Information transfer in complex networks: challenges and new perspectives

Alessandro Montalto

Proefschrift voorgedragen tot het behalen van de graad van
Doctor in de Statistische Data-Analyse
Academiejaar 2015-2016

Promotor:
Prof. Dr. Daniele Marinazzo
Co-Promotor:
Dr. Luca Faes

Vakgroep Data Analyse
Faculteit Psychologie en Pedagogische Wetenschappen, Universiteit Gent
Henri Dunantlaan 1, B-9000 Gent

"ANDY: You know what the Mexicans say about the Pacific?"
"RED: No."
"ANDY: They say it has no memory. That's where I want to live the rest of my life. A warm place with no memory."

– From *Shawshank Redemption* by Stephen King

Acronyms

AVG averaging. 94

CE Conditional Entropy. 21

DWS downsampling. 95

ECG Electrocardiogram. 36

EEG Electroencephalogram. 35

GC Granger Causality. 14

ISS Innovations State-Space. 92

MA Moving Average. 89

MuTE **Mu**ltivariate **T**ransfer **E**ntropy toolbox. 4

NeuNet NUE neural networks estimator used with the non-uniform embedding approach. 72

NeuNet UE Neural networks estimator used with the uniform embedding approach. 72

NNGC Neural Networks Granger Causality. 59

NUE Non-Uniform Embedding framework. 19

PDF Probability Density Function. 31

SS State-Space. 89

TE Transfer Entropy. 15

UE Uniform Embedding framework. 19

VAR Vector Auto Regressive. 21

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

First came the idea, then implementation and simulations: the creative process that gives rise to what might become a beautiful and useful outcome is very complex.

In this thesis we address the problem of detecting the information exchanged by systems that might have different nature. In order to get more insights about how those systems communicate with each other we developed an organic framework that contains several approaches, several points of view according to which the information exchanges can be retrieved. One of the most difficult problems when we try to study a set of interacting systems is how to model it and its interactions. A key role in understanding the system set is played by two scientific fields: complex system analysis and information theory.

To fully grasp the meaning of the term "complex", it needs to be grounded in a particular discipline. Here we would like to address the word complex from the point of view of the analysis of such systems that might be considered different than the sum of its parts. The analysis turned to a new investigation field under the name of *complex systems analysis*.

There is not a generalized agreement about the definition of what a complex system is. We can try to understand what can be considered as a complex system

by taking into account the varied approaches used to model systems composed of several interacting parts that exhibit highly unpredictable behaviours.

The first approach we are going to mention is the one that takes into account the structure of the system. The chief goal is to first determine an underlying, interconnecting structure, then to explore the relationships among the different elements of such a system. As an example we can mention how biologists study structures at different levels of hierarchical organization, ranging from molecular biology to population mechanisms, not neglecting the brain as a complex systems consisting of billions (in the case of the human brain) of subsystems called neurons. Therefore, psychology, as well as social scientific fields such as linguistics and anthropology, was the most natural field to which the study of the structure connecting the variable involved could be applied.

A convenient way to provide a mathematical formalism to the analysis of the structure is graph theory, according to which the elements of a system can be modelled by nodes connected by edges that represent the interactions between nodes, figure 1.1. Thus, graph theory can give insights about the topology of the system under study. If a complex system could be thought as a graph, we would simplify the object of our study too much because we would not take into account the behaviour that a complex system usually shows.

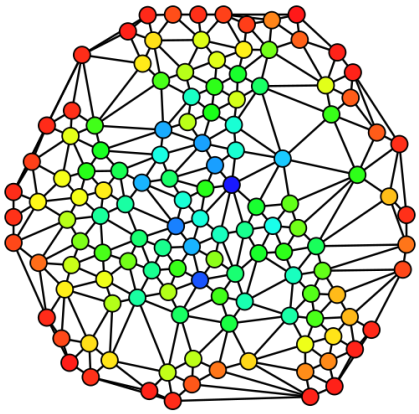

Figure 1.1: Example of graph: the coloured dots are the nodes and the straight lines are the edges structurally connecting the nodes.

Although the structure can give very useful insights about the nature of the system set under study, it does not seem to provide a complete overview of the

relationships connecting the systems.

We then come to the second approach that considers the *dynamical* evolution of the systems with regard to temporal processes. The main assumption of the temporal dynamical point of view is to consider past events as possibly the cause of future events. In this sense, the temporal dynamical approach assesses a relationship between cause and effect. The challenge is to distinguish the history of which subsystems are responsible for the future evolution of the whole system. In order to perform such a distinction the temporal dynamical effects can not be disentangled by the topology of the system. One of the most prominent modern approaches to look at reality is by explaining how things work in a mechanistic and deterministic way. This vision implies that assuming some initial conditions as a starting point, the time evolution of the model of whatever system is under study will lie in a specific range of likely directions. Mathematical models can be accurate enough to predict the behaviour of several systems, even if not as accurately as we wish. The lack of accuracy is due to what we can name randomness, a sort of unpredictable group of events that perturbs the surrounding environment.

The amount of unpredictability and the effects of randomness are then responsible for the behaviour of a system: the more randomness is present, the more unpredictable the system is, and the greater the effect on the system. When randomness can be taken into account as playing a marginal role in the dynamics of the system, then we are able to develop models reliable enough to make predictions. Accordingly, we are able to apply the mechanistic and deterministic approaches with a consequent dismantlement and understanding of the mechanisms that create the system. In this case we are dealing with a *simple system*.

A complete different scenario challenges scientists when randomness can not be accounted for and effects can easily lead the model towards an unstable state. The mechanistic and deterministic hypotheses no longer hold and even if scientists are able do reproduce a small scale version of the entire system, the assumptions are such that the model is very far from representing reality. We are then studying a *complex system* that exhibits another very interesting feature: no matter what kind of relations hold at the very fundamental scale (e.g. neurons in a brain), some other properties are shown at the higher level scale (e.g. simple processes underlying imagination in the human brain). Those properties are referred to as *emergent*. It is

not yet clear how self organization of a complex system can lead to the appearance of the emergent properties. An improvement in the comprehension of how a complex system can be modelled has certainly come to light: the relationships occurring between subsystems might be very simple assuming that they have a very small range effect and they are the common among the interactions between the whole set of subsystems. For instance, we can think about a bird flock forming awesome shapes in the sky as a complex system. One of the reasons why the flock does not disperse and it is capable of avoiding obstacles seems to be that each bird follows three simple rules. The first one is to stay close, but do not bump into its neighbours. The second rule is to fly as fast as its neighbours, implying that the birds surrounding it should always be the same. Finally, according to the third rule each bird should fly towards the centre of the flock. It is possible to simulate the complex flying patterns of a bird flock merely by making each bird fly according to the previous rules. There is not any bird that is in charge to lead the whole flock. Therefore, when a bird changes position, the rest of the birds will do the same, thus triggering the emergent flying patterns.

Nowadays, we can recognise complex systems in different fields such as economics with the analysis of organizations made up of people, social and behavioural science with the spread of epidemics and the analysis of swarming insects, biology and physiology, ranging from the brain to the pacemaker cells in the heart tissue to the entirety of mechanisms controlling human physiology, to name but a few examples.

My contribution to this PhD project deals with the development of methods able to better detect the dynamical influences occurring within a complex system.

This thesis is structured as follows: chapter 2 is an introduction to information theory and the mathematical building blocks that will be further used in the subsequent chapters. We will explain the reasoning that led to a formal mathematical definition of concepts such as information and entropy. We will then link the building blocks to the two measures that we will use throughout the whole thesis. In this way, the reader will be gently introduced to the mathematical framework adopted in this work.

In chapter 3 we are going to describe the **Mu**ltivariate **T**ransfer **E**ntropy toolbox (MuTE) and the first six methods implemented in it. We will provide an exhaustive

explanation of the methods from both theoretical and experimental points of view, comparing the performances of the approaches, but not neglecting to mention the pros and cons of each method in order to lead the user to best choose which method will suit his needs.

Chapter 4 is devoted to the description of the artificial neural networks approach that provides MuTE with two additional estimators. We will briefly describe artificial neural networks from a general point of view in order to facilitate a basic understanding of the relationship between machine learning and information theory. We will stress the relevance of this chapter because it represents a completely new bridge between two scientific fields with the hope that further efforts will be made in closing the gap between machine learning and information theory.

Chapter 5 will deal with the multiscale approach applied to the model-based method. We will explain the problems related to the two preprocessing techniques of filtering and downsampling. A theoretical proof of how preprocessed signals differ from the original signals is given. Furthermore, a possible way to overcome the problems related to the preprocessing techniques is described.

Chapter 6 is devoted to summarizing the main ideas emerging from the thesis and some suggestions regarding future directions for this research are made.

In Appendix A we will explain what we have done in order to promote MuTE to a larger audience than that reached by traditional scholarly publications. We will also describe the efforts to make MuTE more user-friendly in order to encourage relatively inexperienced programmers to use it.

CHAPTER 2

Information Transfers

Information theory can evaluate interactions in several ways providing directed or indirected edges detected relying on the amount of information that a time series is exchanging with another one. Thus, information theory offers a framework in which it is possible to model and study the transmission, processing, utilization, and extraction of information.

Every language is meant to address a communication task; whatever the language is, its main purpose is to provide a reliable tool to exchange information. The current spoken and written languages that are used nowadays adopt a sequence of symbols that represents physical or non-physical objects[1] in order to deliver a message. In our everyday life we continually experience multiple languages such as street art, advertisement, traffic signals and so on, thus experiencing different levels of communication. If we stick with the example of the street signals we can say that the physical object is the piece of matter arranged as signal. The non-physical object might be the destination where we are headed. The message might eventually be the direction to take in order to get a certain place. Communication is thus

[1] Here the terms physical and objects are meant to provide a simplified idea of the relationship between whatever we can call "physical" and "object". A rigorous detailed explanation of the two terms is not addressed in this thesis and it is left to more exhaustive philosophical debates [1].

possible only if whoever reads the signal is able to decode what is written on it. The information carried by every message relies on the capability to effectively relate encoding and decoding of a message. In the case of the street signal the codification of the direction might be represented by an arrow, while the decoding is the interpretation of the arrow: what would we think if an apple appeared on the sign in place of the arrow?

Depending on the device used to deliver the message, the coding may change. If we think about a telegraph the electromagnetic impulses can code a binary representation composed by two levels of magnetizations thought of as the levels 0 and 1. Sequences of ones and zeros can model the letters of the alphabet and, consequently, words. The same reasoning holds in order to obtain numbers, colors, music and whatever we can think about that can be digitized.

Let us imagine an electrified wire that connects two villages. We would like to send electrical pulses along the wire in order to transmit a message. As a start, posit that we wish to send a list of 10 coin flips. We can develop several ways to send that particular message, but we would like to define a method, a codification, that will allow us to send any message with the shortest numbers of ones and zeros. How can we accomplish this task? Let us think about the previous example in order to answer the question. If we want to send the outcome of a coin flip we can think about the two possible pulses, zero and one, as representing a “no” or a “yes” answer asked in order to determine the outcome. We could ask “is the outcome of the first coin flip head?”. To this question we can answer 0, meaning no, or 1, meaning yes. Whenever we send 1, the decoding must interpret it as “head”, while 0 must be decoded as “tail”. So in our binary system we would only need one answer to send one coin flip. We will consequently need ten questions to send ten coin flips.

We then want to sent a message of six letters. Assuming that we are using the English alphabet, we want to send the first letter of our message, let it be the letter f. The challenging problem to solve is the following: how many questions should we ask before we can completely be sure that the decoding will correctly interpret our message? It turns out that the most effective strategy to ask the smallest number of questions is to always divide the number of possible choices n in two and check in which subset the letter is. If n is even we will have two subsets of

exactly $n/2$ elements each. If n is odd we will have two subsets of $n_1 = (n+1)/2$ and $n_2 = (n-1)/2$ elements. We can start dividing the 26 letters of the English alphabet in two subsets of 13 letters each. The first question would then be: is f in the first subset containing letters from a to m? The answer is yes, represented by a 1. We then split the set of variables in two subsets containing letters from a to g and from h to m respectively. Is f in the first subset? Yes, so the coding is 1 1 so far. If we iterate the procedure we will end up with the following code for f: 1 1 0 1 0. This code will uniquely identify the letter f. In figure 2.1 a visual explanation of the algorithm used to code a letter is shown. As soon as we start asking questions, we go down the binary decision tree until we are able to select the desired letter. As depicted in Figure 2.1, the simple rule of thumb is that the code of a letter is derived by going down the decision tree and assigning a 1 every time we choose an arrow pointing to the left. We assign a 0 every time we go down choosing an arrow pointing to the right. There is then a one-to-one correspondence between each letter and its code.

We can now try to figure out whether we can have a clue about the average number of questions that we should ask in order to select a letter. If we apply the algorithm to whatever letter we want to select, we see that the depth of the decision tree consists of either five questions in the worse case or four questions in the best case. At every step the number of subsets is split in two. This means that if the tree were full we would have $2^5 = 32$ possible symbols that means 32 letters. Since we only have 26 letters and the tree is not full, the number of steps may vary between four and five. To evaluate the average number of questions we can apply the inverse function of the exponential: $\log_2 26 \simeq 4.7$, representing the average length of symbols needed to code each possible symbol.

Ralph Hartley in his paper "Transmission of Information", presented at the International Congress of Telegraphy and Telephony in 1927, writes that "a quantitative measure of 'information' is developed which is based on physical as contrasted with psychological considerations". He called information a quantity H that is proportional to the logarithm of the number of possible symbol sequences: $H = n \log s = \log s^n$, where s is the number of possible states the system can take at each question and n is the average number of states used to code a symbol. Taking into account the previous example, we can notice that $s = 2$, $n = 4.7$.

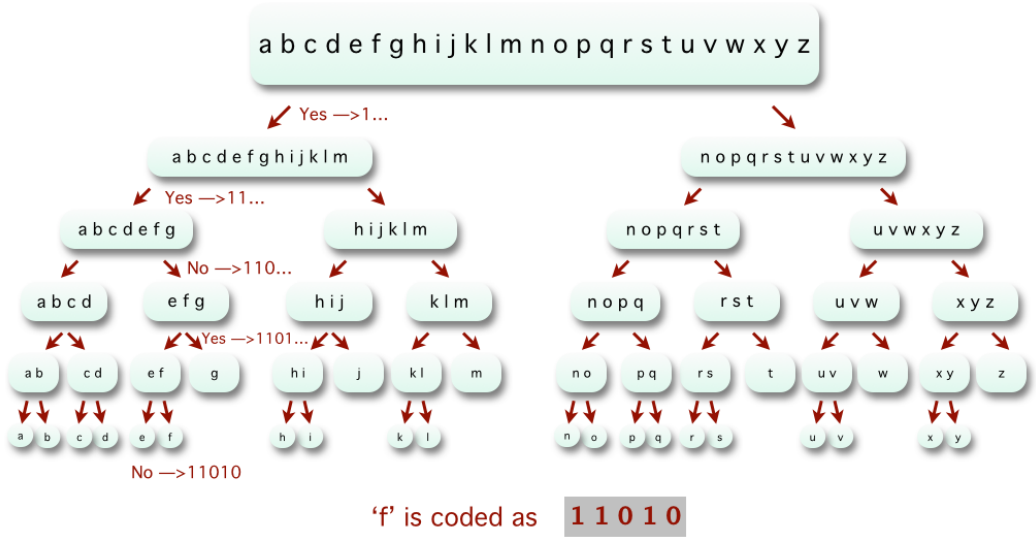

Figure 2.1: Binary decision tree explaining the algorithm used to find a letter asking the minimum number of questions. Whenever we want to code a letter we can ask whether the letter is in the first subset. If so we assign 1, otherwise 0, and we continue until the letter is selected. We can easily see that "f" is coded as 1 1 0 1 0.

Hartley paved the way to what will become the topic studied by information theory. One of the main purposes of information theory is to provide a model of communication such that the amount of information exchanged can be evaluated. Facing the problem of quantifying the information is not trivial. Claude Shannon in his paper "A mathematical theory of communication", 1948, derived a "measure of how much 'choice' is involved in the selection of the event or of how uncertain we are of the outcome" called entropy and represented by H. To understand his idea we can consider an alphabet of four letters only, "a", "b", "c", "d" and a message transmitted. If we assume that the letters sent in a message are uniformly randomly chosen we can arrange the decision tree as shown in figure 2.2I where all the symbols need two questions to be selected. If the letters are not uniformly distributed, then another way to arrange the decision tree is possible. We would first need to evaluate the ratio between the number of occurrences of each symbol in the message and the number of symbols contained in the message. We can think about the ratio as the probability of occurrence of the symbol. Thus, we might have $p_a = 0.5\,; p_b = 0.125\,; p_c = 0.125\,; p_d = 0.25$. The decision tree would be better

built as shown in figure 2.2II.

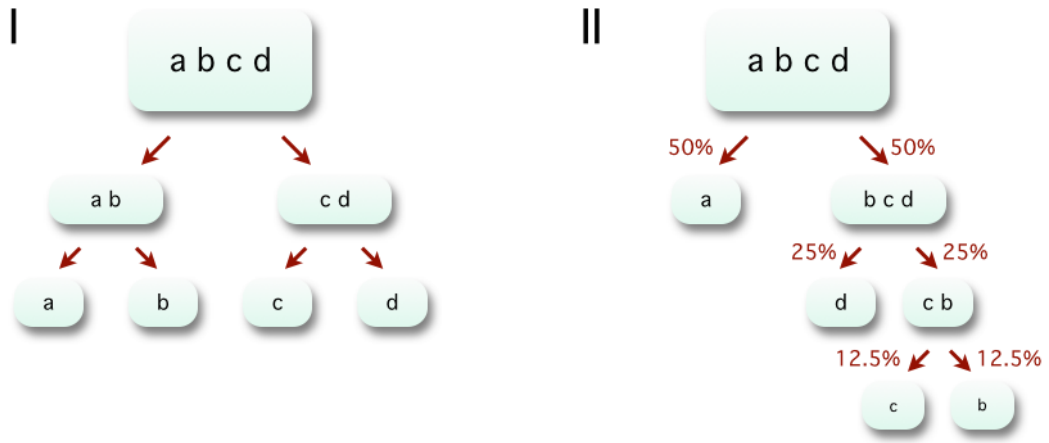

Figure 2.2: In I a uniformly distributed tree is shown. In II the probability distribution changes and the number of questions to select a letter changes accordingly.

Shannon tried to find a measure that can give an idea about the amount of "choice" involved in the selection of a letter or, in other words, about the amount of uncertainty of the outcome.

If such a measure exists, let us call it $H(p_i)$, where $i \in [1, \cdots, n]$ and $n = 4$ in our example, then asking the following properties seems reasonable:

1. H should be continuous in the p_i.

2. If all p_i are equal then $p_i = 1/n$ and H is a monotonic increasing function of n. This property represents the decreasing amount of choice, or uncertainty, as some events become more possible than others.

3. If a choice is split into two successive choices then the original H should be the weighted sum of the singular values of H.

It can be proven that the only function satisfying the three assumptions required is of the form:

$$H = -K \sum_{i=1}^{n} p_i \log p_i \tag{2.1}$$

We can understand equation (2.1) thinking about the examples shown in figure 2.2. We showed that every time we build a binary decision tree the average height of the tree equals the logarithm in base two of the number of possible states ($4.7 \simeq \log_2 26$ questions to ask in order to select a letter). If we want to know the height of the tree at the level i we should evaluate $\text{height}_{p_i} = \log_2(\text{number of choices at } p_i \text{ level}) = \log_2 1/p_i$, where p_i is the probability to occur of a choice at level i. Taking into account figure 2.2 we can notice that in 2.2I all the choices are on the same level so they have the same probability to occur, satisfying the second assumption above. In II the probability distribution that determines how often a letter is selected on average is not uniform. The entropy is then the average summation of the contribution of each letter. This contribution can be mathematically expressed as

$$\begin{aligned} H &= \sum_{i=1}^{4} (\text{height of the tree at } p_i \text{ level} \times\ p_i) \\ &= \sum_{i=1}^{4} p_i \log_2(1/p_i) \\ &= -\sum_{i=1}^{4} p_i \log_2 p_i \end{aligned} \tag{2.2}$$

In I $H = 4(p_i \log_2(1/p_i)) = 4(0.25 \log_2(1/0.25)) = 2$ while in II

$$\begin{aligned} H &= (p_a \log_2(1/p_a)) + (p_b \log_2(1/p_b)) + (p_c \log_2(1/p_c)) \\ &\quad + (p_d \log_2(1/p_d)) \\ &= 0.5 \log_2(1/0.5) + (0.125 \log_2(1/0.125)) \\ &\quad + (0.125 \log_2(1/0.125)) + (0.25 \log_2(1/0.25)) \\ &= 1.75 \end{aligned}$$

As we can see, the greater the uncertainty, figure 2.2 I, the higher the entropy. If we consider the $\log_2(\cdot)$ function then the entropy is expressed in "bits". We can use other bases for the logarithm, changing the unit of measure accordingly. Without

loss of generality we will use the $\log(\cdot)$ function from now on.

As Shannon went further in his reasoning we have other relations to play with. In fact, he not only derived the entropy related to the joint occurrence of two states i, j representing two different events x, y

$$H(x,y) = -\sum_{i,j} p(i,j) \log p(i,j) \tag{2.3}$$

where $p(i,j)$ is the joint probability associated to the occurrence of i and j, but also what he called conditional entropy.

Suppose we are dealing with the same events x, y and the same respective individual states i, j. We can say that for any particular value i that x can assume corresponds to a conditional probability $p(j|i)$ that y has the value j expressed by the following relation:

$$p(j|i) = \frac{p(i,j)}{\sum_i (p(i,j))} \tag{2.4}$$

The conditional entropy of y, $H_(y|x)$ is then defined as the average entropy of y for each value of x, weighted according to the probability that a particular state of x will occur:

$$H(y|x) = -\sum_{i,j} p(i,j) \log p(j|i) \tag{2.5}$$

The equations above give us an idea of what happens when a variable assumes a particular value. Furthermore, we are interested in revealing the dynamics according to which past events can affect the present state in order to (i) try to picture the dynamical influences occurring over time and (ii) build a topology of the influences by means of a graph. If we assume that past events can influence the present, then we might assume that the past is causing the present when the former is helping to predict the latter. In other words, we are interested in detecting whether and to what extent past states can influence the present state. We then need to define what influence means and how to measure it. In the following, we are going to introduce two measures that can quantify the influence occurring within a time series and among several time series.

2

Complex systems can be studied by perturbing one part of the system and investigating the outcome obtained in another part. A very different approach consists of identifying causal relationships as the predictability of ongoing activity in one as compared with that in another. A new method that was introduced by the pioneering work of Wiener (1956) [2] and Granger (1969) [3] allowed this new approach to be further investigated. The WienerGranger method does not require direct intervention in the system. It relies on the estimation of causal statistical influences between simultaneously recorded variables that can be represented by time series. Causality in the WienerGranger sense (GC) is based on the statistical predictability of one time series that derives from knowledge of one or more others.

The definition of "causality" is non-trivial for complex systems where cause and effect are very difficult concepts to understand. In 1956 Norbert Wiener introduced the idea that a time series could "cause" another if the prediction of the second variable is improved by taking the information about the first into account [2]. However, the implementation of the Wiener's idea was introduced only in 1969 by the econometrician Clive Granger in the context of linear autoregressive models of stochastic processes.

The basic idea of GC is straightforward. Let us assume that we have two variables X, and Y, and we want the present state of X, X_t, using only past states of X. We also want to predict X_t using past states of both X and Y. If the second prediction is significantly more successful, then we can infer that the past of Y helps in predicting X_t more than the past of X only. In this case, Y is said to G-cause X. We can easily see that Y G-causing X does not necessarily imply X G-causing Y, GC resulting in an asymmetric measure that is able to detect directed information transfers.

Because the values of a variable at one time are predicted by values of other variables at earlier times, it is often said that GC depends on "temporal precedence". However, it is not sufficient that events in the other variables simply precede similar events temporally in the first variable. For GC to be significant, statistically significant predictability must be established. In other words, non-zero values for GC can usually be obtained from any set of time series, but these values are meaningless unless it is determined that they are statistically significant.

Another useful measure to detect directed dynamical links is called Transfer

Entropy (TE). TE was defined by Schreiber [4] in the realm of information theory and it permits an evaluation of the influences occurring within a system without making any assumptions as to the nature of the influences. The relationships might be linear or non-linear: TE does not need a model in order to detect whether the past states are helping to predict the present state.

In order to give a qualitative description of what transfer entropy is, we can think about a common scenario. Let us imagine that we listen to a completely unknown song on the internet. A great amount of causal interactions take place in order to transfer the music to our brain. For instance, the causal, physical interactions occur between the data downloaded and the way the program that we are using to browse the internet reassembles the data, all the processes occurring to turn that stream of sorted data into the desired music and so on, up to the conversion of air pressure into neural signals in the cochlea that finally triggers activity in our brain. In this case, there is a clear information transfer from the driver that is represented by the music and the target that is represented by the brain. If the sound track contains repetitions, the brain can easily predict the melody. In this case the driver is no longer able to transfer information. The neural activity might theoretically be completely predictable by taking only the past of the neural activity into account. If we estimated TE between the driver and the target, we would find a TE higher than zero when the music is completely unknown, while we would find TE equal to zero when repetitions are heard.

While GC needs a model in order to evaluate the directed dynamical influences, TE is a model-free measure. They can be merged into the same framework if we assume that the whole set of time series is drawn by the same Gaussian distribution. According to this assumption, GC and TE are equivalent and they can be compared [5].

CHAPTER 3

MuTE: A MATLAB Toolbox to Compare Established and Novel Estimators of the Multivariate Transfer Entropy

Paper published on PLoS ONE; doi: 10.1371/journal.pone.0109462

A challenge for physiologists and neuroscientists is to map information transfer between components of the systems that they study at different scales, in order to derive important knowledge on structure and function from the analysis of the recorded dynamics. The components of physiological networks often interact in a non-linear way and through mechanisms which are in general not completely known. It is then safer that the method of choice for analysing these interactions does not rely on any model or assumption on the nature of the data and their interactions.

Transfer entropy has emerged as a powerful tool to quantify directed dynamical interactions. In this chapter we compare different approaches to evaluate transfer entropy, some of them already proposed, some novel, and implement them in a freeware MATLAB toolbox. Applications to simulated and real data are presented in this chapter that might be considered as the first release of the toolbox.

In the next chapter we are going to describe another estimator that has been integrated in the toolbox. The performances of the new estimator will be compared to the methods proposed in this chapter in order to better lead the reader toward the comprehension of the pros and cons of the whole set of approaches.

Furthermore, we are going to extensively describe the toolbox itself further in this thesis. The reader will have the opportunity to understand how to set MuTE due to the detailed description of the parameters involved and the several examples provided. We also implemented a graphical user interface to widely spread the toolbox so that even users without a strong programming background can easily use MuTE.

3.1 Introduction

Since its first introduction by Schreiber [4] transfer entropy (TE) has been recognized as a powerful tool to detect the transfer of information between joint processes. The most appealing features of TE are that it has a solid foundation in information theory and it naturally incorporates directional and dynamical information. Moreover, the formulation of TE does not assume any particular model as underlying the interaction between the considered processes, thus making it sensitive to all types of dynamical interaction. The popularity of this tool has grown even more with the recent elucidation of its close connection with the ubiquitous concept of Granger causality [5], which has led to formally bridge information-theoretic and predictive approaches to the evaluation of directional interactions between processes. Given all these advantages, the TE has been increasingly used to assess the transfer of information in physiological systems with several applications in neurophysiology [6, 7, 8, 9]. It is worth noting that in this chapter when speaking of the transfer of information measured by TE we refer to the "predictive information transfer" intended as the amount of information added by the past (and present) states of a source process to the present state of a target process.

The estimation of TE from time series data which constitute realizations of the investigated physiological processes is complicated by a number of practical issues that need to be addressed and that are contributing to the development of several

recipes to compute this measure.

In this study we discuss three different approaches to evaluate the probability distribution function which constitutes the basis for TE in multivariate systems. In turn, each approach has to be paired with the choice of the time series past values which contribute information to the knowledge of the present state of a given target time series. The first choice is the classical Uniform Embedding framework (UE) that considers a fixed amount of past terms for each series; the second approach is quite recent and employs a Non-Uniform Embedding framework (NUE) [10, 11] iteratively selecting the most informative terms through an optimization criterion.

These recipes, some of them already established, some novel, are accordingly revisited or explained. Then, in order to contribute to the foundation of a common framework for the application of TE, we describe their implementation in a modular MATLAB toolbox. Several examples are presented allowing a critical comparison of UE and NUE approaches for all the three entropy estimators.

The chapter is organized as follows. We first provide an overview of TE. We then distinguish between UE and NUE approaches to the representation of the history of the observed processes. We describe in detail the methods used to estimate the probabilities involved in the evaluation of the TE and their implementation in the toolbox. The approaches are then validated on synthetic time series and then tested on real data: the electroencephalogram of an epileptic patient and cardiovascular measurements in healthy subjects.

3.2 Materials and Methods

3.2.1 Transfer entropy

Let us consider a composite system described by a set of M interacting dynamical subsystems and suppose that, within the composite system, we are interested in evaluating the information flow from the source system $\mathcal{X}$ to the destination system $\mathcal{Y}$, collecting the remaining systems in the vector $\mathcal{Z} = \left\{\mathcal{Z}^k\right\}_{k=1,\ldots,M-2}$. We develop our framework under the assumption of stationarity, which allows to perform estimations replacing ensemble averages with time averages (for non-stationary formulations see, e.g., [12] and references therein). Accordingly, we

denote X, Y and $\mathbf{Z}$ as the stationary stochastic processes describing the state visited by the systems $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ over time, and X_n, Y_n and $\mathbf{Z}_n$ as the stochastic variables obtained by sampling the processes at the present time n. Moreover, we denote $X_n^- = [X_{n-1}X_{n-2}\ldots]$, $Y_n^- = [Y_{n-1}Y_{n-2}\ldots]$, and $\mathbf{Z}_n^- = [\mathbf{Z}_{n-1}\mathbf{Z}_{n-2}\ldots]$ as the vector variables representing the whole past of the processes X, Y and $\mathbf{Z}$. In some cases it can be desirable to take into account also the instantaneous influences of the candidate drivers. In this case, the vectors X_n^- and $\mathbf{Z}_n^-$ defined above should contain also the present terms X_n and $\mathbf{Z}_n$. Then, the multivariate transfer entropy from X to Y conditioned to $\mathbf{Z}$ is defined as:

$$TE_{X\to Y|\mathbf{Z}} = \sum p\left(Y_n, Y_n^-, X_n^-, \mathbf{Z}_n^-\right) log \frac{p\left(Y_n|Y_n^-, X_n^-, \mathbf{Z}_n^-\right)}{p\left(Y_n|Y_n^-, \mathbf{Z}_n^-\right)} \tag{3.1}$$

where the sum extends over all the phase space points forming the trajectory of the composite system. $p(\mathbf{a})$ is then the probability associated with the vector variable $\mathbf{a}$ while $p(b|\mathbf{a}) = p(\mathbf{a},b)/p(\mathbf{a})$ is the probability of observing b knowing the values of $\mathbf{a}$. The conditional probabilities used in (3.1) can be interpreted as transition probabilities, quantifying to which extent the transition of the target system $\mathcal{Y}$ towards its present state is affected by the past states visited by the source system $\mathcal{X}$. Specifically, the TE quantifies the information provided by the past of the process X about the present of the process Y that is not already provided by the past of Y or any other process included in $\mathbf{Z}$.

The formulation presented in (3.1) is an extension of the original TE measure proposed for pairwise systems [4] to the case of multiple interacting processes. The conditional TE formulation, also denoted as partial TE [8, 11], rules out the information shared between X and Y that is mediated by their common interaction with $\mathbf{Z}$. Note that the TE can be seen as a difference of two conditional entropies (CE), or equivalently as a sum of four Shannon entropies:

$$\begin{aligned} TE_{X\to Y|\mathbf{Z}} &= H(Y_n|Y_n^-, \mathbf{Z}_n^-) - H(Y_n|Y_n^-, X_n^-, \mathbf{Z}_n^-) \\ &= H(Y_n, Y_n^-, \mathbf{Z}_n^-) - H(Y_n^-, \mathbf{Z}_n^-) \\ &\quad - H(Y_n, Y_n^-, X_n^-, \mathbf{Z}_n^-) + H(Y_n^-, X_n^-, \mathbf{Z}_n^-) \end{aligned} \tag{3.2}$$

TE has a great potential in detecting information transfer because it does not assume any particular model that can describe the interactions governing the system

dynamics, it is able to discover purely non-linear interactions and to deal with a range of interaction delays [7]. Recent research has proven that TE is equivalent to Granger Causality (GC) for data that can be assumed to be drawn from a Gaussian distribution, a case in which the data covariance is fully described by a linear parametric model [5, 13]. This establishes a convenient joint framework for both measures. Here we evaluate GC in the TE framework and compare this model-based approach with two model-free approaches.

3.2.2 Reconstruction of the system's past states and TE evaluation

We will discuss here the crucial issue of how to approximate the infinite-dimensional variables representing the past of the processes. This problem can be seen in terms of performing suitable conditioned embedding of the considered set of time series [14].

The main idea is to reconstruct the past of the whole system represented by the processes X, Y, $\mathbf{Z}$ with reference to the present of the destination process Y, in order to obtain a vector $V = [V_n^Y, V_n^X, V_n^{\mathbf{Z}}]$ containing the most significant past variables to explain the present of the destination. Once V is computed it is easy to evaluate TE as the difference of two CEs or through the four entropies using the whole V or convenient subsets of it according to equation (3.2).

Uniform embedding

The large majority of approaches applied so far to estimate TE implicitly follow uniform conditioned embedding schemes where the components to be included in the embedding vectors are selected a priori and separately for each time series. For instance the vector Y_n^- is approximated using the embedding vector $V_n^Y = [Y_{n-m} Y_{n-2m} \dots Y_{n-dm}]$, where d and m are respectively the embedding dimension and embedding delay (the same for X_n^- and $\mathbf{Z}_n^-$, approximated by V_n^X and $V_n^{\mathbf{Z}}$). In this way it is possible to distinguish between a first phase during which the past states are collected and a second phase during which the estimate of the entropy, and consequently of the CE, is evaluated by means of the chosen estimator, according

to the following pseudo-code:

1. build the vector $V = [V_n^Y, V_n^X, V_n^{\mathbf{Z}}]$;
2. use V and Y_n to evaluate the last two entropies of (3.2) and, consequently, the lowest CE term (CE2);
3. use $V \backslash V_n^X$ to evaluate the first two entropies of (3.2) and, consequently, the highest CE term (CE1);
4. compute TE as equal to the difference CE1 - CE2.

The obvious arbitrariness and redundancy associated with this strategy are likely to cause problems such as overfitting and detection of false influences [14]. Moreover one should assess which TE values are significant. The significance tests associated with TE estimation based on UE are different for model-based and model free estimators, and are described in the respective following subsections.

Non-uniform embedding

Non-uniform embedding constitutes the methodological advance, with respect to the state of art, that we implement as a convenient alternative to UE. This approach is based on the progressive selection, from a set of candidate variables including the past of X, Y, and $\mathbf{Z}$ considered up to a maximum lag (*candidate set*), of the lagged variables which are most informative for the target variable Y_n. At each step, selection is performed maximizing the amount of information that can be explained about Y by observing the variables considered with their specific lag up to the current step. This results in a criterion for maximum relevance and minimum redundancy for candidate selection, so that the resulting embedding vector $V = [V_n^X V_n^Y V_n^Z]$ includes only the components of X_n^-, Y_n^- and $\mathbf{Z}_n^-$, which contribute most to the description of Y_n. Starting from the full candidate set, the procedure which prunes the less informative terms is described below:

1. Get the matrix with all the candidate terms
   $\mathrm{MC} = [X_{n-1} \dots X_{n-l_X} Y_{n-1} \dots Y_{n-l_Y} \mathbf{Z}_{n-1} \dots \mathbf{Z}_{n-l_Z}]$, with l_X, l_Y, l_Z representing the maximum lag considered for the past variables of the observed processes;

these matrices will contain also the terms X_n and $\mathbf{Z}_n$ in case one wants to take into account instantaneous effects.

2. Run the procedure to select the most informative past variables and the optimal embedding vector:

   (a) Initialize an empty embedding vector $V_n^{(0)}$

   (b) Perform a while loop on k, where k can assume values from 1 to the number of initial available candidates, *numC*, in the MC matrix. At the k-th iteration, after having chosen $k-1$ candidates collected in the vector $V_n^{(k-1)}$:
   for $1 \leq i \leq$ number of current candidate terms

   - add the i-th term of MC , $W_n^{(i)}$, to a copy of $V_n^{(k-1)}$ to form the temporary storage variable $V_n' = [W_n^{(i)} V_n^{(k-1)}]$

   - compute the mutual information between Y_n and V_n', estimating the probability density function according to the chosen estimator

   (c) Among the tested $W_n^{(i)}$, select the term $\hat{W}_n$ which maximizes the mutual information

   (d) **if** $\hat{W}_n$ fulfills a test for *candidate significance*, as described below, include it in the embedding vector, $V_n^{(k)} = [\hat{W}_n V_n^{(k-1)}]$, delete it from MC and set $k = k+1$.

   (e) **else** end the procedure setting $k = numC + 1$ and returning $V = V_n^{(k-1)}$

3. Use Y_n and the full embedding vector $V = [V_n^X \, V_n^Y \, V_n^Z]$ to evaluate the third and fourth entropy values of (3.2) and, consequently, the lowest CE term (CE2)

4. Take the subset of V without the past states belonging to the source process, $[V_n^Y \, V_n^Z]$ to evaluate the first and the second term of (3.2) and, consequently,

the highest CE term (CE1)

5. compute TE subtracting CE2 from CE1.

As described above, candidate selection is performed maximizing the mutual information between the target variable and the vector of the candidates already selected, incremented by the candidate under examination. As we will see in the following sections, the practical implementation of this general criterion consists of an optimization process (i.e., minimization of the conditional entropy or maximization of the conditional mutual information, depending on the estimator chosen). The performances of the processes mentioned above in the reconstruction of the optimal embedding for an assigned target process are also discussed in [11].

The complexity of the algorithm concerns mainly step 2, in particular step 2(b), involving a *for* loop nested inside a *while* loop: in the worst case the body of the *for* loop is executed $numC^2$ times resulting in a complexity $\mathcal{O}(numC^2)$.

At step 2(d), the test for candidate significance is performed at the k-th step comparing the conditional mutual information between the target variable and the selected candidate given the candidates previously selected up to the $(k-1)$-th step, $I(Y_n;\hat{W}_n|V_n^{(k-1)})$, with its null distribution empirically built by means of a proper randomization procedure applied to the points of $\hat{W}_n$. The test for candidate significance is fulfilled if the original measure $I(Y_n;\hat{W}_n|V_n^{(k-1)})$ is above the $100(1-\alpha)^{\text{th}}$ percentile (where α is the desired significance level) of its null distribution. In order to maximize detection accuracy, the adopted randomization procedure is varied for each estimator, and is thus described in the relevant section.

Summarizing, the non-uniform embedding is a feature selection technique selecting, among the available variables describing the past of the observed processes, those who are the most significant - in the sense of predictive information - for the target variable. Moreover, given the fact that the variables are included into the embedding vector only if associated with a statistically significant contribution to the description of the target, the statistical significance of the TE estimated with the NUE approach results simply from the selection of at least one lagged component of the source process. In other words, if at least one component from X is selected by NUE, the estimated TE is strictly positive and can be assumed as statistically

significant. If this is not the case, the estimated TE results exactly zero and is assumed as non-significant. This latter case occurs also when the first candidate ($k = 1$) does not reach the desired level of significance, meaning that none of the candidates provides statistically significant information about the target variable. In such a case, that is encountered for instance when the target process is a white noise, the code returns an empty embedding vector and assigns a value of zero to the TE.

3.2.3 Entropy estimators

Estimation of the TE, performed according to either UE or NUE presented above, results from the application of estimators of entropy and CE to the various terms in (3.2). The toolbox contains three of such estimators. The first is the linear estimator (LIN) that assumes that data are drawn from a Gaussian distribution. Under this assumption, the two CE terms defining the TE can be quantified by means of linear regressions involving the relevant variables taken from the embedding vector [5]. The second estimator is the classical binning estimator (BIN), which consists of coarse-graining the observed dynamics using Q quantization levels, and then computing entropies by approximating probability distributions with the frequencies of occurrence of the quantized values [15]. The third estimator is based on k-nearest neighbor techniques (NN) which exploit the statistics of distances between neighboring data points in the embedding space to estimate entropy terms; we adopted the bias-reduction method of estimating entropies through neighbor search in the space of higher dimension and range searches in the subspaces of lower dimension [16].

A problem that can arise dealing with UE and NUE procedures when we use entropy estimators that does not assume any probability distribution concerns the curse of dimensionality. Indeed the more candidates we work with, the more the data points will be spread in the phase space, the more the probability density function will assume a constant value. Consequently the NUE should be the most apt method to avoid the curse of dimensionality because it reduces the dimension of the phase space. We will prove this statement in the Results section when it will be clear from the comparison between UE and NUE for the BIN and NN estimators in

multidimensional spaces. We are now going to introduce each estimator in detail.

Linear estimator (LIN)

The linear estimator method works under the assumption that the overall process $\{X, Y, \mathbf{Z}\}$ has a joint Gaussian distribution. This assumption allows to work with well-known expressions for the probability density functions. Under this assumption, the two CE terms defining the TE in (3.2) are expressed by means of linear regressions involving the past states of the systems collected in the vector variables [17]. When the UE is implemented, X_n^- is approximated with the vector of length p, $V_n^X = [X_{n-1}, \ldots, X_{n-p}]$, and the same for Y_n^- and Z_n^- which are approximated by $V_n^Y = [Y_{n-1}, \ldots, Y_{n-p}]$ and $V_n^{\mathbf{Z}} = [\mathbf{Z}_{n-1}, \ldots, \mathbf{Z}_{n-p}]$ (here $m = 1, p = d$). When the NUE is implemented, the embedding vectors will contain only the components resulting from the selection procedure. Then, an unrestricted regression of Y_n on the full vector $V^{(u)} = [V_n^X\, V_n^Y\, V_n^{\mathbf{Z}}]^T$, and a restricted regression of Y_n on the reduced vector $V^{(r)} = [V_n^Y\, V_n^{\mathbf{Z}}]^T$, are performed as follows:

$$Y_n = A^{(u)} V^{(u)} + \varepsilon_n^{(u)} \tag{3.3}$$

$$Y_n = A^{(r)} V^{(r)} + \varepsilon_n^{(r)} \tag{3.4}$$

where $A^{(u)}$ and $A^{(r)}$ are vectors of linear regression coefficients. The terms $\varepsilon_n^{(u)}$ and $\varepsilon_n^{(r)}$ are scalar white noise residuals with variance $\sigma^{(u)}$ and $\sigma^{(r)}$. Under the joint Gaussian assumption, it has been demonstrated [5] that the entropy of Y_n conditioned to the unrestricted or restricted regression vectors is, respectively, $H(Y_n|V^{(u)}) = 0.5(\log \sigma^{(u)} + 2\pi e)$ and $H(Y_n|V^{(r)}) = 0.5(\log \sigma^{(r)} + 2\pi e)$, from which follows immediately that:

$$\mathrm{TE}_{X \to Y|Z} = \frac{1}{2} \log \frac{\sigma^{(r)}}{\sigma^{(u)}} \tag{3.5}$$

In this study, the unrestricted and restricted regression models in (3.3) and (3.4) were estimated by the least-squares method. In the UE implementation, the order p of the regressions was selected by the Bayesian information criterion [18]; in the NUE implementation, the order resulted implicitly from the selection procedure. In NUE, maximization of the mutual information between the component $\hat{W}_n$ selected at the

step k and the target variable Y_n (step 2d) was obtained in terms of minimization of the CE $H(Y_n|\hat{W}_n, V_n^{k-1}) = 0.5(\log \sigma^{(k)} + 2\pi e)$, where $\sigma^{(k)}$ denotes the variance of the residuals of the linear regression of Y_n on $[\hat{W}_n, V_n^{k-1}]$. Here, the randomization procedure applied to test candidate significance consisted time-shifting the points of $\hat{W}_n$ by a randomly selected lag (of at least 20 lags, set to avoid autocorrelation effects) [19].

The statistical significance of the TE estimated through the UE approach is assessed by a parametric F-test for the null hypothesis that the p coefficients of $A^{(u)}$ which weigh the past components of the driving process, collected in V_n^X, are all zero [20]. In this case, the test statistic is $F = ((RSS_r - RSS_u)/p)/(RSS_u/(N - Mp))$, where RSS_r and RSS_u are the residual sum of squares of the restricted and the unrestricted model, and N is the time series length. The TE is considered statistically significant if F is larger than the value of the Fisher distribution with $(p, N-p)$ degrees of freedom at the significance level $\alpha = 0.05$.

Binning estimator (BIN)

Here we describe the estimator based on fixed state space partitioning. This approach consists of an uniform quantization of the time series followed by estimation of the entropy approximating probabilities with the frequency of visitation of the quantized states [15]. This is the classical approach adopted in the first definition of TE [4]. A time series y, realization of the generic process Y, is first normalized to have zero mean and unit variance, and then coarse grained spreading its dynamics over ξ quantization levels of amplitude $r = (y_{max} - y_{min})/\xi$, where y_{max} and y_{min} represent minimum and maximum values of the normalized series. Quantization assigns to each sample the number of the level to which it belongs, so that the quantized time series y^{ξ} takes values within the alphabet $\mathcal{A} = (0, 1, \ldots, \xi - 1)$. Uniform quantization of embedding vectors of dimension d results in an uniform partition of the d-dimensional state space into ξ^d disjoint hypercubes of size r, such that all vectors V falling within the same hypercube are associated with the same quantized vector V_{ξ}, and are thus indistinguishable within the tolerance r. The

entropy is then estimated as:

$$H(V_\xi) = -\sum_{V_\xi \in A^d} p(V_\xi) log\, p(V_\xi) \tag{3.6}$$

where the sum is extended over all vectors found in the available realization of the quantized series, and the probabilities $p(V_\xi)$ are estimated for each hypercube simply as the fraction of quantized vectors V_ξ falling into the hypercube (i.e., the frequency of occurrence of V_ξ within A_d). According to this approach, the estimate of TE based on binning results from the application of (3.6) to the four embedding vectors defined in (3.2) and determined either by UE or by NUE.

In the NUE implementation, maximization of the mutual information between the component $\hat{W}_n$ selected at the step k and the target variable Y_n (step 2d) was obtained in terms of minimization of the CE $H(Y_n|\hat{W}_n, V_n^{(k-1)}) = H(Y_n, \hat{W}_n, V_n^{(k-1)}) - H(\hat{W}_n, V_n^{(k-1)})$, with the two entropy terms estimated through the application of (3.6) . As for the LIN estimator, the randomization procedure applied to test candidate significance consisted in time-shifting the points of $\hat{W}_n$ by a randomly selected lag [19].

The statistical significance of the TE estimated through the BIN UE approach exploited the method of surrogate data implemented by the time-shift procedure proposed in [14, 21, 19]. Specifically, the estimated TE is tested against its null distribution formed by the values of TE computed on replications of the original series, where in each replication the source series is time-shifted by a randomly selected lag, set to exclude autocorrelation effects.

Nearest Neighbor estimator (NN)

Since its first introduction in 1967 [22], the nearest neighbor method has been shown to be a powerful non-parametric technique for classification, density estimation, and regression estimation. This method can be used to estimate the entropy of a d-dimensional random variable X, $H(X)$, starting from a random sample $(x_1, \ldots, x_n)$ of N realizations of X. Following the reasoning in [16], if we consider the probability distribution $P_k(\varepsilon)$ for the distance between x_i and its k-th nearest neighbor, the probability $P_k(\varepsilon)d\varepsilon$ is equal to the chance that there is one point lying within a

distance $r \in [\varepsilon/2, \varepsilon/2 + d\varepsilon/2]$ from x_i, that there are $k-1$ other points at smaller distances from it, and that $N-k-1$ points have larger distances from x_k. Let p_i be the mass of the ε-sphere centered at x_i, $p_i(\varepsilon) = \int_{\|\xi - x_i\| < \varepsilon/2} \mu(\xi) d\xi$, where $\mu(\xi)$ is the density of the variable ξ. Then, the expectation value of $log(p_i(\varepsilon))$ is

$$E(log(p_i)) = \int_0^{\inf} P_k(\varepsilon)\, log(p_i(\varepsilon)) d\varepsilon = \psi(k) - \psi(N) \tag{3.7}$$

where $P_k(\varepsilon)$ is evaluated through the trinomial formula and $\psi(\cdot)$ is the digamma function. The expectation is taken here over the positions of all other $N-1$ points, with x_i kept fixed. An estimator for $log(\mu(x))$ is then obtained by assuming that $\mu(x)$ is constant in the entire ε-sphere. The latter gives

$$p_i(\varepsilon) \approx c_d \varepsilon^d \mu(x_i) \tag{3.8}$$

where d is the dimension of x and c_d is the volume of the d-dimensional unit sphere. For the maximum norm one has simply $c_d = 1$, while $c_d = \pi^{\varepsilon/2}/\Gamma(1+d/2)/2^d$ for the Euclidean norm. From (3.7) and (3.8) we can evaluate $log(\mu(x_i))$ and finally:

$$H(X) = -\psi(k) + \psi(N) + log(c_d) + \frac{d}{N} \sum_{i=1}^{N} log(\varepsilon(i)) \tag{3.9}$$

The NN estimator faces the issue of the bias in the estimation of multiple entropies for vector variables of different dimensions by computing entropy sums through a neighbor search in the space of higher dimension, and range searches in the projected sub-spaces of lower dimensions [16]. This approach can be fruitfully exploited for the estimation of the TE, as previously done, e.g., in [6, 7]. To do this, we first rewrite the expression for TE in (3.2) in terms of the components of the embedding vector $V = [V_n^Y, V_n^X, V_n^{\mathbf{Z}}]$ spanning a space of dimension $(d_X + d_Y + d_Z)$:

$$TE_{X \to Y|\mathbf{Z}} = H(Y_n, V_n^Y, V_n^{\mathbf{Z}}) - H(V_n^Y, V_n^{\mathbf{Z}}) - H(Y_n, V) + H(V) \tag{3.10}$$

The term $H(Y_n, V)$ is estimated through neighbor search in the $(d_X + d_Y + d_Z + 1)-$dimensional space, while the three other terms are estimated through range

searches in the spaces of dimension $(d_Y + d_Z + 1)$, $(d_Y + d_Z)$ and $(d_X + d_Y + d_Z)$. Accordingly, adaptation of (3.9) to the four terms in (3.10) yields the equation for TE based on the nearest neighbor estimator:

$$TE_{X\to Y|\mathbf{Z}} = \psi(k) + \left\langle \psi(N_{V_n^Y V_n^Z} + 1) - \psi(N_{Y_n V_n^Y V_n^Z} + 1) - \psi(N_V + 1) \right\rangle \quad (3.11)$$

where $N_{V_n^Y V_n^Z}$,$N_{Y_n V_n^Y V_n^Z}$ and N_V are the number of points whose distance from $[V_n^Y, V_n^{\mathbf{Z}}]$, $[Y_n, V_n^Y, V_n^{\mathbf{Z}}]$ and V, respectively, is strictly less than the distance from $[Y_n, V]$ to its k-th neighbor, and $< \cdot >$ denotes average over all n.

In the NUE implementation of the NN estimator, maximization of the mutual information between the component $\hat{W}_n$ selected at the step k and the target variable Y_n (step 2d) was obtained in terms of maximization of the conditional mutual information $I(Y_n; \hat{W}_n | V_n^{(k-1)})$, which was computed as described above by estimating the four relevant entropies through a neighbor search in the complete space, and range searches in the projected sub-spaces of lower dimensions. Moreover, the randomization procedure applied to test candidate significance consisted in shuffling randomly and independently both the points of $\hat{W}_n$ and those of Y_n. These techniques have been recently shown to be optimal for the selection of candidates in a non-uniform embedding approach using nearest neighbor entropy estimators [11]. As for the BIN UE method, the statistical significance of the TE estimated through the NN UE approach exploited the method of surrogate data implemented by the time-shift procedure proposed in [14, 19, 21].

3.2.4 Toolbox structure

This section describes how the three TE estimators presented above are implemented in the toolbox, exploiting either the UE or the NUE approach for system state reconstruction.

The same main structure, consisting of the following steps, is common to all methods:

1. normalize the data and perform quantization when needed;
2. evaluate the probability density function (PDF);

3. evaluate CE2 (the second conditional entropy in (3.2)). This term, accounting for the present state of the target series conditioned to the past of the remaining series including the driver, is evaluated first since it is needed to obtain the complete set of conditional terms including all the series;

4. evaluate CE1 (the first conditional entropy in (3.2)): this term accounts for the present state of the target series conditioned to a vector including the past of the target series and of the all other series except the driver; such a vector is obtained subtracting the candidates belonging to the driver series from the set of candidates evaluated in the previous step.

Keeping this general scheme in mind, specific steps will be performed for any method of choice. For instance, when using the NUE with the BIN estimator, the steps to be performed are:

1. data quantization;

2. estimation of the PDF, as described in *Binning estimator* section;

3. evaluation of the first and second transfer entropy terms according to *Non-uniform embedding* section.

Given the modularity of the structure shown previously it has been possible to build a user friendly toolbox that allows one to compare all the methods at the same time. The toolbox is available at this link `http://mutetoolbox.guru/downloads/` The package also contains two existing MATLAB toolboxes which are used in some of the calculations: ARFIT [23], a collection of modules for modeling and analyzing multivariate time series with autoregressive models, used for choosing the model order in LIN UE, and OPENTSTOOL [24], a software package for signal processing with emphasis on nonlinear time-series analysis, and used in searching for neighbor in NN. In order to optimize the toolbox for speed, the routine *evaluateEntropy*, that estimates the entropy among variables according to $entropy = -\sum p\,log(p)$, has been converted in a .cpp executable substantially reducing the computation time.

In the following we provide guidelines for the use of the toolbox. Let's start from a hypothetical *main* function and let's explore how a user should set the

parameters to chose which methods to use and, possibly, how to build a new method to be inserted within the toolbox.

In the *exampleMain* file, included in the folder */MuTE/exampleToolbox*, some commented lines remind the method order that has to be kept in mind when setting the parameters, and the parameters available for each method. A first part then follows, devoted to setting the name of the folder that contains data as, for instance, *.mat* files. Each file should contain a matrix with the time series as the rows. Then the folders in which all the output files will be stored are defined. In the second part the function *parametersAndMethods* is called.

The function *parametersAndMethods* requires the following inputs, as reported in table 3.1

- the number of data realizations;
- the sampling rate;
- the subset of interest;
- a value to cut the series length if necessary
- a vector specifying whether each method will take into account all the pairwise combinations of chosen variables. By default the instantaneous effects won't be considered;
- a vector specifying whether the user will set by hand all the pairwise combinations of the chosen variables. This vector will also be used for visualizing the output. It is worth noting that in this case the user should provide as input also the sequence of the destination series and the driver series;
- the folder in which results can be stored, previously defined;
- the folder in which data are stored, previously defined;
- the folder in which results can be eventually copied;
- the number of processors if the code can be run in parallel on several nodes;

- the name of the method chosen and all the relevant parameters as shown in the comments. Here attention should be paid in setting four parameters if the instantaneous effects have to be considered. First of all the function choosing the candidate terms should be set and consequently the variable *usePresent*: *generateConditionalTerm*, and *usePresent* $= 0$ if the instantaneous effects do not have to be taken into account, *generateCondTermLagZero* and *usePresent* $= 1$ otherwise. Then, if one is interested in the action of more than one driver on a target series, for each driver it can be specified whether its instantaneous effect should be considered by writing twice in a row the number of the driver series. One can also choose which variables belonging to the $\mathbf{Z}$ set can be considered with their instantaneous effects, filling the vector *idOtherLagZero*, table 3.2, third column.

  For an example of how these parameters should be set, let's consider 5 variables; a conditioned analysis and a vector *idTargets* $= [1\,2\,3\,4\,5]$ would result in the situation shown in table 3.2, second column, in which no instantaneous effect are set and the variable *idDrivers* contains on the columns the id of the driver series only once and the variable *idOtherLagZero* is the null vector. An example considering instantaneous effects is reported in table 3.2, third column, when looking at how drivers 1 and 4 influence the target 2 and how drivers 5 and 2 influence the target 3, with series 5 and 2 as conditioning variables.

The input parameters, including the methods of choice, specified in the function *createNameMethodParams* are stored in a structure called *params* by the function *parametersAndMethods*. This function then computes TE according to the chosen methods, via the function *callingMethods*, and stores the significant results through the function *storingOutput*. In case of multiple realizations/data sets to analyze, the computation can be performed in parallel on separate pools.

The description of the toolbox structure should take into account the structure of the function *callingMethods* that receives in input the data matrix with the time series points in row and the structure *params*. The function reads the names of the methods stored in the *params* structure and computes the TE with all the chosen methods (in parallel if the hardware architecture allows it). This function will return

a cell array containing the output of each method.

The open structure of the toolbox allows users to integrate in it their own method. The *main* function should in this case be modified with some comments showing which parameters should be passed as an input to *parametersAndMethods*, and in which order. Each new method should then be implemented following the steps described above using all the necessary parameters conveniently grouped via the function *createNameNewMethodParams*. The new method will be called by setting the appropriate name in *callingMethods*.

The execution time for a single run of the system 3.14 ranged from 0.4 s for the LIN UE to 90.4 s for NN NUE on a Dell Mini Tower Computer, OptiPlex 990 with four Intel Core i5-2400 CPU at 3.10GHz, 16 GB of RAM.

One of the purposes of this toolbox is to provide a common framework for all the researchers interested in the application of Transfer Entropy to their data. As part of this effort, MuTE will soon be interfaced with the toolbox TRENTOOL [25]. Readers and users are invited to check periodically the webpages of both toolboxes, that will announce when this interface has been set up.

3.2.5 Simulated data

The first set of simulated data, implemented to validate the simplest approach to TE, BIN UE, consists of two coupled chaotic maps:

$$\begin{aligned} X_{1,n} &= 1 - \beta\, b_1^2 + d\,\varepsilon_n \\ X_{2,n} &= (1 - C_1)\,(1 - \beta\, b_2^2) + C_1\,(1 - \beta b_1^2) + d\,\varepsilon_n \end{aligned} \tag{3.12}$$

where $C_1 = 0.2$ is the coupling coefficient according to which X_1 is influencing X_2, $b_1 = |mean(x_{1,n-1})|$, $b_2 = |mean(x_{2,n-1})|$, $\beta = 1.8$, $d = 0.03$ is the coefficient that regulates the noise and ε is a Gaussian noise [26]. The function generating these data is *multichaoticmap* available in the folder */MuTE/commonFunctions*.

In the second experiment we simulated five time series in two cases: linear time series, for which we can assume a normal distribution of the variables, and non-linear ones, both generated by an autoregressive (AR) model, equations (3.13), (3.14) [27]. The following equations are for the linear Gaussian autoregressive

model:

$$
\begin{aligned}
X_{1,n} &= 0.95\sqrt{2}X_{1,n-1} - 0.9025X_{1,n-2} + w_{1,n} \\
X_{2,n} &= 0.5X_{1,n-2} + w_{2,n} \\
X_{3,n} &= -0.4X_{1,n-3} + w_{3,n} \\
X_{4,n} &= -0.5X_{1,n-2} + 0.25\sqrt{2}X_{4,n-1} + 0.25\sqrt{2}X_{5,n-1} + w_{4,n} \\
X_{5,n} &= -0.25\sqrt{2}X_{4,n-1} + 0.25\sqrt{2}X_{5,n-1} + w_{5,n}
\end{aligned}
\tag{3.13}
$$

where $w_{1,n}, w_{2,n}, w_{3,n}, w_{4,n}, w_{5,n}$ are drawn from Gaussian noise with zero mean and unit variance. The following are the equations for the non-linear model:

$$
\begin{aligned}
X_{1,n} &= 0.95\sqrt{2}X_{1,n-1} - 0.9025X_{1,n-2} + z_{1,n} \\
X_{2,n} &= 0.5X_{1,n-2}^2 + z_{2,n} \\
X_{3,n} &= -0.4X_{1,n-3} + z_{3,n} \\
X_{4,n} &= -0.5X_{1,n-2}^2 + 0.25\sqrt{2}X_{4,n-1} + 0.25\sqrt{2}X_{5,n-1} + z_{4,n} \\
X_{5,n} &= -0.25\sqrt{2}X_{4,n-1} + 0.25\sqrt{2}X_{5,n-1} + z_{5,n}
\end{aligned}
\tag{3.14}
$$

where $z_{1,n}, z_{2,n}, z_{3,n}, z_{4,n}, z_{5,n}$ are drawn from Gaussian noise with zero mean and unit variance. A schematic representation of the simulated couplings, valid for both systems, is reproduced in figure 3.1. The function generating these data is *generateTS* available in the folder */MuTE/commonFunctions*.

3.2.6 Electroencephalogram in epilepsy

The second experiment is performed on intracranial electroencephalography (EEG) measurements recorded from a patient with refractory epilepsy. The dataset consists of time series from 76 contacts. The first sixty-four of these contacts were placed on a 8x8 grid at the cortical level, while the other 12 were along two six-electrode strips that were implanted in deeper brain structures. Eight sets of measurements were taken on this patient, corresponding to eight different epileptic seizures. An epileptologist, examining the data for each seizure, identified two key periods relating to the seizure i.e., a pre-ictal period, just before the clinical onset, and an

ictal one, corresponding to the seizure spread and to the clinical symptoms. Each epoch contained 10 seconds of data recorded at 400 Hz. The data are available at `http://math.bu.edu/people/kolaczyk/datasets.html` and described in [28]. In order to reduce overfitting, in this application data were downsampled to 100 Hz.

3.2.7 Cardiovascular and Cardiorespiratory time series

We considered cardiorespiratory time series measured from 15 young healthy subjects (25.7 ± 2.7 years old) undergoing a standard head-up tilt testing protocol [29]. The acquired signals were the surface electrocardiogram (ECG), the finger arterial blood pressure, and the respiratory nasal flow, measured at 1 kHz sampling rate for 15 minutes in the resting supine position, and 15 further minutes in the 60° position after passive head-up tilting of the bed table. From these signals, the beat-to-beat variability series of heart period (RR interval), *RR(n)*, systolic arterial pressure (SAP), *Sap(n)*, and respiratory activity, *Resp(n)*, were offline measured respectively as the temporal interval occurring between the n-th and the $(n+1)$-th R waves of the ECG, as the local maximum of the systolic arterial pressure signal inside the n-th heartbeat, and as the nasal flow taken at the onset of the n-th heartbeat. The time series are available in the folder */MuTE/cardiovascular_data*. This measurement convention allows instantaneous effects from *Sap(n)* to *RR(n)*, as well as from *Resp(n)* to *Sap(n)* and to *RR(n)*, which were implemented using the relevant feature of the toolbox. The subsequent data analysis was performed on stationary windows of 300 beats taken in supine and upright body positions; inside these windows, the series were normalized to zero mean and unit variance, obtaining the dimensionless series *resp(n)*, *sap(n)*, *rr(n)*.

3.3 Results

3.3.1 Simulated data

The aim of testing the BIN UE approach on the coupled maps of eq. 3.12 was to show a simple case of applicability for this method, which constitutes the most

basic approach to the model-free evaluation of TE. We generated 100 realizations of eq. 3.12, each of 512 points, and performed the analysis setting 1 as maximum lag for the candidates, 100 surrogates, $\alpha = 0.05$ and 6 quantization levels. As we can see in figure 3.2, the method detected correctly the information transfer returning 100 significant realizations for the link $X_1 \to X_2$ and an average TE much higher than the average TE for the link $X_2 \to X_1$ by means of the detection of only 2 significant realizations over 100. We tested also the other methods, which gave similar positive results as the BIN UE, thus demonstrating the applicability of the toolbox for simulations of bivariate systems with short memory.

Then we moved to a more challenging situation in terms of number of interacting systems and lag of the interaction effects, considering the time series simulated with equations 3.13 and 3.14, which involve five systems and contain influences up to 3 points in the past. The experiments were run on 100 realizations of eqs. 3.13 and 3.14, of length equal to 512 points. We investigated the TE between each pair of variables conditioned to the other three. The setup of the experiment was the following: for all estimators, used either in the UE or in the NUE framework, the maximum lag for the candidates was set as 5, the number of surrogates was fixed to 100 and $\alpha = 0.05$. We set 6 quantization levels for BIN and 10 nearest neighbor for NN estimator.

In order to check whether the methods were able to detect the right information transfers, taking into account figure 3.1, we expect the estimators to find a TE greater than zero with the highest significance at the following matrix elements: (1,2), (1,3), (1,4), (4,5), (5,4). Figures 3.3 and 3.4 report the analysis results obtained respectively for the linear system and the non-linear system. Looking at Figure 3.3 one can notice that LIN UE has very good performances: this reflects the fact that this approach is, in this case of a linear AR system, “by construction”, the most likely to correctly detect information transfer. Its NUE version can detect the same links between the variables, though with a slightly higher number of false positives. The LIN estimator, therefore, is able to reveal the correct information flows for this simulation. On the contrary, BIN UE suffers from the curse of dimensionality mentioned in *Entropy estimators* section: evaluating the influences up to the first 5 past points for all the series implies that the uniform embedding procedure projects the data into a phase space of $M \times 5$ dimensions, where M is

the number of time series, resulting in a phase space with 25 dimensions, with the points spread enough to lose relevant information about the transfer entropies in the system. As a consequence, no significant link is retrieved with this approach. NN UE retrieves all the true links, but also detects a number of false interactions. Its better performance compared with BIN UE reflects the ability of the nearest neighbor approach to achieve bias compensation in the estimation of entropies of variables of different dimension. Still, the performance of NN UE is not optimal due to the curse of dimensionality. On the other hand, BIN and NN used in the NUE framework are able to recover all the correct links, with only a few false positives. Moving to Figure 3.4 depicting TE analysis for the non-linear systems, one can notice that the LIN estimator cannot detect all the correct information flows, returning in addition some false positives. Again, BIN UE cannot detect any link because of the curse of dimensionality; conversely BIN NUE, in which the dimensionality of the space is considerably reduced, has high specificity and sensitivity. NN NUE can achieve almost the same performance as BIN NUE but its specificity is lower, especially along the direction $X_2 \to X_4$. NN UE this time is not able to detect all the correct information transfers ($X_5 \to X_4$ remains undetected) and reveals some false positives ($X_2 \to X_4$, $X_2 \to X_5$).

To better clarify whether and how much the methods are able to distinguish between the true information transfer links and the false ones, in Figures 3.5 and 3.6 we plotted the average TE with respect to the number of significant realizations found by the methods. Each retrieved link is a point in this bidimensional space. The true links should be in the upper right corner of the plot corresponding to high TE and high number of significant realizations, and they should be apart from the false links, whose natural location would be around the origin of the plot (low TE and low number of significant links). Looking at figure 3.5 one can notice that for all the methods, except BIN UE and partly NN UE, the two groups of links are well separated and the false links with an averaged TE greater than zero in figure 3.3 can be neglected. The opposite reasoning holds for BIN UE that is not able to distinguish between false and true links. For the non-linear system (figure 3.6) only BIN NUE can separate well true positive from false positive links.

To understand how stable the performance of the methods is, in terms of sensitivity and the specificity, with respect to the length of the analyzed data set, we

computed the analysis varying the series length from 128 to 1024 points. Figures 3.7 and 3.8 depict, respectively for the systems 3.13 and 3.14, the Receiver Operating Characteristic (ROC) curves obtained for all methods as a function of the series length. Evaluating the amount of TP (true positives), TN (true negatives), FP (false positives) and FN (false negatives) after grouping together all coupled directions (positives) and all uncoupled directions (negatives), we computed sensitivity as $TP/(TP+FN)$ and specificity as $TN/(TN+FP)$. In the case of the linear system (Figure 3.7), all methods except the BIN UE provide good performance, with the LIN estimator providing the best sensitivity and specificity. All methods provided robust results with respect to the series length, with only a limited decay in the performance observed for 128 points. In the case of the non-linear system (Figure 3.8), the performance was optimal for BIN NUE and NN NUE (with a slightly lower specificity), while the methods implementing either the LIN estimator or the UE approach were considerably less sensitive.

3.3.2 Electroencephalogram in epilepsy

In such high dimensional and redundant data, a non-uniform embedding approach is intuitively the most appropriate to identify the patterns of information transfer specific to the onset and spread of the epileptic seizure. The aim of the experiment was to use the NUE approach in order to characterize the dynamical interactions in the epileptic brain by looking at the information transfer between the variables during the pre-ictal and ictal phases. The embedding size in the embedding matrix (EM) was set to eight. The results are reported in figure 3.9. The regions corresponding to one of the depth strips (contacts 70 to 76) and the lower left corner of the grid (contacts 1-4, 9-11 and 17) were resected during anterior temporal lobectomy as they were identified by the epileptologists as the seizure onset zone. The Binning approach to NUE seems to be the one which best identifies these areas as those most influential at the start of the seizure and in the early phases of the spread, signature of a putative seizure onset zone. The Binning approach is more selective with respect to the target variables for each driver and less sensitive to the confounding effect of volume conduction resulting in the diagonal patterns observed with the other methods and probably due to conduction effects on the grid.

3.3.3 Cardiovascular data

The analysis of the information transfer for cardiovascular and cardiorespiratory time series was focused on the directions of interaction that are more studied from a physiological point of view: the link from SAP to RR which is related to the so-called cardiac baroreflex, and the links originating from Resp and directed either to RR or to SAP, related respectively to cardiopulmonary or vasculo-pulmonary regulation mechanisms [29]. The particular protocol considered allows to establish a sort of verifiable ground truth. Indeed, in the studied protocol, the transition from supine to upright is known to evoke an activation of the sympathetic nervous system and a concurrent deactivation of the parasympathetic nervous system [30]. Accordingly, the two main physiological regulation mechanisms that are expected to be solicited by this transition are: (i) a substantial increase of the baroreflex regulation (direction $sap \rightarrow rr$), reflecting the necessity of the cardiovascular system to react with changes in the heart rate to the higher fluctuations in the arterial pressure induced by the sympathetic activation; and (ii) a substantial decrease of cardiopulmonary regulation (direction $resp \rightarrow rr$), reflecting the dampening of respiratory sinus arrhythmia consequent to the parasymphatetic deactivation [31]. On the contrary, no known alterations of the vasculo-pulmonary regulation (direction $resp \rightarrow sap$) are expected when moving from supine to upright [29]. In our analysis all these trends are well reflected in terms of information transfer when the multivariate TE is estimated using the BIN NUE and NN NUE methods. Figure 3.10 reports the distribution of the multivariate TE computed along these directions using all methods, with subjects studied in the supine and upright body positions. We observe in Figure 3.10 that BIN NUE and NN NUE reveal, moving from supine to upright, a substantial increase of the TE from Sap to RR, a substantial decrease of the TE from Resp to RR, and an unchanged TE from resp to SAP. These trends were also observed, though with less evident differences, computing the TE according to the LIN estimator. These results suggest the appropriateness of model free TE estimators based on NUE for detecting the information transfer in physiological time series. On the contrary, the BIN UE estimator shows different trends of difficult physiological interpretation, thus suggesting also in experimental data that the estimated TE may be unreliable due to the curse of dimensionality.

3.4 Conclusions

In this work we have considered three entropy estimators able to reveal the information transferred among variables represented by time series. We implemented the estimators in two different ways according to UE and NUE approaches, resulting in six methods, two of which are novel, BIN NUE and NN NUE. We compared all the methods validating them on simulated data first and then on real data. We checked whether and how the methods were affected by the number of variables and by the time lag at which the series influenced each other. From the results obtained we can conclude that the new methods introduced, not assuming any model to explain the data and exploiting the NUE strategy for component selection, can detect the correct information flows and are less affected by the number of involved processes and by their interaction lags. The NUE approaches are indeed prone to work in high dimensional spaces as well as in low dimensional spaces because of their ability to reduce the effective dimension of the phase space, choosing only the right variables at the specific time lag that are better able to explain the destination series. On the contrary, BIN UE and NN UE suffer from the curse of dimensionality when several time series and longer interaction delays are present. Finally, looking at LIN UE and LIN NUE performances we can conclude that, even though the equivalence between Granger causality and TE establishes a convenient joint framework for these two measures, there are some drawbacks in having a predefined model to explain the data when these are non-linear. The better performances obtained by the new methods appear when looking at the ROC curves: BIN NUE and NN NUE have high sensitivity and specificity both for linear and non-linear systems.

All the methods have been implemented in an organic toolbox in MATLAB, allowing straightforward comparisons between the methods, and flexible enough to allow other users to implement their own methods.

Acknowledgments

This work is supported by: the Belgian Science Policy (IUAP VII project CEREBNET P7 11); the University of Gent (Special Research Funds for visiting researchers)

Figures and Tables

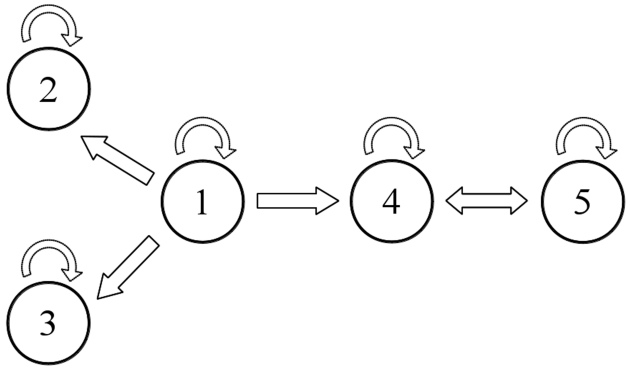

Figure 3.1: Simulated system. Interactions between the variables of the simulated system.

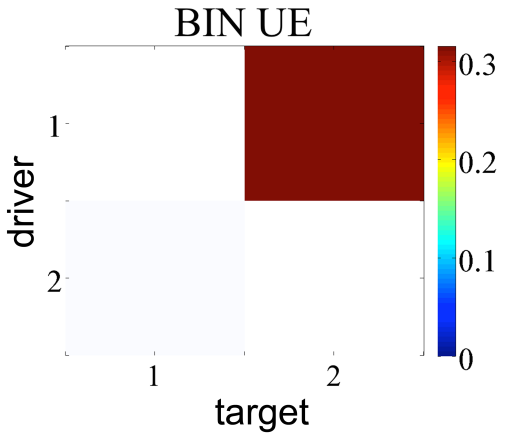

Figure 3.2: TE matrix representation for the BIN UE estimator applied to the system 3.12. The two bars depict the TE evaluated from X_1 to X_2 (up) and from X_2 to X_1 (down). Each bar is color-coded according to the value of the mean TE over 100 realizations of the simulation, with the bar height representing the number of realizations for which the TE was detected as statistically significant.

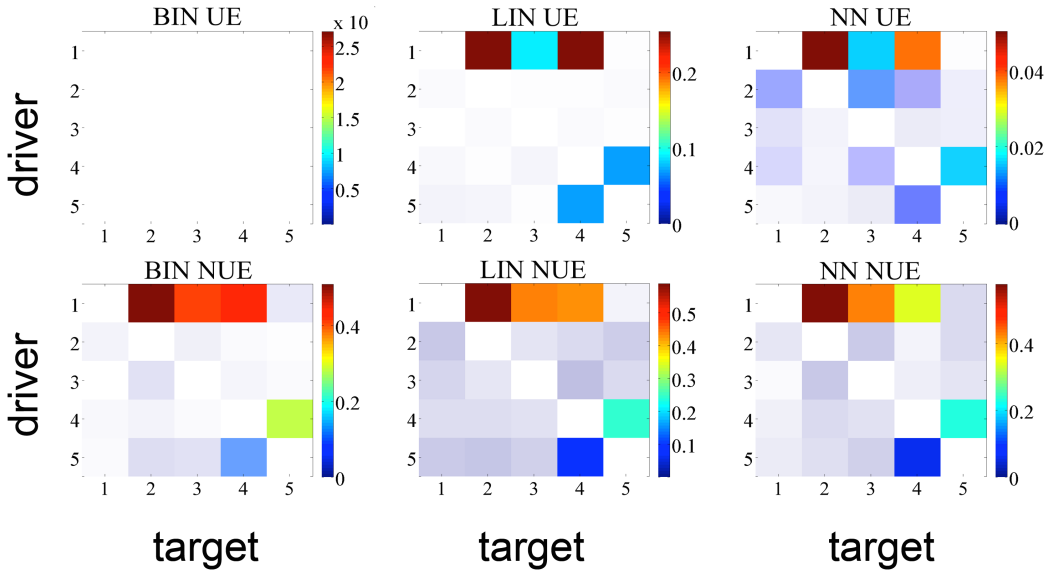

Figure 3.3: TE matrix representation for all the methods with linear time series of 512 points. Bars depict the TE evaluated for all the possible combinations of pairs of variable, conditioned to the other three variables (source: index on the left; destination: index on the right). Bars are color-coded according to the value of the mean TE over 100 realizations of the simulation, with the bar height representing the number of realizations for which the TE was detected as statistically significant

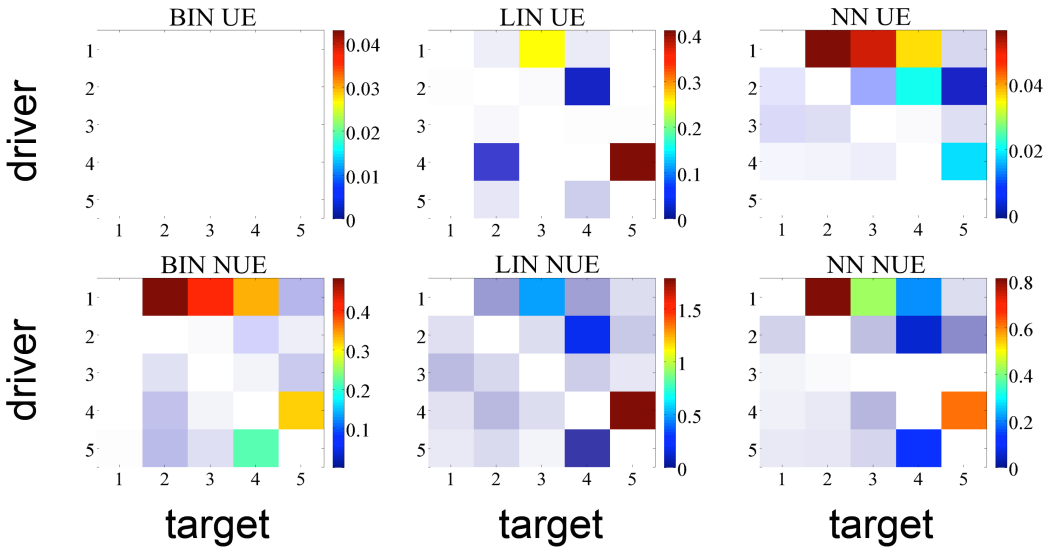

Figure 3.4: TE matrix representation for all the methods with non linear time series of 512 points. Bars depict the TE evaluated for all the possible combinations of pairs of variable, conditioned to the other three variables (source: index on the left; destination: index on the right). Bars are color-coded according to the value of the mean TE over 100 realizations of the simulation, with the bar height representing the number of realizations for which the TE was detected as statistically significant.

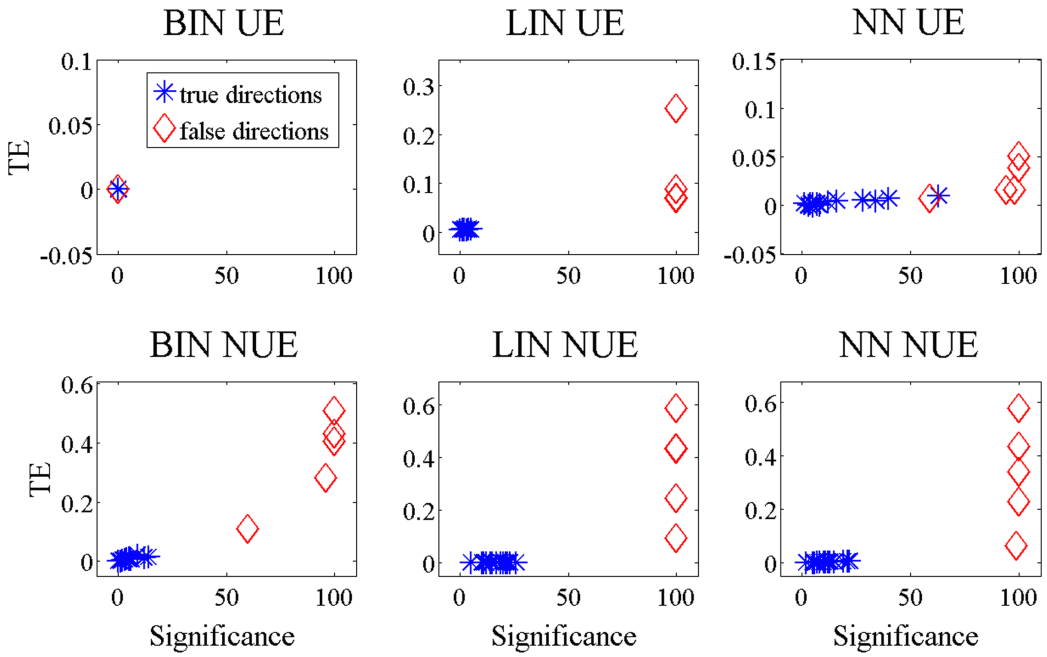

Figure 3.5: TE values versus the number of significant realizations. We plotted each link with time series of 512 points, for the different methods, on system 3.13. In red the five true links and in blue the other ones.

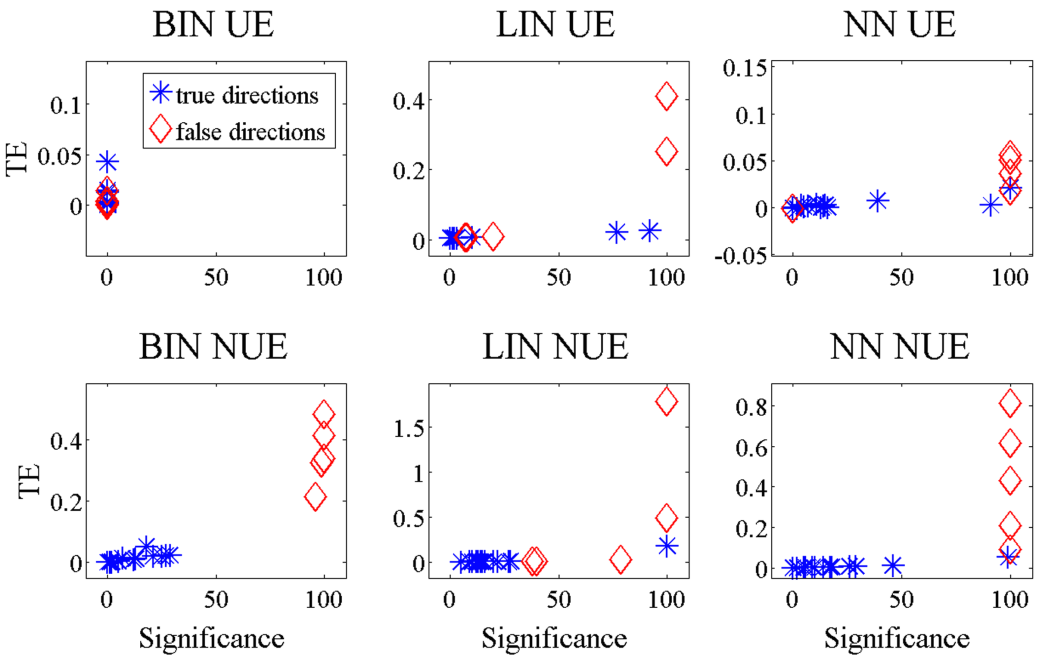

Figure 3.6: TE values versus the number of significant realizations. We plotted each link with time series of 512 points, for the different methods, on system 3.14. In red the five true links and in blue the other ones.

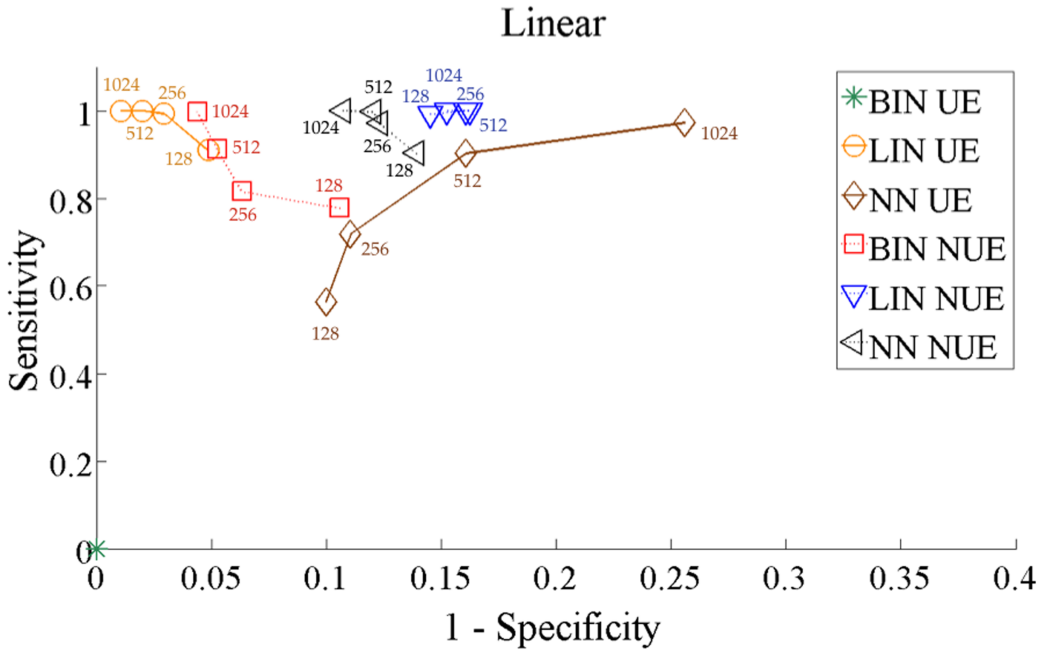

Figure 3.7: ROC curves for all methods for the system 3.13. We varied the series length from 128 up to 1024 points.

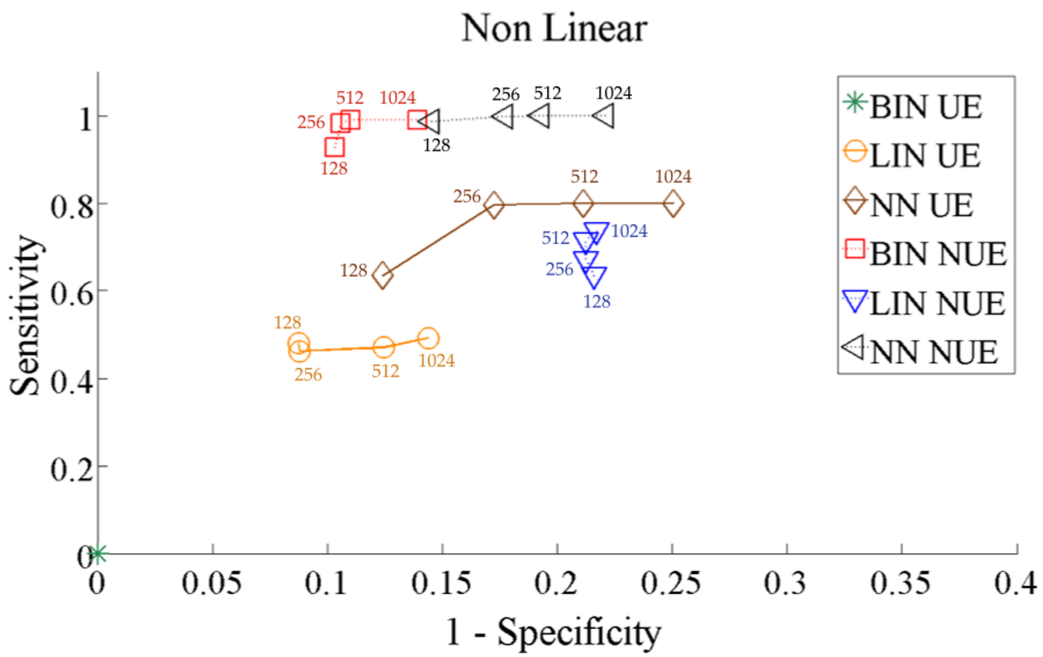

Figure 3.8: ROC curves for all methods for the system 3.14. We varied the series length from 128 up to 1024 points.

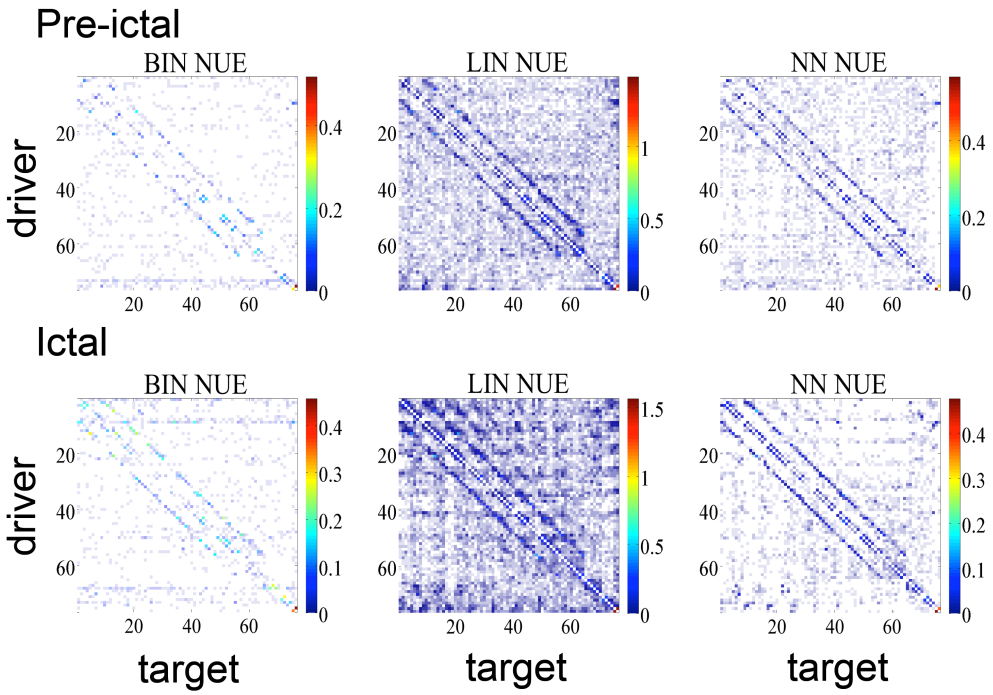

Figure 3.9: TE matrices for human EEG recordings. The pre-ictal (top) and ictal phases (bottom) were obtained with three different approaches to nonuniform embedding.

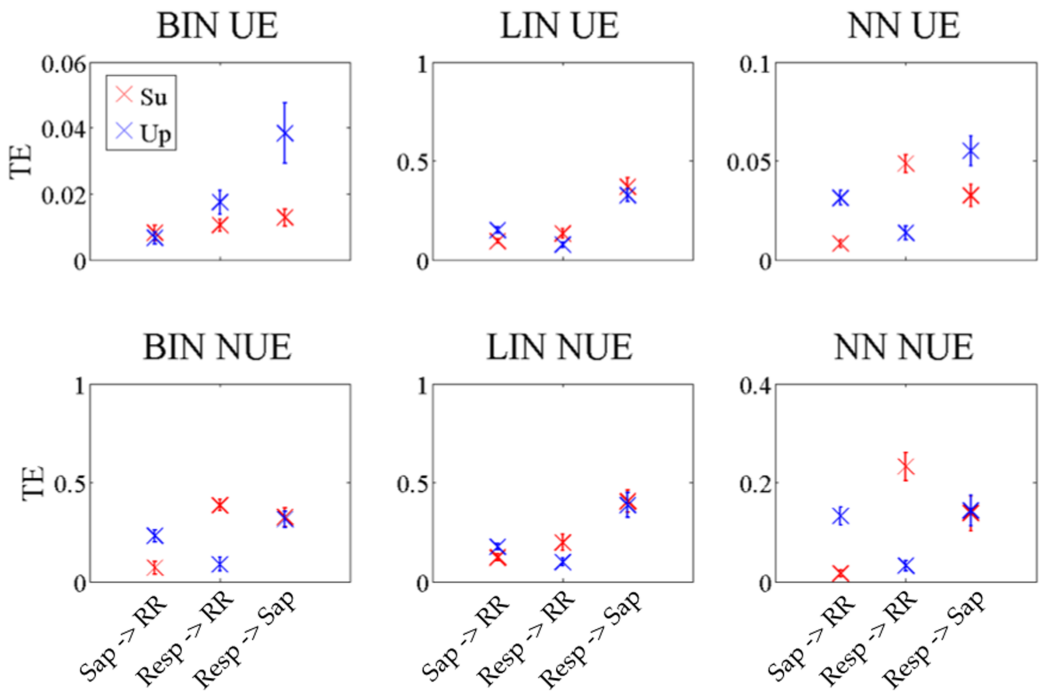

Figure 3.10: Transfer entropy for the links of interests in the cardiovascular example. In red the TE computed for the subjects in supine position, in blue the TE evaluated for the subjects in upright position. The error bars represent the standard error.

Table 3.1: How to set the input parameters: an example.

Name Parameter	**Description**
dataDir	folder containing data to be analysed
numProcessors	number of processors used for the parallel session
dataType	filename extension
resDirGenTS	folder in which results will be stored
dataFileName	data filename
channels	vector containing the series id, among the available series, chosen for the analysis
samplingRate	variable used to resample data
endPoint	value to cut the series length if necessary
autoPairwiseTarDriv	vector containing a 1 or a 0 for each chosen method, reflecting whether TE has to be computed among all the pairs or not. In this latter case, the desired drivers and targets will be specified by *idTargets* and *idDrivers*. By default the instantaneous effects of the drivers are not considered. This can be changed in *parametersAndMethods*, by setting *params_nameMethod.idDrivers = [tarDrivRows(2,:) ; tarDrivRows(2,:)]*.

Table 3.2: Example of the parameters required to define the methods for an experiment on 5 variables. In the second column the instantaneous effects are neglected both for targets and conditioning. In the third column we set instantaneous effects for some drivers and the respective targets. For example, when the target is 1, instantaneous effects are taken into account for driver 2 (first two rows, right column, parameter *idDrivers*) and conditioning variable 3 (first row, right column, parameter *idOtherLagZero*).

	Without Instantaneous Effects	With Instantaneous Effects
Name Parameter	**Parameter Value**	**Parameter Value**
genCondTermFun	generateConditionalTerm.m	generateCondTermLagZero.m
usePresent	0	1
idTargets	1 2 3 4 5	1 2 3 4 5
idDrivers	2 3 1 1 3 5 1 0 2 0 0 4 0 0 0	2 3 1 1 3 2 1 0 1 0 5 1 0 2 0 0 4 0 0 0 0 4 0 0 0
idOtherLagZero	0 0 0 0 0	3 0 5 3 1 0 0 2 0 0

CHAPTER 4

Neural Networks with Non-Uniform Embedding and Explicit Validation Phase to Assess Granger Causality

Paper published on Neural Networks; doi:10.1016/j.neunet.2015.08.003

A challenging problem when studying a dynamical system is to find the interdependencies among its individual components. Several algorithms have been proposed to detect directed dynamical influences between time series. Two of the most used approaches are a model-free one (transfer entropy) and a model-based one (Granger causality). Several pitfalls are related to the presence or absence of assumptions in modeling the relevant features of the data. We tried to overcome those pitfalls using a neural network approach in which a model is built without any a priori assumptions. In this sense this method can be seen as a bridge between model-free and model-based approaches. The experiments performed will show that the method presented in this chapter can detect the correct dynamical information flows occurring in a system of time series. Additionally we adopt a non-uniform embedding framework according to which only the past states that actually help the prediction are entered into the model, improving the prediction and avoiding the

risk of overfitting. This method also leads to a further improvement with respect to traditional Granger causality approaches when redundant variables (i.e. variables sharing the same information about the future of the system) are involved. Neural networks are also able to distinguish between dynamics which are drawn from the same distribution of the training set and dynamics completely different from the training set, as shown further in this chapter.

4

4.1 Introduction

A fundamental problem in the study of dynamical systems is how to find the interdependencies among their individual components, whose activity is recorded and stored in time series. Over the last few years, considerable effort has been dedicated to the development of algorithms for the inference of causal relationships among subsystems, a problem which is strictly related to the estimate of the information flow among subsystems [32, 33]. Two major approaches to accomplish this task are Granger causality (GC) [34, 35] and transfer entropy (TE) [4]. GC is based on regression, testing whether a source variable (driver) is helpful to improve the prediction of a destination variable (target) beyond the degree to which the target predicts its own future. GC is a model-based approach, implying that the corresponding statistics for validation can be derived from analytic models, resulting in a fast and accurate analysis. A pitfall, however, is inherent to model-based approaches: the model assumed to explain the data often implies strong assumptions and the method is not able to detect the correct directed dynamical networks when these assumptions are not met. On the other hand non-parametric approaches, such as transfer entropy, allow the pattern of influences to be obtained in the absence of any guidance or constraints from theory; the main disadvantages of non-parametric methods are the unavailability of analytic formulas to evaluate the significance of the transfer entropy and the computational burden, typically heavier than those required by model-based approaches.

Feed-forward neural networks, consisting of layers of interconnected *artificial neurons* [36], are among the most widely used statistical tools for non-parametric regression. Relying on neural networks, the proposed approach to Granger causality

will be both non-parametric and based on regression, thus realizing the Granger paradigm in a non-parametric fashion.

In this chapter we address the implementation of Granger's original definition of causality in the context of the artificial neural networks approach [37]. The metrics used to validate the hypothesis of directed influence is the *prediction error*: the difference between the network output and the expected target. The choice of the correct prediction error, and consequently the choice of the past states of the time series that will be fed to the model, has to be accompanied by a validation phase. Only under optimization of the *generalization error* one can be sure that the network is not overfitting.

In order to deal with an increasing number of inputs, each one representing a specific candidate source of directed influence, we will adopt a non-uniform embedding procedure [38] that is an iterative procedure to select only the most informative past states of the system to predict the future of the target series among a wider number of available past states. In line with this procedure the network will be trained with an increasing number of inputs, each of them representing a precise past state of the variables that are most helpful to predict the target. Also this selection process will be implemented using the notions of prediction error and generalization error, the former quantifying how well the training data are reproduced, the latter describing the goodness of the validation on a novel set of data.

It is worth stressing that a neural networks approach to GC has been already proposed in [39], where neural networks with a fixed number of inputs, together with other estimators of information flow, are used to evaluate GC. In [39] neural networks are trained without a validation set and an empirical method to avoid overfitting is adopted. To our knowledge the present approach is the first time that non-uniform embedding and a regularization strategy by a validation set are used together in the context of neural network approaches to detect dynamic causal links. Moreover, the neural networks built by our approach will accomplish not only the task of estimating information flows among variables, they may also be used for dynamic classification task as well, as better explained in Subsection 4.6.6. The new

method presented in this chapter has been integrated in MuTE MATLAB toolbox[1] [40] and it will be compared here with the linear Granger causality as well as with the Transfer Entropy, both implemented in the non-uniform embedding framework.

4.2 Introduction to neural networks

Artificial neural networks (ANN) are a very popular branch of machine learning. Here we give a brief introduction to neural networks to make this chapter self-consistent.

Neural networks can be represented as oriented graphs whose nodes are simple processing elements called "neurons" handling their local input, consisting of a weighted summation of the outputs from the parents nodes [36]. The input signal is processed by means of a function, called "activation function", and the corresponding outcome, called "output", is then sent to the linked nodes by a weighted connection; the weight is a real number that represents the degree of relevance of that connection inside the neural network. The most common architecture of a neural network consists of neurons ordered into layers. The first one is called "input layer" that receives the external inputs. The last layer is called "output layer" that gives the result of the computations made by the whole network. All the layers between the input and output layer are called "hidden layers".

Neural networks with at least one hidden layer and activation functions as the sigmoid function on the hidden nodes are able to adequately approximate all kinds of continuous functions defined on a compact set from a d-dimensional input space $\mathbb{R}^d$, the domain, to a c-dimensional output space $\mathbb{R}^c$, the codomain given a sufficient number of hidden nodes: in this sense one can say that neural networks can perform any mapping between two different vector spaces [37]. In order to allow a neural network to find the correct mapping, a so-called "learning phase" is needed. For our purposes, we use *supervised* learning, during which inputs are presented to the network and its output is compared to a known output. The weights are adjusted by the network that tries to minimize a cost function that depends upon the network output and the known output. This kind of learning allows a network to discover

[1] MuTE is a freeware toolbox. A detailed explanation is available on www.mutetoolbox.guru

hidden patterns inside the data.

We implement a growing neural network to study dynamical interactions in a system made up of several variables, described by time series, interacting with each other. The aim is not only to find a directional relationship of influence between a subset of time series, the *source*, and a *target* time series taking into account the rest of the series collected in a set, called *conditioning*, but also to determine the delay at which the source variables are influencing the target. We will then see how the neural networks approach can be useful to accomplish, under the same framework, several tasks such as: finding the directed dynamical influences among variables chosen at a certain delay; predicting a target series when the network is fed with a novel realization of a dynamical system whose connectivity structure has been previously learned; classifying a new data set, giving information about how close the causal relationships are to those observed in data sets used during the learning phase.

4.2.1 Mathematical framework

We deal with growing feed-forward neural networks to better infer the directed dynamical influences in a composite system. Each stochastic variable at hand is assumed to be zero mean (the mean of the data sample is subtracted from data), hence we will deal with neural networks without bias terms. A classical feed-forward neural network without bias is usually described by: a finite set of O nodes $S = 1,2,\ldots,O$ divided in d inputs, c output nodes and $O-(d+c)$ hidden nodes; a finite set of one way direction connections C each one connecting a node belonging to the k-th layer to a node belonging to the h-th layer, with $\mathsf{h} > \mathsf{k}$. A weight w_{hk} is associated with each connection from the node $k \in \mathsf{k}$ to the node $h \in \mathsf{h}$. Each node o is characterized by an input function s_o, an input value i_o, an activation function f_o, and an output value z_o [37]. Let us now define $\mathbf{w}_h$ as the weights vector of the connections which leave the nodes of the k-th layer and reach the node h. Let us define $\mathbf{z}$ as the output vector of a generic layer of nodes. The input i_h is given by

$i_h = s_h(\mathbf{w}_h, \mathbf{z})$. Usually we have: $i_h = \sum_k w_{hk} \cdot z_k$. Each z_h is given by

$$z_h = f_h \left(\sum_k w_{hk} \cdot z_k \right) \tag{4.1}$$

To evaluate the output of a multilayer network, consecutive applications of (4.1) are needed to activate all network nodes. Figure 4.1 depicts the schematic structure of the feed-forward networks under discussion here.

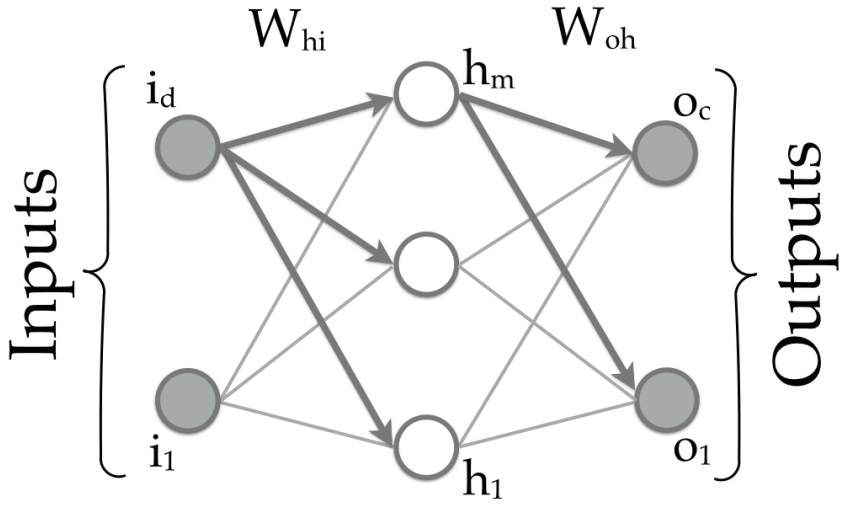

Figure 4.1: Schematic representation of a feed-forward neural network.

To summarize: the output values of a network can be expressed as deterministic functions of the inputs. Assuming that the network has only one hidden layer h, we can say that the whole network represents a function, linear or non-linear depending on the linearity or non-linearity of the f_h, between the d-dimensional input space and the c-dimensional output space, with parameters $\mathbf{w}$ given by the network weights. The relation holding between inputs and outputs of the network can be approximated in the multidimensional space spanned by the hidden nodes by either an hyperplane if linear functions are used as activation functions of the hidden nodes or by a smoother approximation when non-linear functions are set as activation functions of the hidden nodes. Usually, the activation functions of the output nodes are set to be the identity function that does not modify the input values of the output nodes because the outputs of the network are not supposed to be bounded in order to assume values as close as possible to the training target

values, assuming that overfitting is avoided.

So far we have shown how neural networks can process inputs and how they can be mapped onto a parametric function $\mathcal{F}(\mathbf{x};\mathbf{w}) : \mathbb{R}^d \to \mathbb{R}^c$.

We can now assume that there is a function $f : \mathbf{x} \in \mathbb{R}^d \to f(\mathbf{x}) \in \mathbb{R}^c$ to be modelled and we know a finite set of N couples $(\mathbf{x}^n, \mathbf{t}^n)$, where $n \in [1,\dots,N]$, $\mathbf{t}^n$ is the value of the function $f(\mathbf{x})$ evaluated in $\mathbf{x}^n$ plus an error $\varepsilon(\mathbf{x}^n)$. We want to approximate f using the parametric function $\mathcal{F} : \mathbf{w} \in \mathbb{R}^p, \mathbf{x} \in \mathbb{R}^d \to \mathcal{F}(\mathbf{x};\mathbf{w}) \in \mathbb{R}^c$. The function $\mathcal{F}$ can be found through the minimization of a certain error function $E(\mathbf{w})$. For instance a classical error function is the sum of squares function (4.2) to minimize by means of an iterative procedure that requires the data to be presented to the network several times through consecutive realizations called *epochs*:

$$E = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{d}(y_k^n - x_k^n)^2 \tag{4.2}$$

The training of the network is the process to determine, starting from a finite set of couples $(\mathbf{x}^n, \mathbf{t}^n)$, the weights $\tilde{\mathbf{w}}$ that can better shape $\mathcal{F}$ to be as close as possible to f. After each epoch of the training phase the weights in the network are adjusted. At this point a definition of *close* is in order. Let us suppose a noisy dataset consisting of $\mathbf{x}^n$ and $\mathbf{t}^n = f(\mathbf{x}^n) + \varepsilon(\mathbf{x}^n)$ where ε is the noise term. If we train the network until the input can be exactly reproduced then $\mathcal{F}$ is not only reproducing f, but the noise too. It is easy to understand that the more specialized the network the less it will be able to predict the right $\mathbf{t}^{n'} = f(\mathbf{x}^{n'}) + \varepsilon(\mathbf{x}^{n'})$ when a $\mathbf{x}^{n'}$ never seen before is presented to the network. In this case we say that the network is not able to *generalize*. To overcome this issue the *validation phase* is embedded in the learning. The validation phase is paramount because it allows the network to both model the function from which the data could have been drawn and to avoid modeling fluctuations produced by noise in the training set. In order to accomplish these two modeling tasks at the same time, the whole learning procedure is divided into two well distinguished steps:

1. the whole data set is divided in two groups. One group is used for the training step during which the weights are updated

2. the second group is used for the validation step. These data have not been

used in the previous step. We used a maximum amount of *training epochs* and a smaller number of epochs called *validation epochs*. The validation phase is embedded in the learning phase: this combination of training and validation avoids erroneous use of the training procedure, thus avoiding overfitting.

In the following section, we present the algorithm used for the learning phase:

1. training step: adjust the weights after a number of *training epochs*
2. validation step: evaluate the generalization error and store it in a vector *VEvect*
3. repeat steps 1. and 2. continuing to train the network until one of the following three stop conditions is verified:
   - the relative error evaluated as
     $$\frac{\|\text{current VEvect entry} - \textit{mean}(\text{previous VEvect entries})\|}{\text{current VEvect entry}}$$
     is less than a *validation threshold* set to 10^{-3}. Preliminary experiments were performed. According to them, setting the value of *previous VEvect entries* to 5 was a good compromise;
   - $$\frac{(\text{current VEvect entry} - \textit{mean}\,(\text{previous VEvect entries}))}{\text{current VEvect entry}} \geq 0$$
   - the maximum number of *training epochs* is reached.

The *previous VEvect* and *validation threshold* values have been chosen taking into account a cautious gradient descent implying small updating steps as the main concern here is the risk of overfitting.

4.3 Granger Causality with neural networks

The aim we are going to address is to find directed dynamical influences among variables, modeled as time series, using neural networks as a powerful tool to

compute the prediction errors needed to evaluate causality in the Granger sense. According to the original definition, Granger causality (GC) deals with two linear models of the present state of a target variable. The first model does not include information about the past states of a driver variable, while the second model contains such information. If the second model's error is less than that of the first model in predicting the present state of the target, then we can safely say that the driver is causing the target in the sense of Granger [3]. Here we introduce a new Granger causality measure called *Neural Networks Granger Causality (NNGC)* defined as

$$NNGC = err_{\text{reduced}} - err_{\text{full}} \tag{4.3}$$

where err_{reduced} is the prediction error obtained by the network that does not take into account the driver's past states, while err_{full} is the prediction error evaluated by the network that takes into account the driver's past states.

Therefore, instead of fitting predefined models, (linear ones in the original proposal by Granger) we train a neural network to estimate the target using only the past states that can better explain the target series, by using the non-uniform embedding technique. Such strategy leads to growing neural networks, with an increasing number of input neurons, each input neuron representing a past state chosen from the amount of past states available, considering all the variables in the system. The architecture of the network and choice of the most suitable past states, performed through the non-uniform embedding approach, are described in detail in the next sections. Relying on neural networks, this method realizes the Granger paradigm in a non-parametric fashion, like in [41, 42] where radial basis function networks were employed. This article improves such previous work by (i) using non-uniform embedding and (ii) employing training and validation phases concurrently to ensure a more robust detection of dynamical interactions.

4.4 Non-uniform embedding (NUE)

We first introduce in this section the NUE approach which is the basis of the algorithm used to build a neural network able to find the correct mapping between

the input and the output spaces in an optimal way. The uniform embedding (UE) approach relies on a predefined set of candidates making a strong assumption about which past states are better able to explain the future of the target series. This approach, lacking a specific criterion according to which the candidates are chosen, is likely to cause problems such as overfitting and detection of false influences [43, 26]. NUE framework, instead, is an iterative procedure aimed at detecting only the time series' past states that can effectively help to predict the target series. To evaluate whether a new candidate should be chosen, an hypothesis and, eventually, a significance test, should be satisfied. In this way of exploring causality, once this hypothesis and significance test (when needed) are no longer satisfied, the procedure is unable to find additional candidates to help predict the target.

Let us consider a composite system described by a set of M interacting dynamical (sub) systems and suppose that, within the composite system, we are interested in evaluating the information flow from the source system $\mathcal{X}$ to the destination system $\mathcal{Y}$, collecting the remaining systems in the vector $\mathcal{Z} = \left\{\mathcal{Z}^k\right\}_{k=1,\ldots,M-2}$. We develop our framework under the assumption of stationarity, which allows to perform estimations replacing ensemble averages with time averages (for non-stationary formulations see, e.g., [12] and references therein). Accordingly, we denote X, Y and $\mathbf{Z}$ as the stationary stochastic processes describing the state visited by the systems $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$ over time, and X_n, Y_n and $\mathbf{Z}_n$ as the stochastic variables obtained by sampling the processes at the present time n. Moreover, we denote $X_n^- = [X_{n-1}X_{n-2}\ldots]$, $Y_n^- = [Y_{n-1}Y_{n-2}\ldots]$, and $\mathbf{Z}_n^- = [\mathbf{Z}_{n-1}\mathbf{Z}_{n-2}\ldots]$ as the infinite-dimensional vector variables representing the whole past of the processes X, Y and $\mathbf{Z}$. In some cases, taking the instantaneous influences of the candidate drivers into account as well may also be desirable. In such cases, the vectors X_n^- and $\mathbf{Z}_n^-$ defined above should also contain the present terms X_n and $\mathbf{Z}_n$.

We will discuss here the crucial issue of how to approximate the infinite-dimensional variables representing the past of the processes. This problem can be seen in terms of performing suitable conditioned embedding of the considered set of time series [14].

The main idea is to reconstruct the past of the whole system represented by the processes $X, Y, \mathbf{Z}$ with reference to the present of the destination process Y, in order to obtain a vector $V = [V_n^Y, V_n^X, V_n^{\mathbf{Z}}]$ containing the most significant past variables

to explain the present of the destination.

Non-uniform embedding constitutes the methodological advance with respect to the state of the art that we propose as a convenient alternative to UE. This approach is based on the progressive selection, from a set of candidate variables including the past of X, Y, and $\mathbf{Z}$ considered up to a maximum lag (*candidate set*), of the lagged variables which are more informative for the target variable Y_n. At each step, selection is performed maximizing the amount of information that can be explained about Y by observing the variables considered with their specific lag up to the current step. This results in a criterion for maximum relevance and minimum redundancy for candidate selection, so that the resulting embedding vector $V = [V_n^X V_n^Y V_n^Z]$ includes only the components of X_n^-, Y_n^- and $\mathbf{Z}_n^-$, which contribute most to the description of Y_n. Starting from the full candidate set, the procedure which prunes the less informative terms is described below:

1. Get the matrix with all the candidate terms
   $\mathrm{MC} = [X_{n-1} \dots X_{n-l_X} Y_{n-1} \dots Y_{n-l_Y} \mathbf{Z}_{n-1} \dots \mathbf{Z}_{n-l_Z}]$, with l_X, l_Y, l_Z representing the maximum lag considered for the past variables of the observed processes; these matrices will contain also the terms X_n and $\mathbf{Z}_n$ in case one wants to take into account instantaneous effects. The values of l_X, l_Y, l_Z can be set by the experimenter according to a known feature of the data, or set to a reasonably large value for exploratory purposes. If values of l_X, l_Y and l_Z are set too low, an incorrect estimation of Granger causality may result, but higher values should not issues with non-uniform embedding.

2. Run the procedure to select the most informative past variables and the optimal embedding vector:

   (a) Initialize an empty embedding vector $V_n^{(0)}$

   (b) Perform a while loop on k, where k can assume values from 1 to the number of initial available candidates, *numC*, in the MC matrix. At the $k-$th iteration, after having chosen $k-1$ candidates collected in the vector $V_n^{(k-1)}$:
   for $1 \leq i \leq$ number of current candidate terms

- add the $i-$th term of MC , $W_n^{(i)}$, to a copy of $V_n^{(k-1)}$ to form the temporary storage variable $V_n' = [W_n^{(i)} V_n^{(k-1)}]$

- compute the information exchanged between Y_n and V_n'

(c) Among the tested $W_n^{(i)}$, select the term $\hat{W}_n$ which maximizes the information exchanged

(d) **if** $\hat{W}_n$ satisfies a *termination criterion*, delete it from MC and set $k = k+1$.

(e) **else** end the procedure setting $k = numC + 1$ and returning $V = V_n^{(k-1)}$

3. Use Y_n and the full embedding vector $V = [V_n^X V_n^Y V_n^Z]$ and to evaluate err_{full}. err_{reduced} is obtained excluding from err_{full} the candidates belonging to the variables considered as drivers. Both errors are evaluated as the root mean squared error (RMSE) between the neural network output and Y_n. If the error resulting from the network that contains the inputs representing the driver's past states (err_{full}) is lower than the error resulting from the network that does not take into account the driver's past states (err_{reduced}), then the driver assessed is determined to help predict the target more than the network that excludes the driver.

The complexity of the algorithm concerns mainly step 2, in particular step 2(b), involving a *for* loop nested inside a *while* loop: in the worst case the body of the *for* loop is executed $numC^2$ times resulting in a complexity $\mathcal{O}(numC^2)$.

Summarizing, the non-uniform embedding is a feature selection technique selecting, among the available variables describing the past of the observed processes, the most significant - in the sense of predictive information - for the target variable. Moreover, given the fact that the variables are included into the embedding vector only if associated with a significant contribution to the description of the target, the significance of the NNGC estimated with the NUE approach results simply from the selection of at least one lagged component of the source process. In other words, if

at least one component from X is selected by NUE, the estimated NNGC is strictly positive and can be assumed to be significant. If not, the estimated NNGC is exactly zero and is assumed to be non-significant. This latter also occurs when the first candidate ($k = 1$) does not reach the desired level of significance, meaning that none of the candidates provides significant information about the target variable. This may also be encountered, for instance, when the target process consists of white noise: the code will return an empty embedding vector and assign a zero value to the NNGC.

4.5 Non-uniform embedding using neural networks (NeuNet NUE)

Here we want to investigate the opportunity to use neural networks to create the two models needed to evaluate NNGC and, at the same time, to better choose the right candidates to be considered as terms of the models. In this sense our method is a model-free approach because we do not assume any model a priori that can explain the data, but we allow the network to explore the parameters space in order to find the model we need. The procedure will be able to model a function from the input space, spanned by the time series' past states, and the output space, spanned by the present state of the target series: $Y_n = f(V)$. It will be possible to estimate a function $\mathcal{F}$ as close as possible to f. This will ensure a precise prediction of Y from Y itself, X and $\mathbf{Z}$. It is easy to see that from $\mathcal{F}$ it is possible to assess whether for another data set $Y', X', \mathbf{Z}'$, the same relation $\mathcal{F}$ holds: in this case the network will be able to *generalize*.

In this study a three-layers feed-forward neural network is used [37], trained by means of the resilient back propagation technique that is one of the fastest learning algorithms [44]. Briefly, the resilient back propagation is an optimized algorithm to update the weights of a neural network based on the gradient descent technique. Let Δ_{ij} be the weight update value that only determines the size of the weight update

and E the error function. Then the resilient back propagation rule is the following:

$$\Delta_{ij} = \begin{cases} \eta^+ \times \Delta_{ij}^{(t-1)} & \text{if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} \times \frac{\partial E}{\partial w_{ij}}^{(t-1)} > 0 \\ \eta^- \times \Delta_{ij}^{(t-1)} & \text{if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} \times \frac{\partial E}{\partial w_{ij}}^{(t-1)} < 0 \\ \Delta_{ij}^{(t-1)} & \text{else} \end{cases} \tag{4.4}$$

where $0 < \eta^- < 1 < \eta^+$. To summarize: every time the partial derivative of the current weight w_{ij} changes in sign, i.e. the error function slope changes indicating that a local minimum has been avoided, the updated value Δ_{ij} is decreased by the factor η^- allowing a reversal, or "coming back", in the parameters space towards the local minimum. If the derivative does not change sign, then the updated value Δ_{ij} is increased by the factor η^+ accelerating towards a local minimum.

Once the updated value is evaluated, the weight update is quite straightforward as shown by the following equations:

$$\Delta_{w_{ij}}^{(t)} = \begin{cases} -\Delta_{ij}^{(t)} & \text{if } \frac{\partial E}{\partial w_{ij}}^{(t)} > 0 \\ +\Delta_{ij}^{(t)} & \text{if } \frac{\partial E}{\partial w_{ij}}^{(t)} < 0 \\ 0 & \text{else} \end{cases} \tag{4.5}$$

so that $w_{ij}^{(t+1)} = w_{ij}^{(t)} + \Delta_{w_{ij}}^{(t)}$. However, we should also take into account the case when the partial derivative changes sign: the previous weight update is then reverted as follows:

$$\Delta_{w_{ij}}^{(t)} = -\Delta_{w_{ij}}^{(t-1)} \qquad \text{if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} \times \frac{\partial E}{\partial w_{ij}}^{(t)} < 0. \tag{4.6}$$

Following the NUE scheme, each input corresponds to a candidate, while the minimization criterion is the prediction error between the network output and Y_n. We should keep in mind that the core of the entire procedure lies in the choice of the candidates that can actually help to predict the target series. Once the relative prediction error, defined as $(\text{prediction error}_{k-1} - \text{prediction error}_k)/(\text{prediction error}_1 - \text{prediction error}_k)$ where k can assume values from 1 to the number of initial available candidates, is greater than or equal to a threshold, the procedure stops and no further candidates are chosen. To summarize: the hypothesis of Granger causality evaluates how much information is introduced by adding a new input with respect

to the information carried only by the inputs previously considered. Moreover, it is worth stressing that in this case we do not rely on the comparison with a null distribution in order to choose whether a given candidate must be chosen or not. On the other hand, when a driver-response relationship among variables holds, the algorithm will find, input by input, the candidate that will give the lowest prediction error, this being a condition that can hold only if we ensure the network is not overfitted. The risk of overfitting is the reason why a validation phase, described in detail in the following sections, was implemented and the idea of a fixed amount of training iterations was discarded. As soon as the error on the validation set, called *generalization error* increases, the training of the network stops ensuring the capability of the network to generalize, implying that it has not been overfitted.

To better explain the steps implemented to select the past states as a pseudo-code we can say that a *for* loop is nested within a *while* loop. The *while* loop condition, that takes into account the decrease of the prediction error during the training phase, determines whether or not an additional input should be added to the network. It is worth stressing that during the whole procedure of the candidates' selection, the internal architecture of the network is kept fixed: the number of hidden nodes is set up as a fraction of the maximum number of candidates available, as shown in subsection 4.6.1, and it does not change. The *for* loop, instead, tests each available candidate given the previous inputs already chosen. During this test, at each iteration of the *for* loop, a network is trained taking into account the current candidate and the validation phase takes place according to the procedure explained in Subsection 4.2.1 point 3. Therefore, the validation error is taken into account in order to allow the network itself to reach its best performance, in terms of the generalization task, according to the current candidate. At the end of the *for* loop the candidate which gives the minimum prediction error is selected. If the prediction error satisfies the *while* loop condition, such that the relative prediction error is smaller than a threshold, the candidate is chosen and deleted from the set of the available candidates so that the procedure can continue. Otherwise, the procedure will stop. The pseudo-code of the algorithm is shown in the following:

1: Initialize network parameters;
2: Initialize the embedded matrix $EM = \emptyset$
3: Initialize the prediction error PE vector $= \emptyset$

Initialize the current prediction error *CPE* vector $= \emptyset$
Initialize final candidate matrix FCM $= \emptyset$
Initialize CS $= [X_{n-1} \dots X_{n-l_X}, Y_{n-1} \dots Y_{n-l_Y}, \mathbf{Z}_{n-1} \dots \mathbf{Z}_{n-l_Z}]$. The terms X_n and $\mathbf{Z}_n$ should be considered too in case the instantaneous effects should be taken into account.
k = 1
while CS $\neq \emptyset$ **do**
    if CS is full **then**
        Initialize the network NN with one input, the number of chosen hidden nodes, one output
    else
        Add to NN_{k-1} another input;
        Initialize only the weights between the new input and the hidden nodes keeping all the rest fixed;
    end if
    for $i \in [1, \dots, \text{length(CS)}]$ **do**
        while epoch $\leq$ maxTrainEpochs **do**
        % Learning phase:
            train the network, after 30 training epochs evaluate the prediction error;
            validate the network evaluating the generalized error;
            if remainder after division of epochs by valEpochs == 0 **and** epochs $\leq$ maxTrainEpochs **then**
                evaluate the relative validation error
                if $\|$relative validation error$\|$ $\leq$ validationThreshold **or** relative error ≥ 0 **or** epochs == maxTrainEpochs **then**
                    Store the prediction error in CPE(i)
                    epoch = maxTrainEpochs + 1
                end if
                Store the prediction error in CPE(i);
                epoch = epoch + 1
            end if
        end while
    end for
    NN_k = neural network having in input the candidates that give the minimum prediction error stored in CPE
    PE_k = min(CPE)
    if relative prediction error $\leq$ trainThreshold **then**
        NN = NN_{k-1}
        PE = PE_{k-1}
        CS $= \emptyset$

36: **else**
37: NN = NN_k
38: add to FCM the candidate of CS that returns the minimum prediction error
39: delete from CS the candidate that returns the minimum prediction error
40: k = k + 1
41: **end if**
42: **end while**
43: return NN; FCM; PE

where the strings after the % symbol should be considered as non executable code.

In the following we will explain in more detail the algorithm showed above:

1. In the initialization phase it is worth noting that l_X, l_Y, l_Z represent the maximum lag considered for the past variables of the observed processes. In the following experiments we will set l_X, l_Y, l_Z to take into account more past states than needed.

2. at the k-th step of the while loop at line 8, where k runs on the number of inputs chosen, the network tests all the candidates available by means of the for loop at line 15: there are k inputs. The first k-1 inputs are the ones chosen so far and on the k-th input one candidate per time is considered. The initial conditions are the same for each candidate: the weights have been fixed so the ones departing from the k-1 inputs are the same found as the result of the training at the (k-1)-th step and the weights departing from the k-th input are the same at the beginning of each training session when the RMSE between the network output and Y_n is evaluated. Lines 19-27 take care of whether to stop the training phase for the current candidate according to the generalization error.

3. lines 32-41 check whether it would be worth adding candidates, or it is better to stop the whole procedure because no further meaningful information can be added to better predict the target. The generalization error is not relevant at this stage, since it is only used to stop the training phase.

The network is finally trained to reproduce the best correspondence between the space spanned by the terms of FCM and the space spanned by Y_n. The network is then the model that can be used to explain Y_n, including the driver's candidates.

This model will give err_{full}. To evaluate err_{reduced} the candidates belonging to the source system should be removed from the network, so the corresponding inputs and weights should not be considered. This configuration leaves the weights between the hidden nodes and the output unchanged, so the network now won't be able to approximate Y_n as well as during the evaluation of the previous term if the causal hypothesis holds between the driver and the target. err_{reduced} can be computed projecting the information carried by the inputs representing the candidates belonging to the target and the conditioning variables on the output space and evaluating the RMSE between the network output and Y_n. NNGC is now immediately evaluated by the difference between the two terms. The significance of the causality measure estimated with the neural network method embedded into the NUE approach results simply from the selection of, at least, one lagged component of the driver. In other words, if at least one component from the driver is selected, the Granger causality is strictly positive and can be assumed as significant. If this is not the case, the estimated causality that results is exactly zero and is assumed to be non-significant.

NUE is used here as a feature selection algorithm. Other feature selection algorithms can be used to select the most informative candidates; our choice is in line with other approaches to detect dynamical interactions present in literature, thus offering a coherent framework for all the estimators.

4.6 Applications to simulated data

Before applying the proposed method, the correct initialization parameters were set (see Subsection 4.6.1). First of all, we wanted to prove that neural networks implemented with the non-uniform embedding framework perform better than neural networks implemented with the uniform embedding framework, Subsection 4.6.2. Then, several datasets were simulated to test NeuNetNUE in different situations in order to explore its capability to detect the correct directed dynamical links, see Subsections 4.6.3 - 4.6.4. During those three experiments we compared the neural networks with a model-based approach and two model-free approaches, as described in Subsection 4.6.3, to get a better idea of the performances obtained

by our method. Furthermore, we wanted to check whether NeuNetNUE was both robust with respect to redundant information (see Subsection 4.6.5) and able to outperform an approach based on multivariate Granger causality analysis [45]. Finally we wanted to evaluate the capability of the networks to predict and classify time series (see Subsection 4.6.6).

4.6.1 Choice of the parameters

One of the crucial aspects of neural networks approaches concerns the choice of the optimal parameters. W are interested not only in the parameters involving the architecture and the training of the network, but also in the parameters that are responsible for the number of past states that can be chosen, allowing the approach to be more or less conservative. The parameters can be listed as follows:

- the threshold according to which a certain number of past states are chosen (th). This parameter is taken into account to stop the training of the network and it consequently regulates the amount of past states chosen by the network: for a lower *th* more past states are selected, see Section 4.5 pointed list 3
- the validation threshold, useful to not overfit the network (valTh). This parameter plays an important role in the validation phase, allowing the network not to be overfitted, see Section 4.5 pointed list 2
- the number of hidden nodes (hidNodes), reported as percentage of the total amount of the available past states
- the learning rates for the resilient back propagation, η^+, η^-.

The parameters mentioned above must be set so that the neural network approach is able to detect the expected information flow. The investigation of the best parameters values was performed on linear and non-linear models with memory up to 3 points in the past.

We considered 20 simulations of the systems for each combination of the parameter values shown in table 4.1. We set $l_X = l_Y = l_Z = 5$. Values of η^- and η^+, the parameters of the resilient back propagation, ranged as $\eta^- \in [0.4, \ldots, 0.9]$

with step of 0.1 and $\eta^+ \in [1.1, \ldots, 1.4]$ with step of 0.1. The threshold's values for the prediction error range between 2^{-8} and 0.3. According to this assumption, it is then possible to consider whether the current candidate is significant or not. Small values of the threshold, such as 2^{-8}, represent a weakly conservative network. On the other hand, high values of the threshold, such as 0.3, represent a strongly conservative network. We first investigated how the network performed for higher values of the threshold and we found that the networks were too conservative and, consequently, NueNet NUE only found one candidate belonging to the target series only. The same reasoning holds for the validation threshold that gives the range of values within which the validation error can fall. The assumption on how wide the range is, determines whether the network can be considered to have undergone enough training. Finally the number of hidden nodes ranges between 0.1 and 2.5, with step of 0.2, times the number of available past states. In this way, we allow the network to have a number of hidden nodes so as to allow it to reach the best performance with increasing number of inputs. It turned out that *number of hidden nodes* = 1.3× *number of available candidates* was the best compromise.

For each combination of the parameters we evaluated how many times the method was able to detect the right information flows, estimating the number of true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). We then evaluated sensitivity = TP /(TP+FN), specificity = TN / (TN+FP) and $\mathrm{F1_{score}} = 2\,\mathrm{TP}/(2\,\mathrm{TP} + \mathrm{FP} + \mathrm{FN})$ and we checked how the performances changed as the parameters varied. We found that neural networks obtained high performances on both systems corresponding to different parameters values: parameters values obtained on the linear system allowed NeuNet NUE to be less conservative with regard to neural networks used with parameters values found on the non-linear system. We finally chose the parameters in correspondence to which the network would be considered less conservative: th = 8^{-3}; valTh = 0.6; hidNodes = 0.3; $\eta^- = 0.9$; $\eta^+ = 1.1$. Figure 4.2 shows the performances of the network for different values of th, keeping all the other parameters fixed. For a better visualization, only five points out of the nine showed in the table were plotted. The other four points have been omitted, being too close to the others in the figure: the resulting curve is virtually unchanged. We can notice that for the minimum value of th, 2^{-8}, the network takes into account more past states than needed retrieving more FP and

less TP than for higher values of th. Furthermore, the specificity is lower, almost zero, than the sensitivity. For th = 8^{-3} the network's performances give sensitivity = specificity = 1, while for the maximum value of th, 0.3, the network is not allowed to choose a lot of past states and, consequently, there are less TP and more TN than for lower values of th, resulting in the sensitivity lower than the specificity.

Name Parameter	**Parameter values**
th	$2\cdot10^{-8}$ $2\cdot10^{-6}$ $2\cdot10^{-4}$ $2\cdot10^{-3}$ $5\cdot10^{-3}$ $8\cdot10^{-3}$ $1\cdot10^{-2}$ 0.15 0.30
valTh	0.2 0.4 0.6 0.8 1
hidNodes	0.1 0.3 0.5 0.7 0.9 1.1 1.3 1.5 1.7 1.9 2.1 2.3 2.5
η^-	0.4 0.5 0.6 0.7 0.8 0.9
η^+	1.1 1.2 1.3 1.4

Table 4.1: Parameters values to initialize the network.

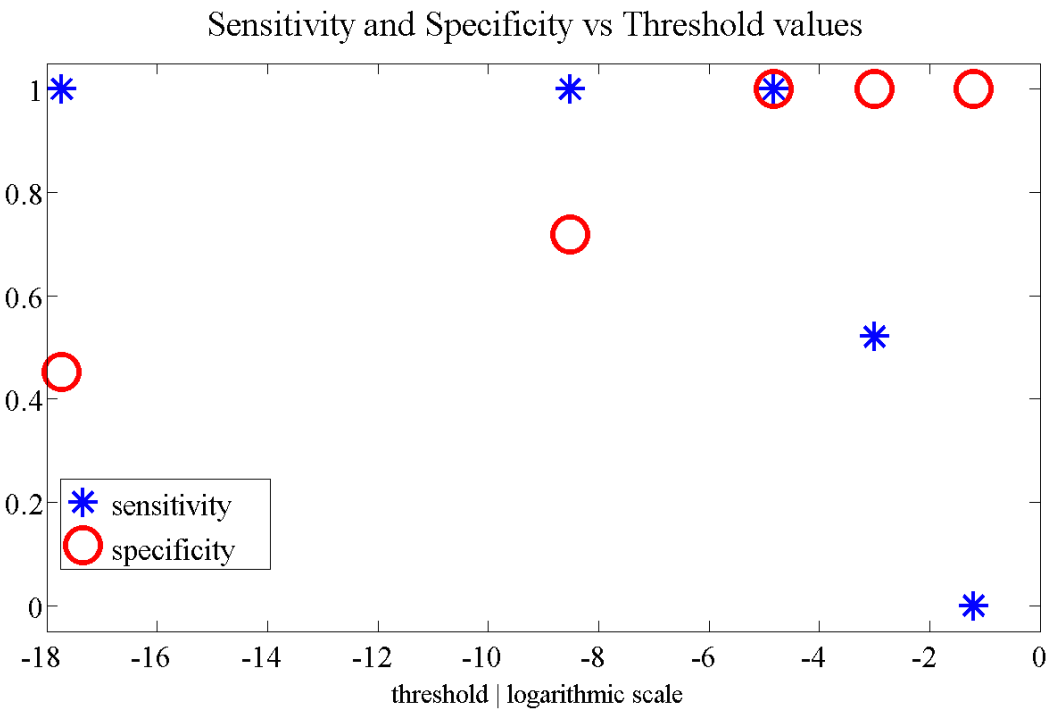

Figure 4.2: Sensitivity and specificity at varying of the threshold values.

4.6.2 Reasoning behind our discarding of the uniform embedding approach

Before explaining the experiments that we performed to test the proposed approach, we would like to underline that the non-uniform embedding framework was chosen because of its theoretical advantages with respect to the uniform embedding. Furthermore, we wanted to check whether those advantages held in the case of the GC neural networks estimator too. Neural networks estimator used with the uniform embedding approach (NeuNet UE) only need one network to be trained when err_{full} is evaluated. Each input of the network represents a past state, therefore the number of inputs equals the number of available past states. The other network parameters have the same values as in the case of neural networks estimator used with the non-uniform embedding approach (NeuNet NUE). The validation phase is still required. Once err_{reduced} is evaluated by only removing the inputs corresponding to the past states that belong to the driver whose influence to a specific target is tested, we can obtain a value of NNGC whose significance still has to be evaluated. This step is addressed using the surrogates technique as implemented in the case of other estimators also used into the uniform embedding framework [40]. This means that for each surrogate another network with the same architecture has to be trained resulting in a dramatic increase of the computational complexity.

We compared NeuNet NUE and NeuNet UE performing a multivariate analysis on 100 realizations of a system composed of five coupled Hénon maps with a length of 2500 time points, built according to the following equations:

$$\begin{aligned}
X_{1,n} &= aV(1) - (0.5c(X_{4,t-1} + X_{5,t-1}) + \\
&\quad (1-c)X_{1,t-1})^2 + aV(2)X_{1,t-2} + w_{1,n} \\
X_{2,t} &= aV(1) - (0.5c(X_{3,t-1} + X_{5,t-1}) + \\
&\quad (1-c)X_{1,t-1})^2 + aV(2)X_{1,t-2} + w_{2,n} \\
X_{3,t} &= aV(1) - X_{3,t-1}^2 + aV(2)X_{3,t-2} + w_{3,n} \\
X_{4,n} &= aV(1) - X_{4,t-1}^2 + aV(2)X_{4,t-2} - 0.02cX_{3,t-2} + w_{4,n} \\
X_{5,t} &= aV(1) - (0.5c(X_{1,t-1} + X_{2,t-1}) + \\
&\quad (1-c)X_{5,t-1})^2 + aV(2)X_{5,t-2} + w_{5,n}
\end{aligned} \tag{4.7}$$

where aV is the characteristic parameter tuned for chaotic behavior, the coupling strength $c = 0.4$ and w is drawn from Gaussian noise with zero mean and unit variance. In figure 4.3 the modeled links are shown.

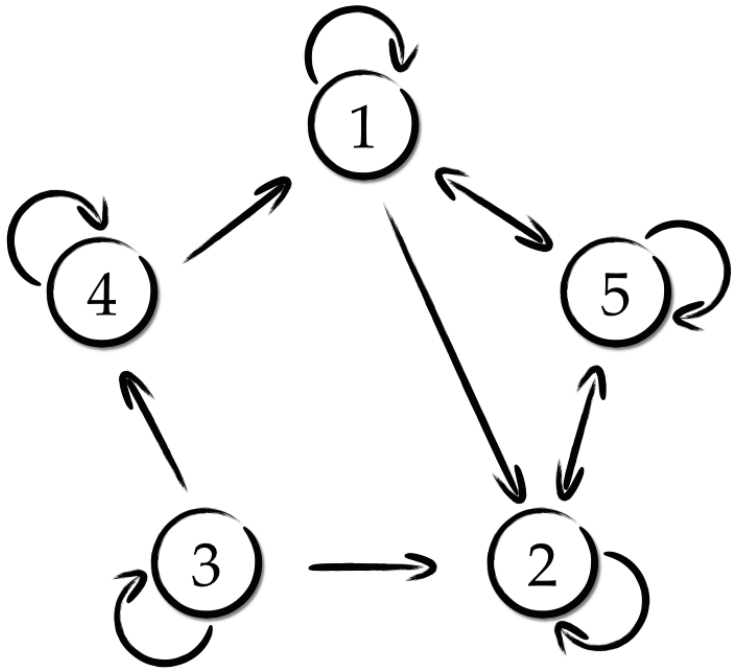

Figure 4.3: Simulated system. Interactions between the variables of the simulated Hénon maps system generated according to equations (4.7).

We performed the analysis setting $l_X = l_Y = l_Z = 5$ and using the rest of the parameter values found in Subsection 4.6.1. Looking at sensitivity, specificity and $\mathrm{F1}_{\mathrm{score}}$, table 4.2, we can clearly notice that NeuNet NUE performs better than NeuNet UE. Considering this result, the heavy computational complexity and the lack of information about the only past states that can give information to the target concerning the use of NeuNet UE, led us to only take into account NeuNet NUE for further investigations.

4.6.3 Simulated data: Hénon maps

In the first experiment we generated 6 Hénon maps rearranging the system (4.7) as shown in figure 4.4 setting the coupling strength $c = 0.2$. The equations are shown

	Sens	Spec	$F1_{score}$
NeuNet NUE	0.86	0.80	0.80
NeuNet UE	0.69	0.93	0.77

Table 4.2: Sensitivity, specificity and $F1_{score}$ values obtained on the system (4.7) by NeuNet NUE and NeuNet UE.

in the following

$$\begin{aligned} X_{1,n} &= aV(1) - X_{1,n-1}^2 + aV(2)X_{1,n-2} + w_{1,n} \\ X_{m,n} &= aV(1) - (0.5c_m(X_{m-1,n-1} + X_{m+1,n-1}) + \\ &\quad + (1-c_m)X_{m,n-1})^2 + aV(2)X_{m,n-2} + w_{m,n} \\ X_{6,n} &= aV(1) - X_{6,n-1}^2 + aV(2)X_{6,n-2} + w_{6,n} \end{aligned} \tag{4.8}$$

where aV is the vector of parameters for chaos, w is drawn from Gaussian noise with zero mean and unit variance and $m \in [2,5]$ is the identifier of the series.

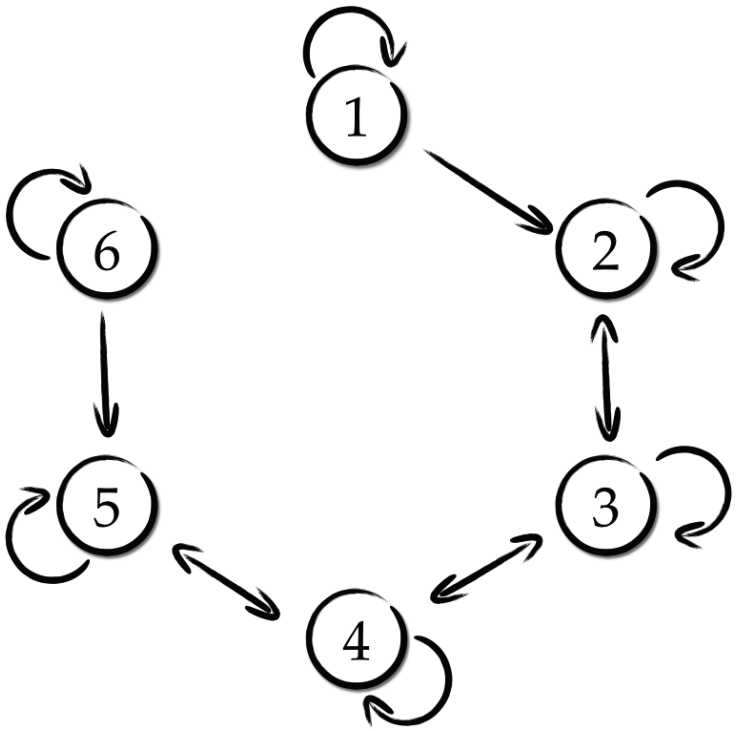

Figure 4.4: Simulated system. Interactions between the variables of the simulated Hénon maps system generated according to equations (4.8).

We generated 100 realizations of the Hénon maps, performed a multivariate analysis keeping the parameters found in section 4.6.1 fixed and setting $l_X = l_Y = l_Z = 5$. We then evaluated the mean values of the NNGC for all the pairwise combinations driver-target as shown in figure 4.5. We compared our method's performance with the binning, linear and nearest neighbor estimators implemented in the non-uniform embedding framework (henceforth BIN NUE, LIN NUE and NN NUE). These three estimators are already implemented in MuTE [40]. The comparison with NeuNet NUE has been performed in terms of sensitivity, specificity and $\mathrm{F1_{score}}$, as shown in table 4.3. We can notice that NeuNet NUE is the second best method after NN NUE.

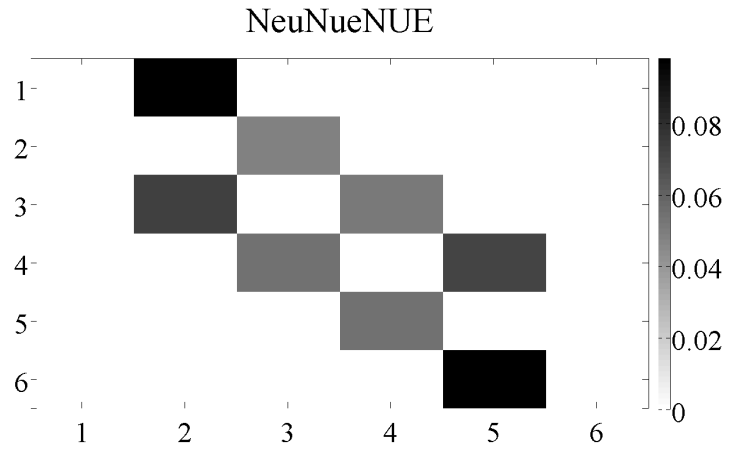

Figure 4.5: GC matrix representation for the NueNet NUE estimator applied to the system (4.8). The color indicates the magnitude of the GC averaged over 100 realizations of the simulation. The targets are plotted on the x-axis while the drivers are plotted on the y-axis.

	Sens	Spec	$\mathrm{F1_{score}}$
BIN NUE	1	0.86	0.84
LIN NUE	0.99	0.74	0.73
NeuNet NUE	1	0.98	0.98
NN NUE	1	1	1

Table 4.3: Sensitivity, specificity and $\mathrm{F1_{score}}$ values obtained on the system (4.8) by the four estimators.

Furthermore, we wanted to investigate whether NeuNet NUE was robust enough

with respect to the coupling strength involved in (4.8) using only 5 time series. Again we performed a comparison with the estimators implemented in MuTE. In figures 4.6 - 4.9 we can see the performances of the four methods, noticing that NeuNet NUE is the only approach able to detect the expected information flows even when the coupling is 0.8. NN NUE detects two false positive information flows for the directions $4 \rightarrow 2$, $2 \rightarrow 4$ from coupling strength value equal to 0.6 on. BIN NUE and LIN NUE obtain the worst performances detecting false positive information flows even for coupling strength equal to 0.3, see figure 4.6 direction $2 \rightarrow 4$. Note the differing number of outliers in figure 4.6 versus figure 4.9, even if the detection criterion is fixed: on each box, the central mark is the median, the edges of the box are the 25th and 75th percentiles, the whiskers extend to the most extreme data points not considered outliers. The four methods show different fluctuations of the exchanged information values as remarked by the different number of outliers.

In figures 4.10, 4.11 we compare the performances of the four methods showing their ROC curves and $\mathrm{F1_{score}}$, respectively. NN NUE and NeuNet NUE ROC curves report the highest sensitivity and specificity as soon as the coupling is greater than zero. For high couplings the ROC curves denote a higher specificity of NeuNet NUE. BIN NUE starts with low sensitivity and specificity, and its specificity generally increases as the coupling increases. $\mathrm{F1_{score}}$ curves belonging to NeuNet NUE and NN NUE are very close. For couplings greater than 0.5 NeuNet NUE $\mathrm{F1_{score}}$ is higher than NN NUE $\mathrm{F1_{score}}$, but both lower than BIN NUE $\mathrm{F1_{score}}$ denoting once again how NeuNet NUE can be much closer to model-free than to model-based approaches. Only at coupling = 0.6 NeuNet NUE has the highest $\mathrm{F1_{score}}$.

Referring to the equations (4.7), we obtained the results shown in table 4.4 in terms of sensitivity, specificity and $\mathrm{F1_{score}}$. This time NeuNet NUE detects causal influences better than BIN NUE, even if it is still not able to perform better than NN NUE.

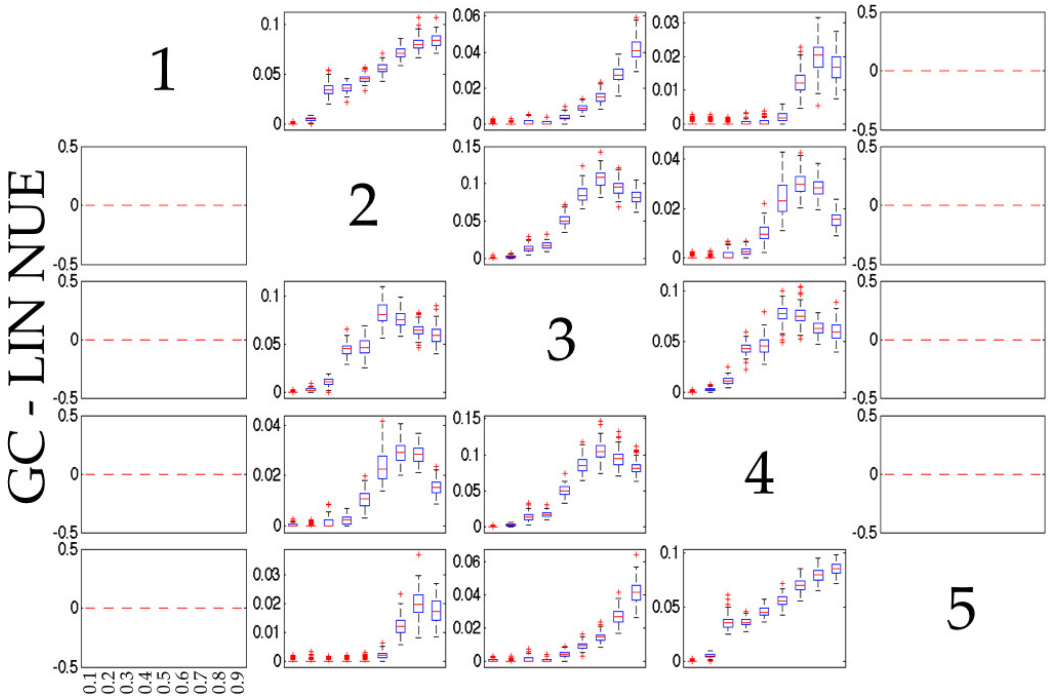

Figure 4.6: LIN NUE performances on Hénon maps at varying of the coupling strength. GC values are plotted on the y-axis, while the coupling strength values are plotted on the x-axis.

4.6.4 Simulated data: Lorenz system

In the third experiment we studied a system composed of five identical Lorenz subsystems defined by the following equations:

$$\begin{aligned} \dot{x}_1 &= -10x_1 + 10x_1, & \dot{x}_i &= -10x_i + 10x_i + C(x_{i-1} - x_i), \\ \dot{y}_1 &= -x_1 z_1 + 28x_1 - y_1, & \dot{y}_i &= -x_i z_i + 28x_i - y_i, \\ \dot{z}_1 &= x_1 y_1 - 8/3 z_1, & \dot{z}_i &= x_i y_i - 8/3 z_i, \end{aligned} \tag{4.9}$$

where $i \in [2,5]$. The differential equations are solved by means of the Runge-Kutta method implemented in MATLAB and the time series are generated at a sampling rate of 0.01 time units. The subsystems, ranging from X_1 to X_5, influence each other according to the following rule: i-th time series is influenced only by the $(i-1)$-th time series except for X_1 that only gives influence to X_2. The coupling strength $C = 5$ is the same for the whole set on influences.

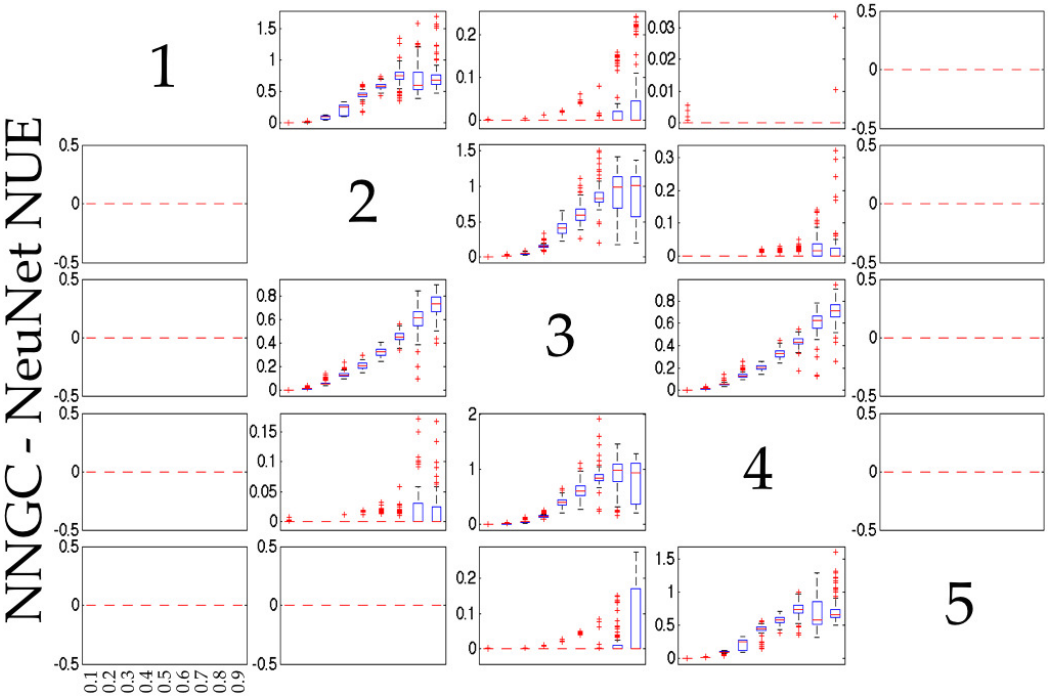

Figure 4.7: NeuNet NUE performances on Hénon maps at varying of the coupling strength. NNGC values are plotted on the y-axis, while the coupling strength values are plotted on the x-axis.

The nature of the Lorenz system results in a more challenging system than the Hénon systems. The parameters set up have been kept the same as in the other experiments. Even in this scenario NeuNet NUE can reach good performances with both high sensitivity and specificity, as shown in table 4.5. Our method, on this system too, reaches performances in the middle between the model-free approaches and the model-based approach.

Another experiment on a chaotic Lorenz system was performed in order to check how robust NeuNEt NUE could be with respect to influences occurring at longer delays. We used 150 bidirectionally coupled Lorenz systems as in [46]. The delay at which series 1 influences series 2 was set at 45 points back, while the delay at which series 2 influences series 1 was set at 75 points back. The coupling constant was set as 0.1 for both series. We chose 90 candidates for each series and checked how many times each candidate was chosen. As we can see in figures 4.12-4.15 NN NUE can detect the right delays even if there are many other

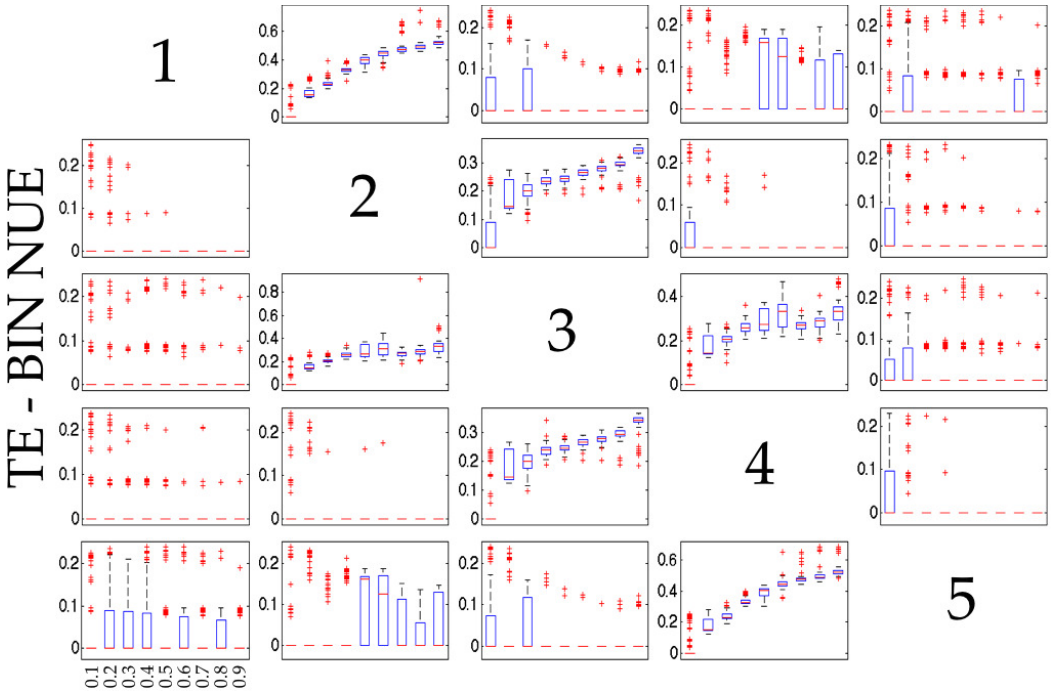

Figure 4.8: BIN NUE performances on Hénon maps at varying of the coupling strength. TE values are plotted on the y-axis, while the coupling strength values are plotted on the x-axis.

candidates chosen. NeuNet NUE was successful in retrieving the correct influences at the corresponding delays, more often than NN NUE as it can be seen from the height of the peaks. The other two estimators clearly failed in detecting the right influences and delays.

4.6.5 Redundant data

An issue that complicates the correct detection of GC is the presence of redundant variables. In this case the conditioning approach is misled and the analysis results in false negatives (see [45] for a complete explanation of this phenomenon). We applied neural networks Granger causality analysis to redundant data to check whether the approach was able to detect the right information flows with an increasing number of redundant variables. We used data generated by the following

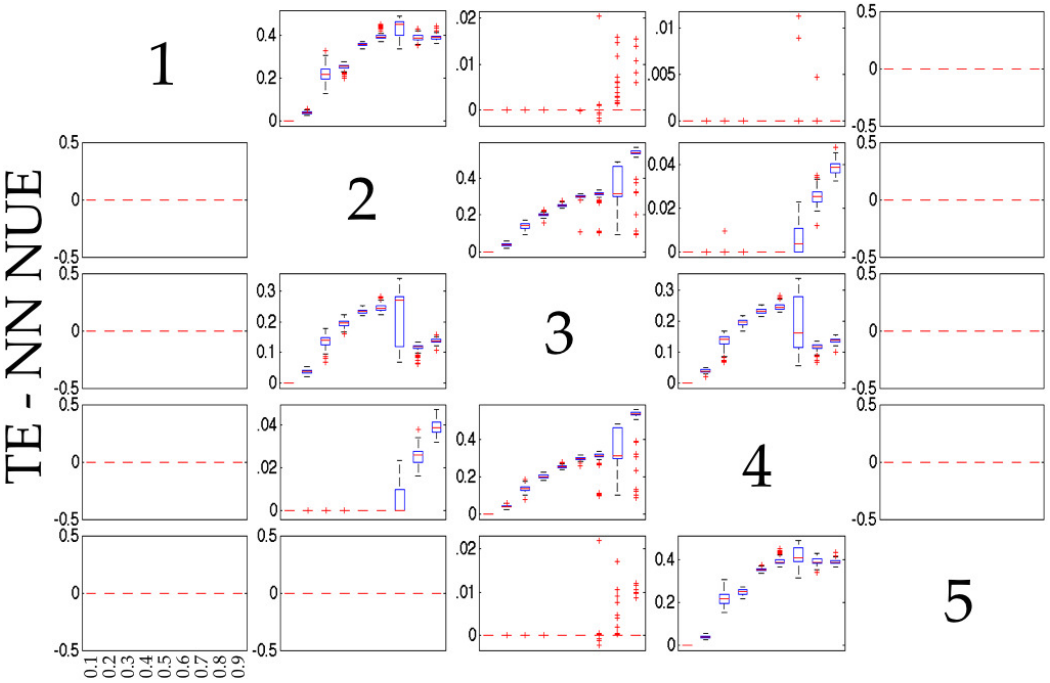

Figure 4.9: NN NUE performances on Hénon maps at varying of the coupling strength. TE values are plotted on the y-axis, while the coupling strength values are plotted on the x-axis.

equations:

$$\begin{aligned} t_n &= h_{n-2} + c\varepsilon_n \\ d_{i,n} &= h_{n-1} + c\varphi_{i,n} \end{aligned} \tag{4.10}$$

where the process h and the noises ε, φ are drawn from a Gaussian distribution with zero mean and unit variance. The coefficient c modulates the noise. The system represent a chain of influences, for which redundancy arises when $i \geq 3$, (the first two variables share information on the future of the third one), and so on.

We compared NeuNet NUE with the fully conditioned non-linear kernel Granger causality as in [45]. The experiments were performed with $l_X = l_Y = l_Z = 5$ and keeping the parameters found in Subsection 4.6.1 fixed. We generated 20 trials of the system (4.10) varying the number of redundant variables from 1 up to 20, with 2500 time points. The analyses were performed taking into account the variable t as targets and each variable d_i as driver, conditioning on the remaining $d_{(i-1)}$

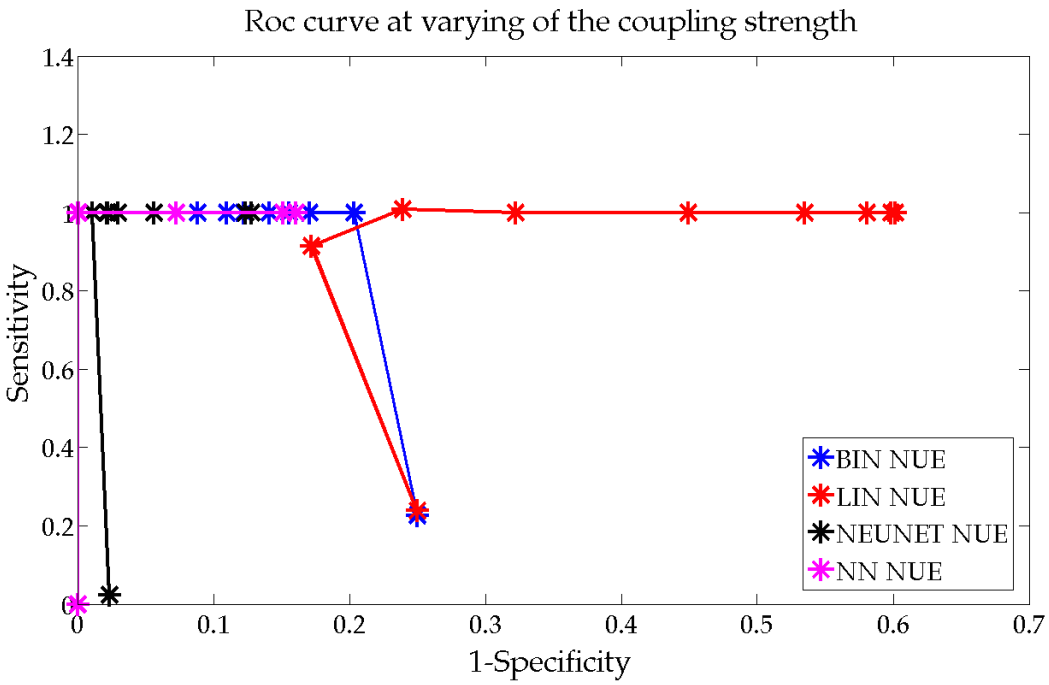

Figure 4.10: ROC curves obtained on Hénon maps at varying of the coupling strength.

variables, with $i \in [1, \ldots, 20]$. We then evaluated GC for both methods averaging over the number of trials, varying the number of redundant variables. According to the results shown in figure 4.16, we can notice how GC detected by the neural networks never drops to zero, as it happens for kernel GC. Table 4.6 reports the number of false negatives given by NeuNet NUE. It is worth noting that the amount of false negatives is zero up to 10 redundant variables. Conversely, due to the different construction of the method, the values of kernel Granger causality are always significant, albeit very low, at least for this system size.

4.6.6 Classification task

Our final goal was to test whether NeuNet NUE could correctly classify dynamics. We trained NeuNet NUE on the system (4.8). We have randomly chosen one of the networks trained to detect the causal influences towards a certain target. Then we fed the network with 100 realizations of the system (4.8), never used in the learning phase, and with 100 realizations of an autoregressive system, represented by the

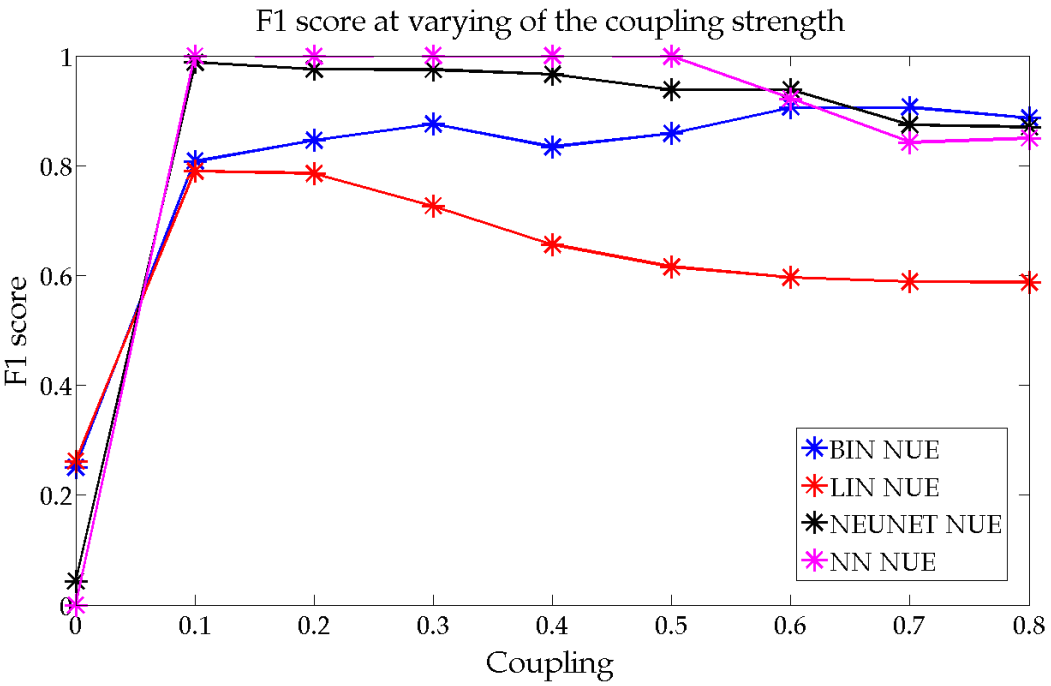

Figure 4.11: $F1_{score}$ obtained on Hénon maps at varying of the coupling strength.

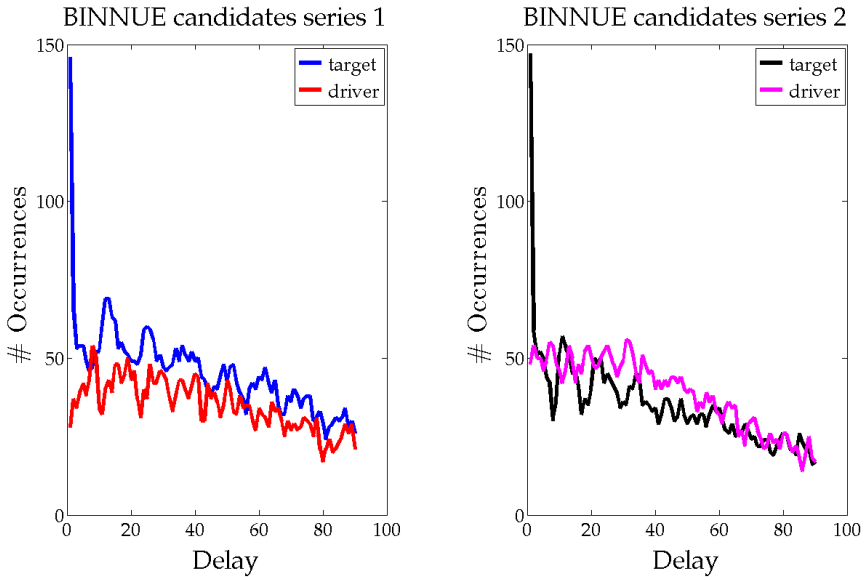

Figure 4.12: BIN NUE performances on bidirectionally coupled Lorents system.

following equations:

$$
\begin{aligned}
X_{1,n} &= 0.95\sqrt{2}X_{1,n-1} - 0.9025X_{1,n-2} + z_{1,n} \\
X_{2,n} &= 0.5X_{1,n-2}^2 + z_{2,n} \\
X_{3,n} &= -0.4X_{1,n-3} + z_{3,n} \\
X_{4,n} &= -0.5X_{1,n-2}^2 + 0.25\sqrt{2}X_{4,n-1} + 0.25\sqrt{2}X_{5,n-1} + z_{4,n} \\
X_{5,n} &= -0.25\sqrt{2}X_{4,n-1} + 0.25\sqrt{2}X_{5,n-1} + z_{5,n}
\end{aligned}
\tag{4.11}
$$

	Sens	Spec	$F1_{score}$
BIN NUE	0.85	0.68	0.73
LIN NUE	0.88	0.45	0.65
NeuNet NUE	0.86	0.80	0.80
NN NUE	0.87	0.91	0.87

Table 4.4: Sensitivity, specificity and $F1_{score}$ values obtained on the system (4.7) by the four estimators

	Sens	Spec	$F1_{score}$
BIN NUE	0.94	0.91	0.82
LIN NUE	0.51	0.72	0.39
NeuNet NUE	0.70	0.95	0.74
NN NUE	1	0.86	0.77

Table 4.5: Sensitivity, specificity and $F1_{score}$ values obtained on the Lorenz system by the four estimators.

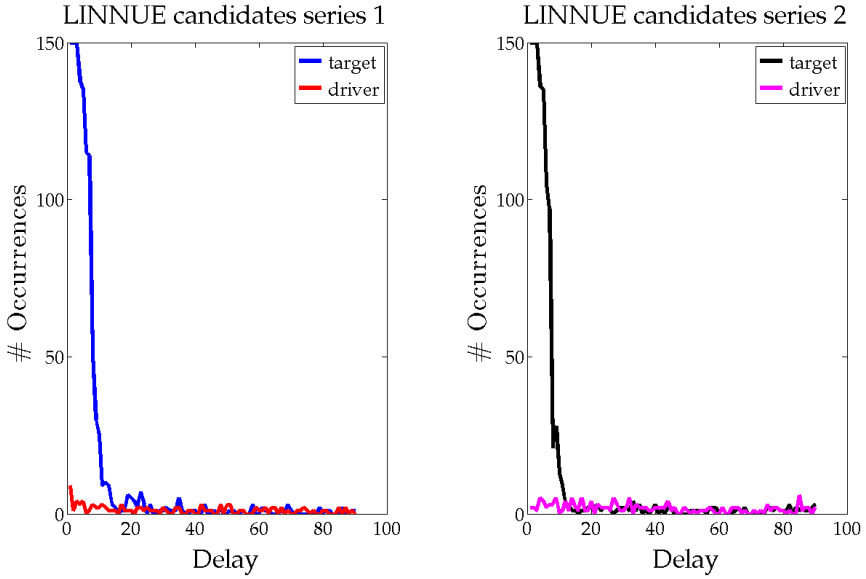

Figure 4.13: LIN NUE performances on bidirectionally coupled Lorents system.

where $z_{1,n}, z_{2,n}, z_{3,n}, z_{4,n}, z_{5,n}$ are drawn from Gaussian noise with zero mean and unit variance.

System 4.8 is considered, this time with only five variables to be consistent with the size of the autoregressive model. We then evaluated the average RMSE for both

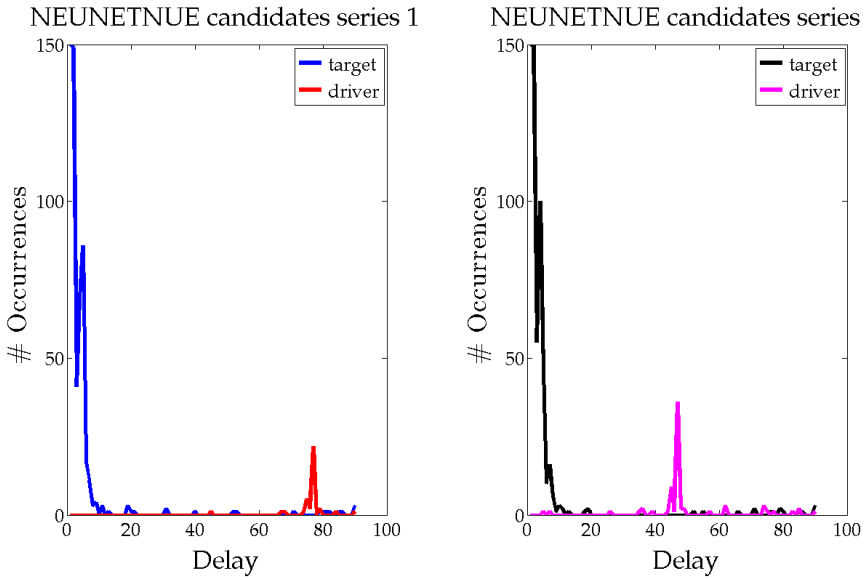

Figure 4.14: NeuNet NUE performances on bidirectionally coupled Lorents system.

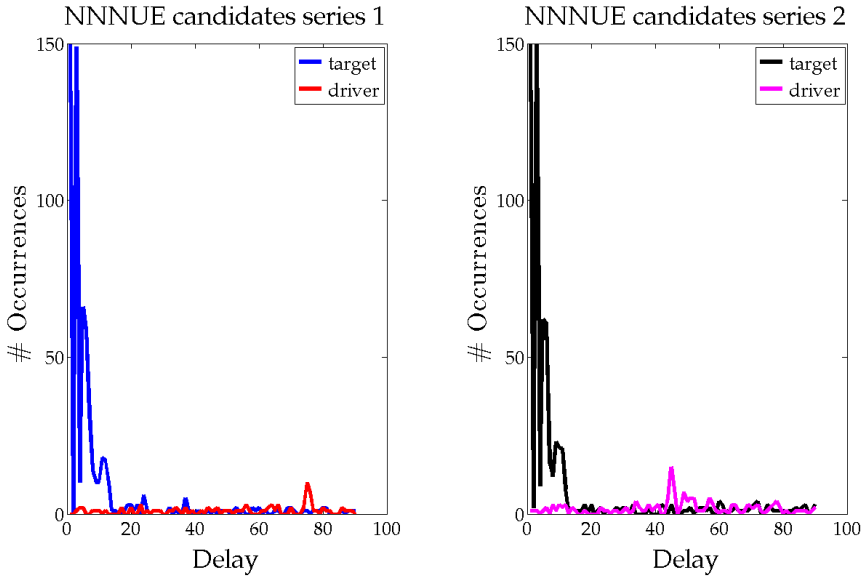

Figure 4.15: NN NUE performances on bidirectionally coupled Lorents system.

systems. We repeated the procedure for 30 different noise values, ranging from 0 to 0.7, and different couplings strength, ranging in the interval [0,0.8] with step of 0.2. The results are shown in figure 4.17. We plotted the average RMSE with respect to the different noise level for each coupling strength value. We can notice

# RV	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
# FN	0	0	0	0	0	0	0	0	0	0	3	4	14	13	17	29	41	49	57	79

Table 4.6: Number of false negatives, FN, returned by NeuNet NUE for 20 trials at varying of the number of redundant variables, RV.

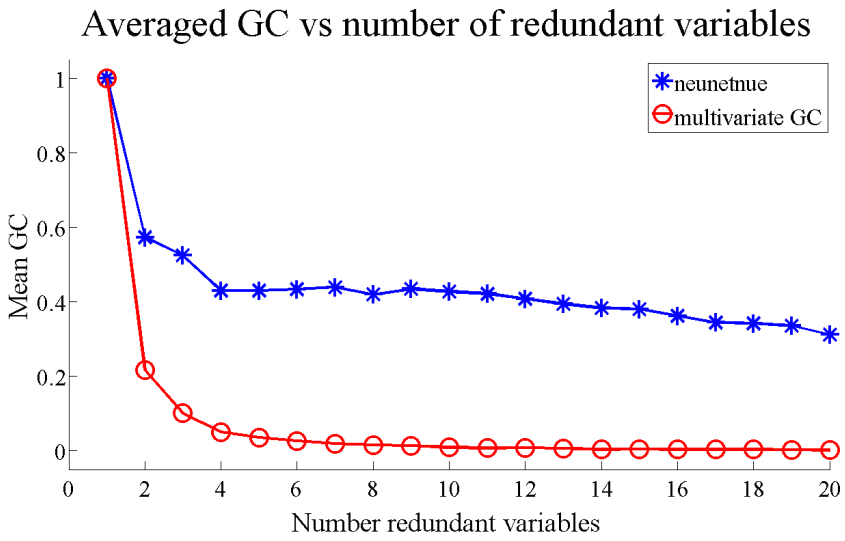

Figure 4.16: NeuNet NUE and multivariate GC performances on the redundant system.

that the errors obtained when the two systems are given to NeuNet NUE as test sets lie in linear separable portions of space. This represents an encouraging result as it may be useful to classify systems, given that our approach has been trained with a known system.

4.7 Conclusions

In this chapter we have implemented the Granger paradigm for detection of dynamical influences in the frame of feed-forward neural networks. The novelty of the present approach arises from the use of non-uniform embedding for variable selection and generalization error for the assessment of Granger causality. We have demonstrated the theoretical and experimental advantages of implementing the neural network approach with non-uniform embedding compared to the uniform one. Due to the universal character of function approximation of neural networks,

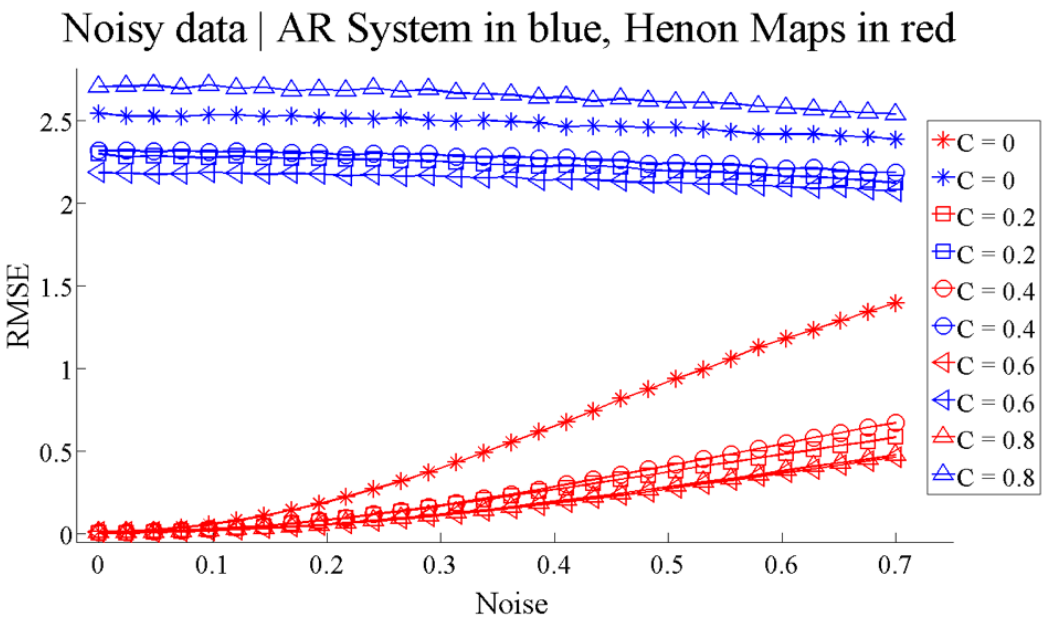

Figure 4.17: RMSE versus the noise level. The red curves are obtained testing NeuNet NUE on the same kind of data with which it was trained. The blue curves are obtained testing NeuNet NUE on the system (4.11). Each curve represents the network trained to detect the influence towards a specific target with a different coupling strength. The couplings C are shown in the legend.

the proposed approach is intermediate between the classical Granger linear implementation and the non-parametric estimator corresponding to transfer entropy: by means of several examples, we have shown that there are situations where our approach outperforms both approaches. The proposed method differs from the kernel Granger causality not only by providing a validation phase, but also by letting the neural networks explore the parameters space and building the best model to explain the information transfers among variables. Kernel Granger causality, instead, is still a model-based approach, for which the type of kernel and the degree of non-linearity have to be specified beforehand. We would like to remark that so far neural networks have been used to detect GC only when combined with other estimators. Furthermore the training phase was stopped only when a certain number or training epochs was reached. This choice seems quite approximate because of the lack of knowledge about the exact amount of training epochs needed to both minimize the error function and to avoid overfitting the neural networks. Therefore,

the validation phase is necessary in our opinion to ensure that the network fully explores the parameters space, converges to a minimum and avoids the risk of overfitting. We conclude remarking that other wrappers can be taken into account and many deep learning architectures are built from artificial neural networks, therefore we expect that further developments of our approach will be the implementations of Granger causality both using other feature selection algorithms and in the frame of deep learning [47].

Acknowledgments

This work is supported by: the Belgian Science Policy (IUAP VII project CEREBNET P7 11); the University of Ghent (Special Research Funds for visiting researchers). The neural network toolbox has been developed by Roberto Prevete and Giovanni Tessitore: rprevete@unina.it; tessitore@na.infn.it

The authors would like to thank Dr. Christopher Stewart for his valuable feedback.

CHAPTER 5

Multiscale Analysis of Information Dynamics for Linear Multivariate Processes

Paper published on ArXiv; arXiv:1602.06155 and presented at the 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society.

In the study of complex physical and physiological systems represented by multivariate time series, an issue of great interest is the description of the system dynamics over a range of different temporal scales. While information-theoretic approaches to the multiscale analysis of complex dynamics are being increasingly used, the theoretical properties of the applied measures are poorly understood. This study introduces for the first time a framework for the analytical computation of information dynamics for linear multivariate stochastic processes explored at different time scales. After showing that the multiscale processing of a Vector Auto Regressive (VAR) process introduces a Moving Average (MA) component, we describe how to represent the resulting VARMA process using State-Space (SS) models and how to exploit the SS model parameters to compute analytical measures of information storage and information transfer for the original and rescaled processes. The framework is then used to quantify multiscale information dynamics for simulated unidirectionally and bidirectionally coupled VAR processes,

showing that rescaling may lead to insightful patterns of information storage and transfer but also to potentially misleading behaviours.

5.1 Introduction

Several physiological systems, including the brain and the cardiovascular system, coordinate their activity according to regulatory mechanisms operating across multiple temporal scales [48, 49]. Due to this multiscale behavior, the output signals of these systems (e.g., the EEG or cardiovascular variability time series) need to be analyzed through scaling techniques to get full insight about the system dynamics. A typical approach is to resample the originally measured physiological time series at various temporal scales, yielding a collection of rescaled series from which various dynamical measures can be calculated. Exploiting information-theoretic functionals that may be subsumed within the framework of information dynamics [50], this approach has been followed both to describe the individual dynamics of single time series through the so-called multiscale entropy [51], and to explore the joint dynamics of multiple time series through the multiscale transfer entropy (TE) [52].

In spite of its potential, the computation of multiscale measures of information dynamics may be complicated by theoretical and practical issues [53] [54]. These issues arise from the procedure for the generation of the rescaled time series, which essentially consists in a filtering step eliminating the fast temporal scales (usually performed through averaging) followed by a downsampling step coarse-graining the time series around the selected scale. While it is expected that these two steps may be problematic, their impact on the computation of multiscale information dynamics has never been investigated systematically. To fill this gap, the present study introduces a framework for the analytical computation of information dynamics for linear Gaussian dynamic processes subjected to averaging and downsampling. The framework is based on the theory of state-space (SS) models, and builds on very recent theoretical results [55] [56] to study the exact values of information storage (storage entropy, SE) and information transfer (TE) for coupled processes observed at different time scales. While this study concentrates on the theoretical

formulation and analysis of simulated linear processes, future extensions will be devoted to practical estimation, study of nonlinear dynamics and application to real time series.

5.2 Multiscale Representation of Linear Processes

Let us consider a set of M time series of length N, $y_{m,n}, m = 1,\dots,M; n = 1,\dots,N$, as a finite length realization of the zero mean stationary vector stochastic process $Y_n = [y_{1,n}\cdots y_{M,n}]^T$. In the linear signal processing framework, the process is classically described as a Vector Autoregressive (VAR) process of order p:

$$Y_n = \sum_{k=1}^{p} \mathbf{A}_k Y_{n-k} + U_n \tag{5.1}$$

where A_k are $M \times M$ matrices of coefficients, and $U_n = [u_{1,n}\cdots u_{M,n}]^T$ is a vector of M zero mean Gaussian processes with covariance matrix $\Sigma \equiv \mathbb{E}[U_n U_n^T]$.

According to the traditional procedure for multiscale analysis [51], each scalar process y_m can be rescaled using an integer scale factor τ to get the process $\bar{y}_m$:

$$\bar{y}_{m,n} = \frac{1}{\tau}\sum_{l=0}^{\tau-1} y_{m,n\tau-l}, n = 1,\dots,N/\tau \tag{5.2}$$

The change of scale in (5.2) corresponds to transform the original process Y through a two step procedure that consists of the following *averaging* and *down-sampling* steps, yielding respectively the processes $\tilde{Y}$ and $\bar{Y}$:

$$\tilde{Y}_n = \frac{1}{\tau}\sum_{l=0}^{\tau-1} Y_{n-l}, n = \tau,\dots,N \tag{5.3a}$$

$$\bar{Y}_n = \tilde{Y}_{n\tau}, n = 1,\dots,N/\tau \tag{5.3b}$$

Now, substituting (5.1) in (5.3a), one can show that the averaging step yields

the following process representation:

$$\tilde{Y}_n = \sum_{k=1}^{p} \mathbf{A}_k \tilde{Y}_{n-k} + \sum_{l=0}^{\tau-1} \mathbf{B}_l U_{n-l} \tag{5.4}$$

where $\mathbf{B}_l = 1/\tau I_M$ for each $l = 0, \ldots, \tau-1$ (I_M is the $M \times M$ identity matrix). This shows that the change of scale introduces a moving average (MA) component of order $q = \tau - 1$ in the original VAR(p) process, transforming it into a VARMA(p,q) process. As we will show in the next Section, the downsampling step (5.3b) keeps the VARMA representation altering the model parameters.

5

5.3 State Space Models

5.3.1 Formulation of SS Models

The general linear state space (SS) model describing an observed vector process Y is in form:

$$X_{n+1} = \mathbf{A}X_n + W_n \tag{5.5a}$$

$$Y_n = \mathbf{C}X_n + V_n \tag{5.5b}$$

where the state equation (5.5a) describes the update of the $L-$dimensional state (unobserved) process through the $L \times L$ matrix $\mathbf{A}$, and the observation equation (5.5b) describes the instantaneous mapping from the state to the observed process through the $M \times L$ matrix $\mathbf{C}$. W_n and V_n are zero-mean white noise processes with covariances $\Xi \equiv \mathbb{E}[W_n W_n^T]$ and $\Psi \equiv \mathbb{E}[V_n V_n^T]$, and cross-covariance $\Upsilon \equiv \mathbb{E}[W_n V_n^T]$. Thus, the parameters of the SS model (5.5) are $(\mathbf{A}, \mathbf{C}, \Xi, \Psi, \Upsilon)$.

Another possible SS representation is that evidencing the *innovations* $E_n = Y_n - \mathbb{E}[Y_n|Y_n^-]$, i.e. the residuals of the linear regression of Y_n on its infinite past $Y_n^- = [Y_{n-1}^T Y_{n-2}^T \cdots]^T$. This new SS representation, usually referred to as Innovations State-Space (ISS) model, is characterized by the state process $Z_n = \mathbb{E}[X_n|Y_n^-]$ and by the $L \times M$ Kalman Gain matrix $\mathbf{K}$:

$$Z_{n+1} = \mathbf{A}Z_n + \mathbf{K}E_n \tag{5.6a}$$

$$Y_n = \mathbf{C}Z_n + E_n \tag{5.6b}$$

The parameters of the ISS model (5.6) are $(\mathbf{A}, \mathbf{C}, \mathbf{K}, \Phi)$, where Φ is the covariance of the innovations, $\Phi \equiv \mathbb{E}[E_n E_n^T]$. Note that the ISS (5.6) is a special case of (5.5) in which $W_n = \mathbf{K}E_n$ and $V_n = E_n$, so that $\Xi = \mathbf{K}\Phi\mathbf{K}^T$, $\Psi = \Phi$ and $\Upsilon = \mathbf{K}\Phi$.

Given an SS model in the form (5.5), the corresponding ISS model (5.6) can be identified by solving a so-called discrete algebraic Ricatti equation (*DARE*) formulated in terms of the state error variance matrix $\mathbf{P}$:

$$\begin{aligned}\mathbf{P} &= \mathbf{A}\mathbf{P}\mathbf{A}^T + \Xi \\ &\quad - (\mathbf{A}\mathbf{P}\mathbf{C}^T + \Upsilon)(\mathbf{C}\mathbf{P}\mathbf{C}^T + \Psi)^{-1}(\mathbf{C}\mathbf{P}\mathbf{A}^T + \Upsilon^T)\end{aligned} \tag{5.7}$$

Under some assumptions [56], the *DARE* (5.7) has an unique stabilizing solution, from which the Kalman gain and innovation covariance can be computed as

$$\begin{aligned}\Phi &= \mathbf{C}\mathbf{P}\mathbf{C}^T + \Psi \\ \mathbf{K} &= (\mathbf{A}\mathbf{P}\mathbf{C}^T + \Upsilon)\Phi^{-1}\end{aligned} \tag{5.8}$$

5.3.2 SS Models for Averaged and Downsampled Processes

Exploiting the close relation between VARMA models and SS models, first we show how to convert the VARMA model (5.4) into an ISS model in the form of (5.6) that describes the averaged process $\tilde{Y}_n$. To do this, we exploit the Aoki's method [57] defining the state process $\tilde{Z}_n = [Y_{n-1}^T \cdots Y_{n-p}^T U_{n-1}^T \cdots U_{n-q}^T]^T$ that, together with $\tilde{Y}_n$, obeys the state equations (5.6) with parameters $(\tilde{\mathbf{A}}, \tilde{\mathbf{C}}, \tilde{\mathbf{K}}, \tilde{\Phi})$, where

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_1 & \cdots & \mathbf{A}_{p-1} & \mathbf{A}_p & \mathbf{B}_1 & \cdots & \mathbf{B}_{q-1} & \mathbf{B}_q \\ \mathbf{I}_M & \cdots & \mathbf{0}_M & \mathbf{0}_M & \mathbf{0}_M & \cdots & \mathbf{0}_M & \mathbf{0}_M \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots \\ \mathbf{0}_M & \cdots & \mathbf{I}_M & \mathbf{0}_M & \mathbf{0}_M & \cdots & \mathbf{0}_M & \mathbf{0}_M \\ \mathbf{0}_M & \cdots & \mathbf{0}_M & \mathbf{0}_M & \mathbf{0}_M & \cdots & \mathbf{0}_M & \mathbf{0}_M \\ \mathbf{0}_M & \cdots & \mathbf{0}_M & \mathbf{0}_M & \mathbf{I}_M & \cdots & \mathbf{0}_M & \mathbf{0}_M \\ \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots \\ \mathbf{0}_M & \cdots & \mathbf{0}_M & \mathbf{0}_M & \mathbf{0}_M & \cdots & \mathbf{I}_M & \mathbf{0}_M \end{bmatrix}$$

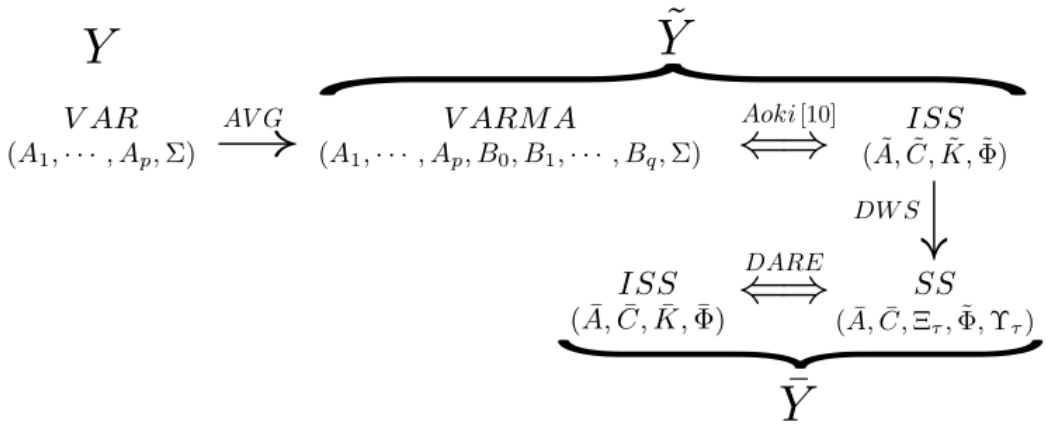

Figure 5.1: Schematic representation of the parametric representation of linear multivariate processes. See text for details.

$$\tilde{\mathbf{C}} = \begin{bmatrix} \mathbf{A}_1 & \cdots & \mathbf{A}_p & \mathbf{B}_1 & \cdots & \mathbf{B}_q \end{bmatrix}$$

$$\tilde{\mathbf{K}} = \begin{bmatrix} \mathbf{I}_M & \mathbf{0}_{M\times M(p-1)} & \mathbf{B}_0^{-T} & \mathbf{0}_{M\times M(q-1)} \end{bmatrix}^T$$

and $\tilde{\Phi} = \mathbf{B}_0 \Sigma \mathbf{B}_0^T$, where $\tilde{\Phi}$ is the covariance of the innovations $\tilde{E}_n = \mathbf{B}_0 U_n$.

Now we turn to show how the downsampled process $\bar{Y}_n$ can be represented through an ISS model directly from the ISS formulation of the averaged process $\tilde{Y}_n$. According to a very recent result (theorem III in [56]), we have that the process $\bar{Y}_n = \tilde{Y}_{n\tau}$ has an ISS representation with state process $\bar{Z}_n = \tilde{Z}_{n\tau}$, innovation process $\bar{E}_n = \tilde{E}_{n\tau}$, and parameters $(\bar{\mathbf{A}}, \bar{\mathbf{C}}, \bar{\mathbf{K}}, \bar{\Phi})$, where $\bar{\mathbf{A}} = \tilde{\mathbf{A}}^\tau$, $\bar{\mathbf{C}} = \tilde{\mathbf{C}}$, and where $\bar{\mathbf{K}}$ and $\bar{\Phi}$ are obtained solving the *DARE* (5.7,5.8) for the SS model $(\bar{\mathbf{A}}, \bar{\mathbf{C}}, \Xi_\tau, \tilde{\Phi}, \Upsilon_\tau)$ with

$$\begin{aligned} \Upsilon_\tau &= \tilde{\mathbf{A}}^{\tau-1} \tilde{\mathbf{K}} \tilde{\Phi} \\ \Xi_\tau &= \tilde{\mathbf{A}} \Xi_{\tau-1} \tilde{\mathbf{A}}^T + \tilde{\mathbf{K}} \tilde{\Phi} \tilde{\mathbf{K}}^T, \tau \geq 2 \\ \Xi_1 &= \tilde{\mathbf{K}} \tilde{\Phi} \tilde{\mathbf{K}}^T, \tau = 1 \end{aligned} \tag{5.9}$$

5.4 Multiscale Information Dynamics

Fig. 5.1 depicts the relations and parametric representations of the original process Y, the averaged process $\tilde{Y}$, and the downsampled process $\bar{Y}$. As seen up to now, the averaging (AVG) over segments of length τ applied to a VAR(p) process yields

a VARMA($p, \tau - 1$) process, which is equivalent to an ISS process [57], and the subsequent downsampling (DWS) yields a different SS process, which in turn can be converted to the ISS form by solving the *DARE* (Fig. 5.1). Thus, both the averaged process $\tilde{Y}_n$ and the downsampled process $\bar{Y}_n$ can be represented as ISS processes with parameters $(\tilde{\mathbf{A}}, \tilde{\mathbf{C}}, \tilde{\mathbf{K}}, \tilde{\Phi})$ and $(\bar{\mathbf{A}}, \bar{\mathbf{C}}, \bar{\mathbf{K}}, \bar{\Phi})$ which can be derived analytically from knowledge of the parameters $(\mathbf{A}_1, \ldots, \mathbf{A}_p, \Sigma)$ of the original process and of the scale factor τ. In this section we show how to compute analytically the measures of information dynamics starting from the ISS model parameters, thus opening the way to the analytical computation of these measures for multiscale (averaged and downsampled) processes.

Given a generic vector observation process Y_n, let us consider the scalar subprocess $y_{j,n}$ as the *target*, and the $(M-1)-$dimensional vector $Y_{i,n} = Y_n \backslash y_{j,n}$ as the *driver* ($i = \{1 \cdots, M\} \backslash j$). In the framework of information dynamics [50], the predictive information of the target of a multivariate process, P_j, measures how much of the information carried by $y_{j,n}$ can be predicted from the knowledge of Y_n^-. This amount can be decomposed as the sum of the information storage S_j and the information transfer $T_{i \to j}$, quantifying respectively the amount of information carried by $y_{j,n}$ that can be predicted from its own past $y_{j,n}^-$ and the additional amount that can be predicted from the whole past Y_n^-. The information storage and transfer are quantified by the so-called storage entropy (SE) and transfer entropy (TE) [58] which, for linear Gaussian processes, are given by:

$$S_j = \frac{1}{2} ln \frac{\lambda_j}{\lambda_{j|j}} \tag{5.10a}$$

$$T_{i \to j} = \frac{1}{2} ln \frac{\lambda_{j|j}}{\lambda_{j|ij}} \tag{5.10b}$$

where $\lambda_j = \mathbb{E}[y_{j,n}^2]$ is the variance of the target process, and $\lambda_{j|j} = \mathbb{E}[e_{j|j,n}^2]$ and $\lambda_{j|ij} = \mathbb{E}[e_{j|ij,n}^2]$ are the partial variance of the target given its own past, $e_{j|j,n} = y_{j,n} - \mathbb{E}[y_{j,n}|y_{j,n}^-]$, and the partial variance of the target given the past of the whole process, $e_{j|ij,n} = y_{j,n} - \mathbb{E}[y_{j,n}|Y_n^-]$.

Now we report how to compute the variances appearing in (5.10) from the parameters of an ISS model in the form of (5.6). First, we note that the variance of

$e_{j|ij,n}$ is simply the $j-th$ diagonal element of the innovation covariance: $\lambda_{j|ij} = \Phi(j,j)$. The variance of $y_{j,n}$ corresponds to the $j-th$ diagonal element of the zero-lag autocovariance of the whole process $\Gamma \equiv \mathbb{E}[Y_n Y_n^T]$: $\lambda_j = \Gamma(j,j)$; for an ISS process, the latter can be computed as $\Gamma = \mathbf{C}\Omega\mathbf{C}^T + \Phi$, where $\Omega = \mathbb{E}[Z_n Z_n^T]$ satisfies the discrete Lyapunov equation $\Omega = \mathbf{A}\Omega\mathbf{A}^T + \mathbf{K}\Phi\mathbf{K}^T$. Computation of the partial variance of the target given its past is less straightforward, involving the formation of a subprocess of the original ISS process. Specifically, one needs to consider the submodel with state equation (5.6a) and observation equation

$$y_{j,n} = \mathbf{C}^{(j)} Z_n + e_{j,n} \tag{5.11}$$

where $\mathbf{C}^{(j)}$ is the $j-th$ row of $\mathbf{C}$. The submodel (5.6a, 5.11) is *not* in innovations form, but is rather an SS model with parameters $(\mathbf{A}, \mathbf{C}^{(j)}, \mathbf{K}\Phi\mathbf{K}^T, \Phi(j,j), \mathbf{K}\Phi^{T^{(j)}})$. As such, solving the *DARE* (5.7,5.8) it can be converted to an ISS model with innovation covariance $\lambda_{j|j}$.

5.5 Simulation Experiment

In order to study the multiscale patterns of information dynamics for linear interacting processes, we analyze the bivariate VAR process with equations:

$$y_{1,n} = a_1 y_{1,n-b_1} + c_1 y_{2,n-d_1} + u_{1,n} \tag{5.12a}$$

$$y_{2,n} = a_2 y_{2,n-b_2} + c_2 y_{1,n-d_2} + u_{2,n} \tag{5.12b}$$

with iid noise processes $u_{1,n}, u_{2,n} \sim \mathcal{N}(0,1)$ so that $\Sigma = \mathbf{I}_2$. The parameters in (5.12) are set to generate autonomous dynamics with strength a_i and lag b_i for each scalar process y_i, and causal interactions with strength c_i and lag d_i from y_j to y_i $(i,j = 1,2)$. We consider two parameter configurations: unidirectional interaction $y_1 \to y_2$, obtained setting $c_1 = 0$ and $c_2 = 0.5, d_2 = 2$, where also autonomous dynamics were generated for y_1 $(a_1 = 0.25, b_1 = 1)$ but not for y_2 $(a_2 = 0)$; bidirectional interactions between processes with autonomous dynamics $(a_1 = 0.25, b_1 = 2; a_2 = 0.25, b_2 = 5)$ obtained setting $c_2 = 0.5, d_2 = 7$ (direction $y_1 \to y_2$) and $c_1 = 0.75, d_1 = 3$ (direction $y_2 \to y_1$).

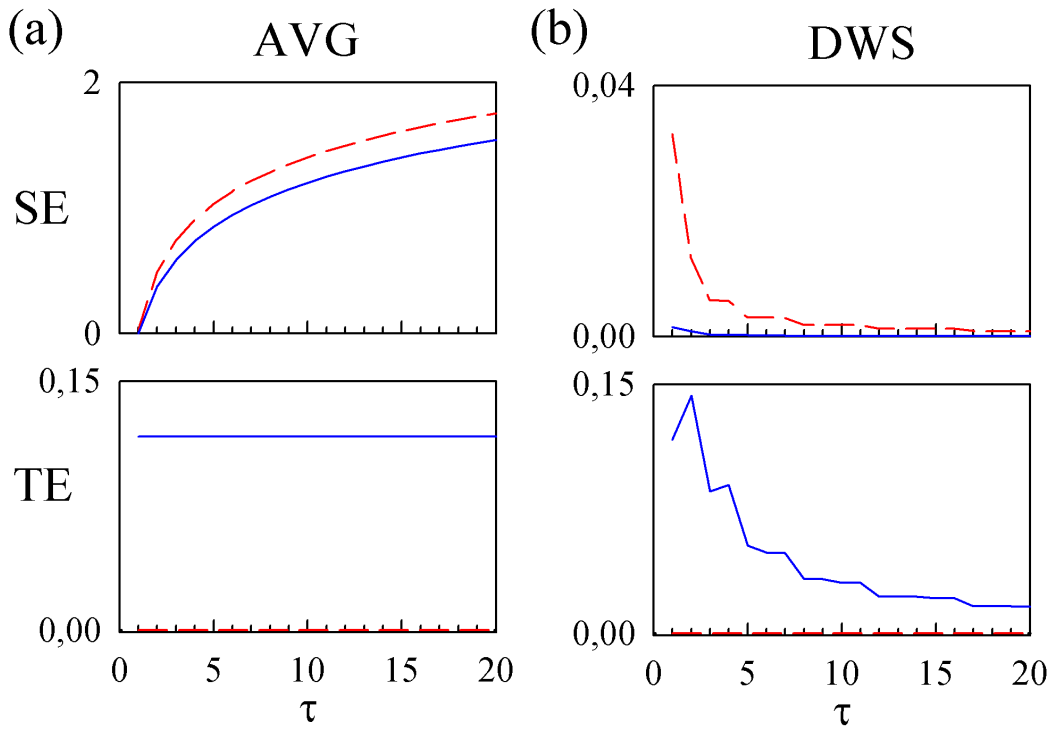

Figure 5.2: Multiscale information dynamics for the unidirectionally coupled VAR process (5.12). Plots depict the information storage (SE) and transfer (TE) computed after averaging (AVG) and downsampling (DWS) the process at scale τ; red-dashed: $S_1, T_{2\to1}$, blue-solid: $S_2, T_{1\to2}$. (a,b) parameter $a_1 = 0.25$; (c,d) parameter $a_1 = 0.95$.

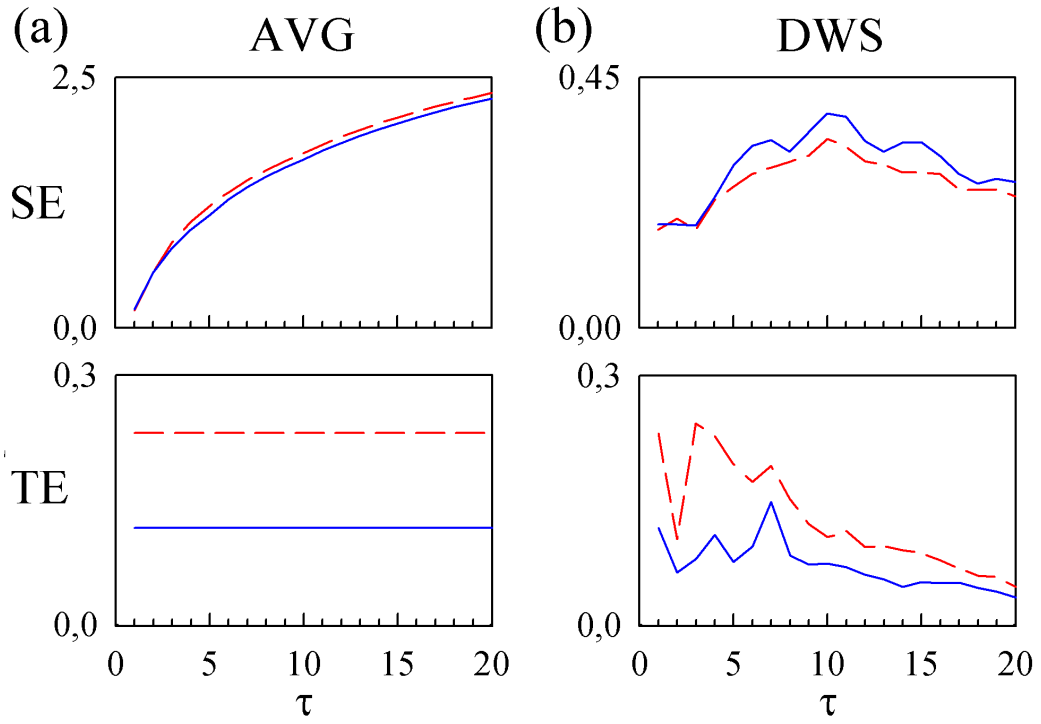

Figure 5.3: Multiscale information dynamics for the bidirectionally coupled VAR process (5.12). Plots and symbols are as in Fig. 5.2.

The results of multiscale analysis of SE and TE performed for the two configurations are shown in Figs. 5.2, 5.3. The values of information dynamics for the original processes, reported in the figures for $\tau = 1$, indicate that the SE reflects auto-dependencies in the target process (e.g., $S_1 > S_2$ in Fig. 5.2 where $a_1 > a_2$, and $S_1 = S_2$ in Fig. 5.3 where $a_1 = a_2$), and that the TE reflects causal interactions from driver to target (e.g., $T_{2\to1} = 0$ in Fig. 5.2 where $c_1 = 0$ and $T_{2\to1} > T_{1\to2}$ in Fig. 5.3 where $c_1 > c_2$). The averaging procedure associated with the change of scale always leads to a progressive increase of the information stored in each individual process (Figs. 5.2a, 5.3a). Moreover, averaging does not alter the amount of information transferred between the processes, as documented by the constant values of the TE across scales observed in all configurations. The downsampling step introduces more substantial alterations in the patterns of information dynamics. The information storage is reduced substantially and reflects the multiscale regularity of each individual process, with higher SE around the scales at which the processes exhibit their lagged interactions (i.e., very low lags in Fig.5.2(b) and higher lags in Fig. 5.3(b)). The information transfer reflects causal interactions between the processes at different time scales, with the TE showing a peak at the lags of the imposed causal interactions (i.e., $\tau = 2$ for $T_{1\to2}$ in Fig. 5.2(b), $\tau = 7$ for $T_{1\to2}$ and $\tau = 3$ for $T_{2\to1}$ in Fig. 5.3(b)).

The behaviors described above are general, in the sense that they were observed also for different parameter configurations. Nevertheless, some particular parameter settings led to unexpected, potentially misleading results. An example is reported in Fig. 5.4, showing the information transfer computed for the first configuration (unidirectional coupling) but with stronger autonomous dynamics of y_1 ($a_1 = 0.95$). In this case the TE $T_{1\to2}$ still shows a peak at the scale corresponding to the lag of the imposed causal relation ($\tau = d_2 = 2$), but a significant TE emerges at large scales along the uncoupled direction ($T_{2\to1} > 0$ for $\tau > 2$).

5.6 Conclusions

We presented a framework for the multiscale computation of the information stored and transferred in multivariate linear processes, assessed respectively through the

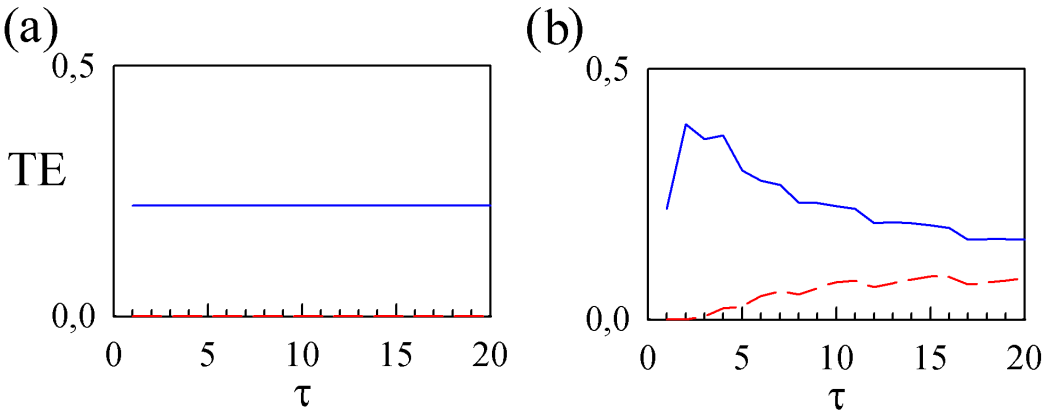

Figure 5.4: Multiscale information dynamics for the unidirectionally coupled VAR process (5.12) with stronger driver autonomous dynamics. Plots and symbols are as in Fig. 5.2.

SE and TE measures, starting from the parameters of the VAR model describing the process and from the scale factor τ.

Our simulation results show that the first step of multiscale analysis, i.e. the averaging of each individual process across τ consecutive points, introduces an auto-correlation in the process that is reflected by the progressive increase with τ of the SE. Moreover, as this step leaves the coefficients regulating the linear interaction across the processes unchanged, the TE does not vary with τ; this result is related to the invariance of Granger causality with filtering [53].

The second analysis step, i.e. the downsampling of the averaged process at fixed time intervals τ, removes the autocorrelation of the innovations inflating the SE, thus allowing a more informative evaluation of the multiscale complexity of the individual time series [51]. Moreover, this step makes the TE scale-dependent, with a peak shown at the time scale corresponding to the lags of the causal interactions occurring between the processes. A negative behavior is the possible occurrence of spurious TE at scales much higher than the true coupling delays.

These results suggest that the multiscale analysis of information storage and information transfer can be useful to shed light on patterns of regularity and causality of coupled dynamic processes which are not fully disclosed working at one single time scale, but can also provide patterns with difficult physical interpretation.

Acknowledgments

Research supported by Healthcare Research and Innovation Program, IRCS-PAT-FBK, Trento.

CHAPTER 6

Conclusion and Future Research

This thesis is integrated into a wider scientific discussion that aims to understand and model information transfers within a complex system. Several application scenarios can be found in neuroscience [5, 59, 60, 61, 9], [62, 63, 64, 8, 7, 6], physiology [65, 10, 66, 67, 68], climatology [69], complex system theory [70, 71, 72] and economics [73, 74]. Furthermore, this work can also be considered as an attempt to unify different approaches in a user-friendly, modular framework. Other toolboxes are present in the literature and they all have had a great scientific impact [25, 75, 76].

In this thesis we provide a detailed explanation of eight estimators of the influences that might occur within a system represented by time series. We were interested in showing the pros and cons of each estimator in order to provide a general overview of the implemented methods. In this way, the reader should be able to easily choose which estimator he might need. We show that the best performances are obtained by BIN NUE, NN NUE and NeuNet NUE, all of which are able to detect both linear and non-linear relationships. It is worth noting that the best estimators are the ones that do not need any a priori assumptions about the model that can represent the data. Furthermore, the best performances are obtained

in the framework of the non-uniform embedding, showing that the careful choice of which past states have to be taken into account to evaluate TE or NNGC is not a trivial problem. This issue can be overcome by non-uniform embedding, even though this technique is not the only possible selection criterion that can be adopted.

One of the most important outcomes of this thesis is that we were able to bring together the disciplines of information theory and machine learning by implementing the neural networks estimator. We showed several advantages of the new methods, such as the robustness of neural networks with respect to redundancy. We would like to point out that the presence of redundant variables is a very important problem to tackle because redundancy might lead to detecting several false influences. NeuNet NUE seems to not be strongly affected by the presence of redundant variables. We showed neural networks' performances in revealing influences from each redundant variable considered as a driver to a certain target in the multivariate framework. Another advantage of NeuNet NUE is its capability to detect long range interactions when most of the proposed methods fails.

MuTE toolbox was implemented in order to provide a unique environment that is able to handle all the methods. MuTE is meant to be a user friendly toolbox. We decided to try to give people who are not so familiar with MATLAB the opportunity to use our toolbox. We did so by building a graphical user interface that can easily be set up to run MuTE for various analyses. The toolbox is modular and flexible enough to allow other users to modify some steps in the estimation procedure by replacing certain functions only. For instance, a new criterion to select the most informative past states can be easily replaced, giving rise to a new estimator. The performances of the new estimator can be immediately evaluated with respect to the other methods. A detailed explanation of how to set MuTE and how to use the graphical user interface are provided in Appendix A.

We are aware that there is plenty of room for improvements. Considering the methods that have been implemented thus far, we propose to first introduce a criterion based on the autocorrelation function in order to better optimize the choice of the embedding. We can then apply a *backward* feature selection that will be combined with the already existing non-uniform embedding feature selection. This combination would allow one to discard erroneous past states that might be chosen due to the fluctuation of the surrogate threshold. We can modify the criterion

according to which we set the binning size by choosing the equiprobable binning approach in order to better estimate the entropies. Several other improvements will concern the optimization of the neural networks method such as a different candidate selection criterion and different learning algorithms. Furthermore, we are taking other machine learning approaches into account in order to both reinforce the bridge between machine learning and information theory and improve the neural networks estimator.

We would like to remark that MuTE has been successfully applied both by groups internal to our current network [67, 68] and by groups that decided to use the toolbox independently [77], Schulz et al., in preparation, presentation at ESGCO2016 conference. Furthermore, several other groups are currently interested in using MuTE. In the spirit of an open scientific collaboration, Dr. Andrea Brovelli is collaborating with us in order to implement his Granger causality method within MuTE.

Appendices

A

Efforts in Disseminating MuTE

We are going to better explain how to set MuTE in order to perform several analyses. We will provide screenshoots and we will describe how to modify the code providing several examples of possible settings scenarios trying to cover the most common experimental set ups. Furthermore, the graphical user interface (GUI) is described. Its implementation will allow an easier and more flexible use of MuTE.

The following description can be found on the following website: `http://mutetoolbox.guru/`. We spent several weeks to build a website that is devoted to divulge MuTE in order to be a useful reference for whoever would like to study the methods and get used to the toolbox. The web page is thought not only to provide theoretical and practical explanations about MuTE, but also to be a reliable source of news concerning the most recent publications in the field of complex systems and information theory by means of a blog updated with the abstracts of the most interesting articles. Furthermore, we strongly encourage other scientists to integrate new methods in MuTE. Whoever feels like contributing in expanding MuTE will take the credits of his work by creating a personal page on the website where his curriculum and contribution to MuTE are shown.

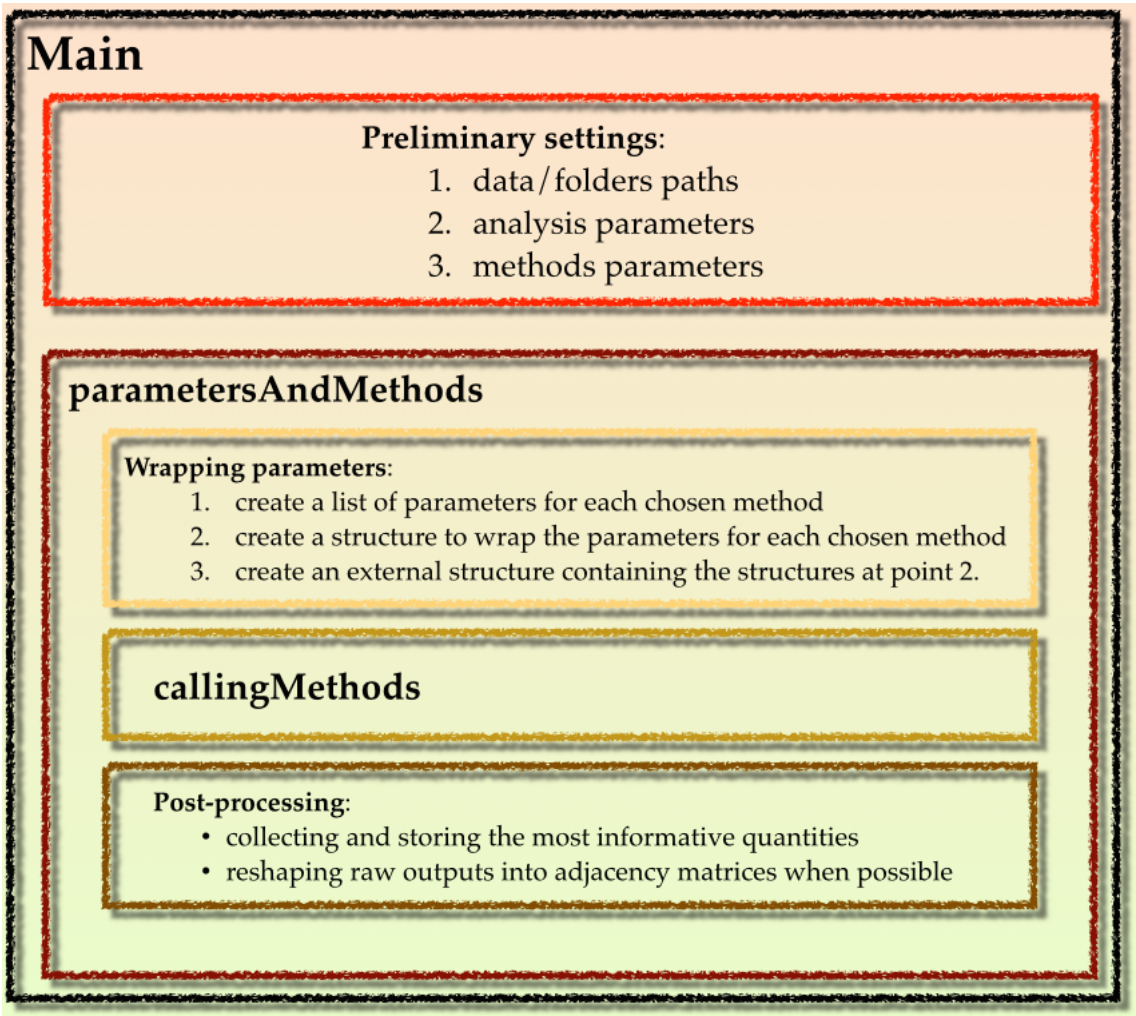

Figure A.1: Toolbox Structure.

A.1 Toolbox Structure

MuTE has a very simple structure as we can notice from A.1. The Main function allows the user to define the preliminary settings and to call the parametersAndMethods function. parametersAndMethods simply wraps the parameters defined in Main, calls the callingMethods function and calls the post precessing functions responsible to store the most informative quantities. In A.1 the schematic representation of the main toolbox's parts and their relationship are shown.

In the following, the toolbox structure is illustrated in detail, step by step, in

order to allow every user to use MuTE. The choice to show the **function names in bold** is aimed to let the user keep track of the most important functions.

NB:

- From now on the user is kindly asked to look at MuTE/exampleToolbox/exampleMain.m function
- Parts of code will be showed either in blue boxes according to what is the current set up or in green providing some examples of possible set up
- The % symbol is used for commented lines

A.2 Prerequisites

1. **Data** should be stored as files containing the field data. data should be a matrix of (number of series $\times$ time points) dimensions. If the user wants to perform an analysis across many trials it is possible to store data as 3D matrices of (number of variables $\times$ number of points $\times$ number of trials) dimensions. It is possible to find an example of data files in MuTE/exampleToolbox/.
2. **Create a folder** in which the data are stored. Let us call this folder as dataFolder from now on.

A.3 Main

The Main script is the only one that should be modified by the user. The analyses can be set outside the specific functions. The examples will show how Main can be customized according to the user needs. The Main gives the opportunity to set the methods parameters, where to read the data, where to store the results. It is also possible to concatenate experiments in order to run Main only once. As soon as the set up is made, MuTE is ready to run and the user should not give any further contribution until the end of the experiments.

A.4 Preliminary Settings

A.4.1 Some Useful Comments

At the very beginning of exampleMain there are some useful comments. First of all we can find the method order.

1: line 5: % Method order: please take the order into account because afterwards
2: line 6: % you should set autoPairwiseTarDriv or handPairwiseTarDriv that need
3: line 7: % the precise order of the methods
4: line 8:
5: line 9: binue
6: line 10: binnue
7: line 11: linue
8: line 12: linnue
9: line 13: nnue
10: line 14: nnnue
11: line 15: neunetue
12: line 16: neunetnue

Afterwards the parameters for each method are listed. We explain them in detail in parametersAndMethods.

A.4.2 Folders Set Up

The folder set up is the first step to take into account. It is necessary to add MuTE to MATLAB path as follows:

1: line 185: % Set MuTE folder path including also all the subfolders, for instance
2: line 186: mutePath = ('/Users/alessandromontalto/Dropbox/MuTE/');
3: line 187: cd(mutePath);
4: line 188: addpath(genpath(pwd));

Change according to where you decide to store MuTE

1: % Set MuTE folder path including also all the subfolders, for instance

2: mutePath = '/Users/.../Desktop/MuTE/';
3: % Adjust according to your path → just an example: mutePath = '/home/alessandro/Desktop/MuTE/';
4: cd(mutePath);
5: addpath(genpath(pwd));

Then, the user should set up the folder where the data are stored.

1: line 190: nameDataDir = 'exampleToolbox/'
2: % Set the directory in which the data files are stored. In this directory the outcome of the experiments will be stored too.
3: dataDir = ['/Users/alessandromontalto/Dropbox/MuTE/' nameDataDir];

Change according to where your data are stored

1: nameDataDir = 'folder containing your data/'
2: % Set the directory in which the data files are stored. In this directory the outcome of the experiments will be stored too.
3: dataDir = ['/Users/.../folder containing your data/'];
4: % Adjust according to your path → just as example: dataDir = ['/home/alessandro/Dropbox/Phd/MuTE/' nameDaraDir];
5: cd(mutePath);
6: addpath(genpath(pwd));

A.4.3 Number of Processors

Set the number of processors you can use to run your experiments changing value to the following variable

1: line 197: numProcessors = 1;

A.4.4 Loading Data

At this point two parameters should be set to load the correct data files:

- **dataFileName** takes the part of the name file common to all the files involved in the analysis.

- **dataLabel** is useful to distinguish files. If no label is required set dataLabel as an empty string.
- **dataExtension** defines the file extension

Variable needed to load the data

```
1: line 207: % Defining the strings to load the data files
2: line 208: dataFileName = 'realization5000p';
3: line 209: dataLabel = '';
4: line 210: dataExtension = '.mat';
```

EXAMPLE 1

We have 100 trials named, for instance, realizations5000p(...).mat where (...) can be a counter ranging from 1 to 100. In this case we do not to set dataLabel.

```
1: line 207: % Defining the strings to load the data files
2: line 208: dataFileName = 'realization5000p';
3: line 209: dataLabel = '';
4: line 210: dataExtension = '.mat';
```

EXAMPLE 2

We have 100 trials named, for instance, henonMaps_circular(...).mat and other 100 trials named henonMaps_sparse(...).mat where (...) can be a counter ranging from 1 to 100. In this case we would set the parameters as follows to perform the analysis taking into account henonMaps_sparse(...).mat.

```
1: line 207: % Defining the strings to load the data files
2: line 208: dataFileName = 'henonMaps';
3: line 209: dataLabel = 'sparse';
4: line 210: dataExtension = '.mat';
```

A.4.5 General Parameters Set Up

Here we are going to describe the more general parameters useful to handle the methods:

- **channels** is useful to select a subset of variables for a certain analysis. It is highly recommended to enter the series id from sorted in ascending order

from left to right.

- **samplingRate** should be greater than 1 if data should be downsampled
- **pointsToDiscard** allows to discard a certain amount of points, starting from the last one. To better clarify **samplingRate** and **pointsToDiscard**, the meaningful part of line 315 in parametersAndMethods function is shown: data(channels,1:samplingRate:(end-pointsToDiscard))
- **realization** may be left unchanged because it stores the files defined in dataFileName, dataLabel and dataExtension. It is worth taking a look at line 244 in exampleMain to get how realization is set: realization = dir([dataDir [dataFileName '*' dataLabel '*' dataExtension]]);
- **autoPairwiseTarDriv** takes into account the opportunity to investigate all the pair wise combinations of the variables chosen by means of channels. autoPairwiseTarDriv is then a vector with either 0 or 1 entries. The methods order shown at the beginning of this section has to be preserved.
- **handPairwiseTarDriv** is useful to when the user already know how many targets is going to choose for the analysis. If the number of targets can be reshaped in a square matrix then it is worth setting as 1 the entriy of handPairwiseTarDriv corresponding to the method choosen. handPairwiseTarDriv is then a vector with either 0 or 1 entries. The methods order shown at the beginning of this section has to be preserved.

Variables useful to handle data

1: line 240: % Defining the string to get data files
2: line 241: channels = 1:5;
3: line 242: samplingRate = 1;
4: line 243: pointsToDiscard = 4500;
5: line 244: realization = dir([dataDir [dataFileName '*' dataLabel '*' dataExtension]]);
6: line 245: autoPairwiseTarDriv = [0 1 0 1 0 1];
7: line 246: handPairwiseTarDriv = [0 0 0 0 0 0];

EXAMPLE 1

Let's say that we want to perform an analysis on henon*.mat files in MuTE/exampleToolbox/. Those files contain a matrix called data with 6 time series of 2500 points. For instance, we want to analyse all the pair wise combinations taking into account only 3 variable out of the 6 available: variables 2, 5, 6. We also want to consider the first 2000 points out of the 2500 available. Finally, we only want to investigate binnue and nnnue performances. In this case we should set the parameters as follows:

1: line 240: % Defining the experiment parameters
2: line 241: channels = [2 5 6];
3: line 242: samplingRate = 1;
4: line 243: pointsToDiscard = 500;
5: line 244: realization = dir([dataDir [dataFileName '*' dataLabel '*' dataExtension]]);
6: line 245: autoPairwiseTarDriv = [0 1 0 0 0 1];
7: line 246: handPairwiseTarDriv = [0 0 0 0 0 0];

A

EXAMPLE 2

Taking into account the same data as "Example 1", we do not want to analyse all the pairwise combinations, but we already know that we are going to choose a number of targets that can be reshaped in a square and that will involve all the variables. We want to downsample the time series at the half of the sampling rate and we want to take into account all the time points. This time we want to investigate binue, linue and linnue performances.

1: line 240: % Defining the experiment parameters
2: line 241: channels = [1:6];
3: line 242: samplingRate = 2;
4: line 243: pointsToDiscard = 0;
5: line 244: realization = dir([dataDir [dataFileName '*' dataLabel '*' dataExtension]]);
6: line 245: autoPairwiseTarDriv = [0 0 0 0 0 0];
7: line 246: handPairwiseTarDriv = [1 0 1 1 0 0];

```
%% STATISTICAL METHODS

fprintf('\n*******************************\n\n');
disp('Computing statistical methods...');
fprintf('\n\n');

tic
[output1,params1]               = parametersAndMethods(listRealization,samplingRate,pointsToDiscard,channels,autoPairwiseTarDriv,...
                                  handPairwiseTarDriv,resultDir,dataDir,copyDir,numProcessors,...
                                  'linue',[],[],[],5,'multiv',5,5,'bayesian',@linearEntropy,[1 0],[1 1],@generateConditionalTerm,0,...
                                  'linnue',[],[],[],5,'multiv',@evaluateLinearNonUniformEntropy,[1 1],100,0.05,@generateConditionalTerm,0,...
                                  'binue',[],[],[],5,'multiv',6,@conditionalEntropy,@quantization,[1 0],[1 1],100,0.05,20,...
                                  @generateConditionalTerm,0,...
                                  'binnue',[],[],[],5,'multiv',6,@evaluateNonUniformEntropy,@quantization,[1 1],100,0.05,@generateConditionalTerm,0,0,...
                                  'neunetue',[],[],[],5,[1 1],'multiv',[],[],{@sigmoid @identity},30,0,4000,2/3,15,...
                                  valThreshold,@resilientBackPropagation,1.1,0.9,1,numHiddenNodes,100,20,0.05,@generateConditionalTerm,0,...
                                  'neunetnue',[],[],[],[],5,[1 0],[1 1],'multiv',[],[],{@sigmoid @identity},30,0,4000,threshold,2/3,15,...
                                  valThreshold,@resilientBackPropagation,1.1,0.9,1,numHiddenNodes,@generateConditionalTerm,1,...
                                  'nnue',[],[],[],5,'multiv',[1 1],100,'maximum',10,nnMexa64Path,mutePath,0.05,10,@generateConditionalTerm,0,...
                                  'nnnue',[],[],[],5,'multiv',[1 1],100,'maximum',10,@nearNeiConditionalMutualInformation,...
                                  @evalNearNeiTestSurrogates2rand,nnMexa64Path,mutePath,0.05,@generateConditionalTerm,0);
```

A.5 Parameters And Methods

parametersAndMethods is the only function in the main script. It takes in input the parameters seen so far and the list of the methods with their own parameters. Inside parametersAndMethods the methods parameters are rearranged in only one structure used by other functions handle the proper parameters and perform the correct analyses.

It is worth seeing how parametersAndMethods is called in the exampleMain.

In the following the screenshot of the call to parametersAndMethods.

It is useful, then, to take a look in detail at the parameters that the user should set for each method. The parameters can be devided into two big groups: the common parameters that have to be set up for each of the 6 methods and the particular parameters that are useful to a particulr method only. We will provide some examples in order to make the meaning of the parameters as clear as possible.

A.5.1 Common Parameters

1. **idTargets** is a row vector with entries the series id chosen as targets of the analysis. It may contain repeated indeces.

2. **idDrivers** is a matrix with columns the series id whose influence on the corresponding target column wise in idTarget is going to be evaluated.

3. **idOtherLagZero** is a matrix with columns the series id chosen among the conditioning variables for that analysis that should be taken into account with the instantaneous effect.

4. **modelOrder** can be a vector indicating how many past states should be taken into account for each series chosen for the analisys by means of channels. If modelOrder is an integer the same amount of past states is considered for all the series in channels.

5. **multi_bivAnalysis** is a string containing either "biv" or "multiv" according to whether perform a bivariate or multivariate analysis respectively.

6. **genCondTermFun** is a pointer to function assuming either "@generateCondTermLagZero" or "@generateConditionalTerm" values. The first function, generateCondTermLagZero, takes into account the instantaneous effect for the drivers and for the variables indicated in idOtherLagZero. The second function, generateConditionalTerm, will not take into account the instantaneous effect, even if are explicitly mentioned in idOtherLagZero. This parameters should be set in accordance with usePresent.

7. **usePresent** is an integer assuming 1 or 0 values according to whether genCondTermFun is set to "@generateCondTermLagZero" or "@generateConditionalTerm" respectively.

EXAMPLE 1

Let us assume that we have a matrix of (10×3000) dimensions as data. We are interested in evaluating the adjacency matrix of the 10 variables: this means that we want to investigate all the pair wise combinations of the 10 time series taking into account all the time points. The implication is that we set idTargets and idDrivers as empy matrices.

Important: if we set an entry of autoPairWiseTarDriv as 1 idTargets and idDrivers of the correponding method will not be taken into account because autoPairWiseTarDriv has priority over idTargets and idDrivers.

Furthermore, we want a conditioned analysis taking 5 past states for all the variables and we do not want to consider the instananeous effects. We select binnue, linnue and nnnue methods. The parameters should be set as follows:

- channels = [1:10];
- samplingRate = 1;

- pointsToDiscard = 0;

...

- autoPairwiseTarDriv = [0 1 0 1 0 1];
- handPairwiseTarDriv = [0 0 0 0 0 0];
- idTargets = []
- idDrivers = []
- idOtherLagZero = []
- modelOrder = 5
- multi_bivAnalysis = 'multiv'
- genCondTermFun = @generateConditionalTerm
- usePresent = 0

EXAMPLE 2

Let us assume that we have a matrix of (10×3000) dimensions as data. We are interested, for instance, in evaluating the adjacency matrix of 4 variables out of 10.

Important: assuming that we are choosing series 2, 5, 8, 9. the time series chosen for the analysis will be extracted from the original matrix. In this way the user should keep into account that idTargets and idDrivers will assume values from 1 to channels length. In this case they will assume values ranging from 1 to 4.

Furthermore, let us assume that we do not want to investigate all the pair wise combinations. Instead, we only want to check whether there are information flows between certain driver-target couples and we need to have an idea about the directed dynamical links as soon as possible. Then we can use 500 points only. We also want a conditioned analysis taking 8 past states for all the variables and we want to consider the instantaneous effects of certain conditioning variables. We select binnue, linnue and nnnue methods. The parameters should be set as follows:

- channels = [2 5 8 9];

- samplingRate = 1;
- pointsToDiscard = 2500;

...

- autoPairwiseTarDriv = [0 0 0 0 0 0];
- handPairwiseTarDriv = [0 1 0 1 0 1];
- idTargets = [1 1 2 2 3 3 4 4 4]
- idDrivers = [2 4 1 3 1 4 1 2 3]
- idOtherLagZero = [3 0 0 0 2 0 0 1 2; 4 0 0 0 0 0 0 0 1]
- modelOrder = 8
- multi_bivAnalysis = 'multiv'
- genCondTermFun = @generateCondTermLagZero
- usePresent = 1

As we can notice, we chose 9 targets so we could set a 1 in handPairwiseTarDriv vector corresponding to the right method. To better explain the set up we can say that we are going to take into account the instantaneous effects of the variables 3 and 4 (corresponding to the 5th and the 8th time series of the original matrix) when investigating how 2 is influencing 1, conditioned to the other two variables. At this point the instantaneous effects for the drivers are not taken into account.

EXAMPLE 3

In this example we still want to deal with a matrix of (10×3000) dimensions as data. This time, we want a bivariate analysis including the instantaneous effects for the drivers. The procedure is slightly different: the id of the driver should be repeated. The explanation of the difference between the set up of the instantaneous effects for the drivers and for the conditioning variables is that the conditioning variables are taken into account setting multi_bivAnalysis = 'multiv' so it is easy to arrange the specific instantaneous effects in another vector. We only want to

use the first 5 time series without discarding any point. We want to use a vector to set a specific model order for each time series. Binue, linue and linnue will be the methods involved in the analysis.

- channels = [1:5];
- samplingRate = 1;
- pointsToDiscard = 0;

...

- autoPairwiseTarDriv = [0 0 0 0 0 0];
- handPairwiseTarDriv = [0 0 0 0 0 0];
- idTargets = [1 1 2 3 3 4 5]
- idDrivers = [2 4 3 1 4 2 3; 5 0 0 2 1 5 0]
- idOtherLagZero = []
- modelOrder = [4;3;5;3;3]
- multi_bivAnalysis = 'biv'
- genCondTermFun = @generateCondTermLagZero
- usePresent = 1

There are 7 targets so it is better to have handPairwiseTarDriv = [0 0 0 0 0 0] otherwise there will be an error when the results will be reshaped in a square matrix. idOtherLagZero will never be taken into account when multi_bivAnalysis = 'biv'.

EXAMPLE 4

Another example trying to combine the configurations that we have previously seen in Examples 1-3. We are assuming that we have a matrix of (10×3000) dimensions as data. We will first show the set up and then we will provide some comments about it.

- channels = [1 3 4 7 8];
- samplingRate = 3;
- pointsToDiscard = 100;

...

- autoPairwiseTarDriv = [0 0 0 0 0 0];
- handPairwiseTarDriv = [0 0 0 0 0 0];
- idTargets = [1 2 3 4 5]
- idDrivers = [2 3 4 1 2; 3 0 5 0 4; 4 0 2 0 0]
- idOtherLagZero = []
- modelOrder = 8
- multi_bivAnalysis = 'multiv'
- genCondTermFun = @generateCondTermLagZero
- usePresent = 1

We are performing an analysis taking into account series 1, 3, 4, 7 and 8 downsampled at one third of the original sampling rate. We are discarding the last 100 time points. We are investigating how:

1. variables 2, 3 and 4 are influencing 1 conditioned to 5. This is equivalent to say that we are interested in detecting the information flow from variables 3, 4 and 7 towards variable 1 conditioned to variable 8 of the original matrix. The same reasonining can be applied in the following;
2. variable 3 is influencing 2 conditioned to variables 1, 4 and 5;
3. variables 4, 5 and 2 are influencing 3 conditioned to 1;
4. variable 1 is influencing 4 conditioned to the variables 2, 3 and 5;
5. variables 2 and 4 are influencing 5 conditioned to 1 and 3.

A.5.2 Particular Parameters LIN UE

We embedded "arfit" in linue in order to allow the user to look for the best model order. For further references the user is kindly asked to read arfit documentation.

- **minOrder** → integer: lower bound of the interval in which arfit can look for the best model order able to fit the data
- **maxOrder** → integer: upper bound of the interval in which arfit can look for the best model order able to fit the data
- **orderCriterion** → string: order selection criterion for arfit. If orderCriterion is not set to 'bayesian', the Akaike's Final Prediction Error will be chosen as selection criterion
- **entropyFun** → pointer: pointer to the function that will evaluate the entropy
- **firstTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 0]). Set 1 according to wich candidates you want to take into account
- **secondTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 1]). Set 1 according to wich candidates you want to take into account

A.5.3 Particular Parameters LIN NUE

- **entropyFun** → pointer: pointer to the function that will evaluate the entropy
- **secondTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 1]). Set 1 according to wich candidates you want to take into account
- **numSurrogates** → integer: number of surrogates necessary to asses the statistical threshold
- **alphaPercentile** → integer: significance level

A.5.4 Particular Parameters BIN UE

- **numQuantLevels** → integer: number of quantum levels
- **entropyFun** → pointer: pointer to the function that will evaluate the entropy
- **preProcessingFun** → pointer: pointer to the function needed to pre-process of the data
- **firstTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 0]). Set 1 according to wich candidates you want to take into account
- **secondTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 1]). Set 1 according to wich candidates you want to take into account
- **numSurrogates** → integer: number of surrogates necessary to asses the statistical threshold
- **alphaPercentile** → integer: significance level
- **tauMin** → integer: number of shifts to produce a surrogate

A.5.5 Particular Parameters BIN NUE

- **numQuantLevels** → integer: number of quantum levels
- **entropyFun** → pointer: pointer to the function that will evaluate the entropy
- **preProcessingFun** → pointer: pointer to the function needed to pre-process of the data
- **secondTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 1]). Set 1 according to wich candidates you want to take into account
- **numSurrogates** → integer: number of surrogates necessary to asses the statistical threshold

- **alphaPercentile** → integer: significance level

A.5.6 Particular Parameters NN UE

- **firstTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 0]). Set 1 according to wich candidates you want to take into account
- **numSurrogates** → integer: number of surrogates necessary to asses the statistical threshold
- **metric** → string: it is possible to set the metric (either euclidian or maximum) used to evaluate the distance in the phase space
- **numNearNei** → integer: number of nearest neighbors to compute
- **funcDir** → string: mex files path
- **homeDir** → string: MuTE path
- **alphaPercentile** → integer: significance level
- **tauMin** → integer: number of shifts to produce a surrogate

A.5.7 Particular Parameters NN NUE

- **secondTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 1]). Set 1 according to wich candidates you want to take into account
- **numSurrogates** → integer: number of surrogates necessary to asses the statistical threshold
- **metric** → string: it is possible to set the metric (either euclidian or maximum) used to evaluate the distance in the phase space
- **numNearNei** → integer: number of nearest neighbors to compute

- **informationTransCriterionFun** → pointer: pointer to the function needed to evaluate the conditional mutual information;
- **surrogatesTestFun** → pointer: pointer to the function that performs the surrogates test;
- **funcDir** → string: mex files path
- **homeDir** → string: MuTE path
- **alphaPercentile** → integer: significance level

A.5.8 Particular Parameters NeuNet UE

- **secondTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 1]). Set 1 according to wich candidates you want to take into account
- **eta** [1] → real number: learning rate to update the weights with gradient descent and gradient descent with momentum
- **alpha** [2] → real number: parameter required in gradient descent with momentum [3]
- **actFunc** → cell array of pointers: it contains pointers to functions that are used as activation functions. There must be (number hidden layers + output layer) number of entries specifying the activation function for each layer
- **numEpochs** → integer: number of training epochs
- **bias** → integer: it allows to take into account the bias if it is set as 1. If bias nodes are not required, bias has to be set as 0

[1] This parameter is not useful to set the method appeared in literature. Nonetheless, it might be useful for further analyses considering a variation of the published method.

[2] This parameter is not useful to set the method appeared in literature. Nonetheless, it might be useful for further analyses considering a variation of the published method.

[3] For further references [78]

- **candidateEpochs** → integer: number of maximum iterations needed to train the network with the current candidate. It is used in the outer while of the non-uniform wrapper
- **dividingPoint** → real number: amount of points used to train the networks. It is expressed as percentage of the data set number of points
- **valStep** → integer: number of iterations after which the validation phase takes place
- **valThreshold** → real number: threshold needed during the validation phase
- **learnAlg** → pointer: points to the function used as learning algorithm
- **rbpIncrease** → real number: η^-, [44]
- **rbpDescrease** → real number: η^+, [44]
- **rangeW** → real number: it represents the range of values assumed by the weights when initialized. If rangeW is set as 1, the weights will be initialized between -1 and 1
- **coeffHidNodes** → real number: percentage of hidden nodes with respect to the amount of available candidates
- **numSurrogates** → integer: number of surrogates necessary to asses the statistical threshold
- **tauMin** → integer: number of shifts to produce a surrogate
- **alphaPercentile** → integer: significance level

A.5.9 Particular Parameters NeuNet NUE

- **data** → matrix: data arranged differently with respect to data used by the other methods. It might be useful to arrange data in order to line up the realizations that the other methods would analyze separately

- **firstTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 0]). Set 1 according to wich candidates you want to take into account
- **secondTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 1]). Set 1 according to wich candidates you want to take into account
- **eta** [4] → real number: learning rate to update the weights with gradient descent and gradient descent with momentum
- **alpha** [5] → real number: parameter required in gradient descent with momentum [6]
- **actFunc** → cell array of pointers: it contains pointers to functions that are used as activation functions. There must be (number hidden layers + output layer) number of entries specifying the activation function for each layer
- **numEpochs** → integer: number of training epochs
- **bias** → integer: it allows to take into account the bias if it is set as 1. If bias nodes are not required, bias has to be set as 0
- **candidateEpochs** → integer: number of maximum iterations needed to train the network with the current candidate. It is used in the outer while of the non-uniform wrapper
- **dividingPoint** → real number: amount of points used to train the networks. It is expressed as percentage of the data set number of points
- **valStep** → integer: number of iterations after which the validation phase takes place
- **valThreshold** → real number: threshold needed during the validation phase

[4] This parameter is not useful to set the method appeared in literature. Nonetheless, it might be useful for further analyses considering a variation of the published method.

[5] This parameter is not useful to set the method appeared in literature. Nonetheless, it might be useful for further analyses considering a variation of the published method.

[6] For further references [78]

- **learnAlg** → pointer: points to the function used as learning algorithm
- **rbpIncrease** → real number: η^-, [44]
- **rbpDescrease** → real number: η^+,[44]
- **rangeW** → real number: it represents the range of values assumed by the weights when initialized. If rangeW is set as 1, the weights will be initialized between -1 and 1
- **coeffHidNodes** → real number: percentage of hidden nodes with respect to the amount of available candidates

NB: we would strongly recommend to leave firstTermCaseVect and secondTermCaseVect as they are. For a more exhaustive documentation of the nearest neighbor parameters please take a look at the openTSTOOL documentation.

A.6 Wrapping Parameters

In parametersAndMethods lines 12-74 are devoted to convert the list of parameters given in input to a list of variables that will belong a particular method. Lines 180-288 are devoted to encapsulate the parameters of the chosen methods in a unique structure. This step is the needed bridge between the interface of MuTE with the user and the methods. To complete this step the function createNameMethodParams that locks the input parameters in the method structure is needed. After that the callingMethods is called.

A.7 Calling Methods

callingMethods function calls the chosen methods. The function only requires the more general data structure built during the wrapping of the parameters and the data. The output is a structure that groups the results of the methods.

A.8 Storing and Visualization of the Results

Three functions are involved in this step, called in parametersAndMethods:

- **storingOutput** → rearrange and store callingMethods output creating a different file for each realization and method. In this way each method will have its performances stored according to the analyzed file. Each file contains a structure named outputToStore. This is the most important post-processing function: starting from its output (the fields of outputToStore) the user can manage all the most useful informations in order to apply his own post-processing.
- **generateExpReport** → adds other fields to outputToStore structure such as pValue, modelOrder, testThreshold and bestOrder if those fields belong to the chosen method. The function, furthermore, generates the following files in entropyMatrices folder:
  - methodName_matrixPValues
  - methodName_testThresholdMtx
  - methodName_transferEntropyMtx
- **reshapeResults** → If either autoPairwiseTarDriv = 1 or handPairwiseTarDriv = 1 and the number of targets can be reshaped in a square, the function reshapes and stores the significance matrix and evaluates the mean values of the transfer entropy with respect to the number of trials obtaining a vector the same length as the number of targets. Then the mean transfer entropy values are reshaped and stored as well.

A.9 Output Returned

Let us take a closer look at the output files stored during the post-processing phase. We can find two folders where the outputs are stored, entropyMatrices and results folders.

In entropyMatrices the following files can be found:

- methodName_transferEntropyMtx contains a first column with the id of the data files. The other columns representing the entropy values: if the user sets allPairWiseCombinatins = 1 and he is dealing with 5 time series, he is going to evaluate 5 $\times$ 4 = 20 pairwise combinations not considering each series as target of itself. This means that the number of targets is 20. transferEntropyMtx will have 21 columns and as much rows as the number of trials.

- methodName_significanceOnDriv only contains as much columns as the number of targets and as much rows as the number of trials. Each entry may assume 1 or 0 values according to whether the link is significant or not.

- methodName_reshapedSignificance contains the significance values reshaped to form the adjacency matrix if the experimental set up allows the reshaping.

In results the following files are stored:

- methodName_typeAnalysis_patient_idTrial contains the outputToStore structure. This is the basic file from which everything can be evaluated during the post-processing phase.

- methodName_meanReshapeMtx is provided in results folder. This file contains the average of the values stored in transferEntropyMtx reshaped (if the experimental set up allows the reshaping) in order to show the adjacency matrix and be able to see the directed dynamical links.

NB: the user is kindly invited to develop his own functions to evaluate what may be the most useful informations according to his particular experiment. The post processing that we provide should be considered as an example of what can be evaluated.

A.10 The GUI

The MuTE GUI has been developed to make the MuTE toolbox more user-friendly. Please be aware that this tutorial only provides information on how to use the

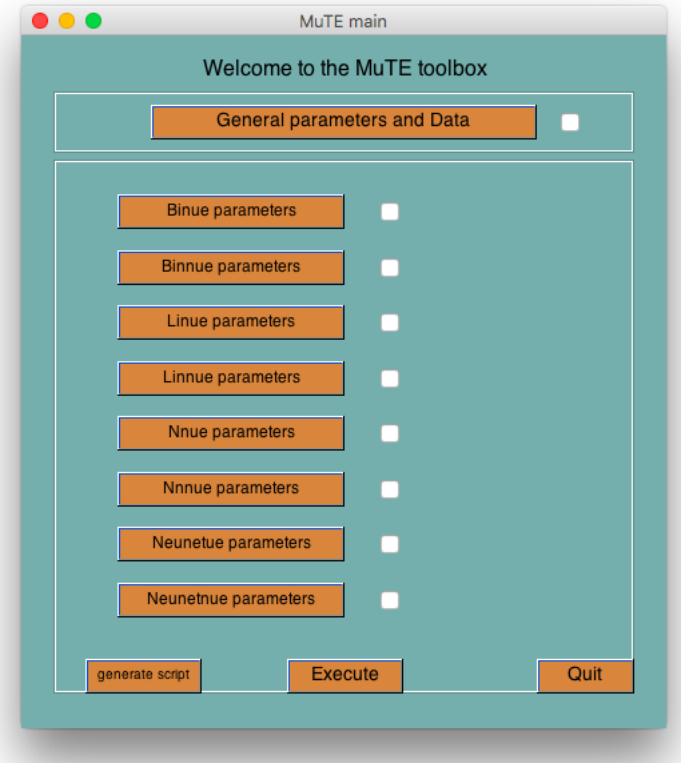

Figure A.2: MuTE GUI main window.

A

GUI. More information regarding the analysis methods can be found at the mute website and references therein `http://mutetoolbox.guru/`. It is strongly recommended to get familiarized with these methods before using the toolbox. In order to use the GUI, the latest version of the MuTE toolbox must be downloaded and added to your MATLAB path (add with subfolders). The most recent version of MuTE can be downloaded at the following address for free: `http://mutetoolbox.guru/downloads/`. To launch the GUI, type "mute" at the MATLAB command prompt. The following window will appear in figure A.2.

A.11 How to use the GUI

The work flow can be separated into 3 parts: 1) Setting the general parameters and data selection, 2) setting method specific parameters and 3) generate script/execute the analysis.

A.11.1 General Parameters and Data

The general parameters refer to those parameters that apply to all the methods used (i.e. the methods selected in the 2nd step). These can be set by clicking the 'General parameters and Data' button on top of the main window. A new window will appear where the data can be selected and general parameters can be set, figure A.3. In the new window, all the relevant parameter that need to be specified are listed. Most of the parameters have default values. In addition, every parameter is accompanied with a info ("?") button. More info regarding specific parameters can be obtained by clicking on the info button. Once all the relevant parameters are set, you can save the settings by clicking the 'Save settings' button. The default values can be restored by clicking the 'Restore default' button.

The GUI does an automatic check with respect to the selected data. The number of data files selected by the MuTE GUI will be displayed at the command prompt. If for some reasons the GUI is unable to find the data files, an error will pop up. If no data is found, it will be impossible to save the settings and thus also to generate a script/execute the analysis. Once the correct general parameters are saved, the general parameters window will be closed and the checkbox next to the 'general parameters and data' button in the main window will be checked.

As soon as the general parameters are saved, if everything is set up correctly the following warning is printed on the workspace in order to remind the user to wait until the computation is done and to not worry about the prompt shown in the workspace: "WARNING: the computation is taking place in another workspace. Please wait until '...COMPUTATION DONE!' is displayed."

A.11.2 Method-specific Parameters

In the middle of the main window, there are 8 buttons listed vertically. Each of these buttons will open a new window where you can specify method specific parameters, figure A.4.

The methods listed from top to bottom are the following:

- Binue: Binning uniform embedding
- Binnue: Binning non-uniform embedding

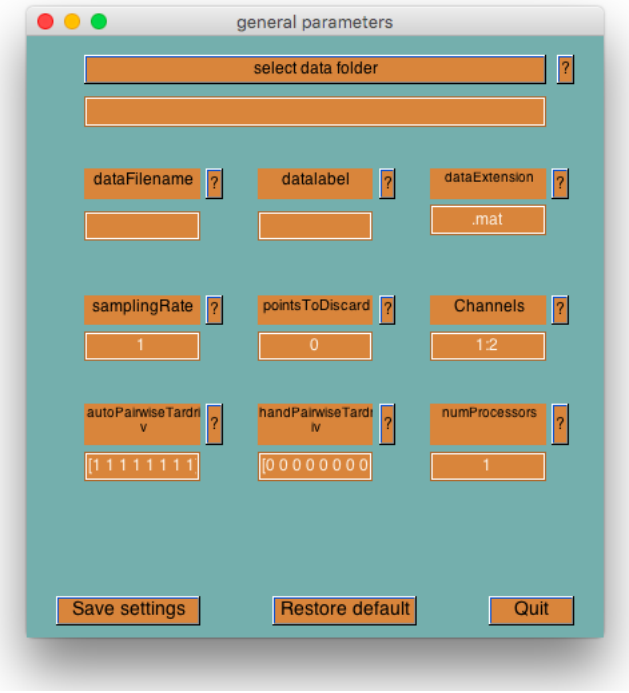

Figure A.3: The general parameters and data pop-up window.

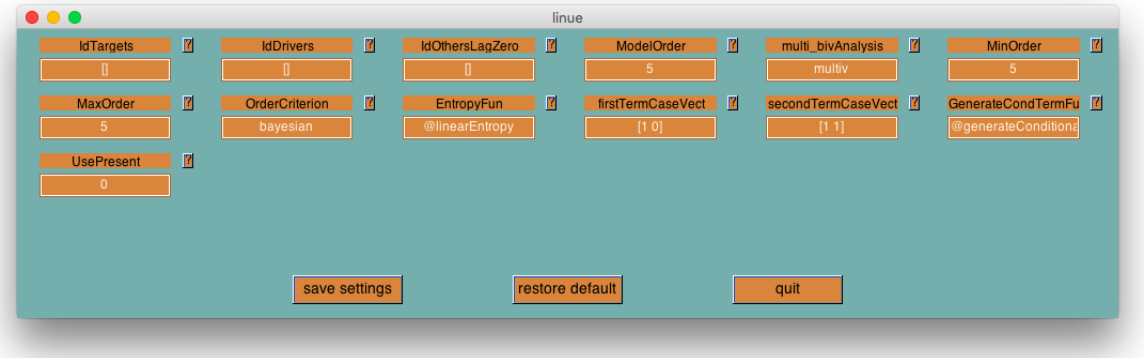

Figure A.4: An example of a method specific pop-up window. Here the method specific parameters can be specified. In figure, the linue method settings are displayed.

- Linue: Linear uniform embedding
- Linnue: Linear non-uniform embedding
- Nnue: Nearest neighbor, uniform embedding
- Nnnue: Nearest neighbor, non-uniform embedding
- Neunetue: Neural networks uniform embedding
- Neunetnue: Neural networks non-uniform embedding

The method specific window is similar to the general parameters window. All the parameters have default values. It is recommended to keep the default values for most parameters. Each parameter is again provided with a ("?") info button. The info button opens a window where more information regarding that specific parameter can be obtained. Once the settings are saved, the window will be closed and the checkbox next to the parameter will be checked. If one of the method specific checkboxes is unchecked, then that method will not be performed during the analysis.

A.11.3 Generate Script/Execute the Analysis

Once all the setting have been set, the user has two options: 1) generate a script or 2) execute the analysis. When clicking the "generate" button, a matlab scrip, a .m file with the date and hour of generation in the name e.g.

$$mute\_analysis\_2016\_1\_28\_9h\_56min.m$$

, will be generated in the data folder. The analysis can than be run by simply running the script. When clicking the "execute" button, the same script will be generated but now it will also be executed automatically. Our goal of generating a script is to make the transition from point-and- click to command line analysis easier. The user must specify at least one of the methods and the general parameters, otherwise an error will pop-up and no script will be generated/executed. The analyses are done once the main window is closed. The results can be found in a folder inside the data directory.

Acknowledgments

The GUI was developed in collaboration with Frederick Van de Steen who arranged the code. For further references about the GUI please do not hesitate to contact him at *frederik.VandeSteen@ugent.be* or send an email by filling in the form available at `http://mutetoolbox.guru/contacts/`.

CHAPTER 7

Personal Contributions

7.1 Papers

- In “Lag-Specific Transfer Entropy as a Tool to Assess Cardiovascular and Cardiorespiratory Information Transfer” [79] We performed some analyses useful for the application part of the paper.

- In “MuTE: A new Matlab toolbox for estimating the multivariate Transfer entropy in physiological variability series” [80] We implemented the first version of MuTE, performed the analyses and actively contributed to write the paper accepted at the Cardiovascular Oscillations (ESGCO), 2014 8th Conference of the European Study Group where I also presented a poster to show the paper outcomes.

- The paper “A linear approach for sparse coding by a two-layer neural network” [81] was published in collaboration with the University of Naples taking advantage of my Master’s Thesis. We contributed to the development of the paper with the implementation of the approaches presented in the work based on the neural network MATLAB package developed by the my supervisor in

Naples and his staff and which I helped to improve.

- "Comparing model-free and model-based transfer entropy estimators in cardiovascular variability" [82] was another application of the first version of MuTE accepted at the Computing in Cardiology Conference (CinC), 2013 where I gave an oral presentation of the paper. I contributed by taking care of the performed analyses and writing the paper.
- In "Cardiorespiratory Information Dynamics during Mental Arithmetic and Sustained Attention" [67] I contributed by helping the setting of the methods used to analyse the data and the interpretation of the results.
- In "Information dynamics in cardiorespiratory time series during mental stress testing" [83] I contributed by helping the setting of the methods used to analyse the data and the interpretation of the results.
- In "Information dynamics in cardiorespiratory analyses: Application to controlled breathing" [84] I contributed by helping the setting of the methods used to analyse the data and the interpretation of the results.
- In "Interictal cardiorespiratory variability in temporal lobe and absence epilepsy in childhood" [68] I contributed by helping the setting of the methods used to analyse the data and the interpretation of the results.
- Information decomposition of short-term cardiovascular and cardiorespiratory variability [85]
- In "MuTE: A MATLAB Toolbox to Compare Established and Novel Estimators of the Multivariate Transfer Entropy" [86], chapter 4, I was in charge of the setting of the experiments making the comparison between the methods as much clear as possible. I used some techniques usually adopted in classification tasks so that the pros and cons of the methods could be easily detected. I actively contributed to write the paper.
- "Neural networks with non-uniform embedding and explicit validation phase to assess Granger causality" [87], chapter 5, was completely thought, implemented, and developed by me. The co-authors gave a high impact contribution

in shaping the reasoning to that an organic framework could come out of my idea.

- In “Multiscale Analysis of Information Dynamics for Linear Multivariate Processes” [88], chapter 6, I actively worked with Dr. Luca Faes at the University of Trento in order to pave the way for the future article. I took care of the experiments too.

7.2 Conferences, Awards and Grants

- Computing in Cardiology Conference (CinC), 2013. I gave an oral presentation of the paper.
- ‘Workshop on Neural Information Dynamics, Causality and Computation near Criticality”, 2014. I presented MuTE.
- 8th European Study Group on Cardiovascular Oscillations (ESGCO 2014); Best Poster Prize
- FWO Grant For Long Stay Abroad

CHAPTER 8

Nederlandse samenvatting

Voor wetenschappers blijft het een uitdaging om de transfer van informatie tussen systeemcomponenten in kaart te brengen. Hierbij is het moeilijk om op basis van systemen, die gebruik maken van verschillende schalen, kennis over structuren en functie te infereren op basis van de geregistreerde dynamische gegevens. De componenten van complexe netwerken interageren namelijk vaak op een niet-lineaire manier en via mechanismen die algemeen genomen niet helemaal gekend zijn. In die gevallen is het veiliger dat de gekozen analysemethode, om deze interacties te bestuderen, niet op een model gebaseerd is en ook geen assumpties maakt over de natuur van de data en de interacties.

De bijdrage van deze dissertatie bestaat uit de ontwikkeling van methodes die in staat zijn om de dynamische invloeden beter te detecteren in dergelijke complexe systemen. In wat volgt wordt de inhoud van de 6 hoofdstukken kort toegelicht.

Hoofdstuk 2 is een introductie tot ”information theory” en haar onderliggende mathematische bouwstenen die doorheen dit werk gebruikt zullen worden. We leggen er eveneens de rationale uit die geleid heeft tot de formele en mathematische definitie van concepten zoals ”information” en ”entropy”. Vervolgens wordt de relatie tussen deze bouwstenen en de twee relevante maten uitgelegd. Deze maten

worden verder doorheen dit werk gebruikt.

In hoofdstuk 3 wordt de ”Multivariate Transfer Entropy toolbox” (MuTE) besproken. Hiernaast worden ook zes methoden geïmplementeerd. Er wordt voor elk van deze methoden een exhaustieve uitleg gegeven vanuit een theoretische en experimentele invalshoek. Hierbij wordt de performantie eveneens vergeleken. Tot slot worden ook de voor- en nadelen van elke methode besproken zodat de eindgebruiker beter op de hoogte is welke methode het beste aan zijn/haar noden tegemoet komt.

Hoofdstuk 4 is gewijd aan de beschrijving van de ”artificial neural networks” benadering. Aan de MuTE toolbox werden twee schattingsmethodes toegevoegd. We beschrijven kort deze benadering zodat een vergelijking kan gemaakt worden tussen ”machine learning” en ”information theory”. In dit hoofdstuk wordt de nadruk gelegd op de relevantie van de brug tussen ”machine learning” en ”information theory”. Onze hoop is dat toekomstig onderzoek de kloof tussen deze twee wetenschappelijke velden verder kan dichten.

In Hoofdstuk 5 wordt de ”multiscale” aanpak toegepast op een modelgebaseerde methode. We geven uitleg over de problemen die rijzen bij pre-processing technieken zoals filtering en ”down-sampling”. Er wordt een theoretisch bewijs geleverd voor de verschillen tussen enerzijds de signalen die ”preprocessing” ondergingen en anderzijds de originele signalen die deze bewerking niet ondergaan hebben. In dit hoofdstuk wordt ook een manier beschreven om problemen die gerelateerd zijn aan ”preprocessing” technieken aan te pakken.

In Hoofdstuk 6 wordt een overzicht en samenvatting gegeven van de belangrijkste ideeën en bevindingen van deze thesis. Het bevat bovendien een aantal paden en suggesties voor toekomstig onderzoek.

In Appendix A beschrijven we de verschillende manieren waarop de MUTE toolbox gepromoot werd met als doel om verder te reiken dan wat men typisch bereikt bij het publiceren van een wetenschappelijk artikel. Hiernaast wordt ook beschreven hoe de toolbox gebruiksvriendelijker gemaakt werd. Op die manier is de toolbox ook toegankelijk voor gebruikers zonder programmeerervaring.

Bibliography

[1] Ferdinand De Saussure. *Cours de linguistique générale: Édition critique*, volume 1. Otto Harrassowitz Verlag, 1989.

[2] Norbert Wiener. The theory of prediction, 1956.

[3] Clive WJ Granger. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica*, pages 424–438, 1969.

[4] Thomas Schreiber. Measuring information transfer. *Phys Rev Lett*, 85(2):461, 2000.

[5] L. Barnett, A.B. Barrett, and A.K. Seth. Granger causality and transfer entropy are equivalent for gaussian variables. *Phys Rev Lett*, 103(23):238701, 2009.

[6] Michael Wibral, Benjamin Rahm, Maria Rieder, Michael Lindner, Raul Vicente, and Jochen Kaiser. Transfer entropy in magnetoencephalographic data: Quantifying information flow in cortical and cerebellar networks. *Prog Biophys Mol Biol*, 105(1):80–97, 2011.

[7] Raul Vicente, Michael Wibral, Michael Lindner, and Gordon Pipa. Transfer entropy a model-free measure of effective connectivity for the neurosciences. *J Comput Neurosci*, 30(1):45–67, 2011.

[8] Vasily A Vakorin, Natasa Kovacevic, and Anthony R McIntosh. Exploring transient transfer entropy based on a group-wise ica decomposition of eeg data. *Neuroimage*, 49(2):1593–1600, 2010.

[9] Boris Gourévitch and Jos J Eggermont. Evaluating information transfer between auditory cortical neurons. *J Neurophysiol*, 97(3):2533–2543, 2007.

[10] Luca Faes, Giandomenico Nollo, and Alberto Porta. Information-based detection of nonlinear granger causality in multivariate processes via a nonuniform embedding technique. *Phys Rev E*, 83:051112, 2011.

[11] D. Kugiumtzis. Direct-coupling information measure from nonuniform embedding. *Phys Rev E*, 87:062918, Jun 2013.

[12] Anders Ledberg and Daniel Chicharro. Framework to study dynamic dependencies in networks of interacting processes. *Phys Rev E Stat Nonlin Soft Matter Phys*, 2012.

[13] Katerina Hlavácková-Schindler. Equivalence of granger causality and transfer entropy: A generalization. *App Math Sci*, 5(73):3637–3648, 2011.

[14] Ioannis Vlachos and Dimitris Kugiumtzis. Nonuniform state-space reconstruction and coupling detection. *Phys Rev E*, 82(1):016207, 2010.

[15] Katerina Hlaváčková-Schindler, Milan Paluš, Martin Vejmelka, and Joydeep Bhattacharya. Causality detection based on information-theoretic approaches in time series analysis. *Phys Rep*, 441(1):1–46, 2007.

[16] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Phys Rev E*, 69(6):066138, 2004.

[17] Adam B Barrett, Lionel Barnett, and Anil K Seth. Multivariate granger causality and generalized variance. *Phys Rev E*, 81(4):041907, 2010.

[18] Gideon Schwarz. Estimating the dimension of a model. *Ann Stat*, 6(2):461–464, 1978.

[19] R Quian Quiroga, A Kraskov, T Kreuz, and Peter Grassberger. Performance of different synchronization measures in real data: a case study on electroencephalographic signals. *Phys Rev E*, 65(4):041903, 2002.

[20] P.T. Brandt and J.T. Williams. *Multiple Time Series Models*. Number No. 148 in Multiple Time Series Models. SAGE Publications, 2007.

[21] Luca Faes, Alberto Porta, and Giandomenico Nollo. Mutual nonlinear prediction as a tool to evaluate coupling strength and directionality in bivariate time series: comparison among different strategies based on k nearest neighbors. *Phys Rev E*, 78(2):026201, 2008.

[22] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE Trans Inf Theory*, 13(1):21–27, 1967.

[23] Tapio Schneider and Arnold Neumaier. Algorithm 808: Arfit: a matlab package for the estimation of parameters and eigenmodes of multivariate autoregressive models. *ACM Trans Math Softw*, 27(1):58–65, 2001.

[24] Christian Merkwirth, Ulrich Parlitz, Immo Wedekind, David Engster, and Werner Lauterborn. Opentstool user manual. *Göttingen, Germany: Drittes Physikalisches Institut, Universität Göttingen*, 2009.

[25] Michael Wibral, Raul Vicente, Viola Priesemann, and Michael Lindner. Trentool: an open source toolbox to estimate neural directed interactions with transfer entropy. *BMC Neurosci*, 12(Suppl 1):P200, 2011.

[26] Daniele Marinazzo, Mario Pellicoro, and Sebastiano Stramaglia. Kernel method for nonlinear granger causality. *Phys Rev Lett*, 100(14):144103, 2008.

[27] Luiz A Baccalá and Koichi Sameshima. Partial directed coherence: a new concept in neural structure determination. *Biol Cybern*, 84(6):463–474, 2001.

[28] Mark A Kramer, Eric D Kolaczyk, and Heidi E Kirsch. Emergent network topology at seizure onset in humans. *Epilepsy Res*, 79(2):173–186, 2008.

[29] Luca Faes, Giandomenico Nollo, and Alberto Porta. Information domain approach to the investigation of cardio-vascular, cardio-pulmonary and vasculopulmonary causal couplings. *Front Physiol*, 2(80):80, 2011.

[30] Michael A Cohen and J Andrew Taylor. Short-term cardiovascular oscillations in man: measuring and modelling the physiologies. *Physiol J*, 542(3):669–683, 2002.

[31] Giandomenico Nollo, Luca Faes, Renzo Antolini, and Alberto Porta. Assessing causality in normal and impaired short-term cardiovascular regulation via nonlinear prediction methods. *Philos Trans A Math Phys Eng Sci*, 367(1892):1423–1440, 2009.

[32] Michael Wibral, Raul Vicente, and Joseph T Lizier. *Directed Information Measures in Neuroscience*. Springer, 2014.

[33] Koichi Sameshima and Luiz Antonio Baccala. *Methods in Brain Connectivity Inference through Multivariate Time Series Analysis*. CRC Press, 2014.

[34] C.W.J. Granger. Investigating causal relations by econometric models and cross-spectral methods. *Econometrica*, 3:424–438, 1969.

[35] Steven L Bressler and Anil K Seth. Wiener–granger causality: a well established methodology. *Neuroimage*, 58(2):323–329, 2011.

[36] David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group, editors. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. MIT Press, Cambridge, MA, USA, 1986.

[37] C.M. Bishop. Neural networks for pattern recognition. 1995.

[38] Luca Faes, Giandomenico Nollo, and Alberto Porta. Information-based detection of nonlinear granger causality in multivariate processes via a nonuniform embedding technique. *Phys. Rev. E*, 83:051112, May 2011.

[39] A Attanasio and U Triacca. Detecting human influence on climate using neural networks based granger causality. *Theoretical and Applied Climatology*, 103(1-2):103–107, 2011.

[40] Alessandro Montalto, Luca Faes, and Daniele Marinazzo. Mute: A matlab toolbox to compare established and novel estimators of the multivariate transfer entropy. *PloS one*, 9(10):e109462, 2014.

[41] Nicola Ancona, Daniele Marinazzo, and Sebastiano Stramaglia. Radial basis function approach to nonlinear granger causality of time series. *Phys. Rev. E*, 70:056221, Nov 2004.

[42] Daniele Marinazzo, Mario Pellicoro, and Sebastiano Stramaglia. Nonlinear parametric model for granger causality of time series. *Phys. Rev. E*, 73:066216, Jun 2006.

[43] Milan Paluš and Martin Vejmelka. Directionality of coupling from bivariate time series: How to avoid false causalities and missed connections. *Phys. Rev. E*, 75:056211, May 2007.

[44] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Neural Networks, 1993., IEEE International Conference on*, pages 586–591. IEEE, 1993.

[45] Sebastiano Stramaglia, Jesus Cortes, and Daniele Marinazzo. Synergy and redundancy in the granger causal analysis of dynamical networks. *NEW JOURNAL OF PHYSICS*, 16:18, 2014.

[46] Michael Wibral, Nicolae Pampu, Viola Priesemann, Felix Siebenhühner, Hannes Seiwert, Michael Lindner, Joseph T Lizier, and Raul Vicente. Measuring information-transfer delays. *PloS one*, 8(2):e55809, 2013.

[47] Li Deng and Dong Yu. Deep learning: Methods and applications. *Foundations and Trends in Signal Processing*, 7(34):197–387, 2013.

[48] P.Ch. Ivanov, L.A. Nunes Amaral, A.L. Goldberger, S. Havlin, M.G. Rosenblum, Z.R. Struzik, and H.E. Stanley. Multifractality in human heartbeat dynamics. *Nature*, 399(6735):461–465, 1999.

[49] X. Kang, X. Jia, R.G. Geocadin, and N. V. Thankor. Multiscale entropy analysis of eeg for assessment of post-cardiac arrest neurological recovery under hypothermia in rats. *IEEE Trans. Biomed. Eng.*, 5(4):1023–1030, 2009.

[50] L. Faes, A. Porta, and G. Nollo. Information decomposition in bivariate systems: Theory and application to cardiorespiratory dynamics. *Entropy*, 17(1):277–303, 2015.

[51] Madalena Costa, Ary L Goldberger, and C-K Peng. Multiscale entropy analysis of complex physiologic time series. *Phys. Rev. Lett.*, 89(6):068102, 2002.

[52] M. Lungarella, A. Pitti, and Y. Kuniyoshi. Information transfer at multiple scales. *Phys. Rev. E*, 76(5), 2007.

[53] L. Barnett and A.K. Seth. Behaviour of granger causality under filtering: Theoretical invariance and practical application. *J. Neurosci. Methods*, 201(2):404–419, 2011.

[54] J.F. Valencia, A. Porta, M. Vallverd, F. Clari, R. Baranowski, E. Orowska-Baranowska, and P. Caminal. Refined multiscale entropy: Application to 24-h holter recordings of heart period variability in healthy and aortic stenosis subjects. *IEEE Trans. Biomed. Eng.*, 56(9):2202–2213, 2009.

[55] Lionel Barnett and Anil K Seth. Granger causality for state-space models. *Phys. Rev. E*, 91(4):040101, 2015.

[56] Victor Solo. State space methods for granger-geweke causality measures. *arXiv preprint arXiv:1501.04663*, 2015.

[57] M. Aoki and A. Havenner. State space modeling of multiple time series. *Econ. Rev.*, 10(1):1–59, 1991.

[58] L. Faes, D. Kugiumtzis, G. Nollo, F. Jurysta, and D. Marinazzo. Estimating the decomposition of predictive information in multivariate systems. *Phys. Rev. E*, 91(3), 2015.

[59] Pierre-Olivier Amblard and Olivier JJ Michel. On directed information theory and granger causality graphs. *Journal of computational neuroscience*, 30(1):7–16, 2011.

[60] Mario Chávez, Jacques Martinerie, and Michel Le Van Quyen. Statistical assessment of nonlinear causality: application to epileptic eeg signals. *Journal of neuroscience methods*, 124(2):113–128, 2003.

[61] Matteo Garofalo, Thierry Nieus, Paolo Massobrio, and Sergio Martinoia. Evaluation of the performance of information theory-based methods and cross-correlation to estimate the functional connectivity in cortical networks. *PloS one*, 4(8):e6482, 2009.

[62] Niklas Lüdtke, Nikos K Logothetis, and Stefano Panzeri. Testing methodologies for the nonlinear analysis of causal relationships in neurovascular coupling. *Magnetic resonance imaging*, 28(8):1113–1119, 2010.

[63] Samuel A Neymotin, Kimberle M Jacobs, André A Fenton, and William W Lytton. Synaptic information transfer in computer models of neocortical columns. *Journal of computational neuroscience*, 30(1):69–84, 2011.

[64] Shivkumar Sabesan, Levi B Good, Konstantinos S Tsakalis, Andreas Spanias, David M Treiman, and Leon D Iasemidis. Information flow and application to epileptogenic focus localization from intracranial eeg. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on*, 17(3):244–253, 2009.

[65] Luca Faes and Giandomenico Nollo. Bivariate nonlinear prediction to quantify the strength of complex dynamical interactions in short-term cardiovascular variability. *Medical and Biological Engineering and Computing*, 44(5):383–392, 2006.

[66] Luca Faes, Giandomenico Nollo, and Alberto Porta. Non-uniform multivariate embedding to assess the information transfer in cardiovascular and cardiorespiratory variability series. *Computers in biology and medicine*, 42(3):290–297, 2012.

[67] Devy Widjaja, Alessandro Montalto, Elke Vlemincx, Daniele Marinazzo, Sabine Van Huffel, and Luca Faes. Cardiorespiratory information dynamics during mental arithmetic and sustained attention. *PLoS ONE*, 10(6):e0129112, 06 2015.

[68] Carolina Varon, Alessandro Montalto, Katrien Jansen, Lieven Lagae, Daniele Marinazzo, Luca Faes, and Sabine Van Huffel. Interictal cardiorespiratory variability in temporal lobe and absence epilepsy in childhood. *Physiological Measurement*, 36(4):845, 2015.

[69] Bernd Pompe and Jakob Runge. Momentary information transfer as a coupling measure of time series. *Physical Review E*, 83(5):051122, 2011.

[70] Joseph T Lizier, Jakob Heinzle, Annette Horstmann, John-Dylan Haynes, and Mikhail Prokopenko. Multivariate information-theoretic measures reveal directed information structure and task relevant changes in fmri connectivity. *Journal of Computational Neuroscience*, 30(1):85–107, 2011.

[71] Joseph T Lizier, Mikhail Prokopenko, and Albert Y Zomaya. Local information transfer as a spatiotemporal filter for complex systems. *Physical Review E*, 77(2):026110, 2008.

[72] Joseph T Lizier, Mikhail Prokopenko, and Albert Y Zomaya. Information modification and particle collisions in distributed computation. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 20(3):037109, 2010.

[73] Jinkyu Kim, Gunn Kim, Sungbae An, Young-Kyun Kwon, and Sungroh Yoon. Entropy-based analysis and bioinformatics-inspired integration of global economic information transfer. *PloS one*, 8(1):e51986, 2013.

[74] Okyu Kwon and J-S Yang. Information flow between stock indices. *EPL (Europhysics Letters)*, 82(6):68003, 2008.

[75] Joseph T Lizier. Jidt: An information-theoretic toolkit for studying the dynamics of complex systems. *arXiv preprint arXiv:1408.3270*, 2014.

[76] Lionel Barnett and Anil K Seth. The mvgc multivariate granger causality toolbox: a new approach to granger-causal inference. *Journal of neuroscience methods*, 223:50–68, 2014.

[77] Benoît Randuineau. *Interactions between pathgogens*. PhD thesis, Université Paris VI, 2015.

[78] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[79] L. Faes, D. Marinazzo, A. Montalto, and G. Nollo. Lag-specific transfer entropy as a tool to assess cardiovascular and cardiorespiratory information transfer. *Biomedical Engineering, IEEE Transactions on*, 61(10):2556–2568, Oct 2014.

[80] A. Montalto, L. Faes, and D. Marinazzo. Mute: A new matlab toolbox for estimating the multivariate transfer entropy in physiological variability series. In *Cardiovascular Oscillations (ESGCO), 2014 8th Conference of the European Study Group on*, pages 59–60, May 2014.

[81] Alessandro Montalto, Giovanni Tessitore, and Roberto Prevete. A linear approach for sparse coding by a two-layer neural network. *Neurocomputing*, 149, Part C:1315 – 1323, 2015.

[82] A. Montalto, D. Marinazzo, D. Kugiumtzis, G. Nollo, and L. Faes. Comparing model-free and model-based transfer entropy estimators in cardiovascular variability. In *Computing in Cardiology Conference (CinC), 2013*, pages 747–750, Sept 2013.

[83] D. Widjaja, A. Montalto, E. Vlemincx, D. Marinazzo, L. Faes, and S. Van Huffel. Information dynamics in cardiorespiratory time series during mental stress testing. In *Cardiovascular Oscillations (ESGCO), 2014 8th Conference of the European Study Group on*, pages 23–24, May 2014.

[84] D. Widjaja, L. Faes, A. Montalto, I. Van Diest, D. Marinazzo, and S. Van Huffel. Information dynamics in cardiorespiratory analyses: Application to controlled breathing. In *Engineering in Medicine and Biology Society (EMBC), 2014 36th Annual International Conference of the IEEE*, pages 6353–6356, Aug 2014.

[85] L. Faes, A. Montalto, G. Nollo, and D. Marinazzo. Information decomposition of short-term cardiovascular and cardiorespiratory variability. In *Computing in Cardiology Conference (CinC), 2013*, pages 113–116, Sept 2013.

[86] Alessandro Montalto, Luca Faes, and Daniele Marinazzo. Mute: A matlab toolbox to compare established and novel estimators of the multivariate transfer entropy. *PLoS ONE*, 9(10):e109462, 10 2014.

[87] Alessandro Montalto, Sebastiano Stramaglia, Luca Faes, Giovanni Tessitore, Roberto Prevete, and Daniele Marinazzo. Neural networks with non-uniform embedding and explicit validation phase to assess granger causality. *Neural Networks*, pages –, 2015.

[88] Luca Faes, Alessandro Montalto, Sebastiano Stramaglia, Giandomenico Nollo, and Daniele Marinazzo. Multiscale analysis of information dynamics for linear multivariate processes. *CoRR*, abs/1602.06155, 2016.