Pontificia Universidad Javeriana Cali
Faculty of Engineering & Sciences
Department of Electronics and Computer Sciences
Electronic Engineering
Undergraduate Final Project

DATA-DRIVEN SYSTEM IDENTIFICATION OF THE BARABÁSI–ALBERT MODEL

David Contreras Franco

Supervised by: Ph.D. Jorge Finke
Ph.D. Luis Eduardo Tobón

December 16, 2021

Contents

List of Abbreviations

AIC Akaike Information Criterion

BA Barabási–Albert

BIC Bayesian Information Criterion

CII Correct Individual Identification

COI Correct Overall Identification

DNN Deep Neural Network

ER Entropic Regression

ERR Error Reduction Ratio

FPE Final Prediction Error

FROLS Forward Regression Orthogonal Least Squares

GAT Graph Attention Network

GFRF Generalised Frequency Response Function

IHT Iterative Hard-Thresholding

LASSO Least Absolute Shrinkage and Selection Operator

LILC Law of Iterated Logarithm Criterion

LS Least Squares

MAE Mean Absolute Error

MBE Mean Bias Error

MISO Multiple-Input Single-Output

MPC Model Predictive Control

MSE Mean Squared Error

NARMAX Nonlinear AutoRegressive Moving Average with eXogenous input

NN Neural Network

ODE Ordinary Differential Equation

OLS Orthogonal Least Squares

OMP Orthogonal Matching Pursuit

PICND Parameter Identification of Complex Network Dynamics

RLS Recursive Least Squares

RMSE Root Mean Squared Error

RRSE Root Relative Squared Error

SINDy Sparse Identification of Nonlinear Dynamics

SISO Single-Input Single-Output

STLSQ Sequential Thresholded Least-Squares

Abstract

This work compares the Sparse Identification of Nonlinear Dynamics (SINDy) and Nonlinear AutoRegressive Moving Average with eXogenous input (NARMAX) methods, in the task of dynamics identification, on ODE systems, such as chaotic systems, nonlinear transfer functions with random input, and node degree evolution of the Barabási–Albert (BA) model with the extension of their capability to identify different equations between two nodes with different *fitness* values in a Bianconi-Barabási model. The comparison metrics selected are based on model representation, sparsity, estimation error, and time. The comparison shows an initial indication that SINDy finds sparser models, while NARMAX finds models with minimised estimation error.

Acknowledgement

First, I would like to express my gratitude towards my family; without their support, this work would not have been done in the time it did. I also extend my gratitude to my advisors, Jorge Finke and Luis Eduardo Tobón; their guidance and teachings helped improve this work and my desire to become a researcher and engineer.

Chapter 1

Introduction

There are numerous examples of interconnected systems: neurons that compose the brain through synapses, fish schools that swim in a synchronous movement to evade predators, ants that build underground networks to form a colony, and thousands of humans who interact to form societies. The collective behaviour of so-called complex systems cannot be explained by understanding the isolated behaviour of individual components.

The inability to fall back on reductionism creates a problem. The classical approach to model a system, reducing the whole to more straightforward explanations, is no longer feasible. Additionally, the classical approach requires that experts define a model and fit its parameters to the data. However, as systems increase complexity, this model tuning requires more and more time and resources. Consequently, novel approaches for system identification that automate this process are required.

The lofty aim of system identification methods for nonlinear systems is to facilitate, from available data, the estimation of nonlinear dynamics of a system. To better understand and interpret complex interconnected systems, this work proposes to identify the equations that govern the evolution of systems of interconnected entities based on a well-studied mathematical framework. In particular, we want to recreate identifying a system using data generated by the BA model, a stochastic network model with simple rules. The aim is to compare the performance of two different system identification methods in estimating the dynamics that result from the Barabási–Albert model.

Chapter 2

State of the Art

Data-driven dynamic identification in complex networks has gone through many contributions. Wang *et al.* proposed a method based on compressive sensing for predicting and reconstructing complex dynamical networks and showed that nonlinear node dynamics could be accurately predicted with near-zero error, with examples of dynamics only in the polynomial vector field [1]. Later, Wang *et al.* continued with the compressive sensing approach to reconstruct complex networks under game-based interactions, validated on different topologies, including the scale-free network [2]. From this, Ri-Qi *et al.* showed the capability of the compressive sensing methods to estimate collective dynamics that emerge from the complex network by starting with synchronisation [3].

Barzel and Barabási developed a formal universal framework to distinguish classes with varied characteristics based on their dynamics [4]. Moreover, later on, developed a method to infer the micro-dynamics of a complex network from macro observations of the network through the response to external perturbations, constructing a general nonlinear pairwise dynamics; from this subspace, there is no discrimination between all possible models [5]. At this stage, model interpretability has not been thoroughly discussed. Additionally, Quaranta *et al.* reviewed different data-driven methods as part of computational intelligence on nonlinear system identification for various models; relevant due to the nature behind the dynamics of complex networks [6].

Brunton *et al.* proposed an algorithm for SINDy, which approached the problem of system discovery as a sparse regression and compressed sensing problem [7]. Bakhtiarnia *et al.* proposed a modified version to discover the dynamics of social networks, removing the necessity of perturbations in the system for identification [8]. Later, the approach was expanded to the Parameter Identification of Complex Network Dynamics (PICND) algorithm, which combines the previous modified SINDy algorithm with a genetic algorithm approach [9].

Finally, AlRahman [10] developed the Entropic Regression (ER) Method as a sparse system identification method. This work is crucial since it was made with Complex Systems in mind, and it was compared to other methods considered for this work like SINDy; the method requires no prior information about the order of the system and proposes applications in complex networks and basis construction.

To the best of the author's knowledge, there is no further work regarding using data-driven methods to identify the dynamics of complex networks.

Chapter 3

Theoretical Background

3.1 Mathematical Model

Since this project aims to work with dynamics identification as a part of model discovery, it is crucial to know a couple of concepts and representations.

3.1.1 *A priori* Information

Any approach to a problem, in this case, system modelling, will start based on the information available; this is known as *A priori* information and means everything that is known. It is decisive because it guides the solution; the more *A priori* information, the clearer the system behaviour appears. Usually, for the analytical approach to system modelling, *A priori* information may represent the nature of the process under study from which to start selecting the most suitable set of equations.

Based on the knowledge available, two types of mathematical models can be determined: black box and white box models. These represent the opposites of the amount of knowledge available, black box meaning absolute no knowledge available and white box meaning all the necessary knowledge (to understand the system) available. A middle point is referred to as Gray Box models, and most systems lie in this category, as there is some information about it but not all of it.

3.1.2 Model Representation

Depending on the branch of science or engineering, different model representations are used to describe the system analysed. System identification methods commonly use the transfer function and state-space representation.

Transfer Function

A transfer function is the mathematical model of a specific output to all the possible inputs. It treats the system as a black box and attempts at thoroughly describing it using only in terms of inputs and outputs. It is more generally used for single-input single-output systems, although easily expandable.

State-Space Representation

State-space representation has a broader approach to the model description since it allows for multiple-input multiple-output systems. It considers the input, the output, and the inner state variables of the system related through functions, which are called dynamics and are the ones that define how the state of the system evolves as time passes.

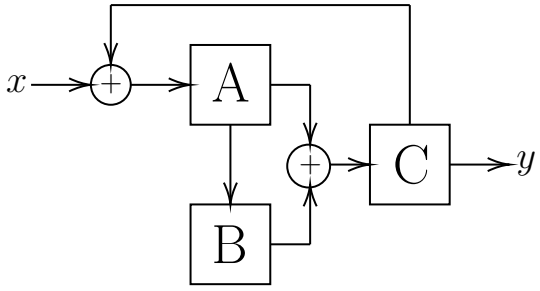


Figure 3.1: State-Space Representation example of a system with three state variables, an input x and output y, related by equations 3.1.

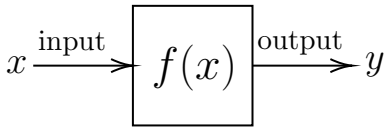


Figure 3.2: Transfer function representation of a system. Here the function f maps any input x to an output y.

$$\frac{dA}{dt} = x + C \qquad \frac{dB}{dt} = A \qquad \frac{dC}{dt} = A + B \tag{3.1}$$

3.1.3 Linearity

A linear system is that which satisfies the superposition principle, *i.e.* the system satisfies the additive and homogeneity properties. It is essential to describe it since the nonlinear systems negate this description.

Nonlinear Systems

A nonlinear system is one, which does not satisfy the superposition principle. Although the description is quite vague, it is needed since it comprises many varied systems. For this work, it is vital to keep in mind that in a nonlinear system, a change in the input does not reflect a proportional change in the output; likewise, it can be said that as a construction (like in a State Space Representation) the behaviour of the system cannot be reduced to the sum of its parts.

3.2 Complex Systems

A definition of "complex" may be a system that is difficult to understand or analyse [11], although a more precise definition is needed. The problem lies in that such definition, like

stated, is not as simple as shown by Wiesner *et al.* [12], since it involves and includes many different systems; as such, a tentative definition of a complex system is a system consisting of interconnected parts, which show an emergent behaviour. This definition requires clarification on interconnection and emergence.

Emergence describes properties of interactions that come into effect only when the parts interact as a whole. The components of the system have individual properties and system properties, or in simpler words, the system is not a result of its parts. This definition does not allow for reductionism, a common assumption to classical analytical approaches to system modelling. The other component is interconnection, which is closely related to emergence. By interconnection, it expresses the necessity for interaction and feedback between the system's parts.

An example of these properties is a school of fishes. Each fish that composes the school has its decision process; alone, one may observe its movement and deduce its operation. However, in a group, each fish interacts with its neighbours, staying close enough to not stand out to lurking predators and far enough not to crash with its neighbours. The behaviour of the school of fishes as a whole can even show a highly ordered form [13].

3.2.1 Complex Networks

Complex networks are one of the many complex systems. Barabási stated that complex networks are at the heart of complex systems [14]; they represent the interconnection of many elements, relating the state and dynamics of many entities, and the scaling in random network models show an emergent behaviour [15]. This simplification enables the representation of the interactions in a complex system, simplified as a set of nodes and edges [16].

With this capability to represent complex systems, it is essential to clarify what constitutes a network. A network represents a system through nodes and edges that construct a specific topology. It is crucial to distinguish them from graphs since networks require meaning in the connections, the system it is representing. The complexity in a network may come from any of the components described; this work focuses on structural complexity [17].

A more specific group of complex networks is random networks, which describes networks in which a connection (edge) between a new node and each existing node has a specific probability. This work focuses on the Barabási–Albert model's specific random network model.

Chapter 4

Methods

Among the methods for consideration are all of those which can correctly identify the proper underlying behaviour of the system; this means that all the linear methods, piecewise linear approximations, and similar, are not considered. Most of the background on methods for nonlinear was extracted from [18] with support from [19].

The method ER reviewed as part of State of the Art appears ideal for this problem. It is an approach based on information theory in which entropy is used for both optimisation and regression, thus looking for the model with high information content according to an information criterion, as opposed to models that satisfy sparsity. However, the main advantage of ER lies in problems with significant noise and many outliers, both concepts outside this project's scope. Therefore, considering the complexity of this method, it will not be one of the methods for comparison.

4.1 Volterra Series

Since the Volterra series has been a popular model for representing nonlinear systems, it is appropriate to discuss it within the methods considered for this work. A Volterra Series describes a nonlinear causal signal, where the output can be described via M lagged input signals.

$$y(k) = h_0 + \sum_{\ell=1}^{L} y_\ell(k) \tag{4.1}$$

$$y_\ell(k) = \sum_{m_1=1}^{M} \cdots \sum_{m_\ell=1}^{M} h_\ell(m_1, \ldots, m_\ell) u(k-m_1) \cdots u(k-m_\ell) \tag{4.2}$$

Where $u(k)$ is the input and $y(k)$ is the output at instance $k \in \mathbb{N}$, and $h_\ell(m_1, \ldots, m_l)$ is the ℓ'th-order nonlinear impulse response, called the ℓ'th-order Volterra Kernel. This method will not be used due to the following challenges:

- The number of Volterra Kernels is required at the start of the identification; this may not be known *a priori*, and considering fewer kernels than needed will lead to incorrect identifications.

- The number of points needed for correct identification may be quite large due to it growing exponentially with the ℓ'th order of the series.

It is important to note that frequency-domain methods considered the Generalised Frequency Response Functions (GFRFs), defined as the multidimensional Fourier transform of the corresponding Volterra kernel in the time-domain, suffers from the same disadvantages.

4.2 Block-Structured Nonlinear Models

Block-structured models describe nonlinear systems by linear dynamics and static nonlinearities. The Hammerstein and Wiener models are the most well known basic block-structured models; from these, a general model is described with a static nonlinearity between two dynamic systems.

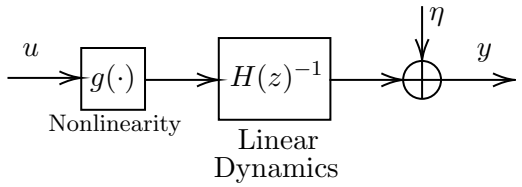


Figure 4.1: Hammerstein Model

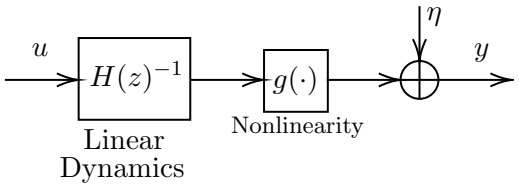


Figure 4.2: Wiener Model

Figure 4.3: Block-Structured Models

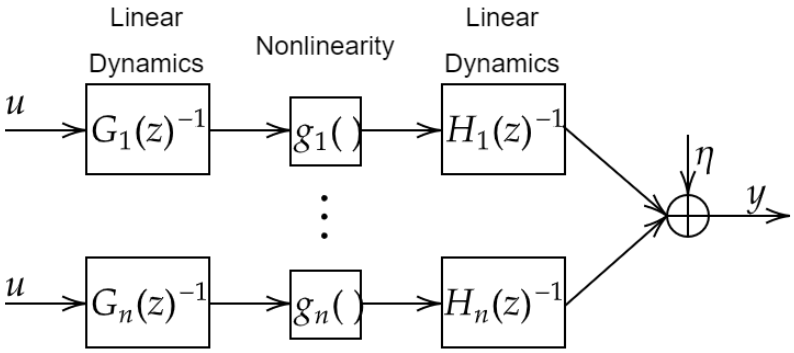


Figure 4.4: Parallel cascade general block-structured model with noise

Even though this model allows for great interpretability because individual components may be related to the underlying system, the estimation may be misguiding without sufficient knowledge. Additionally, these models can be described via Volterra Series.

4.3 NARMAX

NARMAX consists of a general case definition of the identification of a nonlinear system, defined by a function that depends only on previous input and output values and stochastic terms, where the goal is to find the simplest model possible that can be related to the underlying system, instead of the generalisation and estimation. The main idea is to build the underlying model by iteratively picking the most important nonlinear term from a library of candidate functions until the model is adequate and related to the underlying system.

$$y_k = F[y_{k-1}, \ldots, y_{k-n_y}, x_{k-d}, x_{k-d-1}, \ldots, x_{k-d-n_x} + e_{k-1}, \ldots, e_{k-n_e}] + e_k \tag{4.3}$$

Where $F[\cdot]$ is some nonlinear function of the input and output with a time delay d, $x_k \in \mathbb{R}^{n_x}$ is the input, $y_k \in \mathbb{R}^{n_y}$ is the output, and $e_k \in \mathbb{R}^{n_e}$ is the stochastic term at discrete time $k \in \mathbb{N}^n$, with $n_y, n_x, n_e \in \mathbb{N}$ as the maximum lags for the input, output, and noise. Given the general structure of the NARMAX model, the Volterra Series, Block-Structured models and many Neural Network architectures can be considered subsets of the NARMAX model. Subsequently, NARMAX as a method consists of several steps:

1. *Structure detection*: Identify which terms are in the model.
2. *Parameter estimation*: Define the coefficients of the identified terms.
3. *Model validation*: Evaluate the model performance.
4. *Prediction*: Use the identified model to predict future outputs.
5. *Analysis*: From the identified model, analyse the underlying dynamical properties of the system.

This work will only consider the steps until model validation. Additionally, the function $F[\cdot]$ will be described from a Polynomial basis of degree five, since this covers the systems from chapter 5; a Rational Basis is needed for the BA model, but this is solved with the Polynomial Basis with an adjustment detailed in section 8.2.1. The selection for the basis does not represent the limits of the NARMAX method; extreme nonlinear behaviours may be represented via a Bilinear, Rational, Radial basis functions, Wavelet Decomposition, among many possible basis functions, which NARMAX defines as the Extended Model Set, composed by a mix of different basis functions.

Therefore, it is evident that the critical step in system identification with the NARMAX methods is structure detection. This step is fundamentally achieved via the Orthogonal Least Squares (OLS) estimator in par with the Error Reduction Ratio (ERR) test. An extension of this is the Forward Regression Orthogonal Least Squares (FROLS) algorithm, which avoids ordering bias in the ERR and has become one of the most used algorithms for model structure detection. In the sense of sparse regression, it is paramount to note

that Orthogonal Matching Pursuit (OMP) is a slight variation of OLS. The final structure detection algorithm selects the most significant term via FROLS and is either added to the model or discarded with the ERR test until a threshold for the ERR values is achieved.

A significant advantage of the NARMAX model is that many existing nonlinear models, such as the Volterra Series and the Block-Structured, can be considered special cases of the NARMAX model.

4.3.1 SysIdentPy

SysIdentPy is the Python package that implements the NARMAX method, which further motivates the selection of this method. Since SysIdentPy implements a Polynomial NARMAX model and the FROLS algorithm, usage of this package is simplified. Least Squares (LS) and Recursive Least Squares (RLS) were used for the estimator. Regarding the information criterion, SysIdentPy has implemented: Akaike Information Criterion (AIC), Bayesian Information Criterion (BIC), Law of Iterated Logarithm Criterion (LILC), and Final Prediction Error (FPE); from which BIC was selected due to consistently resulting in sparser models during testing.

4.4 SINDy

The SINDy method considers the problem of model discovery from the compressed sensing perspective, whereas out of many candidate functions, only a few of them are needed to describe the dynamics; *ergo*, the terms of the governing dynamics equations are sparse in the nonlinear functions space. So, considering the vector $\mathbf{x}(t) \in \mathbb{R}^n$ represents the state of an n-dimensional system at time t or moment k for discrete-time systems, the dynamics are described by the function $\mathbf{f}$.

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t)) \qquad \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k) \tag{4.4}$$

To determine $\mathbf{f}$, from the states evolution through time matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$, the matrix $\dot{\mathbf{X}} \in \mathbb{R}^{n \times m}$ can either be measured or numerically defined from $\mathbf{X}$ when considering a continuous system with differential equations as dynamics; alternatively, a time delay may be applied to $\mathbf{X}$ for discrete systems with dynamics described through difference equations.

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^T(t_1) \\ \vdots \\ \mathbf{x}^T(t_m) \end{bmatrix} \qquad \mathbf{X}^{m-1} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_{m-1}^T \end{bmatrix} \tag{4.5}$$

$$\dot{\mathbf{X}} = \begin{bmatrix} \dot{\mathbf{x}}^T(t_1) \\ \vdots \\ \dot{\mathbf{x}}^T(t_m) \end{bmatrix} \qquad \mathbf{X}^{m} = \begin{bmatrix} \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_m^T \end{bmatrix} \tag{4.6}$$

The library of candidate functions $\mathcal{D}$ contains all possible basis functions that describe the dynamics, like Polynomial or Trigonometric basis. With each basis function $\theta_i \in \mathcal{D}$ evaluated on the matrix $\mathbf{X}$ as $\theta_i(\mathbf{X}) \in \mathbb{R}^{n \times m}$, the complete library $\Theta(\mathbf{X})$ describes all possible dynamics of $\mathbf{X}$ given $\mathcal{D}$.

$$\Theta(\mathbf{X}) = [\theta_1(\mathbf{X}) \dots \theta_i(\mathbf{X}) \dots] \tag{4.7}$$

Going back to the primary assumption that only a few terms from the candidate functions $\mathcal{D}$ are needed to describe the governing equations, the sparse vectors of coefficients $\Xi = [\xi_1 \dots \xi_i \dots]$ determine the basis functions present in the dynamics. Finally, the sparse regression problem can be defined in continuous and discrete-time, which may be solved through known algorithms like Least Absolute Shrinkage and Selection Operator (LASSO) or OMP. Additionally, noise may be considered as part of the sparse regression problem.

$$\dot{\mathbf{X}} = \Theta(\mathbf{X})\Xi \qquad \mathbf{X}^m = \Theta(\mathbf{X}^{m-1})\Xi \tag{4.8}$$

This method has shown to be quite robust and flexible in a wide range of applications [7], but mainly aims at interpretability and directly solves the main problem of this work.

4.4.1 PySINDy

An additional advantage of choosing this method is the Python package made by the original authors of the method [20]. Although it is pretty easy to use, it is evident that for a successful model discovery using the SINDy method, several decisions must be taken regarding selecting the differentiation method, the library of basis functions, and the optimiser, the algorithm that solves the sparse regression problem.

Regarding the differentiation method, only finite differences will be used for comparison to simplify the number of possible cases for testing. SINDy successfully identified all the systems from chapter 5 and the BA model with this differentiation method. The library of basis functions was restricted to a Polynomial basis of degree five, except for the custom basis function for the BA model as described in section 8.2.2.

Finally, the Sequential Thresholded Least-Squares (STLSQ) algorithm was used as an optimiser; this is due to it being the main one recommended by the authors of the original paper, given its computational efficiency and fast convergence. STLSQ is a type of Iterative Hard-Thresholding (IHT) algorithm, where a solution to Ξ is found with LS, and then the resulting coefficients are thresholded by a value λ with the element-wise hard-thresholding operator H_λ; which means that all values in Ξ that are smaller than λ are set to zero. From the resulting non-zero coefficients, a new solution is found with LS and thresholded again with λ; this is done until the number of coefficients after the thresholding step does not change.

$$H_\lambda(\Xi) = \begin{cases} 0 & \Xi_{ij} \leq \lambda \\ \Xi_{ij} & \Xi_{ij} > \lambda \end{cases} \tag{4.9}$$

$$\Xi^{(n+1)} = H_\lambda \left(L_s \left(\Theta(\mathbf{X}), \dot{\mathbf{X}}, \Xi^{(n)} \right) \right) \tag{4.10}$$

4.5 Neural Networks

Neural networks show excellent behaviour in estimating a system's behaviour, which raises the question of the necessity of models in the pipeline when entirely data-driven techniques already perform pretty well. The main problem lies in the actual interpretability of the system once the network can estimate it accurately; Neural Networks (NNs) are great machines at fitting and generalising based on any data. However, analysis on the resulting model is not viable since the network is not using the nature of the system to make its predictions but the fitting equivalence given the layers it contains.

An exciting approach to make NNs give interpretable results is AgentNet by Ha and Jeong [21]. It is inspired by Graph Attention Networks (GATs), with the regular supervised training of Deep Neural Networks (DNNs) for prediction and using the attention values as the model's interpretability. Although it shows excellent results, the current model is limited to pairwise interaction of the agents, significantly limiting the complexity and basis functions to be identified.

4.6 Summary

NARMAX is a general model that can represent other classical approaches to nonlinear system identification, and this work will compare it to the SINDy method due to its novelty, flexibility in candidate functions, and solution for multidimensional systems. Additionally, both methods aim for the simplest model through algorithms that favour sparsity. Finally, both methods have a Python package, SysIdentPy for NARMAX and PySINDy for SINDy.

Chapter 5

Systems

Given the selection of the methods and tools in chapter 4, the most direct approach considered to compare the performance was to use the base examples proposed by the authors and perform a cross-evaluation of both methods on each example. Evidently, besides the test systems, the BA model was tested, but due to its primary objective, this model will be detailed in chapter 6. All the systems were simulated, and the generated data was split into an initial set for training, identification, and subsequent set for testing the identified model.

Both methods presented limitations based on the underlying representation each method considered: SINDy a state-space representation and NARMAX a transfer function, which means that NARMAX is limited to a single-output system while SINDy is not. On the other hand, NARMAX is adept at identifying time-delays in its input or output, as long as such *a priori* information is given for the model discovery, while SINDy considers only the previous input. Both methods should be used regardless of their limitations; suitable solutions are detailed in sections 8.2.1 and 8.2.2.

5.1 Differential Equation Systems

Most of the capability of SINDy is shown by identifying the Ordinary Differential Equations (ODEs) that describe the dynamics of a system given its state-space representation. Since all these systems are ODE, the simulation consisted only of the integration of the system given some initial conditions and time samples separated by some time difference dt, the package SciPy helped in this manner [22]. The following examples were extracted from the extensive documentation for the package PySINDy [20].

1. Linear 2D ODE from equations 5.1 in a time range $[0, 25]$ with $dt = 0.01$ and a test size of 75% of the generated data.
2. Cubic 2D ODE from equations 5.2 in a time range $[0, 25]$ with $dt = 0.001$ and a test size of 75% of the generated data.
3. Linear 3D ODE from equations 5.3 in a time range $[0, 50]$ with $dt = 0.01$ and a test size of 60% of the generated data.
4. Lorenz System ODE from equations 5.4 with $\sigma = 10$, $\rho = 28$, and $\beta = \frac{8}{3}$, in a time range $[0, 25]$ with $dt = 5e - 4$ and a test size of 75% of the generated data.

Besides the examples used, PySINDy explored more complex chaotic systems, but this sample already gives an insight into the proposed applications. An additional system of ODEs is the Rössler Attractor, defined by equations 5.5 with $\alpha = 0.2$, $\beta = 0.2$, and $\gamma = 5.7$, in a time range $[0, 50]$ with $dt = 1e - 2$ and a test size of 50% of the generated data.

$$\begin{aligned} \dot{x} &= -0.1x + 2y \\ \dot{y} &= -2x - 0.1y \end{aligned} \tag{5.1}$$

$$\begin{aligned} \dot{x} &= -0.1x^3 + 2y^3 \\ \dot{y} &= -2x^3 - 0.1y^3 \end{aligned} \tag{5.2}$$

$$\begin{aligned} \dot{x} &= -0.1x + 2y \\ \dot{y} &= -2x - 0.1y \\ \dot{z} &= -0.3z \end{aligned} \tag{5.3}$$

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= x(\rho - z) - y \\ \dot{z} &= xy - \beta z \end{aligned} \tag{5.4}$$

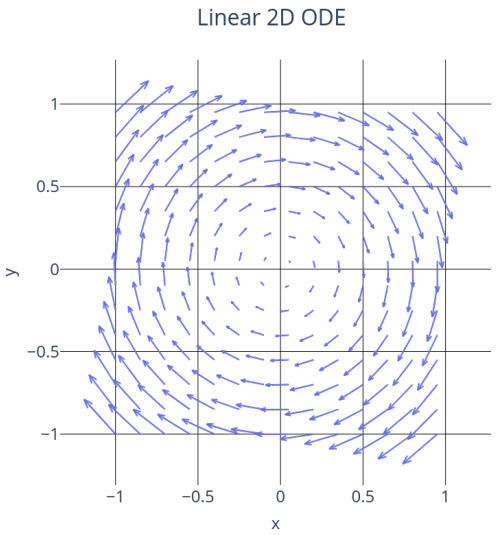
$$\begin{aligned} \dot{x} &= -y - z \\ \dot{y} &= x + \alpha y \\ \dot{z} &= \beta + z(x - \gamma) \end{aligned} \tag{5.5}$$

5.2 Nonlinear Functions

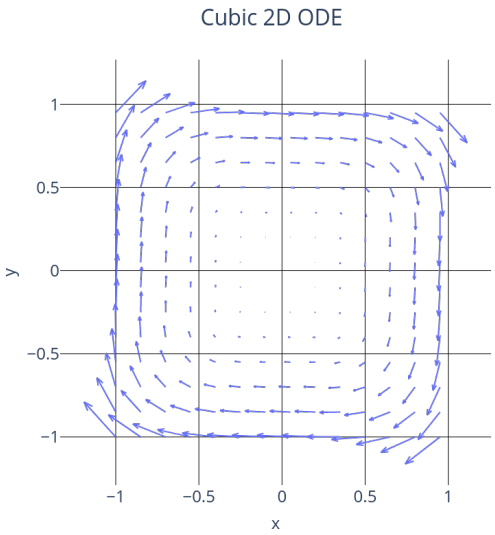
SysIdentPy examples are the Single-Input Single-Output (SISO) function from equation 5.6 and the Multiple-Input Single-Output (MISO) function from equation 5.7, both with a uniformly distributed random variable x as input and some added white noise e to the output [23]. Also, in both cases, 1000 samples were generated, from which 10% was used for testing the identified model. The equations have been rewritten to show time delays more evidently for clearness.

$$y[k+1] = 0.2y[k] + 0.1y[k]x[k] + 0.9x[k-1] + e[k] \tag{5.6}$$

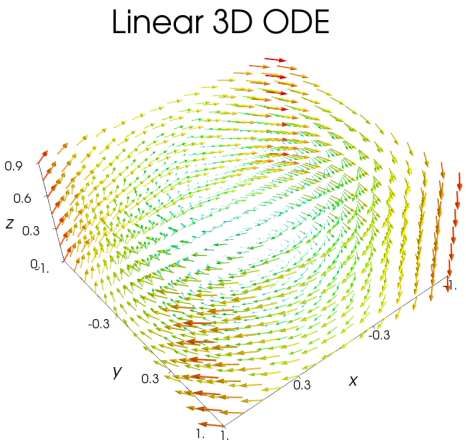
$$y[k+1] = 0.4y^2[k] + 0.1y[k]x_1[k] + 0.6x_2[k] - 0.3x_1[k]x_2[k-1] + e[k] \tag{5.7}$$



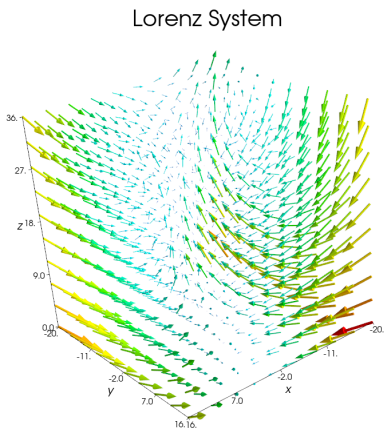
(a) Vector Field for equations 5.1



(b) Vector Field for equations 5.2

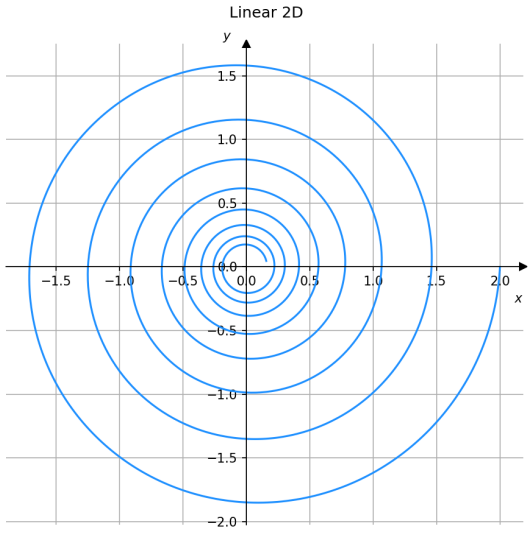


(c) Vector Field for equations 5.3

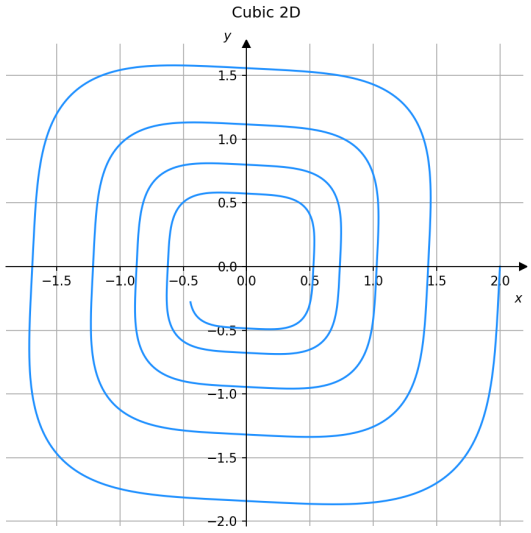


(d) Vector Field for equations 5.4

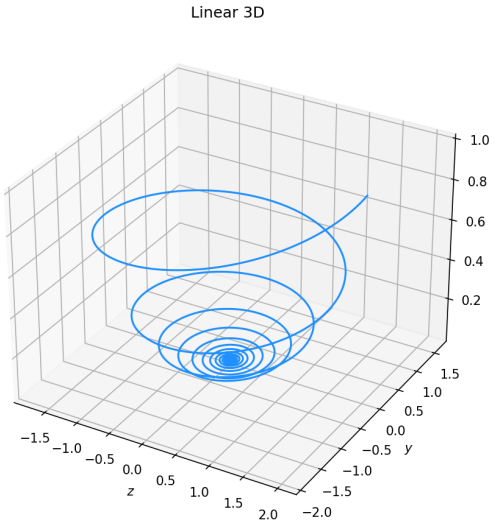
Figure 5.1: ODEs Vector Fields



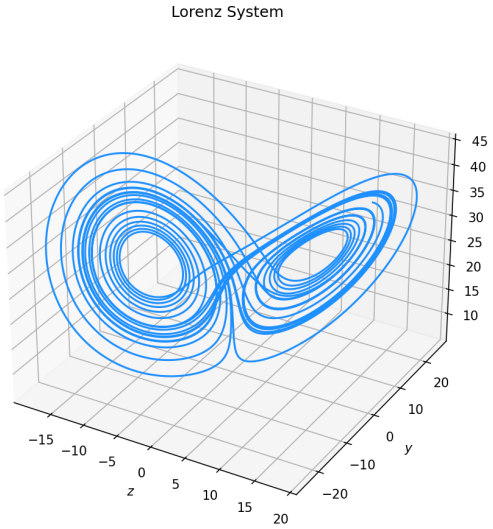
(a) Simulation for equations 5.1 with initial state $[2, 0]$.



(b) Simulation for equations 5.2 with initial state $[2, 0]$.

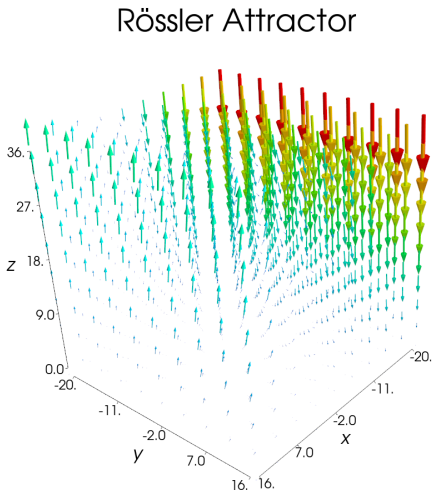


(c) Simulation for equations 5.3 with initial state $[2, 0, 1]$.

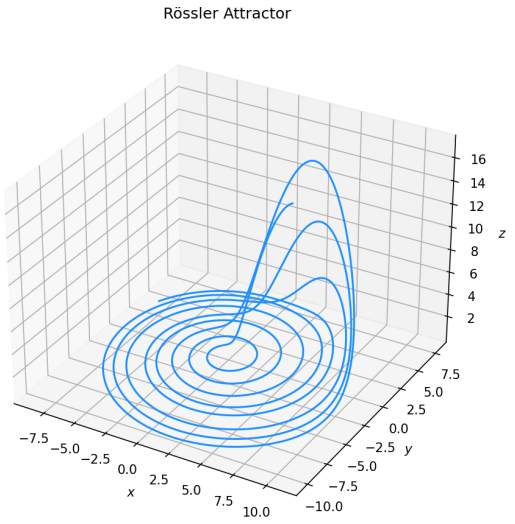


(d) Simulation for equations 5.4 with initial state $[-8, 8, 27]$.

Figure 5.2: ODEs Simulations



(a) Vector Field for equations 5.5



(b) Simulation for equations 5.5 with initial state $[5, 0, 15]$.

Figure 5.3: Rössler Attractor visualisations

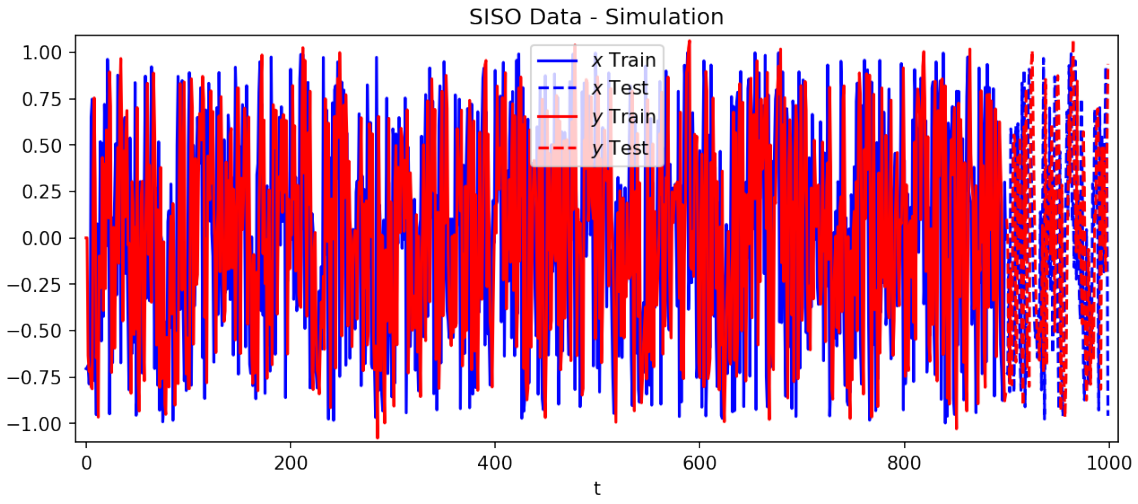


Figure 5.4: Example of the nonlinear function 5.6 where x is a uniform random variable.

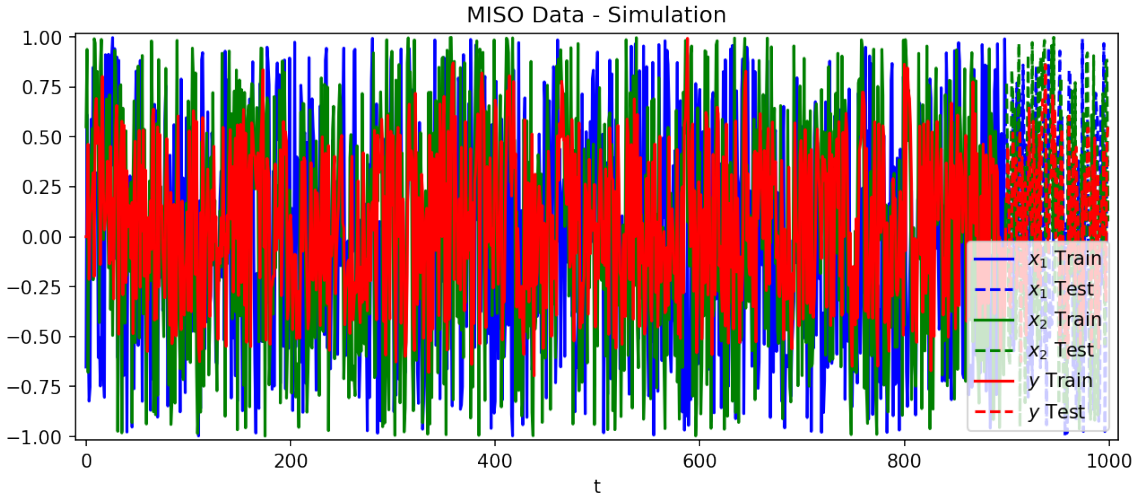


Figure 5.5: Example of the nonlinear function 5.7 where x is a uniform random variable.

Chapter 6

Barabási–Albert Model

The Barabási–Albert model can generate scale-free networks that show emergence behaviour and, therefore, represent many types of complex systems [15]. This scale-free property is achieved through two primary rules in the BA model: Growth and Preferential Attachment. Growth means that a new node is added to the network with m new connections or links at each step, and preferential attachment defines which nodes the new node will most likely connect.

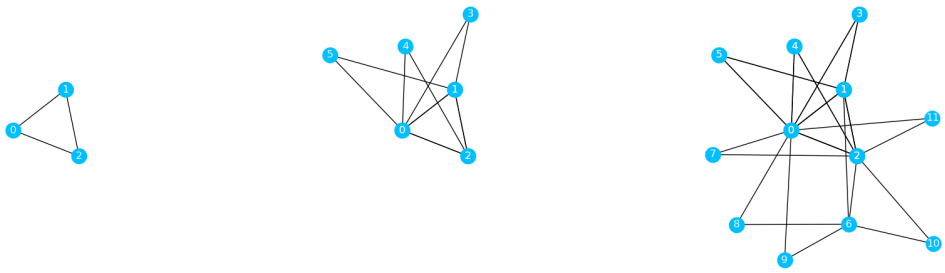
So to build a network with the BA model:

1. Start with m_0 nodes, with arbitrary links between them.
2. Add a new node with m links to the existing nodes in the network. The probability $\prod(k_i)$ that node i is linked to the new node is defined by equation 6.1.
3. Repeat step 2 for growth.

$$\prod(k_i) = \frac{k_i}{\sum_j k_j} \tag{6.1}$$

Understanding that the number of links in a node is the degree of the node, in equation 6.1, k_i refers to the degree of the i-th node. Since the higher the degree, the likelier it is to receive more links, popular nodes (nodes with many links) are likelier to grow.

CHECK W/ FINKE IF THIS CAN BE SAID AND IS TRUE It is important to note that the mathematical definition of the BA model does not give a detailed description of the initial configuration of the nodes nor describes whether the links are added sequentially or simultaneously. For this work, two additional rules are added, the initial state of the network corresponds to $m+1$ nodes wholly connected, and the new connections are distinct to avoid multi-links.



(a) $n = 3$ (b) $n = 6$ (c) $n = 12$

Figure 6.1: Example evolution of the Barabási–Albert model with $m = 2$.

6.1 Degree Dynamics

The dynamics intended to be identified from data by this work is the time evolution of the degree of a single node, in contrast to the analytical approach [24]. Considering a network in which new nodes will link to the network with m edges, the time evolution of the network following this model, *i.e.* the rate at which the degree of a specific node i changes over time, is given by equation 6.2.

$$\frac{dk_i}{dt} = m \prod(k_i) = m \frac{k_i}{\sum_{j=1}^{N-1} k_j} \qquad \text{where} \sum_{j=1}^{N-1} k_j = 2mt - m \tag{6.2}$$

Considering $t \gg 1$, we obtain the degree dynamics expressed in equation 6.3.

$$\frac{dk_i}{dt} = \frac{k_i}{2t} \tag{6.3}$$

Knowing that a node i at time of attachment t_i has m links, *i.e.* $k_i(t_i) = m$, and integrating the equation 6.3, a time function can be derived as seen in equation 6.4.

$$k_i(t) = m \left(\frac{t}{t_i} \right)^{\beta} \qquad \text{where } \beta = 1/2 \tag{6.4}$$

6.1.1 Degree Distribution

Another characteristic of the BA model is the power-law degree distribution described by equation 6.5.

$$p(k) \approx 2m^{\frac{1}{\beta}} k^{-\gamma} \qquad \text{where } \gamma = 3 \tag{6.5}$$

An exact degree distribution that obtains the pre-factors may be calculated [14], but this goes beyond the scope of this work. It is only essential to remember that the underlying degree distribution is a power-law distribution.

6.2 Bianconi-Barabási Model

Going beyond the assumption that a node is attractive simply by its high number of links, the Bianconi-Barabási model expands to the idea that nodes also have a *fitness* level η that makes a node more or less attractive to receive new connections. In this way, a new node with high *fitness* may overpass existing popular nodes with a high degree but not so high *fitness*; this model will be referred to as the BA *Fitness* model. This new attachment rule means that a new node will connect with m links and fitness η_i. The fitness η_i is a random number from the distribution $\rho(\eta)$, and it does not change once set.

To describe this new property of the nodes, the probability of receiving a link from equation 6.1 is changed to equation 6.6. This change in the equation leads to the degree dynamics expressed in equation 6.7

$$\prod(k_i) = \frac{\eta_i k_i}{\sum_j \eta_j k_j} \tag{6.6}$$

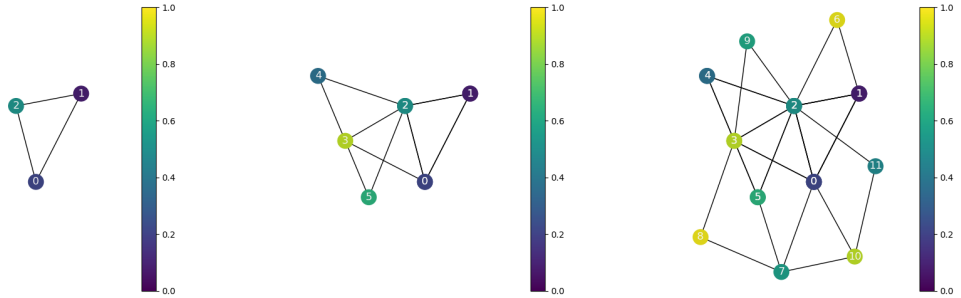
$$\frac{\partial k_i}{\partial t} = m \frac{\eta_i k_i}{\sum_j \eta_j k_j} \tag{6.7}$$

WHAT ELSE MUST GO HERE?

- Complete degree dynamics ?
- Degree Distribution?
- Equal Fitnesses
- Uniform Fitness Distribution?
- Measuring Fitness?

For this work, a simple case with two fitness levels, the consideration lies in whether the methods can differentiate the fitness levels by the identified dynamics. Given the scope of this project, degree evolution is enough. However, much further analysis can be done on the Barabási–Albert and the Bianconi-Barabási model that would lead to significant insights

into the capabilities of the methods for its dynamics identifications Network Science by Albert-László Barabási goes much more into detail [14].



(a) $n = 3$ (b) $n = 6$ (c) $n = 12$

Figure 6.2: Example evolution of the Bianconi-Barabási model with $m = 2$ and random uniform distribution for the fitness.

Chapter 7

Metrics

As part of the performance comparison, metrics are needed to evaluate the resulting model identified by the methods. Three types of metrics were considered: direct consideration of correct model identification, sparsity metrics, and error estimation metrics. Correct model identification shows whether the method successfully identified the correct dynamics, sparsity metrics are measured on the complexity of the identified model and error estimation metrics on the predicted values of the identified model attempt to recreate the system's behaviour. Given the vast amount of existing metrics to measure error, a sample of metrics with enough insight into the performance was chosen [25]. Additionally, execution time was measured to give insight into the feasibility of real-time applications.

7.1 Basis Functions Set

Given that this project aims to show whether the methods can identify the governing set of equations from the set of basis functions that describe the dynamics of the system, we consider that from the dictionary $\mathcal{D}$ of m basis functions, the set of functions $\mathcal{M}_i \subset \mathcal{D}$ exactly describe the dynamics of x_i, where x_i is the time evolution of state i of the system, where $i \in \{1, \ldots, n\}$. If a method estimates that the set of dynamics that describe x_i is $\mathcal{E}_i$, we may say the method correctly identifies the system's dynamics *iff* $\forall_i \; \mathcal{M}_i \subseteq \mathcal{E}_i$. Success in the individual and overall system identification is measured in equations 7.1 and 7.2; these metrics will be referred to as Correct Individual Identification (CII) and Correct Overall Identification (COI).

$$\mathrm{C_i} = [\mathcal{M}_1 \subseteq \mathcal{E}_1 \ldots \mathcal{M}_n \subseteq \mathcal{E}_n]^T \tag{7.1}$$

$$\mathrm{C_o} = \forall_i \; \mathcal{M}_i \subseteq \mathcal{E}_i \tag{7.2}$$

It is important to note that these two metrics are possible due to the restriction of working with systems where there is already an analytical solution. It is helpful in this project to show whether the methods correctly identified the dynamics directly; in further works where a solution is unknown, actual implementations of these methods, it is better to rely on sparse and error based metrics.

Moreover, a method that always returns as a model all the functions in the dictionary $\mathcal{D}$ fitted to the data through a regression approach, like LS, cannot be said to identify a system's dynamics correctly. This behaviour would be a case of over-fitting the recorded data, which may create numerical problems, and instead of revealing the underlying behaviour of the system, it hides it. We are looking for the simplest model that correctly represents the data; this is evaluated through sparsity metrics.

7.2 Sparsity

The number of non-zero entries in a vector is called sparsity and can be obtained via the l_0-norm in equation 7.3. Although this function is not a proper norm, it is pretty helpful to identify how sparse is a vector; therefore, it will be referred to as a norm in this work.

$$\|\mathbf{x}\|_0 = \sum_{n=1}^{N} \mathbf{1}\{x_n \neq 0\} \tag{7.3}$$

where $\mathbf{1}\{\cdot\}$ is the indicator function, as seen in equation 7.4

$$\mathbf{1}\{X\} = \begin{cases} 1 & X \text{ holds} \\ 0 & X \text{ does not hold} \end{cases} \tag{7.4}$$

The l_0-norm can then calculate the sparsity of a resulting model set of dynamics. Considering a dictionary $\mathcal{D}$ of m basis functions, vector $\xi_i \in \mathbb{R}^m$ represents the multiplicative coefficients of the basis functions in $\mathcal{D}$ that describe the dynamics of x_i. Then for a system with n states, the sparsity of the dynamics s_i for state i can be obtained via $\|\xi_i\|_0$, which means that $s \in \mathbb{Z}^n$ represents the sparsity of the model for the n-dimensional system.

$$s = [\|\xi_1\|_0 \dots \|\xi_n\|_0]^T \tag{7.5}$$

Furthermore, given the cases of this project that the real solution, set of equations, is known, the sparsity of the model identified can be compared to the true sparsity of the system. Such Δs represents how many more terms are found in the identified model. Ideally, Δs would be zero, but it is critical to keep in mind that this metric only represents how much more complex the identified model is to the system and alone does not represent the true success of the method. It may have been that the method identified a different set of equations of the equal or even fewer amount of terms, in this case, it is vital to keep track of the error estimation since this will show whether or not the identified model is successful in replicating the dynamics of the system.

A final indicator of the model's simplicity, the complexity $\mathcal{C}$, is defined as the sum over the sparsity, as seen in equation 7.6.

$$\mathcal{C} = \sum_{n=1}^{N} s_n \tag{7.6}$$

7.3 Error Estimation

Given the measurements $Y \in \mathbb{R}^{k \times n}$, where there are n states and k measurements timewise, as the validation data generated from simulating the system with starting conditions Y_0, and the predicted data $\widehat{Y} \in \mathbb{R}^{k \times n}$ generated from the identified model considering Y_0 as starting conditions, the quality of the model as a predictor of the system may be evaluated through the following metrics.

7.3.1 Mean Bias Error

The Mean Bias Error (MBE) represents the predictor's forecast bias; it shows whether the model tends to over or under predict. An MBE of zero means an unbiased estimator, not necessarily without error. MBE is calculated as the average of the actual value minus the predicted value, as seen in equation 7.7.

$$\mathrm{MBE}(\widehat{Y}) = \frac{1}{nk} \sum_{i=1}^{n} \sum_{j=1}^{k} y_{ij} - \widehat{y}_{ij} \tag{7.7}$$

7.3.2 Mean Squared Error

Mean Squared Error (MSE) gives an excellent idea of the amount of error in the prediction by averaging the squares of the errors as seen in equation 7.8. We can also decompose the MSE through the variability and bias of the prediction $\widehat{Y}$, so the predictor needs to have both high accuracy and high precision for it to have a low MSE. Naturally, an MSE of zero would represent a perfect predictor.

$$\mathrm{MSE}(\widehat{Y}) = \frac{1}{nk} \sum_{i=1}^{n} \sum_{j=1}^{k} (y_{ij} - \widehat{y}_{ij})^2 = \mathrm{Var}(\widehat{Y}) + \mathrm{Bias}(\widehat{Y}, Y)^2 \tag{7.8}$$

There is relevance in using the MSE for this work, even though there has been discussion about using the MSE or the Mean Absolute Error (MAE) for model evaluation. As stated by Chai & Draxler, Root Mean Squared Error (RMSE) may reveal better the performance difference of two models under the assumption that the underlying is Gaussian [26]. Although the Barabási–Albert model follows the power-law distribution, for further works, a system whose underlying interactions are complex enough, due to central limit theorem, MSE will be beneficial. Additionally, Wei used MSE extensively for model valuation in systems with a power-law distribution [27], Hassanibesheli *et al.* used MSE to evaluate the

performance of three methods in the reconstruction of dynamics from complex systems [28], Seungwoong & Hawoong used MSE for evaluation of the performance on continuous variables of their data-driven approach to identify the interactions in complex systems [21]; giving precedence in using MSE as a metric in problems closely related to the one in this work.

7.3.3 Root Relative Squared Error

A metric expanding the MSE is the Root Relative Squared Error (RRSE), in which the error of the predictor is compared to the error of the predictions made by a naive predictor as observed in equation 7.10, the average of the values from equation 7.9. Given the importance of finding a simple model that still predicts the system's behaviour correctly, this metric may give insight into the complexity of the model.

$$\overline{y} = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} y_{ij} \tag{7.9}$$

$$\mathrm{RRSE}(\widehat{Y}) = \sqrt{\frac{\sum_{i=1}^{n} \sum_{j=1}^{k} \left(y_{ij} - \widehat{y}_{ij}\right)^2}{\sum_{i=1}^{n} \sum_{j=1}^{k} \left(y_{ij} - \overline{y}\right)^2}} \tag{7.10}$$

Chapter 8

Implementation

This project was done in Python 3.9.7 with PySINDy [20] version 1.4.3 for the SINDy method, SysIdentPy [23] version 0.1.8 for the NARMAX method, and NetworkX version 2.6.3 as a baseline for the simulation of the network models.

Three main implementations were needed to achieve this project: simulation of the network models, adjustments to the implementations of the methods, and metrics calculation. Everything described here is publicly available in this project's repository [29].

8.1 Networks Generation

Although NetworkX has a generator for BA, computing the degree evolution for any node required calculating the degree for every sub-Graph during the generation. Therefore, a different generator was needed, with NetworkX's generator as a basis, that calculated the degree evolution of all the nodes in the network as it was being created; this sped up significantly the process.

Additionally, NetworkX offered no generator for BA *Fitness*; this required creating one that assigned a *fitness* level to each node from a uniformly distributed random variable and considered this to change the probability of new links. To test the capabilities, by the methods, to differentiate *fitness* levels, the generator assigned the *fitness* uniformly based on a given list of two different values for this work, and deterministically the *fitness* of the initial node for degree evolution estimation and comparison.

Lastly, two prominent cases were considered during generation for later method comparison: a single network and the average degree evolution of multiple networks under the same conditions, regarding the m new links and fitness level distribution for BA *Fitness*. The former is the primary goal of this project, and the latter is to observe the performance on the expected degree evolution, given the stochastic nature of the model. This averaging of the degree evolution was achieved through concurrent generation, detailed in appendix A.

8.1.1 Cases Considered

Each network generated had 10000 nodes, each one with two new links. The following types were generated:

- Barabási–Albert
- Bianconi-Barabási with *Fitness* levels $[0.991, 0.223]$.
- Bianconi-Barabási with *Fitness* levels $[0.75, 0.25]$.
- Bianconi-Barabási with *Fitness* levels $[0.625, 0.375]$.
- Bianconi-Barabási with *Fitness* levels $[0.5625, 0.4375]$.
- Bianconi-Barabási with *Fitness* levels $[0.53125, 0.46875]$.

For each case, a single network and the average degree evolution of 100 networks were generated. The BA *Fitness* model cases consisted of assigning to all the nodes one of the two *fitness* values available; from these two cases exist each one analysing the initial node evolution with a different *fitness* value.

8.2 Methods Adjustments

The aim was to test both methods on a broad range of systems. Since some systems do not fit the initial design of the methods implementations, adjustments were needed, even if most of them are more related to the implementation of the methods.

8.2.1 SysIdentPy Adjustments

The NARMAX method has two main limitations: solving models with State-Space Representation and restriction regarding the candidate functions available to build the transfer function. The latter is a limitation of the SysIdentPy version since ensemble basis functions were recently added (v0.1.8) but are limited to the combination of Fourier Basis functions and Polynomial functions; no custom function allowed.

Regarding State-Space models, the NARMAX method found a transfer function for every state, in which the estimated transfer function of the state had as an output the state estimated and all the other states as inputs; generalising this process allowed for simple usage of NARMAX on systems like the ones from section 5.1.

NARMAX Basis functions

As evident from equation 6.3, the dynamic is time-varying and neither from a Polynomial or Fourier basis. A workaround considers the multiplicative inverse time as the input of the transfer functions, with the degree as the output. It is allowed since the network evolves from the initial time-step $t_0 = 1$, letting the basis function $1/t$ be part of the Polynomial basis functions, and therefore NARMAX can identify it.

8.2.2 PySINDy Adjustments

PySINDy allowed both State-Space Representation and custom basis functions, making the overall adjustments easier. For integrating time into the dynamics, equation 8.1 was the auxiliary basis function added to the library of candidate functions. Additionally, time was considered as the input to the system, using SINDy with Model Predictive Control (MPC) [30]. SINDy with control also allowed the usage of time-delays to find the solution to the systems described in section 5.2; here, the inputs were time-shifted to account for different time-delays.

$$f_c(x, y) \triangleq x/y \tag{8.1}$$

8.3 Metrics Data Frame

A data frame with the metrics from chapter 7 was built to simplify the testing and evaluation pipeline. The error estimation metrics from section 7.3 did not require implementation, given that these are pretty common and have been previously implemented. The sparsity-based metrics from section 7.2 are all based on the calculation of the l_0-norm. An important note here is distinguishing between differential equations and difference equations; the Δs metric considers the systems always to be described with differential equations. Therefore, the solution from the NARMAX method and the SINDy method in discrete mode may be wrong by one; this is not corrected in the metrics data frame and needs to be considered for any analysis.

Regarding the model metrics from section 7.1, implementation of the metrics as defined in equations 7.1 and 7.2, both according to the way the methods presented their solutions $\mathcal{E}_i$: PySINDy in a sparse matrix of coefficients where the non-zeros are the selected terms, and SysIdentPy as a $k \times l$ matrix of regressors where k represents the number of terms in the dynamics equation and l the nonlinearity order.

Chapter 9

Results and discussion

All the results are in this project's repository [29]. Each system was simulated, split in training and testing data, and both methods attempted to identify the underlying dynamics from the same training data. COI will not be shown in the following tables because it is easily inferred from CII.

9.1 ODE Systems

Both methods can successfully identify the corresponding dynamics with some advantage to SINDy regarding execution time for a simple linear case.

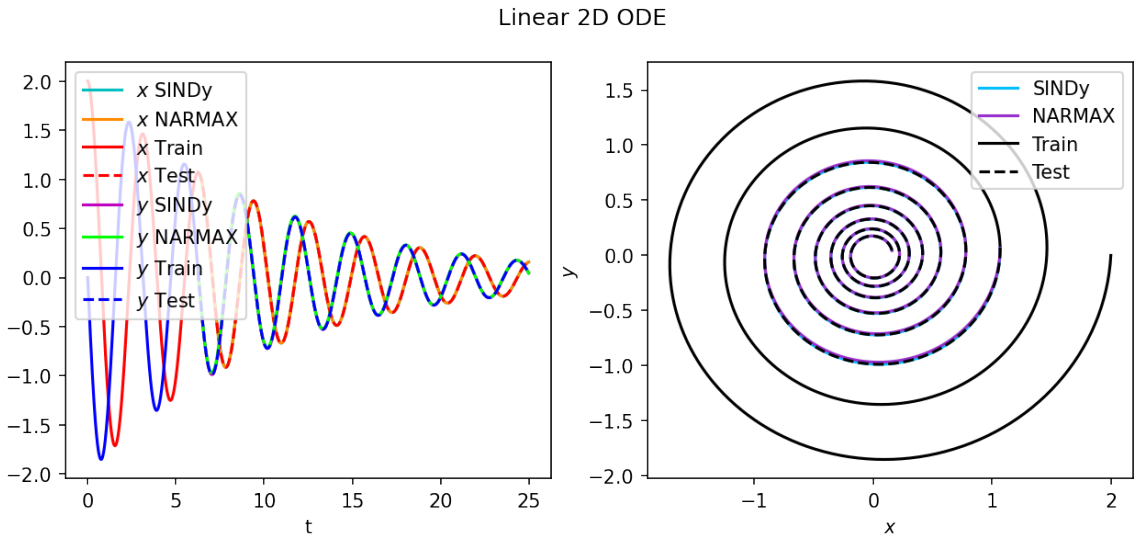


Figure 9.1: Estimations from the SINDy and NARMAX methods of the Linear 2D ODE from equations 5.1.

Method	CII	$\|\cdot\|_0$	Δs	$\mathcal{C}$	MBE	MSE	RRSE	Time (s)
SINDy	[T, T]	[2, 2]	[0, 0]	4	7.00e-6	9.82e-8	8.12e-4	6.25e-3
NARMAX	[T, T]	[2, 2]	[0, 0]	4	-3.12e-3	9.64e-5	2.55e-2	7.63e-1

Table 9.1: Metrics data frame for the methods estimating the Linear 2D ODE system dynamics from equations 5.1.

As soon as the methods attempt a nonlinear system, NARMAX over-fits the number of terms needed. Due to its successful COI and corrections from the additional terms, it can still predict the system behaviour with low error.

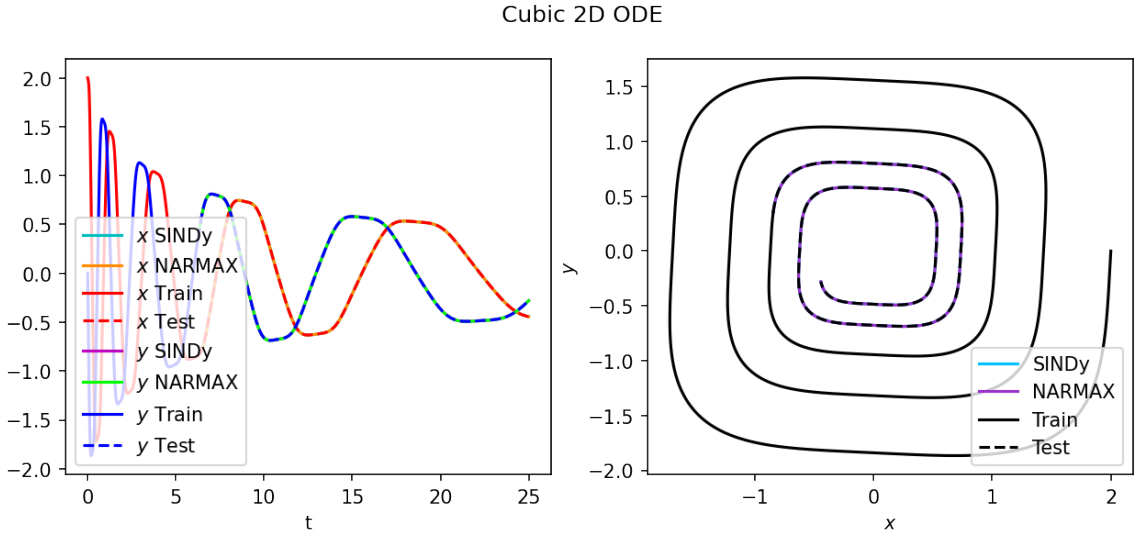


Figure 9.2: Estimations from the SINDy and NARMAX methods of the Cubic 2D ODE from equations 5.2.

Method	CII	$\|\cdot\|_0$	Δs	$\mathcal{C}$	MBE	MSE	RRSE	Time (s)
SINDy	[T, T]	[2, 2]	[0, 0]	4	-2.00e-6	7.99e-11	1.90e-5	1.41e-2
NARMAX	[T, T]	[7, 6]	[5, 4]	13	3.28e-4	3.34e-7	1.23e-3	5.88

Table 9.2: Metrics data frame for the methods estimating the dynamics of the Cubic 2D ODE system from equations 5.2.

Higher linear dimensions show a good performance from NARMAX since the model is fitted to every state individually, which is noticeable in the time needed for complete identification.

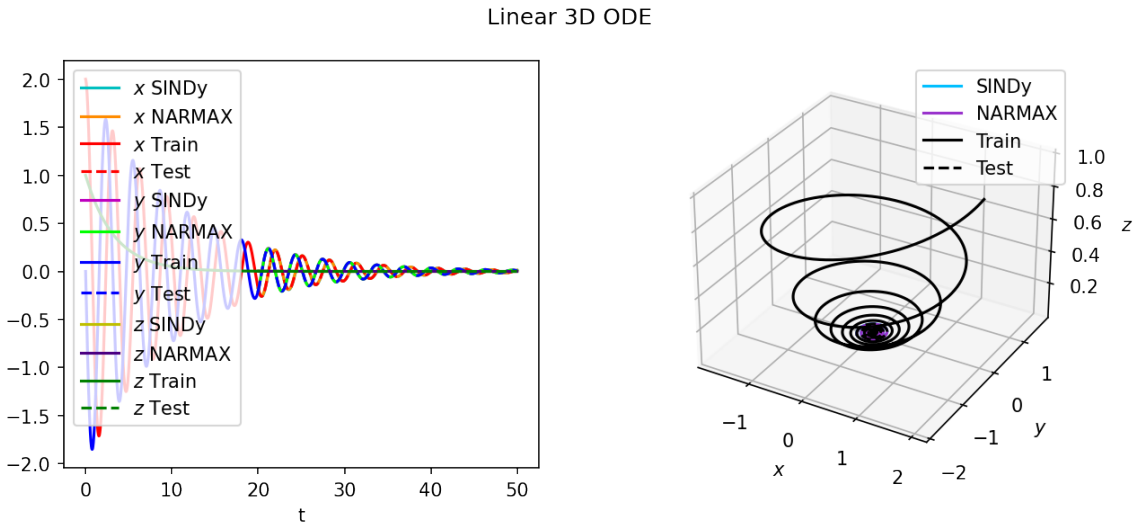


Figure 9.3: Estimations from the SINDy and NARMAX methods of the Linear 3D ODE from equations 5.3.

Method	CII	$\|\cdot\|_0$	Δs	$\mathcal{C}$	MBE	MSE	RRSE	Time (s)
SINDy	[T, T, T]	[2, 2, 1]	[0, 0, 0]	5	3.98e-7	3.41e-9	9.16e-4	1.48e-2
NARMAX	[T, T, T]	[2, 2, 1]	[0, 0, 0]	5	-4.21e-5	2.60e-6	2.54e-2	2.79

Table 9.3: Metrics data frame for the methods estimating the Linear 3D ODE system dynamics from equations 5.3.

The chaotic Lorenz system gives excellent insight into the differences between these two methods. While SINDy successfully identified the correct basis functions, it offsets the actual values over time due to coefficient precision on the long-term prediction. This behaviour is not of huge concern since it is characteristic of chaotic systems. On the other hand, NARMAX seems to perfectly track the system during the whole prediction stage, which is evident in the low error estimation, but this is compensated with a model of three times the complexity required.

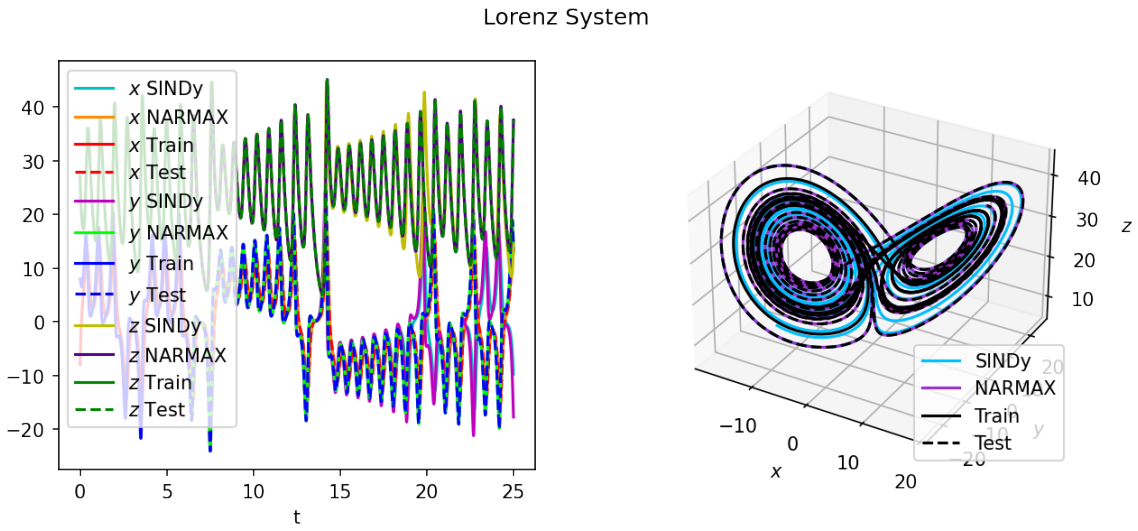


Figure 9.4: Estimations from the SINDy and NARMAX methods of the Lorenz system from equations 5.4.

Method	CII	$\|\cdot\|_0$	Δs	$\mathcal{C}$	MBE	MSE	RRSE	Time (s)
SINDy	[T, T, T]	[2, 3, 2]	[0, 0, 0]	7	-1.17e-1	4.70e+1	8.15e-1	1.69e-1
NARMAX	[T, T, T]	[5, 8, 8]	[3, 5, 6]	21	-1.16e-3	9.55e-4	3.66e-3	1.76e+1

Table 9.4: Metrics data frame for the methods estimating the dynamics of the Lorenz system from equations 5.4

The Rössler attractor gives an entirely different view on the behaviour of the methods. In this case, SINDy can precisely identify the chaotic dynamics and correctly estimate them with low error. In contrast, NARMAX, in this case, has a worse estimation error while still over-fitting the number of terms.

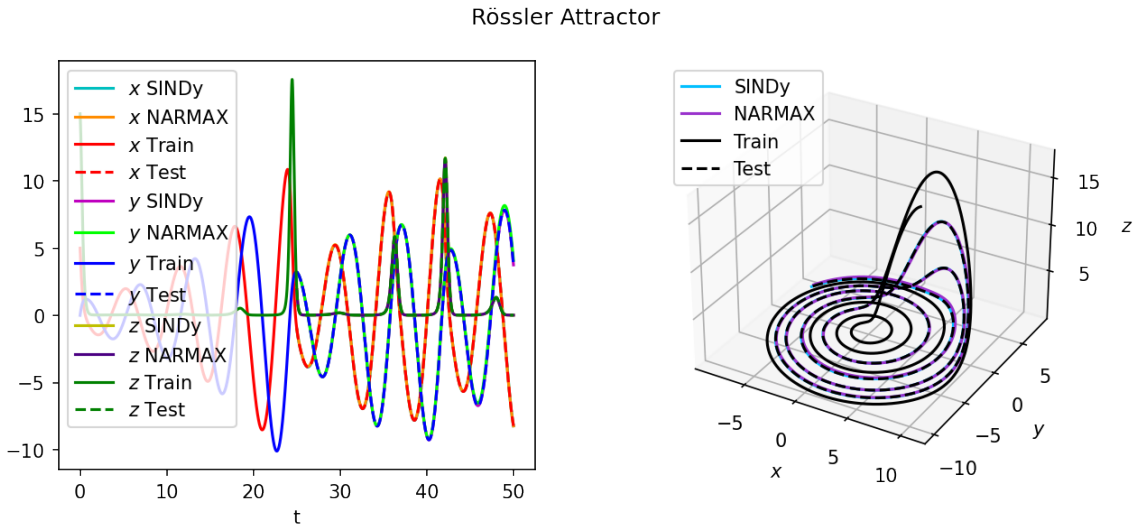


Figure 9.5: Estimations from the SINDy and NARMAX methods of the Rössler Attractor from equations 5.5.

Method	CII	$\|\cdot\|_0$	Δs	$\mathcal{C}$	MBE	MSE	RRSE	Time (s)
SINDy	[T, T, T]	[2, 2, 3]	[0, 0, 0]	7	-2.60e-5	1.10e-5	7.80e-4	4.18e-2
NARMAX	[T, T, T]	[10, 10, 9]	[8, 8, 6]	29	-4.05e-2	9.85e-3	2.37e-2	4.82

Table 9.5: Metrics data frame for the methods estimating the dynamics of the Rössler Attractor from equations 5.5

The main disadvantage NARMAX has against SINDy is its inability to solve for the complete high dimensional system. This handicap dramatically affects the time required to find a solution. On the other hand, over-fitting is not of significant concern; while it is true that the sparsest model is desired, the true aim is for interpretable models, which NARMAX still achieves. It is important to recall that SINDy does a differentiation step to consider the data, as it should, from a continuous-time perspective while NARMAX is attempting to fit the data as if it was discrete-time, the additional terms may come from this wrong initial assumption.

9.2 Nonlinear Functions

In the simplest case of a nonlinear function, the SISO is of no issue to either of the methods, correctly identifying the regressors with low estimation error. The high RRSE for SINDy is unexpected, given that apparently, the estimations between the two methods

seem to be quite similar in all other regards; this requires further insight since it may show that either RRSE is not an appropriate metric or an additional metric that contrasts the error from MBE and MSE is needed.

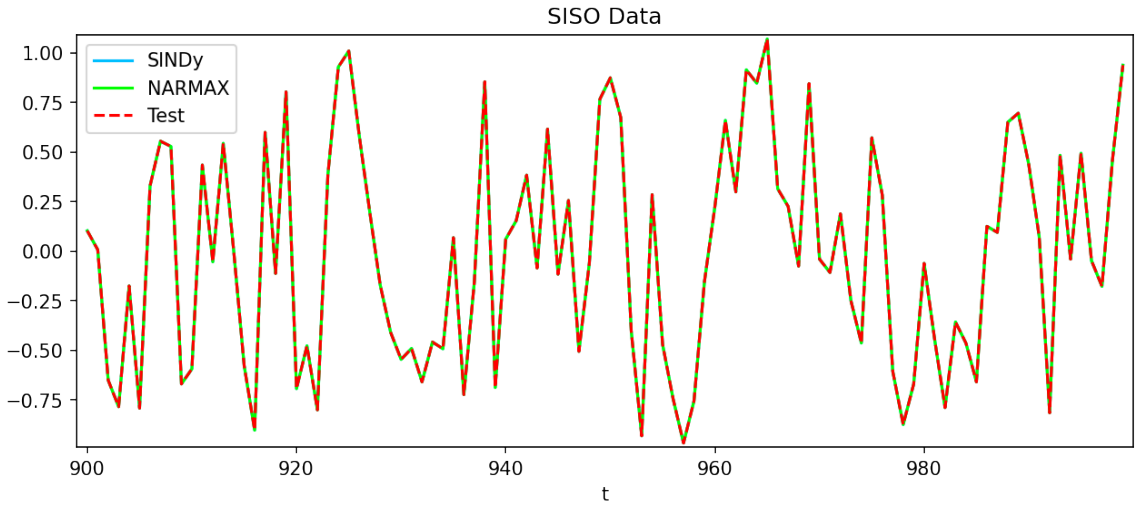


Figure 9.6: Estimations from the SINDy and NARMAX methods of the SISO function from equation 5.6.

Method	CII	$\|\cdot\|_0$	Δs	$\mathcal{C}$	MBE	MSE	RRSE	Time (s)
SINDy	[T]	[3]	[0]	3	-2.20e-5	1.18e-8	1.41e+1	4.87e-3
NARMAX	[T]	[3]	[0]	3	-2.10e-5	1.17e-8	1.96e-4	2.72e-2

Table 9.6: Metrics data frame for the methods estimating the SISO function from equation 5.6

The MISO case further shows the capability of both methods to identify this type of function. The same effect from the SISO case regarding the RRSE is observed.

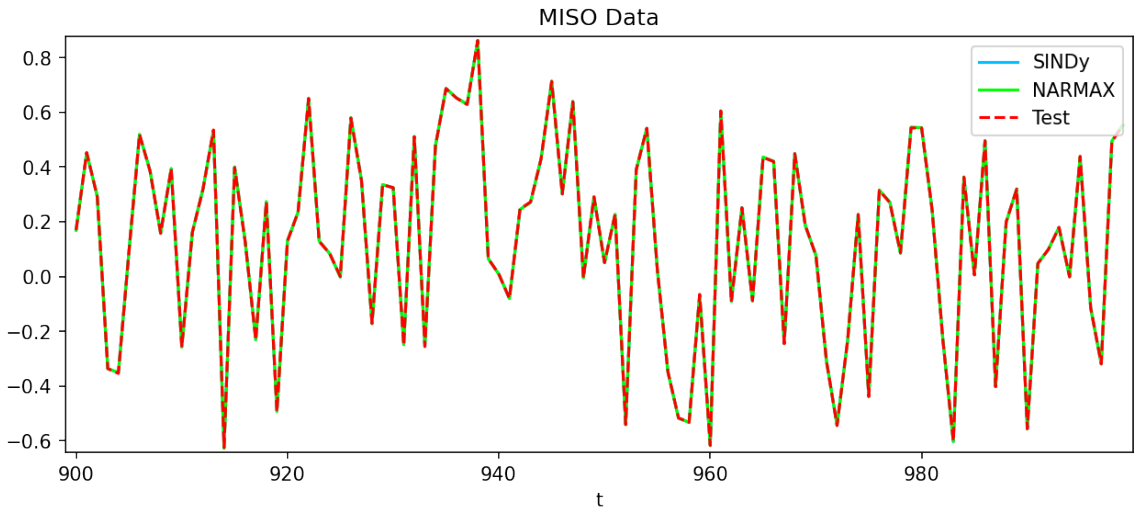


Figure 9.7: Estimations from the SINDy and NARMAX methods of the MISO function from equation 5.7.

Method	CII	$\|\cdot\|_0$	Δs	$\mathcal{C}$	MBE	MSE	RRSE	Time (s)
SINDy	[T]	[4]	[0]	4	-2.66e-4	1.00e-6	1.41e+1	5.47e-3
NARMAX	[T]	[4]	[0]	4	-2.91e-4	1.00e-6	3.30e-3	6.76e-2

Table 9.7: Metrics data frame for the methods estimating the MISO function from equation 5.7

The most exciting result is that SINDy is not designed for lags in the input, but it could identify them correctly in both cases. Maybe an extension of SINDy for an M lagged input would allow for an even broader range of systems to be identified. Additionally, it is essential to remember that for both of these cases, x was uniform random noise, which means that both systems can distinguish this behaviour and still find the coefficients that relate the input to the output.

9.3 Barabási–Albert Model

Regarding the BA model, the objective was to identify the degree evolution of the first node according to equation 6.3. The methods had the first 40% of the evolution for training and attempted to predict the rest. Initially, there is the identification and estimation on the ensemble average of 100 networks considering the degree evolution as a random process using the NARMAX and SINDy methods, followed by the identification and estimation on

the realisation of a single network using the NARMAX method and compared to SINDy considering the evolution in continuous and in discrete time.

Both methods successfully identify the specific term of the evolution according to equation 6.3 in the ensemble average case, with NARMAX adding additional terms that allow it to improve its predictions significantly and, therefore, its estimation error. For reference, the curve describing the dynamics is plotted with "F. Sol." showing the evolution according to equation 6.4 and "Diff. Eq. Sol." showing the evolution according to equation 6.3.

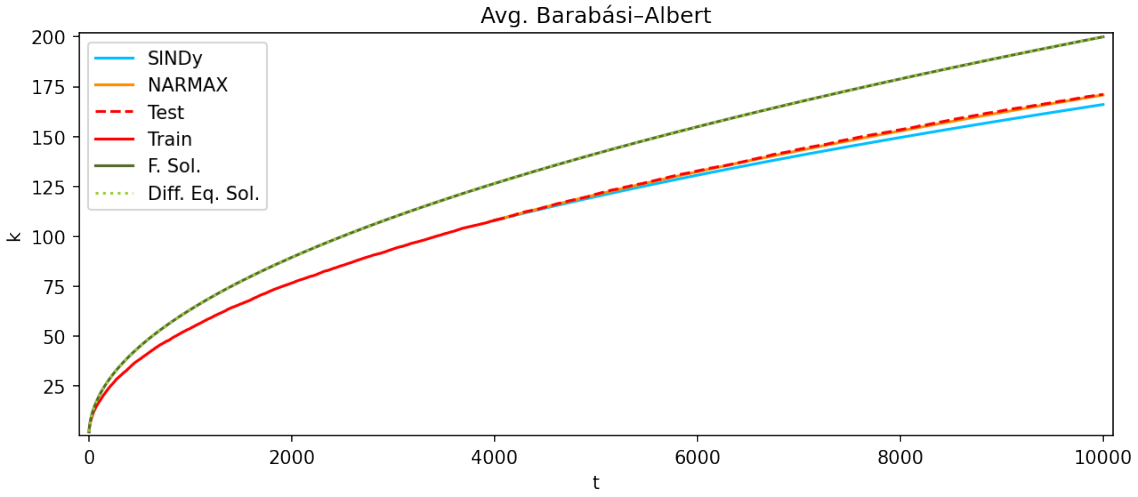


Figure 9.8: Estimations of the SINDy and NARMAX methods from the identified model of the initial node from the BA model averaged over 100 realisations.

Method	CII	$\|\cdot\|_0$	Δs	$\mathcal{C}$	MBE	MSE	RRSE	Time (s)
SINDy	[T]	[1]	[0]	1	2.94	1.11e+1	1.14e+2	3.20e-2
NARMAX	[T]	[4]	[3]	4	4.44e-1	2.65e-1	1.10e+2	1.77e-1

Table 9.8: Metrics data frame for the methods estimating the evolution of the initial node of the BA model averaged over 100 realisations.

The single realisation case gives an insight into the capability of the methods on model identification when the data is from a stochastic nature. SINDy shows excellent performance in this case, but NARMAX's tendency to overfit leads it to a high order term that significantly affects the prediction. This error can be omitted by reducing the order of polynomials considered, but this would mean giving more *a priori* information than the one SINDy needed.

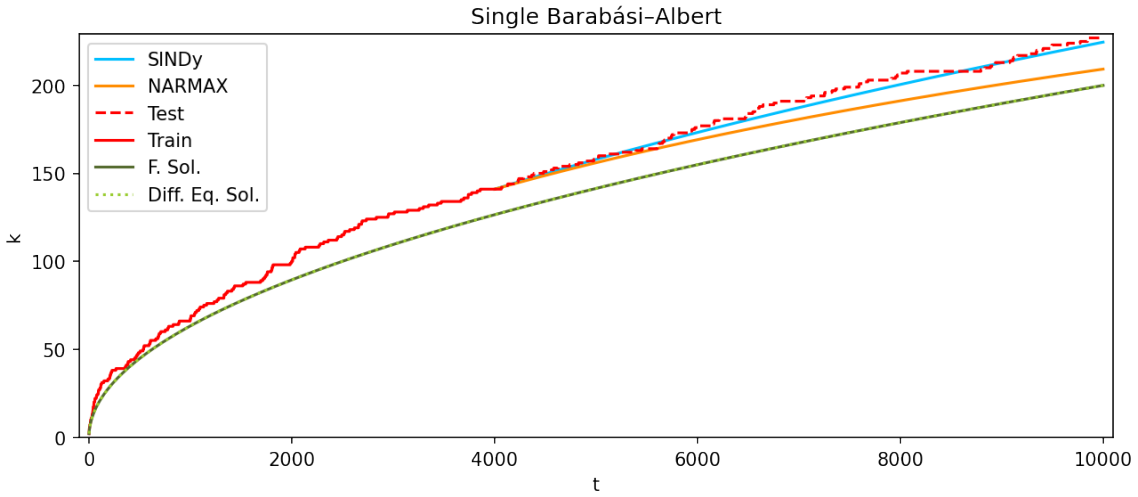


Figure 9.9: Estimations of the SINDy and NARMAX methods from the identified model of the initial node from the single BA model realisation.

Method	CII	$\|\cdot\|_0$	Δs	$\mathcal{C}$	MBE	MSE	RRSE	Time (s)
SINDy	[T]	[1]	[0]	1	2.27	1.04e+1	1.11e+2	1.27e-2
NARMAX	[T]	[3]	[2]	3	9.20	1.15e+2	1.18e+2	1.67e-1

Table 9.9: Metrics data frame for the methods estimating the evolution of the initial node of the single BA model realisation.

The advantage regarding the estimation bias is evident when comparing the continuous and discrete modes for SINDy. It is important to recall that Δs does not recognise the need for an additional term, the previous degree value, in the discrete case. In other words, Δs with the discrete mode of SINDy does not add additional terms.

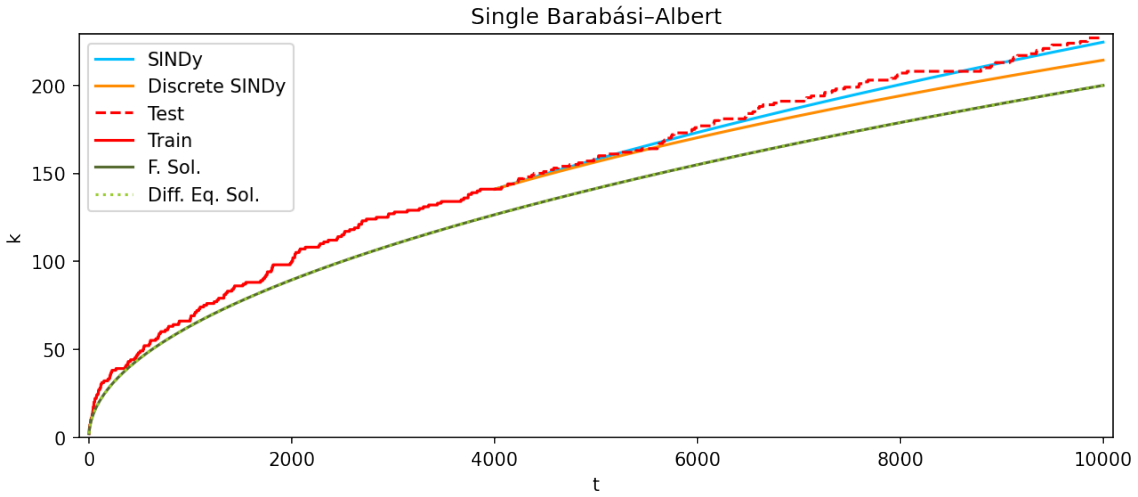


Figure 9.10: Estimations of the SINDy in discrete mode and NARMAX methods from the identified model of the initial node from the single BA model realisation.

Method	CII	$\|\cdot\|_0$	Δs	$\mathcal{C}$	MBE	MSE	RRSE	Time (s)
SINDy	[T]	[1]	[0]	1	2.27	1.04e+1	1.11e+2	1.27e-2
SINDy Discrete	[T]	[2]	[1]	2	7.05	6.75e+1	1.16e+2	1.01e-2

Table 9.10: Metrics data frame for the methods estimating the evolution of the initial node of the single BA model realisation with SINDy in discrete mode.

Overall, both methods correctly identify the expected dynamics, with considerable similarities to the original mathematical model.

9.4 Bianconi-Barabási Model

The different ranges of *fitness* levels for the Bianconi-Barabási model were defined to find whether the methods could differentiate between two nodes on similar networks but with different *fitness* levels, and if so, at which *fitness* difference threshold. Both models were able to identify a dynamic that connected the current degree with the inverse of the passing of time, as is with the BA model in equation 6.3, but it is essential to consider that the BA *Fitness* model has different dynamics, equation 6.7. Accordingly, the identification is considered successful if the time-degree relationship can be identified and the corresponding coefficient is significantly different between the bigger and smaller *fitness* values. For reference, a solution based on equation 6.7 is plotted in green along with the simulated and predicted values. In each plot, the bigger *fitness* value is referred to as "upper" and the

smaller one as "lower".

For the ensemble average identification, both methods successfully identify a dynamic equation that relates the degree with the inverse time, with coefficients that correspond to the *fitness* levels; higher *fitness* level with a higher coefficient. Overall, both methods perform similarly, although NARMAX requires significantly more time to find a solution than SINDy.

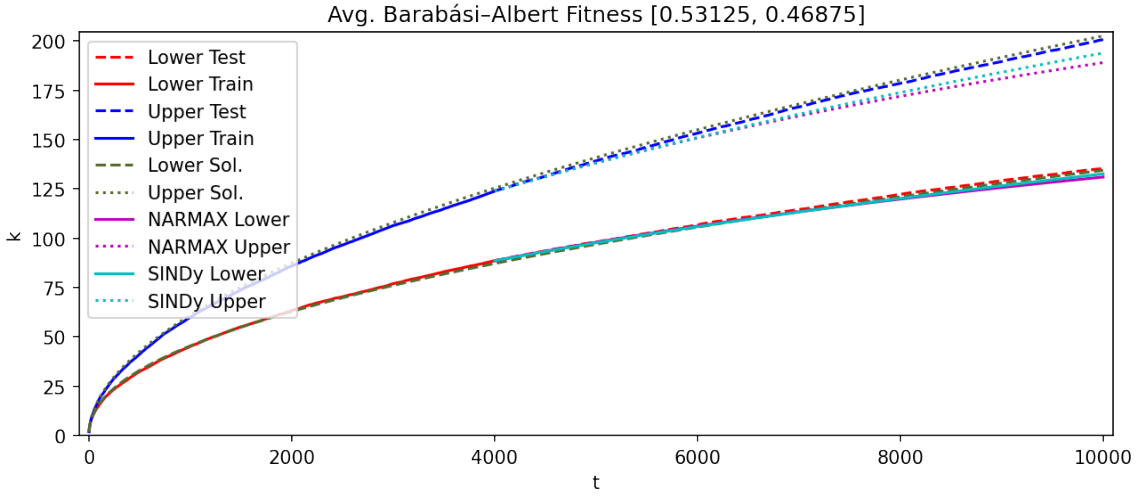


Figure 9.11: Estimations of the SINDy and NARMAX methods from the identified model of the initial node from the BA *Fitness* model with discrete fitness levels $[0.53125, 0.46875]$ averaged over 100 realisations.

Method	CII	$\|\cdot\|_0$	Δs	$\mathcal{C}$	MBE	MSE	RRSE	Time (s)
SINDy	[T]	[1]	[0]	1	1.44	2.86	1.13e+2	8.01e-2
NARMAX	[T]	[3]	[2]	3	1.67	4.74	1.15e+2	5.31

Table 9.11: Metrics data frame for the methods estimating the evolution of initial node of the BA *Fitness* model with discrete fitness levels $[0.53125, 0.46875]$ averaged over 100 realisations estimating a node with *Fitness* 0.46875.

Method	CII	$\|\cdot\|_0$	Δs	$\mathcal{C}$	MBE	MSE	RRSE	Time (s)
SINDy	[T]	[1]	[0]	1	3.41	1.57e+1	1.14e+2	1.26e-2
NARMAX	[T]	[3]	[2]	3	4.80	3.51e+1	1.18e+2	2.50

Table 9.12: Metrics data frame for the methods estimating the evolution of initial node of the BA *Fitness* model with discrete fitness levels $[0.53125, 0.46875]$ averaged over 100 realisations estimating a node with *Fitness* 0.53125.

The single realisation case gives a whole different story from the NARMAX method; it completely fails to identify a sensible dynamic for the bigger *fitness* value. Also, unlike the other cases, there is no estimation error advantage of NARMAX over the SINDy method. Here SINDy shows resilience in model complexity and maintaining a small estimation error.

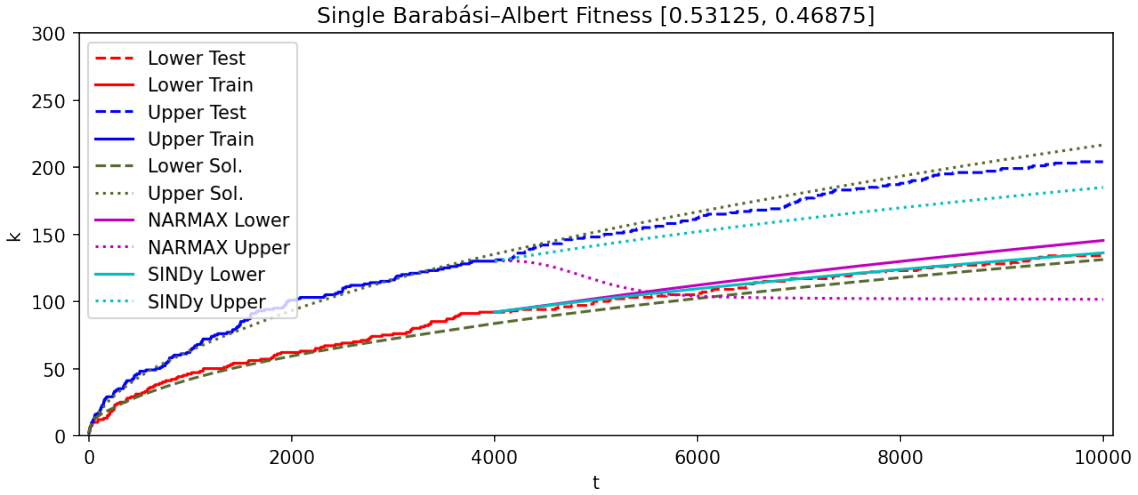


Figure 9.12: Estimation from the SINDy and NARMAX methods of the BA *Fitness* model with discrete fitness levels $[0.53125, 0.46875]$.

Method	CII	$\|\cdot\|_0$	Δs	$\mathcal{C}$	MBE	MSE	RRSE	Time (s)
SINDy	[T]	[1]	[0]	1	-1.37	3.06	1.11e+2	4.03e-2
NARMAX	[T]	[6]	[5]	6	-5.72	3.93e+1	9.90e+1	9.75e-2

Table 9.13: Metrics data frame for the methods estimating the BA *Fitness* model with discrete fitness levels $[0.53125, 0.46875]$ estimating a node with *Fitness* 0.46875.

Method	CII	$\|\cdot\|_0$	Δs	$\mathcal{C}$	MBE	MSE	RRSE	Time (s)
SINDy	[T]	[1]	[0]	1	1.35e+1	2.21e+2	1.13e+2	1.17e-2
NARMAX	[T]	[4]	[3]	4	6.64e+1	5.28e+3	8.15e+1	7.08e-2

Table 9.14: Metrics data frame for the methods estimating the BA *Fitness* model with discrete fitness levels $[0.53125, 0.46875]$ estimating a node with *Fitness* 0.53125.

For the discrete case of SINDy, removing possible stability advantages from the differentiation stage shows improvements over the continuous mode, mainly on identifying the bigger *fitness* value. This improvement gives excellent insight into the robustness against stochastic behaviour, regardless of the *fitness* level.

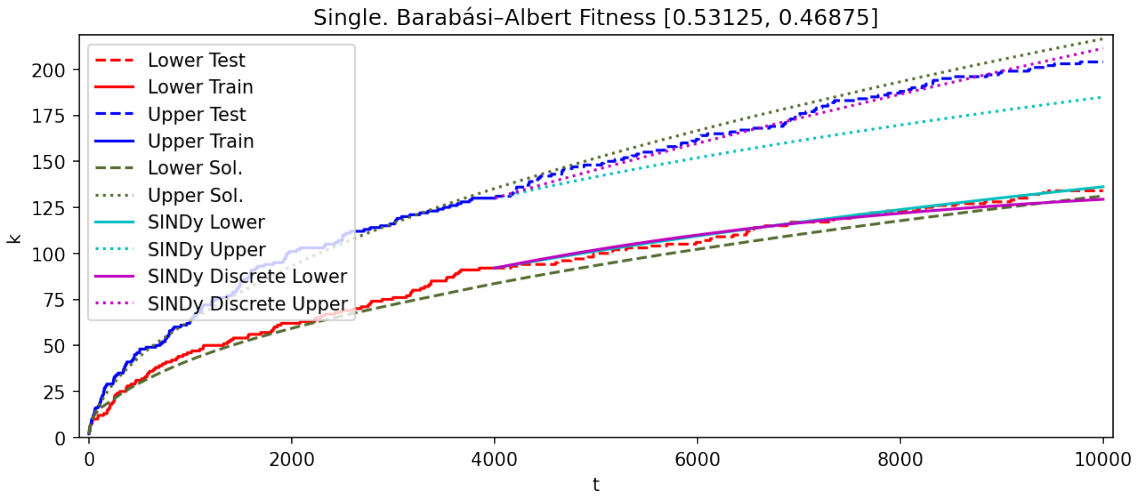


Figure 9.13: Estimation from the SINDy and NARMAX methods of the BA *Fitness* model with discrete fitness levels $[0.53125, 0.46875]$ and SINDy in discrete mode.

Method	CII	$\|\cdot\|_0$	Δs	$\mathcal{C}$	MBE	MSE	RRSE	Time (s)
SINDy	[T]	[1]	[0]	1	-1.37	3.06	1.11e+2	3.10e-2
SINDy Discrete	[T]	[3]	[2]	3	1.48e-2	8.23	1.23e+2	6.09e-3

Table 9.15: Metrics data frame for the methods estimating the BA *Fitness* model with discrete fitness levels $[0.53125, 0.46875]$ and comparing SINDy in continuous and discrete mode estimating a node with *Fitness* 0.46875.

Method	CII	$\|\cdot\|_0$	Δs	$\mathcal{C}$	MBE	MSE	RRSE	Time (s)
SINDy	[T]	[1]	[0]	1	1.35e+1	2.21e+2	1.13e+2	5.45e-3
SINDy Discrete	[T]	[2]	[1]	2	9.51e-1	7.37	1.06e+2	5.81e-3

Table 9.16: Metrics data frame for the methods estimating the BA *Fitness* model with discrete fitness levels $[0.53125, 0.46875]$ and SINDy in continuous and discrete mode estimating a node with *Fitness* 0.53125.

Identifying the degree evolution on BA *Fitness* shows the capability of the SINDy method to identify distinct *fitness* values. The NARMAX method requires ensemble averaging for the correct identification, which is less viable for a real-application scenario. These results do not show that the methods identify the actual *fitness* value but that different equations can be identified between different *fitness* values, as close as 0.0625.

Chapter 10

Conclusions

SINDy and NARMAX were selected, aiming for sparse-based methods that broadly represented the standard nonlinear system identification methods. Which were compared with sparse and error based metrics, as well as per term model identification success; these were selected to give a broad insight into model representation, model sparsity to evaluate the simplicity of the models, accuracy estimation as the most common metrics, and time for insight into real-time applications. As a baseline for testing, the methods were tested on ODE systems, such as chaotic systems, nonlinear transfer functions with random input, and the BA model with the extension of BA with *fitness*. Both methods perform pretty well and correctly identify the underlying dynamics; SINDy tends for sparser models while NARMAX for minimised estimation errors.

The methods' performance on the Barabási–Albert model were compared, giving insight into their capability to identify node degree evolution, both on an ensemble average of the model and a single realisation. Additionally, the methods were compared in their capability to identify different equations between two nodes with different *fitness* values in a Bianconi-Barabási model. Both methods succeeded at identifying the underlying dynamics of the BA model. Although SINDy successfully identifies distinct equations between two nodes with different *fitness* values in a BA Fitness model, NARMAX failed when only a single realisation was considered. Overall, there is some indication that SINDy performs better than NARMAX at this task, and given the flexibility of the model for extensions, some ideas implemented in the NARMAX method could improve SINDy, for example, the usage of information criterion and inputs with lag. Additionally, SINDy consistently identified the dynamics faster than NARMAX, which may allow real-time implementation on some applications.

In conclusion, the main contributions of this work are:

1. This work over-viewed different methods and metrics.
2. Modified the generator for the BA model that gives the degree evolution of the nodes in the network.
3. Created a generator for the BA *Fitness* model, the also return the degree evolution of the nodes in the network.
4. Used the current SINDy with control to consider lag in the input.

5. Adapted the current NARMAX implementation to find the governing equations for multidimensional systems.
6. It proposed an approachable way to include rational basis functions for the Polynomial NARMAX basis library current implementation in SysIdentPy.
7. It proposes a metrics data frame for nonlinear system identification methods comparison.

10.1 Future Works

There are many ways and options to continue this work. Initially, the tests here were limited regarding the optimiser of the SINDy method; neither did it test the extensions that exist now from it, like Ensemble-SINDy [31]. Additionally, the ER method proposes an entirely different approach, information content over sparsity, to find more accurate models. To implement ER in Python, like the equivalent ERFit in MATLAB [32], or using information criterion for model validation in SINDy. A complete testing framework may aid the testing of more methods.

Additionally, this work was limited to simulated data and model identification; it is still needed to test the methods on real data. Different applications can be considered, like system control or extending models that empirically show a lack of precision. In the long term, most methods still depend on an initial correct library of basis function selection, a method that can simplify this process would be even more robust in the whole task of dynamics identifications. There are many possible continuations of this work, aiming for data-driven solutions for systems understanding.

Appendices

Appendix A

Multiprocessing for Stochastic Networks Generation

The average of multiple networks is needed to get the expected degree evolution of a network, which means generating multiple networks, a process that does not require the completion of previous networks generation. Given the need in this project to generate multiple networks with many nodes, a process that sequentially takes much time, a simple structure to generate multiple networks simultaneously was developed. Luckily, Python comes with the multiprocessing package to parallel the execution of a function across many processes; this allows the generation of as many networks as processes defined using the pool object.

```
from multiprocessing import Pool

with Pool(processes=numProc) as process:
    process.map(generate_network, range(N))
```

Where "generate network" is a function that generates a single stochastic network, "N" is the number of networks that we want to generate, and "numProc" is the number of parallel processes, limited by the number of CPU cores. The handler is needed to avoid termination problems. An option here is to generate the "N" networks, collect them and then get the mean of the evolution; the problem here is that as "N" grows, much more memory is required to store that many networks. The alternative proposed is to use a shared memory space, where all the networks are generated, add their degree evolution as they are generated. This shared space is done with the shared "ctypes" RawArray, defined according to the data types and the size of each network generated.

```
from multiprocessing.sharedctypes import RawArray

array_mem = RawArray(ctypes.c_uint32, (nodes-links)*nodes)
```

Where "ctypes.c_uint32" is the data type of 32 bits unsigned integer, "nodes" is the number of nodes in the network, and "links" is the number of new links per node. For it to be accessible to all the processes as a NumPy structure since this is how the network generator returns the degree evolution, the shared memory space, and the appropriate shape is part of the initialisation of the multiprocessing Pool object.

```
def init_process(array_mem, array_shape):
    global process_array_mem, process_array_shape
    process_array_mem = array_mem
    process_array_shape = array_shape
```

Where "array shape" is the shape of the NumPy array that holds the degree evolution of the network, $(n-m)\times n$ for the custom generator of the BA model. Lastly, an attempt by multiple processes to access the same shared space may result in data corruption. A Lock object solves this by restricting the access to the shared space one process at a time; this may seem to make the whole point of multiprocessing useless, but the advantage of the parallel generation still holds, given that generating the network requires more resources than adding it to the shared memory space.

```
from multiprocessing import Lock

lock = Lock()

...

with lock:
    ...
```

These tools allow to generate and average many networks much faster. An appropriate test was run on a Ryzen™ Threadripper™ PRO 3955WX, which has 16 cores with 32 virtual cores, using a single core to sequentially generate the networks and using the described structure with 30 parallel processes. The test consisted of generating eleven types of networks, one BA model and ten BA *Fitness* models with different combinations of *fitness* values, each averaging 200 networks of 10000 nodes each with two new links.

Cores	Time (s)	Time (h)
Single	4.58e+4	12.716
Multi (30)	3.52e+3	0.977

Table A.1: Time comparison of a single and multi-process approach to network generation.

Bibliography

[1] W.-X. Wang, R. Yang, Y.-C. Lai, V. Kovanis, and M. A. F. Harrison, "Time-series–based prediction of complex oscillator networks via compressive sensing," *EPL (Europhysics Letters)*, vol. 94, p. 48006, may 2011.

[2] W.-X. Wang, Y.-C. Lai, C. Grebogi, and J. Ye, "Network reconstruction based on evolutionary-game data via compressive sensing," *Phys. Rev. X*, vol. 1, p. 021021, Dec 2011.

[3] R.-Q. Su, X. Ni, W.-X. Wang, and Y.-C. Lai, "Forecasting synchronizability of complex networks from data," *Phys. Rev. E*, vol. 85, p. 056220, May 2012.

[4] B. Barzel and A.-L. Barabási, "Universality in network dynamics," *Nature Physics*, vol. 9, pp. 673–681, 04 2013.

[5] B. Barzel, Y.-Y. Liu, and A.-L. Barabási, "Constructing minimal models for complex system dynamics," *Nature Communications*, vol. 6, pp. 7186–7193, 05 2015.

[6] G. Quaranta, W. Lacarbonara, and S. F. Masri, "A review on computational intelligence for identification of nonlinear dynamical systems," *Nonlinear Dynamics*, vol. 99, pp. 1709–1761, 01 2020.

[7] S. L. Brunton, J. L. Proctor, and J. N. Kutz, "Discovering governing equations from data by sparse identification of nonlinear dynamical systems," *Proceedings of the National Academy of Sciences*, vol. 113, no. 15, pp. 3932–3937, 2016.

[8] A. Bakhtiarnia, A. Fahim, and E. M. Miandoab, "Data-driven discovery of social network dynamics," in *2020 4th International Conference on Smart City, Internet of Things and Applications (SCIOT)*, pp. 39–45, 2020.

[9] A. Bakhtiarnia, A. Fahim, and E. M. Miandoab, "Parameter identification of complex network dynamics," *Nonlinear Dynamics*, vol. 104, pp. 3991–4005, 06 2021.

[10] A. A. AlMomani, *Prediction Analysis and System Identification of Complex Systems*. PhD thesis, Wallace H. Coulter School Of Engineering, Clarkson University, 08 2019.

[11] Y. Bar-Yam, *Dynamics of Complex Systems*. Reading, Massachusetts: Addison-Wesley, 1997.

[12] J. Ladyman, J. Lambert, and K. Wiesner, "What is a complex system?," *European Journal for Philosophy of Science*, vol. 3, 06 2013.

[13] J. L. Mateos, *Complex Systems and Non-linear Dynamics*, vol. 1, ch. 14, pp. 356–360. Oxford, United Kingdom: Eolss Publishers Co. Ltd., 2009.

[14] A.-L. Barabási and M. Pósfai, *Network Science*. Northeastern University, Boston: Cambridge University Press, 2016.

[15] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, p. 509–512, Oct 1999.

[16] K. A. Zweig, *Network Analysis Literacy: A Practical Approach to the Analysis of Networks*. Vienna, Austria: Springer-Verlag GmbH, 2016.

[17] G. Chen, X. Wang, and X. Li, *Fundamentals of Complex Networks*. Hoboken, New Jersey: John Wiley & Sons, Ltd, 2014.

[18] S. A. Billings, *Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio–Temporal Domains*. Hoboken, New Jersey: John Wiley & Sons, Ltd, 2013.

[19] J. Sjöberg, Q. Zhang, L. Ljung, A. Benveniste, B. Delyon, P.-Y. Glorennec, H. Hjalmarsson, and A. Juditsky, "Nonlinear black-box modeling in system identification: a unified overview," *Automatica*, vol. 31, no. 12, pp. 1691–1724, 1995. Trends in System Identification.

[20] B. de Silva, K. Champion, M. Quade, J.-C. Loiseau, J. Kutz, and S. Brunton, "Pysindy: A python package for the sparse identification of nonlinear dynamical systems from data," *Journal of Open Source Software*, vol. 5, no. 49, p. 2104, 2020.

[21] S. Ha and H. Jeong, "Unraveling hidden interactions in complex systems with deep learning," *Scientific Reports*, vol. 11, p. 12804, 06 2021.

[22] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[23] W. R. Lacerda, L. P. C. da Andrade, S. C. P. Oliveira, and S. A. M. Martins, "Sysidentpy: A python package for system identification using narmax models," *Journal of Open Source Software*, vol. 5, no. 54, p. 2384, 2020.

[24] A.-L. Barabási, R. Albert, and H. Jeong, "Mean-field theory for scale-free random networks," *Physica A: Statistical Mechanics and its Applications*, vol. 272, no. 1, pp. 173–187, 1999.

[25] A. Botchkarev, "A new typology design of performance metrics to measure errors in machine learning regression algorithms," *Interdisciplinary Journal of Information, Knowledge, and Management*, vol. 14, p. 045–076, 2019.

[26] T. Chai and R. R. Draxler, "Root mean square error (rmse) or mean absolute error (mae)? – arguments against avoiding rmse in the literature," *Geoscientific Model Development*, vol. 7, no. 3, pp. 1247–1250, 2014.

[27] Y. Wei, *ESTIMATION, MODEL SELECTION, AND RESILIENCE OF POWER-LAW DISTRIBUTIONS*. doctoralthesis, University of Pittsburgh, June 2016.

[28] F. Hassanibesheli, N. Boers, and J. Kurths, "Reconstructing complex system dynamics from time series: a method comparison," *New Journal of Physics*, vol. 22, p. 073053, jul 2020.

[29] D. Contreras Franco, "Data-driven Dynamics Understanding." `https://github.com/DavidContrerasFranco/Data-driven-Dynamics-Understanding`, 2021.

[30] U. Fasel, E. Kaiser, J. N. Kutz, B. W. Brunton, and S. L. Brunton, "Sindy with control: A tutorial," 2021.

[31] U. Fasel, J. N. Kutz, B. W. Brunton, and S. L. Brunton, "Ensemble-sindy: Robust sparse model discovery in the low-data, high-noise limit, with active learning and control," 2021.

[32] A. A. AlMomani and E. Bollt, "Erfit: Entropic regression fit matlab package, for data-driven system identification of underlying dynamic equations," 2020.