lowCutoffFrequency = 1; // Hz
43      const float highCutoffFrequency = 50; // Hz
44      filter.setup(order, samplingrate, lowCutoffFrequency,
            highCutoffFrequency);
45
46      // Input signal
47      vector<string> vecOfStr;
48
49      // Get the contents of file in a vector
50      bool result = getFileContent("datos.dat", vecOfStr);
51
52      int strsize = vecOfStr.size();
53
54      //turn data into integers
55      vector<int> Data;
56      Data.resize(strsize);
57      for (int i = 0; i < strsize; i++)
58      {
59        int num=stoi(vecOfStr[i]);
60        Data[i]=num;
61      }
62
63      // Apply the filter to the input signal
64      vector<double> outputSignal(Data.size());
65      for (size_t i = 0; i < Data.size(); ++i) {
66          outputSignal[i] = filter.filter(Data[i]);
67      }
68
69      // Print the filtered output
70      for (const auto& value : outputSignal) {
71          std::cout << value << endl;
72      }
73
```

```
74      return 0;
75  }
```

D Power spectal density

Listing 4: Code used to obtain the power spectral density with the welch method

```
1
2  #include <iostream>
3  #include<fstream>
4  #include <cmath>
5  #include <string>
6  #include <vector>
7  #include <algorithm>
8  #include "sigpack.h"
9  #include <armadillo>
10
11
12 using namespace std;
13 using namespace arma;
14 using namespace sp;
15
16
17 //It will iterate through all the lines in file and put them in given
       vector
18 bool getFileContent(string fileName, vector<string> & vecOfStrs)
19 {
20     // Open the File
21     ifstream in(fileName.c_str());
22     // Check if object is valid
23     if(!in)
24     {
25         cerr << "Cannot open the File : "<<fileName<<endl;
26         return false;
27     }
28     string str;
29     // Read the next line from File untill it reaches the end.
30     while (getline(in, str))
31     {
32         // Line contains string of length > 0 then save it in vector
33         if(str.size() > 0)
34             vecOfStrs.push_back(str);
35     }
36     //Close The File
37     in.close();
38     return true;
39 }
40
41
42
43 int main()
44 {
45     // pwelch args
46     int NFFT = 256;
47     int OVERLAP = NFFT/2;
48
```

```
    // Time scale
    double dt = 1.0/60.0;

    // sampling rate
    double fs = 1.0/dt;

    // Frecuency list
    double freq_step_size = fs / NFFT;

    vector<string> vecOfStr;

    // Get the contents of file in a vector
    bool result = getFileContent("eeg.dat", vecOfStr);

    int strsize = vecOfStr.size();

    //turn data into integers
    vector<int> Data;
    Data.resize(strsize);
    for (int i = 0; i < strsize; i++)
    {
      int num=stoi(vecOfStr[i]);
      Data[i]=num;
    }

    // Convert Data to an arma::Col object
    colvec x = conv_to<colvec>::from(Data);

    // Declare x as a const reference
    const Col<double>& signal = x;

    Col<double> psd = pwelch(signal, NFFT, OVERLAP);

    for(int i=0; i < OVERLAP; i++)
    {
        cout<< freq_step_size*i << ' '<< 10*log10(psd[i]) <<endl;
    }

    return 0;
}
```

E

Recurrence plot and recurrence quantification analysis

Listing 5: Code used to genarate the recurrence matrix and the analysis

```
#include <stdlib.h>
#include <iostream>
#include <RecurrenceQuantificationAnalysis.h>
#include <RecurrencePlot.h>
#include<fstream>
#include <cmath>
#include <string>
#include <vector>

using namespace std;
```

```
11
12 //It will iterate through all the lines in file and put them in given
       vector
13 bool getFileContent(string fileName, vector<string> & vecOfStrs)
14 {
15     // Open the File
16     ifstream in(fileName.c_str());
17     // Check if object is valid
18     if(!in)
19     {
20         cerr << "Cannot open the File : "<<fileName<<endl;
21         return false;
22     }
23     string str;
24     // Read the next line from File untill it reaches the end.
25     while (getline(in, str))
26     {
27         // Line contains string of length > 0 then save it in vector
28         if(str.size() > 0)
29             vecOfStrs.push_back(str);
30     }
31     //Close The File
32     in.close();
33     return true;
34 }
35
36
37 double standard_deviation(vector<int> Data)
38 {
39     //Calculate standard deviation
40     double SD=0;
41     double sum=0;
42     double average =0;
43     double sumav = 0;
44     double variance = 0;
45
46     for (int i = 0; i < Data.size(); i++)
47     {
48         sum+=Data[i];
49     }
50
51     average=sum/Data.size();
52
53     for (int i = 0; i < Data.size(); i++)
54     {
55        sumav += pow((Data[i]-average),2);
56     }
57
58     variance = sumav/Data.size();
59     SD=sqrt(variance);
60
61     return SD;
62 }
63
64 int main(void)
65 {
66     vector<string> vecOfStr;
67     // Get the contents of file in a vector
68     bool result = getFileContent("eeg.dat", vecOfStr);
```

```
69
70      int strsize = vecOfStr.size();
71
72      //turn data into integers
73      vector<int> Data;
74      Data.resize(strsize);
75      for (int i = 0; i < strsize; i++)
76      {
77        int num=stoi(vecOfStr[i]);
78        Data[i]=num;
79      }
80
81       double SD=0;
82
83      SD=standard_deviation(Data);
84
85      double time_series[Data.size()];
86      int time_series_length = Data.size();
87      int dimension = 2;
88      int delay = 1;
89      double threshold = 0.1*SD;
90      string norm = "euclidean";
91
92       // Copy the elements from the vector to the integer array
93      for (int i = 0; i < Data.size(); ++i)
94      {
95          time_series[i] = Data[i];
96      }
97
98      RecurrencePlot rp(time_series, time_series_length, dimension, delay,
            threshold, norm);
99
100     rp.write_recurrence_matrix("tryoutput1.dat");
101
102     int** recurrence_matrix = rp.get_recurrence_matrix();
103     int size = rp.get_number_of_vectors();
104     int lmin = 2;
105     int vmin = 2;
106     int wmin = 2;
107
108     RecurrenceQuantificationAnalysis RQA(recurrence_matrix, size, lmin,
          vmin, wmin);
109
110     RQA.write_measures("tryRQAmeasures1.dat");
111
112
113     return 0;
114 }
```

F Fractal Analysis

Listing 6: Code used to calculate the katz and petrosian fractal dimensions

```
1 #include <iostream>
2 #include<fstream>
3 #include <cmath>
```

```
4  #include <string>
5  #include <vector>
6  #include <algorithm>
7
8
9  using namespace std;
10
11 //It will iterate through all the lines in file and put them in given
       vector
12 bool getFileContent(string fileName, vector<string> & vecOfStrs)
13 {
14     // Open the File
15     ifstream in(fileName.c_str());
16     // Check if object is valid
17     if(!in)
18     {
19         cerr << "Cannot open the File : "<<fileName<<endl;
20         return false;
21     }
22     string str;
23     // Read the next line from File untill it reaches the end.
24     while (getline(in, str))
25     {
26         // Line contains string of length > 0 then save it in vector
27         if(str.size() > 0)
28             vecOfStrs.push_back(str);
29     }
30     //Close The File
31     in.close();
32     return true;
33 }
34
35 double EuclideanDistance(double x1, double y1, double x2, double y2)
36 {
37     double x = x1 - x2;
38     double y = y1 - y2;
39     double dist;
40
41     dist = pow(x, 2) + pow(y, 2);       //calculating Euclidean distance
42     dist = sqrt(dist);
43
44     return dist;
45 }
46
47
48 double average(vector<int> Data)
49 {
50     //Calculate standard deviation
51     double sum=0.0;
52     double average =0.0;
53
54     for (int i = 0; i < Data.size(); i++)
55     {
56         sum+=Data[i];
57     }
58
59     average=sum/Data.size();
60
61
```

```
62      return average;
63  }
64
65
66  int main()
67  {
68      vector<string> vecOfStr;
69      // Get the contents of file in a vector
70      bool result = getFileContent("eeg.dat", vecOfStr);
71
72      int strsize = vecOfStr.size();
73
74      //turn data into integers
75      vector<int> Data;
76      Data.resize(strsize);
77
78      for (int i = 0; i < strsize; i++)
79      {
80        int num=stoi(vecOfStr[i]);
81        Data[i]=num;
82      }
83
84      int n = Data.size();
85      double av= average(Data);
86      int deltan = 0;
87
88      //make binary series
89      vector<int> binaryseries;
90      binaryseries.resize(strsize);
91
92      for (int i = 0; i < n; i++)
93      {
94          if(av>Data[i]){binaryseries[i]=1;}
95          else{binaryseries[i]=-1;}
96
97          if (i>0 && binaryseries[i]+binaryseries[i-1]==0){deltan++;}
                //count sign changes
98      }
99
100     double L= 0.0;
101     double d= 0.0;
102
103     for (int i =1; i < n; i++)
104     {
105         L+= EuclideanDistance(i-1, Data[i-1],i, Data[i]);
106         for (int j = 1; j < n ; j++)
107         {
108             if (EuclideanDistance(j, Data[j],i, Data[i])>d){d=
                    EuclideanDistance(j, Data[j],i, Data[i]);}
109         }
110     }
111
112     //Petrosian fractal dimension
113     double Petro = log(n)/(log(n)+log(n/(n+0.4*deltan)));
114     //katz fractal dimension
115     double katz = log(n)/(log(n)+log(d/L));
116
117     cout << "Petrosian fractal dimension: " << Petro << endl;
118     cout << "Katz fractal dimension: " << katz << endl;
```

```
119
120     return 0;
121 }
```

G Entropy analysis

Listing 7: Code used to obtain the sample, permutation and multiscale entropy

```
from pyentrp import entropy as ent

ts = data
std_ts = np.std(ts)
sample_entropy = ent.sample_entropy(ts, 2, 0.2*std_ts)[1]
multiscale_entropy = ent.multiscale_entropy(ts, 2, tolerance=0.1*std_ts,
    maxscale=20)
permutation_entropy = ent.permutation_entropy(ts, normalize=True)
```

H Lempel-Ziv complexity

Listing 8: Code used to calculate the Lempel-ziv complexity

```
#include <iostream>
#include<fstream>
#include <cmath>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

//It will iterate through all the lines in file and put them in given
    vector
bool getFileContent(string fileName, vector<string> & vecOfStrs)
{
    // Open the File
    ifstream in(fileName.c_str());
    // Check if object is valid
    if(!in)
    {
        cerr << "Cannot open the File : "<<fileName<<endl;
        return false;
    }
    string str;
    // Read the next line from File untill it reaches the end.
    while (getline(in, str))
    {
        // Line contains string of length > 0 then save it in vector
        if(str.size() > 0)
            vecOfStrs.push_back(str);
    }
    //Close The File
    in.close();
    return true;
```

```
32 }
33
34 double average(vector<int> Data)
35 {
36     //Calculate standard deviation
37     double sum=0;
38     double average =0;
39
40     for (int i = 0; i < Data.size(); i++)
41     {
42         sum+=Data[i];
43     }
44
45     average=sum/Data.size();
46
47
48     return average;
49 }
50
51
52 int main()
53 {
54     vector<string> vecOfStr;
55     // Get the contents of file in a vector
56     bool result = getFileContent("datos.dat", vecOfStr);
57
58     int strsize = vecOfStr.size();
59
60     //turn data into integers
61     vector<int> Data;
62     Data.resize(strsize);
63
64     for (int i = 0; i < strsize; i++)
65     {
66       int num=stoi(vecOfStr[i]);
67       Data[i]=num;
68     }
69
70     int n = Data.size();
71     double av= average(Data);
72
73     for (int i = 0; i < n; i++)
74     {
75         if(av>Data[i]){Data[i]=1;}
76         else{Data[i]=0;}
77     }
78
79     double LZC =0;
80     int i = 0;//i = p-1, p is the pointer (The index which allows to
           have the longest production)
81     int C = 1;// -LempelZiv Complexity
82     int u = 1;//length of the current prefix
83     int v = 1;//length of the current component for the current pointer
           p
84     int vmax = v;//final length used for the current component (largest
           on all the possible pointers p)
85
86     while (u + v <= n) {
87         if (Data[i + v] == Data[u + v]) {
```

```
            v++;
        } else {
            vmax = max(v, vmax);
            i++;
            if (i == u) {
                C++;
                u += vmax;
                v = 1;
                i = 0;
                vmax = v;
            } else {
                v = 1;
            }
        }
    }

    if (v != 1) {
        C++;
    }

    LZC=log(n)*C/n;

    cout << "C: " << C << endl;

    cout << "LZC: " << LZC << endl;

    return 0;
}
```

I
Post-processing

Listing 9: Code used to generate multiple simulations

```
##################
## Get simulations
##################

## Number of simulations
n_simulations = 30

trest = 3
trelative = 5

for seed in tqdm(range(n_simulations)):
    run_command = [f'{red_neurona_exec}', f'{trest}', f'{trelative}', f'{seed}']
    result = subprocess.run(run_command, capture_output=True, text=True)
    time.sleep(1)

    # save the output
    simulation_data = np.fromstring(result.stdout, sep='\n')

    if len(simulation_data) > 900:
        simulation_name = red_neurona_exec_name.replace('.out','')+f'{trest}_{trelative}_{seed}'
        simulation_path = f'{simulated_eggs}/{simulation_name}.dat'
```

```
        pd.DataFrame({'x':simulation_data}).astype(int).to_csv(
            simulation_path, header=False, index=False)

        ###################
        ## Butterworth
        ###################
        run_command = [f'{butterworth_exec}', f'{simulation_path}']
        result = subprocess.run(run_command, capture_output=True, text=
            True)

        # save output
        butterfilter_output_path = f'{butterfilter_eggs}/{
            simulation_name}.dat'

        butterworth_data = np.fromstring(result.stdout, sep='\n')[50:]
        pd.DataFrame({'x':butterworth_data}).astype(int).to_csv(
            butterfilter_output_path, header=False, index=False)
```

Listing 10: Code used to post-process real and simulated EEGs

```
def get_simulated_data(file):
    return np.loadtxt(file)

def get_egg_plot(data, data_name):
    plt.rcParams.update({'font.size': 15})
    fig = plt.figure()
    ax = plt.axes()
    fig.set_size_inches(w=10, h=5.4)

    ## Plot data
    data = np.array(data)
    ax.plot(data)

    ax.set_title('EGG')
    ax.set_ylabel('Amplitude')
    ax.xaxis.set_ticks(np.arange(0, 1100, 100))
    ax.grid()
    fig.savefig(f'{output_folder}/figures/{data_name}/{data_name}_EGG.
        png', bbox_inches='tight', pad_inches=0.03, dpi = 150)

def get_psdwelch_plot(frecuency_list, data, data_name):
    plt.rcParams.update({'font.size': 15})
    fig = plt.figure()
    ax = plt.axes()
    fig.set_size_inches(w=10, h=5.4)

    ## Plot data
    data = np.array(data)
    ax.plot(frecuency_list, data)

    ax.set_title('Periodogram')
    ax.set_xlabel('Frecuency')
    ax.set_ylabel('Amplitude')

    ax.xaxis.set_ticks(np.arange(0, 32, 2))
    ax.grid()
    fig.savefig(f'{output_folder}/figures/{data_name}/{data_name}
        _periodogram.png', bbox_inches='tight', pad_inches=0.03, dpi =
        150)

```

```
38 def get_post_process(file, file_path, real_eeg=True):
39     file_label = file.replace('.dat','')
40 
41     figures_folder = f'{output_folder}/figures/{file_label}'
42     ## create folder
43     directory = figures_folder
44     if not os.path.exists(directory):
45         os.makedirs(directory)
46         print("Directory '%s' created" %directory)
47 
48     recurrence_data_file_name = f'{output_folder}/processed_data/
           recurrence/recurrence_{file_label}.dat'
49     recurrence_measure_data_file_name = f'{output_folder}/processed_data
           /recurrence/recurrence_measure_{file_label}.dat'
50 
51     ##################
52     ## EEG data
53     ##################
54     if real_eeg:
55         egg_data_file_name = f'{output_folder}/raw_data/{file_label}.dat
               '
56         data = get_real_data(file_path)
57         output_files_label = 'real'
58     else:
59         egg_data_file_name = f'{output_folder}/processed_data/
               butterworth_filter/{file_label}.dat'
60         data = get_simulated_data(file_path)
61         output_files_label = 'simulated'
62 
63     get_egg_plot(data, f'{file_label}')
64 
65     ###################
66     ## PSDwelch
67     ###################
68     pd.DataFrame({'x':data}).to_csv(f'{egg_data_file_name}', header=
           False, index=False)
69 
70     cmd = f'{pwelch_exec} {NFFT} {OVERLAP} {sampling_dt} "{
           egg_data_file_name}"'
71     output_ = subprocess.Popen(cmd, shell=True, executable='/bin/bash',
           stdout=subprocess.PIPE)
72 
73     out_raw, err = output_.communicate()
74     out = str(out_raw).replace('b','').replace("'",'').split('\\n')[:-2]
75 
76     frecuency_list = np.array([float(out[n_data].split()[0]) for n_data
           in range(len(out))])
77     psdwelch_list = np.array([float(out[n_data].split()[1]) for n_data
           in range(len(out))])
78 
79     get_psdwelch_plot(frecuency_list, psdwelch_list, f'{file_label}')
80 
81     ## bands
82     psdwelch_list += -np.min(psdwelch_list)
83     psdwelch_list /= np.linalg.norm(psdwelch_list)
84     delta_band  = np.mean(psdwelch_list[np.where((frecuency_list >= 1) &
           (frecuency_list < 4))])
85     theta_band  = np.mean(psdwelch_list[np.where((frecuency_list >= 4) &
           (frecuency_list < 8))])
```

```
86      alpha_band  = np.mean(psdwelch_list[np.where((frecuency_list >= 8) &
            (frecuency_list < 12))])
87      sigma_band  = np.mean(psdwelch_list[np.where((frecuency_list >= 12)
            & (frecuency_list < 16))])
88      beta_band   = np.mean(psdwelch_list[np.where((frecuency_list >= 16)
            & (frecuency_list < 24))])
89      gamma_band  = np.mean(psdwelch_list[np.where((frecuency_list >= 24)
            & (frecuency_list < 30))])
90
91      np.save(f"{output_folder}/processed_data/bands/{output_files_label}/
            bands_results_{file_label}.npy", np.array([delta_band, theta_band
            , alpha_band, sigma_band, beta_band, gamma_band], dtype=object))
92
93      ##################
94      ## Recurrence plot
95      ##################
96      cmd = f'{recurrence_exec} {egg_data_file_name} {
            recurrence_data_file_name} {recurrence_measure_data_file_name} '
97      outout = subprocess.Popen(cmd, shell=True, executable='/bin/bash',
            stdout=subprocess.PIPE)
98      time.sleep(1)
99
100     #### load data
101     data_ = np.loadtxt(recurrence_data_file_name, delimiter=',')
102
103     #### setup plot
104     plt.rcParams.update({'font.size': 80})
105     fig = plt.figure()
106     ax = plt.axes()
107     fig.set_size_inches(w=30, h=30)
108     fig.canvas.draw()
109
110     #### plot matrix
111     pcm = ax.imshow(data_[recurrence_init_zoom:recurrence_final_zoom,
            recurrence_init_zoom:recurrence_final_zoom], cmap='gray_r',
            origin='lower')
112
113     #### save plot
114     plt.gca().set_aspect('equal')
115     plt.tight_layout()
116     fig.savefig(f'{figures_folder}/{file_label}_recurrence.png',
            bbox_inches='tight', pad_inches=0.03, dpi = 150)
117
118     #### load recurrence extra info
119     df_recurrence = pd.read_csv(f'{recurrence_measure_data_file_name}')
120
121     L       = df_recurrence['L'].to_numpy()
122     L_MAX   = df_recurrence['L_MAX'].to_numpy()
123     V_MAX   = df_recurrence['V_MAX'].to_numpy()
124     DET     = df_recurrence['DET'].to_numpy()
125     LAM     = df_recurrence['LAM'].to_numpy()
126     TT      = df_recurrence['V'].to_numpy()
127
128     np.save(f"{output_folder}/processed_data/recurrence/{
            output_files_label}/recurrence_results_{file_label}.npy", np.
            array([L, L_MAX, V_MAX, DET, LAM, TT], dtype=object))
129
130     ##################
131     ## Entropy analysis
```

```
132     ###################
133     ts = data
134     std_ts = np.std(ts)
135     sample_entropy = ent.sample_entropy(ts, 2, 0.2*std_ts)[1]
136     multiscale_entropy = ent.multiscale_entropy(ts, 2, tolerance=0.1*
            std_ts, maxscale=20)
137     permutation_entropy = ent.permutation_entropy(ts, normalize=True)
138 
139     np.save(f"{output_folder}/processed_data/entropy/{output_files_label
            }/entropy_results_{file_label}.npy", np.array([sample_entropy,
            multiscale_entropy, permutation_entropy], dtype=object))
140 
141     ###################
142     ## Fractal analysis
143     ###################
144     cmd = f'{fractalanalysis_exec} "{egg_data_file_name}"'
145     output_ = subprocess.Popen(cmd, shell=True, executable='/bin/bash',
            stdout=subprocess.PIPE)
146     out_raw = output_.communicate()
147     output_list = str(out_raw).replace('(','').replace('b','').replace("
            '",'').split('\\n')[:2]
148 
149     petrosian_fractal = float(output_list[0].split(':')[1])
150     katz_fractal = float(output_list[1].split(':')[1])
151 
152     np.save(f"{output_folder}/processed_data/fractal/{output_files_label
            }/lempelziv_results_{file_label}.npy", np.array([
            petrosian_fractal, katz_fractal], dtype=object))
153     ###################
154     ## LempelZiv analysis
155     ###################
156     cmd = f'{lempelziv_exec} "{egg_data_file_name}"'
157     output_ = subprocess.Popen(cmd, shell=True, executable='/bin/bash',
            stdout=subprocess.PIPE)
158     out_raw = output_.communicate()
159     output_list = str(out_raw).replace('(','').replace('b','').replace("
            '",'').split('\\n')[:2]
160 
161     C = float(output_list[0].split(':')[1])
162     LZC = float(output_list[1].split(':')[1])
163 
164     np.save(f"{output_folder}/processed_data/lempelziv/{
            output_files_label}/lempelziv_results_{file_label}.npy", np.array
            ([C, LZC], dtype=object))
```