FEDERAL UNIVERSITY OF SANTA CATARINA
TECHNOLOGY CENTER
AUTOMATION AND SYSTEMS DEPARTMENT
UNDERGRADUATE COURSE IN CONTROL AND AUTOMATION ENGINEERING

Pedro Marcolin Antunes

Kernel-Based System Identification with Sparse Dictionary Learning: Applications in Petroleum Reservoir

Florianópolis-SC
2023

Pedro Marcolin Antunes

Kernel-Based System Identification with Sparse Dictionary Learning: Applications in Petroleum Reservoir

Final report of the subject DAS5511 (Course Final Project) as a Concluding Dissertation of the Undergraduate Course in Control and Automation Engineering of the Federal University of Santa Catarina.
Supervisor: Prof. Eduardo Camponogara, Dr.

Florianópolis-SC
2023

Pedro Marcolin Antunes

Kernel-Based System Identification with Sparse Dictionary Learning: Applications in Petroleum Reservoir

This dissertation was evaluated in the context of the subject DAS5511 (Course Final Project) and approved in its final form by the Undergraduate Course in Control and Automation Engineering

Florianópolis, February 24th, 2023.

Prof. Hector Bessa Silveira, Dr.
Course Coordinator

Examining Board:

Prof. Eduardo Camponogara, Dr.
Advisor and Supervisor
UFSC/CTC/DAS

Thiago Lima Silva, Dr.
Evaluator
Research Scientist at SINTEF

Prof. Marcelo De Lellis Costa de Oliveira, Dr.
Board President
UFSC/CTC/DAS

This work is dedicated to my family and friends.

ACKNOWLEDGEMENTS

Research is damn hard. Many times I felt lost, feeling that what I was doing was irrelevant and/or wrong. Then I realized that's how it is, a constant act of humbleness. Being able to look back, revisit, and try to see the big picture. It all pays off in the end. I thank Prof. Camponogara for guiding me through that process over the last two years. He is a true mentor, always watching over and fighting for his student's growth. Thank you.

To my girlfriend and my friends, that provided the best moments of joy, thank you.

Last and most important, I just got here thanks to my family's unconditional love.

“The real is what resists symbolization absolutely.”
(Jacques Lacan)

ABSTRACT

Closed-loop reservoir management typically requires the use of high-fidelity and computationally expensive simulators, where the models need to be executed several times. Proxy modeling consists of a series of methods to build simpler models that aim to reduce computational costs while maintaining adequate levels of accuracy. For the context of reservoir management and optimization, this reduction is crucial and allows the use of techniques that require several simulation iterations. This work proposes a framework for proxy modeling using Kernel-based System Identification and Sparse Dictionary Learning. The models are validated in a synthetic reservoir, with errors between 1% and 2%, and can be used to increase the range of possibilities of control and optimization methods in reservoir management.

Keywords: Reservoir Simulation. Proxy Modeling. Kernel Methods. System Identification.

RESUMO

Gerenciamento de reservatórios em malha fechada tipicamente requer o uso de simuladores de alta fidelidade e alto custo computacional, onde os modelos precisam ser executados diversas vezes. Modelos proxy reúnem uma série de técnicas que buscam reduzir a complexidade do modelo e acelerar sua execução, enqunato mantendo uma precisão adequada. Para o contexto de gerenciamento e otimização de reservatórios, esta redução é crucial para permitir o emprego de técnicas que necessitam diversas iterações de simulação. Esta trabalho propõe um *framework* para síntese de modelos proxy usando métodos de Identificação de Sistemas baseados em *Kernel methods*, juntamente com Aprendizado de Dícionarios Esparsos. Os modelos são aplicados e valiados em um reservatório de petróleo sintético, atingindo erros na faixa de 1% a 2%, podendo servir para aumentar o leque de possibilidades de métodos de controle e otimização de reservatórios.

Palavras-chave: Simulação de Reservatórios. Modelos Aproximados. Kerel Methods. Identificação de Sistemas.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ALD	Almost Linearly Dependent
ANN	Artificial Neural Networks
BHP	Bottom Hole Pressure
EOR	Enhanced Oil Recovery
KRLS	Kernel Recursive Least Squares
KRR	Kernel Ridge Regression
MAPE	Mean Absolute Percentage Error
NPV	Net Present Value
OGI	Oil and Gas Industry
POD	Proper Orthogonal Decomposition
RKHS	Reproducing Kernel Hilbert Space
SDL	Sparse Dictionary Learning
SVD	Singular Value Decomposition

CONTENTS

1 INTRODUCTION

Oil and natural gas are vital assets for the functioning of the world economy, and even with the current trend towards more sustainable energy sources, these assets will still play an important role in the coming decades. The hydrocarbon production chain, from extraction to refining, is fraught with highly complex challenges, among them the synthesis of representative mathematical models for the reservoirs.

Through geological studies and field tests, it is possible to generate high-fidelity computational models of reservoirs. The dynamics of these models are governed by partial differential equations that represent the physical laws of fluid transport, thus defining a large complex nonlinear dynamic system. Using numerical computational methods we can simulate this system, thus obtaining information about the evolution of the system in time.

Since we are dealing with a large scale system, many states, and with complex dynamics, it is natural to expect that its simulation is associated with a high computational cost. In certain contexts, such as reservoir optimization and control, a large volume of simulations and iterative methods are necessary, where this high computational cost imposes a series of limitations.

An alternative is the use of proxy models, which are approximations of lower fidelity, but with much reduced computational cost. These models can be obtained through various methods of system identification, thus obtaining models of various natures, linear or nonlinear, stochastic or deterministic, etc.

If such models have a satisfactory degree of representativeness, they can be used within contexts where high-fidelity models are unusable. The motivating idea for the present work is to develop a proxy model synthesis methodology based on Kernel-based System Identification and Sparse Dictionary Learning. Kernel-based System Identification is a method inspired in machine learning which was first described in (PILLONETTO; DE NICOLAO, 2010). It has been widely studied in recent years in both linear and nonlinear identification.

Kernel-based methods are data-driven methods that typically rely on a large volume of data. In order to mitigate the amount of data needed, a Sparse Dictionary Learning is proposed to discard irrelevant data points.

1.1 OBJECTIVES

The purpose of this work is to study, implement, and evaluate the capacity of the combined Kernel Ridge Regression and Sparse Dictionary Learning to correctly identify dynamical systems, and serve as proxy models.

The complete objectives are:

- To study modern system identification approaches, most based on sparse regularization, and the theoretical foundation of Kernel Methods.
- Implement both Kernel Ridge Regression and Sparse Dictionary Learning algorithms, based on the article (BADDOO et al., 2022).
- Apply and validate the combined method on a simple example system, and to the benchmark petroleum reservoir SPE1, under gas injection.

The main contributions are:

- Application of Kernel Ridge Regression and Sparse Dictionary Learning algorithms to a large scale complex system with control inputs. The original article (BADDOO et al., 2022) only applied it for prediction to the autonomous Lorenz System.
- Development of a proxy model to reproduce a complete reservoir simulation, saturations and pressures, not only well productions.

Chapter 2 consists in a brief introduction to petroleum reservoirs and production system. A modern approach called Regularized System Identification is described in Chapter 3, with an application to the Lorenz Oscillator. The core theoretical foundation of Kernel Methods is exposed in Chapter 4, together with the Kernel Ridge Regression. Chapter 5 presents a Sparse Dictionary Learning that will be used discard redundant data from the data set. Chapter 6 consists of applications in a simple example and in the SPE1 petroleum reservoir. Finally, conclusions are drawn in Chapter 7.

2 PETROLEUM RESERVOIRS

Reservoirs are characterized by a geological trap formed by what is called sealant rock. After the migration processes, the hydrocarbons are trapped inside reservoir rocks, under the sealing rocks, remaining untouched until they are discovered by man through geological processes of oil prospecting (COSSÉ, 1993). To be a reservoir rock, these must have empty spaces inside (porosity) and connections between the pores (permeability) (CRAFT; HAWKINS; TERRY, 1991). To fulfill its role, the sealing rock needs to be impermeable, thus preventing the escape of hydrocarbons out of the reservoir. The Figure 1 shows the structures of a reservoir.

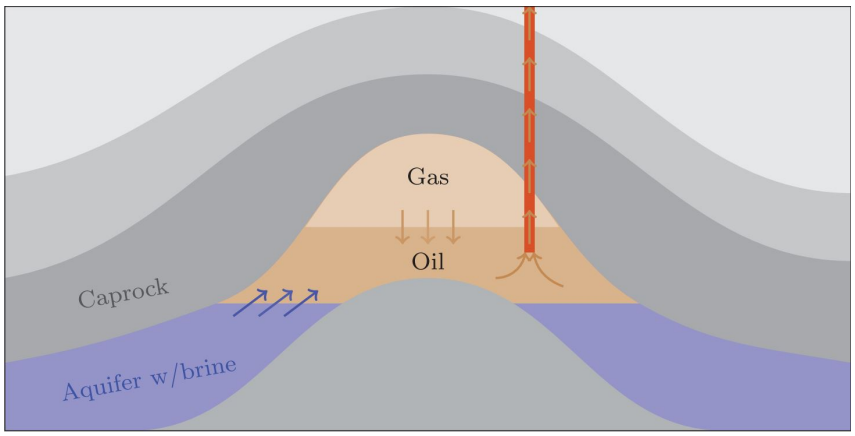

Figure 1 – Reservoir structures illustration. Source: (LIE, 2019)

Depending on the pressure and permeability conditions, the natural gas may be dissolved in the oil or not. In the second case, a gas cap can form in the upper region of the reservoir. There are several classifications as to the type of geological trap and how the components are distributed within the reservoir rock. To thoroughly define each of them is not the objective of this paper.

2.1 OIL RECOVERY METHODS

At the beginning of a reservoir's production, there is a pressure differential between the reservoir and the surface. This condition allows a natural flow of production. As time passes naturally this differential tends to decrease, leading to a slow decrease in production. To extend the life span of the asset, recovery methods, called Enhanced Oil Recovery (EOR), are used to bring this pressure differential back up to desired levels.

Such methods can be classified as primary, secondary, or tertiary. The primary methods seek to reduce the Bottom Hole Pressure (BHP) through equipment connected to the well, to maintain the pressure differential between the reservoir and the well for the flow to occur. After a certain moment, this method becomes ineffective and the

secondary methods come into play, whose principle is the drilling of new wells to inject fluids into the reservoir. These fluids can be polymers, gases, or water, so the accumulation of these components will lead to an increase in the internal pressure of the reservoir, allowing the production to flow. Tertiary methods use chemical-physical manipulation of the reservoir components to change the viscosity of the fluids and their properties.

Each method has its characteristics, as well as its own costs. The economic evaluation of a given intervention requires good models capable of predicting its results.

In this paper, we will specifically study the secondary recovery method based on water injection, Waterflooding. By injecting water we want to move and pressurize the hydrocarbons toward the producing wells, so the injection wells must be positioned correctly and operating at optimal injection rates.

2.2 PRODUCTION OPTIMIZATION

Traditionally the production process in a reservoir works reactively. Water injection rates are set to maximum values and downhole pressures to minimum values. This method is considered aggressive and does not take into account long-term optimization processes of Net Present Value (NPV) and recovery factor.

After a certain period of production, there will come a time when, for a completed well, the production cost will be greater than the revenue, this time is called Water Breakthrough. Naturally, this well is shut in and production continues until all the wells reach this same milestone.

In contrast to the reactive method, one can apply a set of model-based optimal control techniques to maximize the objectives. Such techniques rely on high-quality models of varying degrees of fidelity. This set of techniques, along with production data assimilation, is called Closed Loop Reservoir Management, (SCHIOZER et al., 2022).

The application of closed-loop optimization methods typically involves a higher dynamism of control actions, high variations of injection rates and injection pressures, and a high level of control accuracy. Such dynamism results in better financial returns in the long run.

2.3 RESERVOIRS SIMULATORS

A reservoir simulator, depicted in Figure 2, employs numerical models that represent the reservoir's geophysical characteristics, as well as multi-phase flow patterns and production system characteristics (wells and surface facilities), to analyze and predict the flow of fluids from the reservoir to the surface.

In general, this is an arduous task, as it involves several complex factors, such as the equations governing the dynamics of fluid motion and the physical parameters that

influence the motion (permeability and porosity), which can be estimated using seismic techniques. Since many of these parameters carry high uncertainties, yield predictions can have low accuracy.

To describe component flow processes three items are used. The first is a mathematical model of the flow, usually described by a set of partial differential equations (PDEs), which describe how fluids move through a porous medium. The second item is a geological model of the reservoir that is described by a grid in which each cell has distinct geophysical features that act as inputs to the flow model. Finally, there are the wells and other components of the production system that introduce a communication channel and fluid flow between the reservoir and the surface.

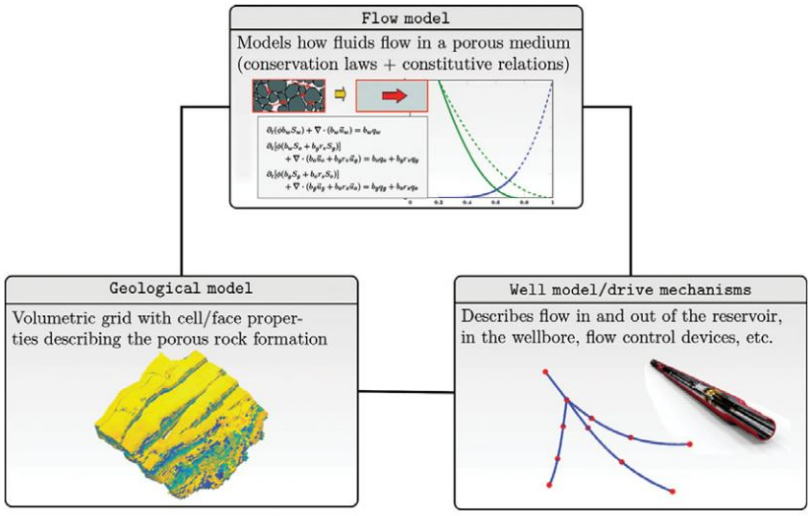

Figure 2 – The building blocks of a reservoir simulator. Fonte: (LIE, 2019)

2.4 PROXY MODELS

The the most widely used reservoir analysis tool is the numerical simulator, but it is always associated with a high computational cost. Given that reservoirs are large-scale dynamic systems represented by a grid with millions or even billions of cells, it is normal that simulations take hours, depending on the degree of fidelity of the model.

In many situations, this high computational cost makes a series of processes unfeasible, especially reservoir and production optimization processes. Optimization processes in general require a high number of simulations, are iterative, and grow in complexity quickly when we wish to consider the constraints of the production system and when we wish to solve robust optimization problems, taking into account the uncertainties of the model.

Within this context arises the possibility of using proxy models, which is nothing more than a statistical mathematical model that seeks to replicate the dynamics of the reservoir approximately. In this way, we drastically reduce the computational cost

associated with a decrease in the quality of the predictions. This balance between simulation speed and accuracy is what defines the quality of a proxy model so that it can be used for certain purposes.

The proxy models can be based on real collected data or on data from high-fidelity numerical simulators, as is the case of this work. They can also have structures that maintain or not physical properties such as conservation of mass and momentum. Models whose structure does not respect the real dynamics of the reservoir and simply seek to represent input-output relationships are said to be purely data-driven models.

There are several methods for synthesizing proxy models, the choice of method will depend on the context in which it will be applied. Among them are:

1. Statistical Methods;
2. Reduced Order Models;
3. Machine Learning Methods.

2.5 MACHINE LEARNING METHODS

In this work, we propose a Machine Learning method to identify reservoir proxy models. Machine learning models have been widely applied in reservoir proxy modeling. Initial approaches sought to build a black-box model of inputs-outputs, not taking into consideration the internal states of the reservoir. These models predict productions in the wells based on past input controls, injections and BHP.

Later approaches started to build models to reproduce the simulators results, where all cells are represented, and the evolution of the states is given by a machine learning model. Methods based on Artificial Neural Networks (ANN) have been proposed several times, (TOMPSON et al., 2016), (SAGHEER; KOTB, 2019), and (NAVRATIL et al., 2020), and are the most common method in the literature.

The method proposed in this work aims to reproduce the results of the simulator, being able to predict pressures and saturations in all cells. It differs from black-box approaches where the aim is to only predict productions. We make use of Kernel Methods and Sparse Dictionary Learning, which will be presented in Chapters 4 and 5, respectively.

3 REGULARIZED SYSTEM IDENTIFICATION

3.1 SPARSITY AND COMPRESSED SENSING

Sparsity plays a massive role in signal and image processing and compression. For example, let us take the image space of a one-megapixel image ($10^3 \times 10^3$ pixels), represented in Figure 3. Considering a grayscale image, with pixel value ranging from 0 to 255, there are 256^{10^6} possible images, a pretty big number. Suppose we would consider only the space of natural images. In that case, this can be pictures of every single moment of your life from all possible angles, fitted in a one-megapixel image. This space is only a tiny little portion of the whole image space.

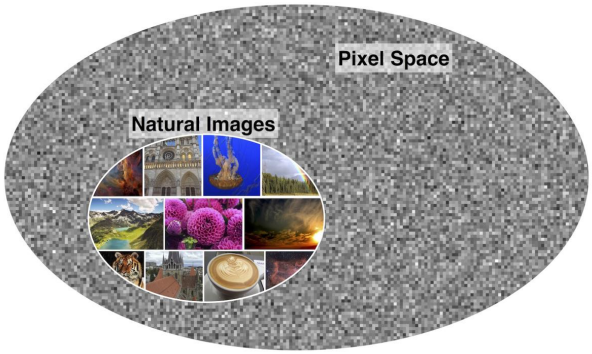

Figure 3 – Natural images represent only an extremely tiny portion of the pixel space. Source: (BRUNTON; KUTZ, 2019).

An image represented by a vector $\mathbf{x} \in \mathbb{R}^n$ can be decomposed into a vector of weights $\mathbf{x} \in \mathbb{R}^n$ and a universal basis $\mathbf{\Psi} \in \mathbb{R}^{n \times n}$ for the pixel space:

$$\mathbf{x} = \mathbf{\Psi s} \tag{1}$$

with $\mathbf{s}$ having only very few nonzero elements. This is why images and signals are so compressible: we can define them only by very few terms, which can be transmitted at low cost and then reconstructed using the universal basis. Even the small active terms can be neglected because their impact on the final image would be so small that humans could not even notice. This is the core of image and audio compression. For example, JPEG images use the Discrete Fourier Transform (DFT) as a universal basis.

We can apply a similar strategy to dynamical systems. The feature space of all possible combinations of up to degree *d* polynomial terms forming an ODE scales quickly as *d* rises. Combine this with trigonometric and exponential functions and your feature space grows ever larger. The same principle from image compression applies in this context. An evolution trajectory of dynamical systems decomposed as a vector of weights and a basis given by the feature space would probably result in a sparse weight vector.

3.2 SPARSE IDENTIFICATION OF NONLINEAR DYNAMICS (SINDY)

Discovering dynamical systems models from data is a central challenge in mathematical physics, with a rich history going back at least as far as the time of Kepler and Newton and the discovery of the laws of planetary motion. Historically, this process relied on a combination of high-quality measurements and expert intuition. With vast data and increasing computational power, the automated discovery of governing equations and dynamical systems is a new and exciting scientific paradigm (BRUNTON; KUTZ, 2019).

Traditional system identification methods choose the structure of the model based on prior physics knowledge of the process or based on heuristics. After that, the parameters are selected in a way that the model fits the training data according to some criteria.

Recent system identification algorithms, such as the Sparse Identification of Nonlinear Dynamics (SINDy) (BRUNTON; PROCTOR; KUTZ, 2016), combine both tasks of defining the structure and fitting the parameters.

In many dynamical systems with complex behaviors, this behavior is a function of only a few terms. For example, the Lorenz Attractor in Figure 4, whose behavior is chaotic and extremely dependent on the initial conditions, can be expressed only by quadratic and bilinear terms

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= x(\rho - z) - y \\ \dot{z} &= xy - \beta z. \end{aligned} \tag{2}$$

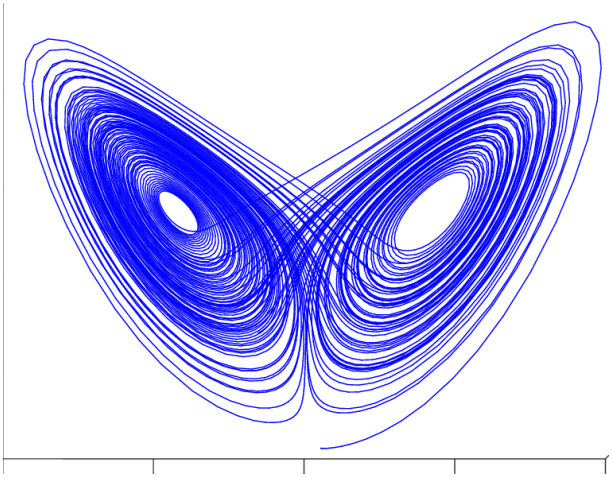

Figure 4 – The Lorenz Attractor. Source: (HART; HOOK; DAWES, 2020).

In general, most processes can be modeled using ordinary differential equations, in the form

$$\frac{d}{dt}\mathbf{x} = \mathbf{f}(\mathbf{x}). \tag{3}$$

In SINDy we aim to approximate $\mathbf{f}$ by a linear model

$$\mathbf{f}(\mathbf{x}) \approx \sum_{k=1}^{p} \theta_k(\mathbf{x})\xi_k = \boldsymbol{\Theta}(\mathbf{x})\boldsymbol{\xi} \tag{4}$$

with $\boldsymbol{\xi}$ being as sparse as possible, just selecting the terms that are relevant to the dynamics. This is achieved by using sparsity-inducing regularizers, which will be presented later in this chapter.

The workflow starts by collecting data from an experiment and stacking it as row vectors

$$\mathbf{X} = [\mathbf{x}(t_1)\ \mathbf{x}(t_2)\ \cdots\ \mathbf{x}(t_m)]^T. \tag{5}$$

where $\mathbf{x}(t_1) \in \mathbb{R}^{n\times 1}$ denotes the measurement at time t_1. Similarly, we construct the matrix of derivatives

$$\dot{\mathbf{X}} = [\dot{\mathbf{x}}(t_1)\ \dot{\mathbf{x}}(t_2)\ \cdots\ \dot{\mathbf{x}}(t_m)]^T. \tag{6}$$

In practice, the derivatives are hardly ever measured, leading to the necessity of computing them numerically. This can be a hard task for noisy data. Several algorithms compute numerically robust derivatives, starting from the basic finite differences until the more complex regularized methods.

The second step is to build a library of candidate nonlinear terms $\boldsymbol{\Theta}(\mathbf{X})$:

$$\boldsymbol{\Theta}(\mathbf{X}) = [\mathbf{1}\ \mathbf{X}\ \mathbf{X}^{P_2}\ \cdots\ \mathbf{X}^{P_d}\ \cdots\ \sin(\mathbf{X})\ \cdots]. \tag{7}$$

The polynomial terms $\mathbf{X}^{P_2}$,$\mathbf{X}^{P_3}$, etc., are denoted as

$$\mathbf{X}^{P_2} = \begin{bmatrix} x_1^2(t_1) & x_1(t_1)x_2(t_1) & \cdots & x_2^2(t_1) & \cdots & x_n^2(t_1) \\ x_1^2(t_2) & x_1(t_2)x_2(t_2) & \cdots & x_2^2(t_2) & \cdots & x_n^2(t_2) \\ \vdots & \vdots & \ddots & \vdots & \ddots & \cdots \\ x_1^2(t_m) & x_1(t_m)x_2(t_m) & \cdots & x_2^2(t_m) & \cdots & x_n^2(t_m) \end{bmatrix}, \tag{8}$$

containing the polynomial relations between states. This library can contain any type of term, polynomial, trigonometric, exponential, etc. The dynamical system (3) can then be approximated by the matrix product

$$\dot{\mathbf{X}} = \boldsymbol{\Theta}(\mathbf{X})\boldsymbol{\Xi}. \tag{9}$$

Here, each column ξ_k of $\boldsymbol{\Xi}$ represents weights associated with the active terms in the k-th row of $\boldsymbol{\Phi}(\mathbf{X})$, these active terms will determine the dynamics of the state associated with the given row. Once $\boldsymbol{\Xi}$ is determined, we can reconstruct each row of the governing equations, each state, as:

$$\dot{\mathbf{x}}_k = f_k(x) = \boldsymbol{\Theta}(\mathbf{x}^T)\xi_k. \tag{10}$$

For estimating each column ξ_k we define a convex minimization problem, using the l_1 norm as a regularizer:

$$\xi_k = \arg\min_{\xi} \left\| \dot{\mathbf{X}}_k - \Theta(\mathbf{X})\xi \right\|_2 + \lambda \|\xi\|_1 , \tag{11}$$

here $\dot{\mathbf{X}}_k$ denotes the k-column of $\dot{\mathbf{X}}$.

For most physical systems, the right-hand side of equation (3) has only a few terms. Therefore, the solution to the regression problem must be sparse in a high-dimension feature space. Notice that the columns of $\Theta(\mathbf{X})$ are the features which are combined linearly by ξ_k to yield the k-th column of $\dot{\mathbf{X}}$. Ideally, the usage of the l_0 norm ($\|\cdot\|_0$) would induce a highly sparse solution, given that it penalizes all nonzero entries of ξ_k. The l_0 norm is not a convex function, using it would result in a NP-Hard problem, which in practice cannot be solved for large problems.

The authors (BRUNTON; PROCTOR; KUTZ, 2016) propose a relaxation to the l_1 norm, as formulated in (11). Although the l_1 norm is not differentiable, it is convex, resulting in a Convex Optimization problem.

3.3 SEQUENTIAL THRESHOLDED LEAST-SQUARE (STLS)

The problem (11) is also called the LASSO regression and can be solved by traditional optimization methods, such as sub-gradient and proximal gradient methods. Depending on the size of the problem, such methods can become intractable.

The authors of SINDy propose a heuristic algorithm based on two steps. The first one is to perform a traditional least-squares without regularization. The second step is to analyze the solution weights and cut off those that do not pass an arbitrary threshold.

Although the algorithm is relatively simple, it possesses strong properties on convergence and quality of the solution, as deeply elaborated in (ZHANG; SCHAEFFER, 2018).

3.3.1 Convergence analysis

For an integer $n \in \mathbb{N}$, let $[n] \subset \mathbb{N}$ be the set defined by $[n] := 1, 2, ..., n$. The support set of a vector $x \in \mathbb{R}^n$ is the set of indices corresponding to nonzero elements:

$$\text{supp}(x) := \{j \in [n] : x_j \neq 0\}. \tag{12}$$

The number of nonzero elements is defined as:

$$\|x\|_0 = \text{card}(\text{supp}(x)), \tag{13}$$

and the vector x is denoted as s-sparse if it has at most s nonzero elements, $\|x\|_0 \leq s$. The complete procedure is given in Algorithm 2.

Algorithm 1: The SINDy algorithm (BRUNTON; PROCTOR; KUTZ, 2016).

Input: $m \geq n; \lambda; \mathbf{A} \in \mathbb{R}^{m \times n}$ with $\text{rank}(\mathbf{A}) = n; b \in \mathbb{R}^m$.
Set $k = 0$; Initialize $x^0 = \mathbf{A}^\dagger b$ and $S^{-1} = \varnothing$;
Set $S^k = \{j \in [n] : |x_j^k| \geq \lambda\}$;
while $S^k \neq S^{k-1}$ **do**
  $x^{k+1} = \arg\min \|\mathbf{A}x - b\|_2$ such that $\text{supp}(x) \subseteq S^k$;
  $S^{k+1} = \{j \in [n] : |x_j^{k+1}| \geq \lambda\}$;
  $k = k + 1$;
end
Output: x^k.

The value of $\lambda > 0$ is a hyper-parameter that must be chosen such that $S^0 \neq \varnothing$. The following theorems are proved, and the complete proofs are detailed in the original article (ZHANG; SCHAEFFER, 2018).

Theorem 3.3.1 (On the Convergence of the SINDy Algorithm.) *Assume that $m \geq n$. Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ with $\|\mathbf{A}\|_2 = 1$, $b \in \mathbb{R}^m$ and $\lambda > 0$. Let x^k be the sequence generated by Algorithm 2. Defining the objective function F by*

$$F := \|\mathbf{A}x - b\|_2^2 + \lambda^2 \|x\|_0 , \; x \in \mathbb{R}^n. \tag{14}$$

The following propositions are true

1. *x^k converges to a fixed point of the Algorithm 2 in at most n steps;*
2. *a fixed point of the algorithm is a local minimizer of F;*
3. *a global minimizer of F is a fixed point of the scheme;*
4. *x^k strictly decreases F unless the iterates are stationary.*

These results provide a solid foundation for the method, as well as justify the performance of the algorithms in previous applications.

3.3.2 Example: Chaotic Lorenz System

.

In this section, we will apply the SINDy algorithm for identification of the Lorenz System equations. The ODEs that describe the systems are defined in (2). The generated data is stacked in a matrix $\mathbf{X}$, where each row is a snapshot at a given time. Similarly, the time derivatives are stacked in a matrix $\dot{\mathbf{X}}$. Here is assumed that the derivatives are measured without any noise, this will be further discussed.

The parameters are $\sigma = 10$, $\beta = 8/3$ and $\rho = 28$. To build the data matrices, the system is being sampled with a period $\Delta t = 10^{-3}$. The next step is to build the library of terms, in this example, we include all polynomial combinations up to degree five.

$$\Theta(\mathbf{X}) = \begin{bmatrix} | & | & | & | & | & | & | & | & | & | \\ \mathbf{1} & \mathbf{x}(t) & \mathbf{y}(t) & \mathbf{z}(t) & \mathbf{x}(t)^2 & \mathbf{x}(t)\mathbf{y}(t) & \mathbf{y}(t)^2 & \mathbf{z}(t)\mathbf{y}(t) & \cdots & \mathbf{z}(t)^5 \\ | & | & | & | & | & | & | & | & | & | \end{bmatrix}. \tag{15}$$

Solving the regression problem (11), with the algorithm proposed in 2 and $\lambda = 0.025$, returns the coefficients displayed in Table 1.

Table 1 – Coefficients identified by the SINDy algorithm, applied to the Lorenz System.

	ξ_1	ξ_2	ξ_3
1	0	0	0
x	-10	28	0
y	10	-1	0
z	0	0	-2.6667
xx	0	0	0
xy	0	0	1
xz	0	-1	0
yy	0	0	0
yz	0	0	0
...	...	...	...
yzzzz	0	0	0
zzzzz	0	0	0

As we can see, SINDy algorithm precisely identifies the correct coefficients, which leads to an exact prediction, as seen in Figure 5.

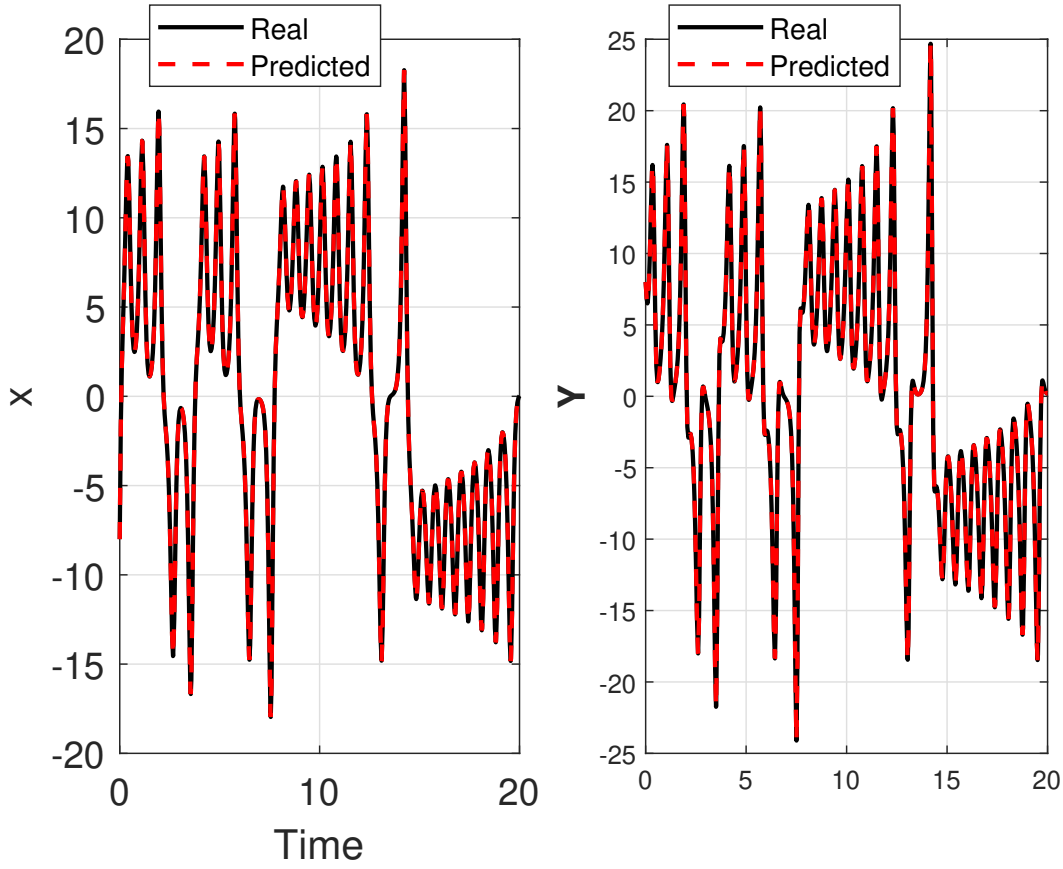

Figure 5 – A comparison between real measured data and the identified model prediction. On the left the x coordinate, and on the right the y coordinate. Source: Author.

The algorithm does not have this perfect performance when the derivatives are corrupted with noise. Using the same setup, but adding a zero-mean white noise with variance $\sigma = 1$ to the results of the derivative in incorrect identification. In Figure 6 we can see that the identified model fails in predicting the states after the time $t = 6.8$.

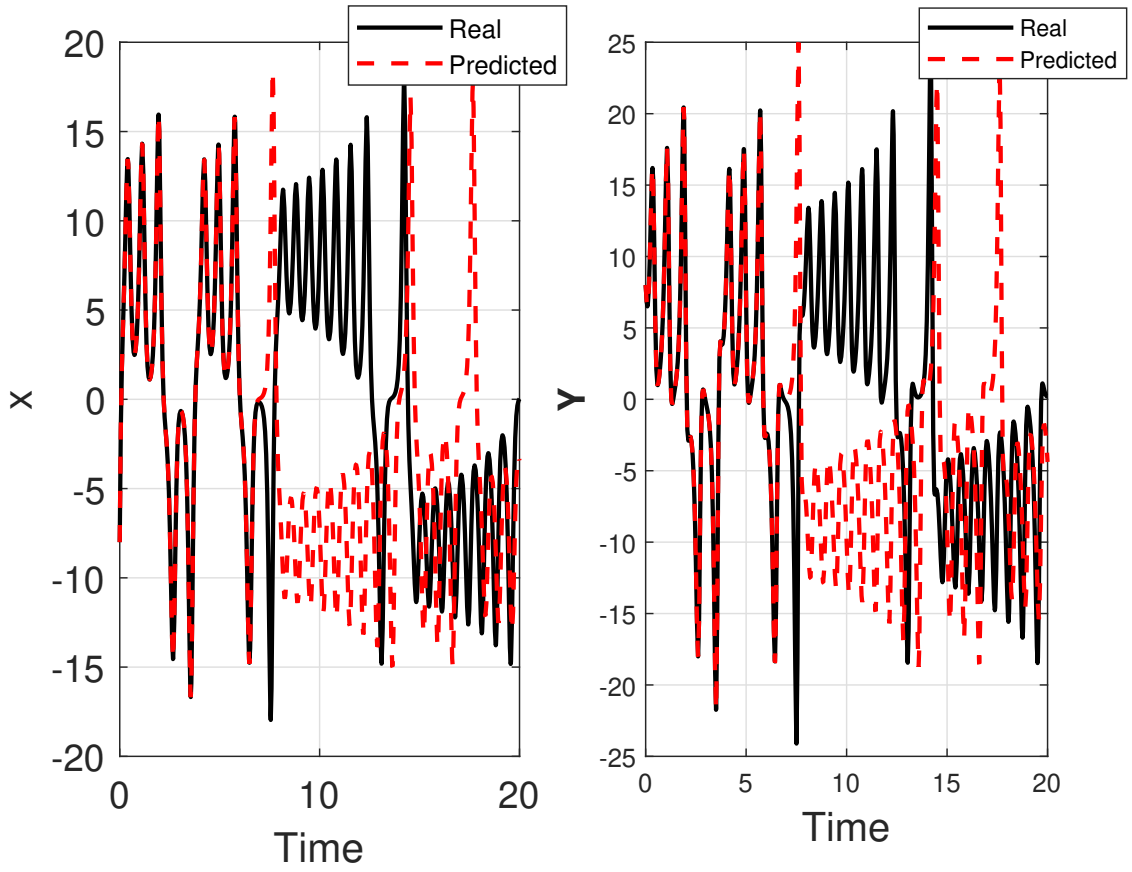

Figure 6 – Impact of measurement white noise with variance $\sigma = 1$ in the derivatives. On the left the x coordinate and in the right the y. Source: Author.

In a practical situation, the data is always corrupted with noise and measurement errors. There are several robust numerical differentiation methods, such as the Total-Variation Regularization (TVR) method to overcome this problem.

3.4 SINDY WITH CONTROLS

For real-world applications, we aim to identify systems that include inputs and controls

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x} = f(\mathbf{x}, \mathbf{u}), \tag{16}$$

with states $\mathbf{x} \in \mathbb{R}^n$ and control inputs $\mathbf{u} \in \mathbb{R}^q$. The original SINDy algorithm is not built for this task.

In (FASEL et al., 2021) the authors propose a framework for SINDy coupled with a Model Predictive Control (MPC). The framework is capable of accurately identifying models with inputs, as well as controlling them in a feedback structure.

In a very similar way, the snapshots of both states and controls are stacked into

two matrices:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_m \end{bmatrix}^T \tag{17}$$
$$\mathbf{U} = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_m \end{bmatrix}^T .$$

In this case, we must also include the control snapshots matrix ($\mathbf{U}$) in the library of terms, as well as combinations between states and controls.

$$\mathbf{\Theta}(\mathbf{X}, \mathbf{U}) = \begin{bmatrix} \mathbf{1} & \mathbf{X} & \mathbf{U} & (\mathbf{X} \otimes \mathbf{X}) & (\mathbf{X} \otimes \mathbf{U}) & (\mathbf{U} \otimes \mathbf{U}) & \cdots \end{bmatrix}, \tag{18}$$

where $(\mathbf{X} \otimes \mathbf{U})$ defines the vector of all product combinations of the components in $\mathbf{u}$ and $\mathbf{x}$, in a similar way as done in (8).

It is crucial that the library includes the true terms of the dynamic, otherwise, SINDy would not successfully build the correct model. After this point, the procedure is identical to the original algorithms, which perform the regression problem using STLS. In Figure 7 we can see the steps of the framework, applied to an infectious disease control model.

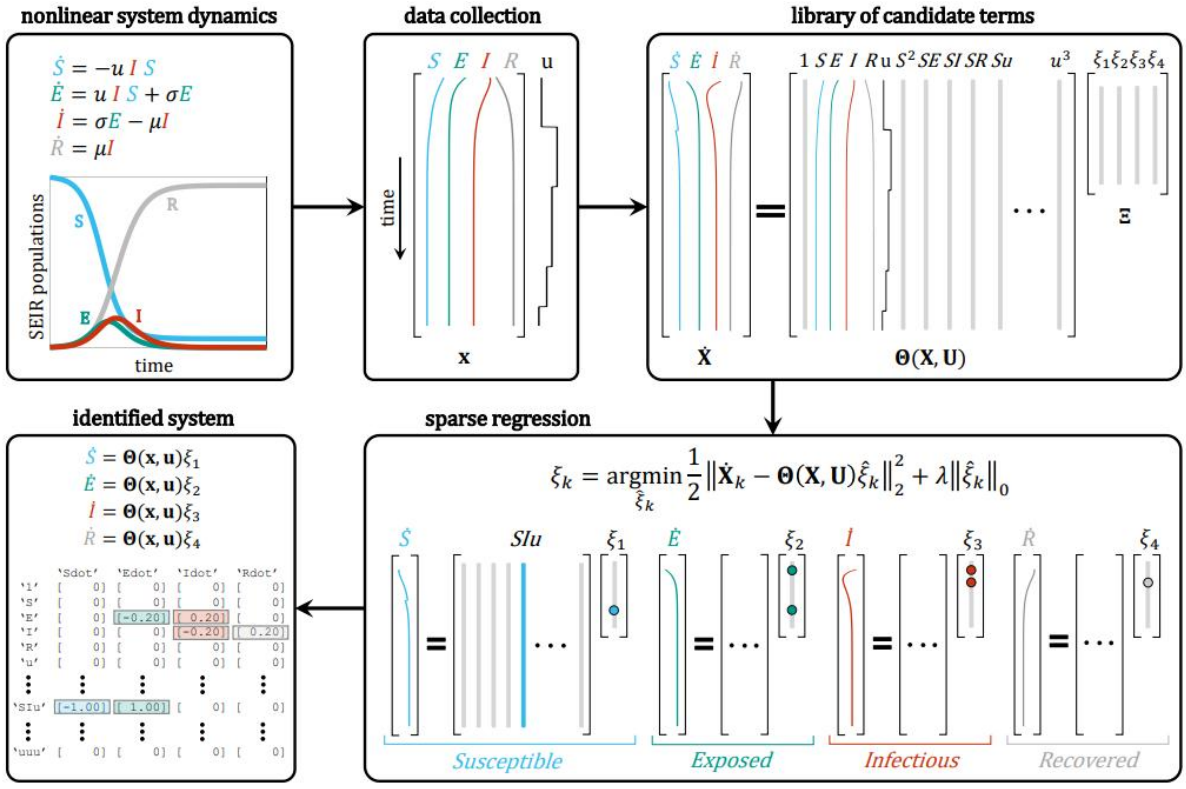

Figure 7 – Schematic of the SINDy with control algorithm. Active terms in a library of candidate nonlinearities are selected via sparse regression. Source: (FASEL et al., 2021).

3.5 SINDY LIMITATIONS AND DRAWBACKS

SINDy algorithm has been successfully applied to a large range of processes. The models identified are interpretable and they usually require less data to train, when

compared to neural networks.

Besides this, SINDy still demonstrates a few important limitations and drawbacks. Perhaps the most important one is the sensitivity with respect to the quality of the derivatives measurement or estimation. From the Lorenz example in Section 3.3.2, it can be seen that the algorithms lose performance in prediction with training data corrupted with noise. This issue will not be explored further in this work, rather we will explore another problem.

Examples of dynamical systems with a high number of states are not rare. For example, in the Oil and Gas Industry (OGI) there are complex processes in the production system and in reservoir management. For systems with dozens of states, the library Θ of candidate terms would be prohibitively large, with respect to memory and computational power.

There are a few alternatives to overcome the high dimensionality problem, one is to project the system in a low dimensional space through Proper Orthogonal Decomposition (POD), which is a common practice in fluid mechanics. This method tailors a hierarchy orthogonal basis for the system based on the Singular Value Decomposition (SVD).

In this work, we try to overcome the problem of dimensionality by the usage of Kernel Methods. These methods allow us to build the library Θ implicitly, using the information of inner-products. This theory will be further developed in Chapter 4.

4 KERNEL METHODS

4.1 INTRODUCTION

Suppose you have a set of training data

$$(x_1, y_1), ..., (x_m, y_m) \in \mathcal{X} \times \{-1, +1\}. \tag{19}$$

How to determine the class (± 1) of a new instance (x_{m+1})? This is the classical problem of binary classification in learning theory. The classification is done by determining a degree of similarity between the new instance (also called input or measurement) and the previous training set. The method used to calculate this degree of similarity lies in the field's core.

In this context, kernel methods arise as a powerful tool to pairwise compare two objects. A kernel function can be described as a function in the form

$$k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$$
$$(x, x') \mapsto k(x, x'),$$

which returns a real number that represents a degree of similarity between two elements in $\mathcal{X}$. We can think of the dot product as a similarity measure between two vectors $x, x' \in \mathcal{R}^N$, defined as

$$\langle x, x' \rangle = \sum_{i=1}^{N} [x]_i [x']_i. \tag{20}$$

where $[x]_i$ denotes the i^{th} element of the vector x.

Geometrically, if both vectors are normalized to length 1, the dot product will give the cosine of the angle between the vectors. In other words, the similarity will be maximum, equal to one, if they are in the same direction, and minimum, equal to zero, if they are orthogonal.

For many problems in Machine Learning and System Identification, similarity measurements that are "richer" than the canonical dot product in $\mathbb{R}^N$ are desired. For instance, one can design a nonlinear mapping (Φ) that takes elements in some set $\mathcal{X}$ to a higher dimension, even infinite, feature space $\mathcal{H}$:

$$\Phi : \mathcal{X} \to \mathcal{H}$$
$$x \mapsto \Phi(x).$$

In this enhanced feature space $\mathcal{H}$, patterns in data unseen in the original space $\mathcal{X}$ may arise. For example, consider the binary classification problem in a scenario where the data is not linearly separable. There may exist a mapping (Φ) to a higher-dimension space where the data might be linearly separable, as is illustrated in Figure 8.

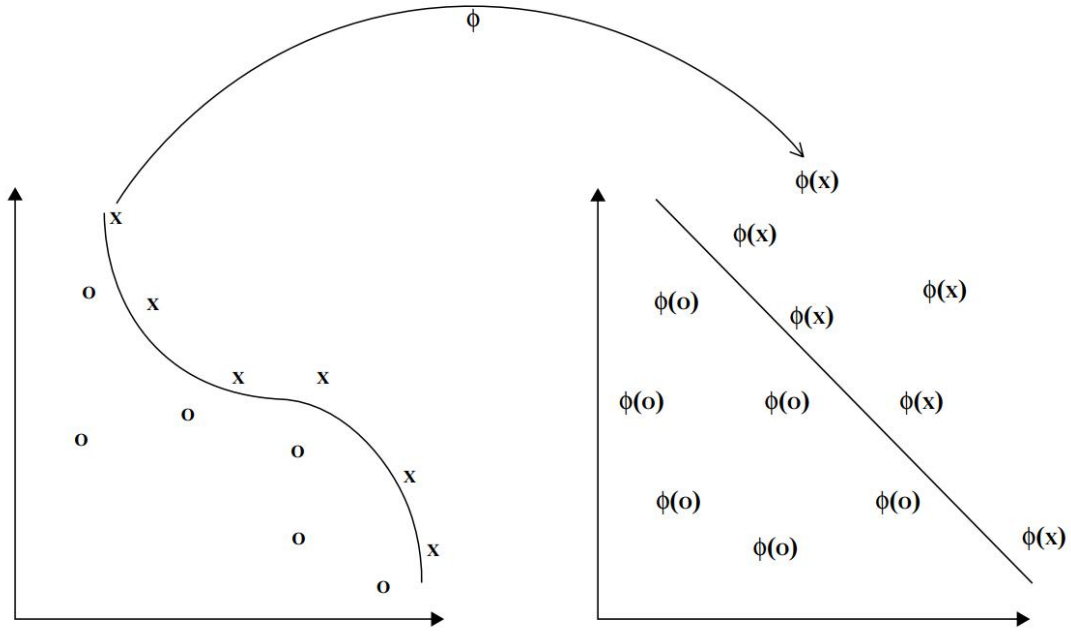

Figure 8 – The mapping Φ embeds the data into a feature space where the nonlinear pattern now becomes linear. Source: (SHAWE-TAYLOR; CRISTIANINI, 2004).

Kernel Functions can be seen as powerful tools for computing inner products in a higher dimensional feature space $\mathcal{H}$, built by a nonlinear mapping Φ, without ever explicitly computing any element of $\mathcal{H}$. In other words, we seek kernel functions where the following relation holds

$$k(x, x') = \left\langle \Phi(x), \Phi(x') \right\rangle_{\mathcal{H}}. \tag{21}$$

Notice that if the relation (21) is true, there is no need to explicitly compute the mapping $\Phi(\mathbf{x})$ in the feature space, for an input $\mathbf{x}$, for decision problems that can be expressed in terms of inner-products of input data in a feature space given by $\Phi(\mathbf{x})$. This indirect computation can be performed for any decision problem that can be expressed in terms of kernel functions, a tool which is known as the *Kernel Trick*.

As an illustrative example, suppose one wishes to construct a feature space of unordered degree-two monomials to perform a Machine Learning task. In this case $\mathcal{X} = \mathbb{R}^2$, and

$$\begin{aligned} &\Phi : \mathcal{X} = \mathbb{R}^2 \mapsto \mathcal{H} = \mathbb{R}^3 \\ &x = (x_1, x_2) \mapsto \Phi(x) = (x_1^2, x_1 x_2, x_2^2). \end{aligned} \tag{22}$$

The inner product in $\mathcal{H}$ is defined by

$$\left\langle \Phi(x), \Phi(x') \right\rangle = x_1^2 x_1'^2 + x_1 x_2 x_1' x_2' + x_2^2 x_2'^2.$$

where $x = (x_1, x_2)$ and $x' = (x_1', x_2')$

This is a computationally inexpensive inner-product in $\mathbb{R}^3$, but for higher order monomials, it scales rapidly. For N-dimensional input vectors and d-degree monomials there exists

$$\binom{d+N-1}{d} = \frac{(d+N-1)!}{d!(N-1)!} \tag{23}$$

different monomials of degree d. By defining (22) slightly differently,

$$\begin{aligned} \Phi : \mathcal{X} &\mapsto \mathcal{H} = \mathbb{R}^3 \\ (x_1, x_2) &\mapsto (x_1^2, \sqrt{2}x_1x_2, x_2^2), \end{aligned} \tag{24}$$

the inner-product in $\mathcal{H}$ would be

$$\left\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \right\rangle_{\mathcal{H}} = x_1^2 x_1'^2 + 2x_1x_2x_1'x_2' + x_2^2x_2'^2 = (x_1x_1' + x_2x_2')^2 = \left\langle x, x' \right\rangle_{\mathcal{X}}^2 .$$

Thus, choosing a kernel function to be $k(x, x') = \left\langle x, x' \right\rangle^2$, allows us to compute the inner-product in $\mathcal{H}$ without ever constructing it.

4.2 REPRODUCING KERNEL HILBERT SPACE

The example from the last section raises the question: does (21) hold for every kernel function? If not, which properties must a kernel possess for it to hold? We will see that yes, there is a family of kernels for which (21) is true. To demonstrate it, we first need the definition of positive definite kernels.

Definition 4.2.1 (Gram Matrix) *Given a function $k : \mathcal{X}^2 \mapsto \mathbb{K}$ (where $\mathbb{K} = \mathbb{C}$ or $\mathbb{K} = \mathbb{R}$) and input patterns $x_1, ..., x_m \in \mathcal{X}$, the $m \times m$ matrix K with elements*

$$[K]_{ij} = k(x_i, x_j) \tag{25}$$

is called the Gram matrix (or kernel matrix) of k with respect to $x_1, ..., x_m$ (SCHÖLKOPF; SMOLA, 2018).

Definition 4.2.2 (Positive Definite Kernel) *A function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is a positive definite (p.d.) kernel if it is symmetric*

$$\forall (x, x') \in \mathcal{X}^2, k(x, x') = k(x', x)$$

and satisfies, for all $N \in \mathbb{N}$, $(x_1, x_2, ..., x_N) \in \mathcal{X}^N$ and $(a_1, a_2, ..., a_N) \in \mathbb{R}^N$:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j k(x_i, x_j) \geq 0.$$

*In other words, a kernel k is p.d. if, and only, if the similarity matrix $[K]_{ij} = k(x_i, x_j)$ is positive **Semidefinite**.*

The simplest example of a p.d. kernel is the linear kernel. Let $\mathcal{X} = \mathbb{R}^d$, and the function $k : \mathcal{X}^2 \mapsto \mathbb{R}$ be defined by:

$$\forall (x, x') \in \mathcal{X}^2,\ k(x, x') = \left\langle x, x' \right\rangle_{\mathbb{R}^d}. \tag{26}$$

We can see that by property of norms, $\left\langle x, x' \right\rangle_{\mathbb{R}^d} = \left\langle x', x \right\rangle_{\mathbb{R}^d}$, defining its symmetry. And the positive definiteness is given by

$$\sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j \left\langle x_i, x_j \right\rangle_{\mathbb{R}^d} = \left\| \sum_{i=1}^{N} a_i x_i \right\|_{\mathbb{R}^d}^2 \geq 0. \tag{27}$$

See that the example of degree two polynomial kernel in (24), was also a positive definite kernel.

All p.d. kernels hold an important property given by Aronszajn's Theorem.

Theorem 4.2.1 (Aronszajn's Theorem) *k is a positive definite kernel on the set $\mathcal{X}$ if, and only if, there exists a Hilbert space $\mathcal{H}$ and a mapping*

$$\Phi : \mathcal{X} \mapsto \mathcal{H} \tag{28}$$

such that, for any $x, x' \in \mathcal{X}$:

$$k(x, x') = \left\langle \Phi(x), \Phi(x') \right\rangle_{\mathcal{H}}. \tag{29}$$

The Hilbert space $\mathcal{H}$ associated with a p.d. kernel is called a Reproducing Kernel Hilbert Space (RKHS), and the kernel is called a reproducing kernel (r.k.).

Revisiting the example (24), we can see that $k(x, x') = \left\langle x, x' \right\rangle^2$ is positive definite, as all norms, and the associated RKHS of k is $\mathcal{H} = \mathbb{R}^3$.

Another important Theorem arises concerning the uniqueness of a r.k.

Theorem 4.2.2 (Uniqueness of r.k. and RKHS) *If $\mathcal{H}$ is a RKHS, then it has a unique r.k. k. Conversely, a function k can be the r.k. of at most one RKHS.*

Stated the existence of a RKHS associated with all p.d. kernels, now we will show properties of those spaces that allow us to make use of the kernel trick for ML problems. One does not even need to know precisely the composition of the RKHS in application to machine learning problems, as long as the kernel is p.d. it can be used to propose candidate functions to a given problem.

4.3 REPRESENTER THEOREM

The RKHS $\mathcal{H}$ is a vector space of functions from $\mathcal{X}$ to $\mathbb{R}$. Each data point $x \in \mathcal{X}$ is represented by a function $\Phi(x) = K_x$ in $\mathcal{H}$.

For example, for any $x, y \in \mathcal{X} = \mathbb{R}$, we may choose to represent them as a Gaussian function, as in Figure 9,

$$\Phi(x) : t \mapsto e^{-\frac{1}{\sigma^2}(x-t)^2} \tag{30}$$

$$\Phi(y) : t \mapsto e^{-\frac{1}{\sigma^2}(y-t)^2}. \tag{31}$$

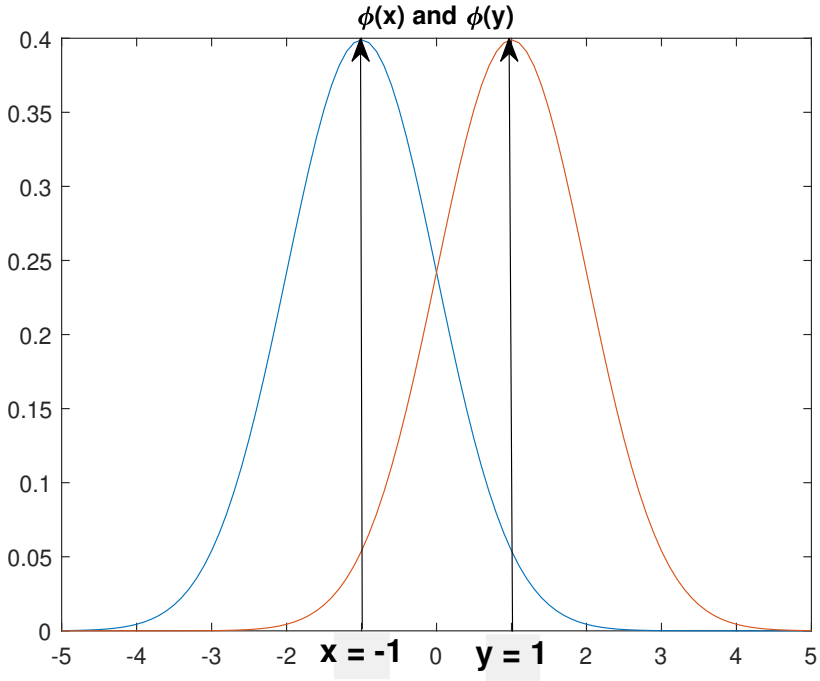

Figure 9 – Example of two Gaussian functions with the same standard deviation, in the RKHS. Source: Author.

By using the Gaussian kernel, which is p.d., we can evaluate the inner product of these functions in the RKHS defined by the kernel

$$k(x, y) = e^{-\frac{1}{\sigma^2}(x-y)^2} = \langle \Phi(x), \Phi(y) \rangle_{\mathcal{H}}. \tag{32}$$

With that, we can compare the degree of similarity between two functions living in the RKHS. Since the RKHS is a vector space, it not only contains Gaussian functions but also all linear combinations of those functions. In this way, the RKHS induced by the Gaussian kernel is much richer and contains much more than Gaussian functions.

The most important advantage of using kernel functions is that we can perform a linear search of a candidate solution of a decision problem living in the space of functions that is the RKHS. This is possible by the Representer Theorem stated below.

Theorem 4.3.1 (Representer Theorem) *Let $\mathcal{X}$ be a set endowed with a p.d. kernel K, $\mathcal{H}$ the corresponding RKHS, and $\mathcal{S} = \{x_1, ..., x_m\} \subseteq \mathcal{X}$ a finite set of points in $\mathcal{X}$. Also,*

let $\Psi : \mathbb{R}^{m+1} \mapsto \mathbb{R}$ *be a function of* $m+1$ *variables, strictly increasing with respect to the last variable. Then, any solution to the optimization problem:*

$$\min_{f \in \mathcal{H}} \Psi(f(x_1), ..., f(x_m), \|f\|_{\mathcal{H}}), \tag{33}$$

admits a representation of the form:

$$\forall x \in \mathcal{X},\ f(x) = \sum_{i=1}^{m} \alpha_i k(x_i, x) = \sum_{i=1}^{n} \alpha_i K_{x_i}(x) = \langle f, K_x \rangle_{\mathcal{H}} = \langle f, \Phi(x) \rangle_{\mathcal{H}}\,. \tag{34}$$

In other words, the solution lies in a finite-dimensional subspace

$$f \in \textbf{\textit{Span}}(K_{x_1}, ..., K_{x_m}).$$

and it can be represented by a linear expression, as an inner product.

In simpler words, the solution of a problem such as (33) can be expressed as a linear combination of functions in the RKHS. Those functions are the mapping $\Phi(x)$ of data points x in $\mathcal{X}$ into functions K_x in $\mathcal{H}$.

The Theorems and tools presented give us a simple framework to solve ML problems:

- Map the data points $x \in \mathcal{X}$ to a high-dimensional Hilbert space $\mathcal{H}$ (the RKHS) through a kernel mapping $\Phi : \mathcal{X} \mapsto \mathcal{H}$, with $\Phi(x) = K_x$.
- In $\mathcal{H}$, consider simple linear models $f(x) = \langle f, \Phi(x) \rangle_{\mathcal{H}}$.
- If $\mathcal{X} = \mathbb{R}^p$, a linear function in $\Phi(x)$ may be nonlinear in x.
- Express the decision problem in terms of inner-products and solve it linearly in the RKHS using the Representer Theorem.

4.4 EXAMPLE: KERNEL RIDGE REGRESSION

In this section, we show how we can structure a Ridge Regression problem in terms of kernel functions. Let $\mathbf{x} = (x_1, x_2, ..., x_n)^T \in \mathbb{R}^n$ be the input vector, $y \in \mathbb{R}$ be the output, $\mathcal{S}_m = (\mathbf{x}_i, \mathbf{y}_i)_{i=1,..,m} \in \mathbb{R}^n \times \mathbb{R}$ a training set of m pairs. The goal is to find a linear function $f : \mathbb{R}^n \mapsto \mathbb{R}$ to predict $y = f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = \langle \mathbf{w}, \mathbf{x} \rangle$ where $\mathbf{w} \in \mathbb{R}^n$.

The function f is the solution to the given problem:

$$\mathbf{w} = \arg\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \lambda \|\mathbf{w}\|^2\,. \tag{35}$$

Here $\lambda \geq 0$ is the regularizing parameter that prevents overfitting by penalizing non-smooth functions. By convention, the matrix $\mathbf{X}$ and vector $\mathbf{y}$ store the elements as

row vectors,

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_m^T \end{bmatrix}, \ \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}. \tag{36}$$

The problem (35) is convex and differentiable with respect to $\mathbf{w}$, therefore its solution can be found by setting the gradient of the objective function to zero. This way the solution is given by

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_m)^{-1}\mathbf{X}^T\mathbf{y}. \tag{37}$$

For $\lambda \geq 0$ the inverse of the matrix $(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I}_m)$ always exists. The computational cost of finding the solution $\mathbf{w}$ is the cost of solving the inverse of $(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I}_m)$ and it requires $\mathcal{O}(n^3)$ operations.

Notice that the solution is not expressed in terms of inner-products, therefore we cannot apply the kernel trick yet. Let $\mathbf{P}$ be an $n \times m$ matrix and $\mathbf{Q}$ be a $m \times n$ matrix, we can use the following relation to express the solution of (35) in different terms

$$(\mathbf{P}\mathbf{Q} + \mathbf{I}_N)^{-1}\mathbf{P} = \mathbf{P}(\mathbf{Q}\mathbf{P} + \mathbf{I}_M)^{-1}. \tag{38}$$

With that we can rewrite (37) as

$$\mathbf{w} = \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I}_m)^{-1}\mathbf{y}. \tag{39}$$

Notice that the term $\mathbf{X}\mathbf{X}^T$ is the matrix of componentwise inner-products,

$$\mathbf{X}\mathbf{X}^T = \begin{bmatrix} \mathbf{x}_1^T\mathbf{x}_1 & \mathbf{x}_1^T\mathbf{x}_2 & \dots & \mathbf{x}_1^T\mathbf{x}_m \\ \mathbf{x}_2^T\mathbf{x}_1 & \mathbf{x}_2^T\mathbf{x}_2 & \dots & \mathbf{x}_2^T\mathbf{x}_m \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_m^T\mathbf{x}_1 & \mathbf{x}_m^T\mathbf{x}_2 & \dots & \mathbf{x}_m^T\mathbf{x}_m \end{bmatrix} = \begin{bmatrix} \langle \mathbf{x}_1, \mathbf{x}_1 \rangle & \langle \mathbf{x}_1, \mathbf{x}_2 \rangle & \dots & \langle \mathbf{x}_1, \mathbf{x}_m \rangle \\ \langle \mathbf{x}_2, \mathbf{x}_1 \rangle & \langle \mathbf{x}_2, \mathbf{x}_2 \rangle & \dots & \langle \mathbf{x}_2, \mathbf{x}_m \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \mathbf{x}_m, \mathbf{x}_1 \rangle & \langle \mathbf{x}_m, \mathbf{x}_2 \rangle & \dots & \langle \mathbf{x}_m, \mathbf{x}_m \rangle \end{bmatrix}. \tag{40}$$

As we can see, now the solution $\mathbf{w}$ is written as a function of inner-products of inputs. Using the kernel trick the inner-products in the original input space ($\mathbb{R}^N$) can be replaced by a kernel function that represents the inner-product evaluation in the RKHS

$$\mathbf{\Phi}(\mathbf{X})\mathbf{\Phi}(\mathbf{X})^T = \mathbf{K} = \begin{bmatrix} \langle \mathbf{\Phi}(\mathbf{x}_1), \mathbf{\Phi}(\mathbf{x}_1) \rangle & \langle \mathbf{\Phi}(\mathbf{x}_1), \mathbf{\Phi}(\mathbf{x}_2) \rangle & \dots & \langle \mathbf{\Phi}(\mathbf{x}_1), \mathbf{\Phi}(\mathbf{x}_m) \rangle \\ \langle \mathbf{\Phi}(\mathbf{x}_2), \mathbf{\Phi}(\mathbf{x}_1) \rangle & \langle \mathbf{\Phi}(\mathbf{x}_2), \mathbf{\Phi}(\mathbf{x}_2) \rangle & \dots & \langle \mathbf{\Phi}(\mathbf{x}_2), \mathbf{\Phi}(\mathbf{x}_m) \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \mathbf{\Phi}(\mathbf{x}_m), \mathbf{\Phi}(\mathbf{x}_1) \rangle & \langle \mathbf{\Phi}(\mathbf{x}_m), \mathbf{\Phi}(\mathbf{x}_2) \rangle & \dots & \langle \mathbf{\Phi}(\mathbf{x}_m), \mathbf{\Phi}(\mathbf{x}_m) \rangle \end{bmatrix}. \tag{41}$$

Where $k(\mathbf{x}_i, \mathbf{x}_j) = \left\langle \mathbf{\Phi}(\mathbf{x}_i), \mathbf{\Phi}(\mathbf{x}_j) \right\rangle$. The matrix of inner-products in (41) is the Gram Matrix ($\mathbf{K}$) defined in (25). Now we write the solution of (35) as

$$\mathbf{w} = \mathbf{\Phi}(\mathbf{X})^T(\mathbf{K} + \lambda\mathbf{I}_m)^{-1}\mathbf{y}, \tag{42}$$

Notice that the mapping Φ was also applied to the first term of the equation, we will see further that it does not need to be computed explicitly. When comparing (42) with (37) we can see that now finding the solution requires computing the inverse of $(\mathbf{K} + \lambda \mathbf{I}_m)$, whose cost is $\mathcal{O}(m^3)$, depending on the number of inputs of the training set S.

By calling $\boldsymbol{\alpha} = (\mathbf{K} + \lambda \mathbf{I}_m)^{-1}\mathbf{y}$, we express $\mathbf{w}$ as

$$\mathbf{w} = \Phi(X)^T \boldsymbol{\alpha}. \tag{43}$$

With this formulation we can make predictions of an unseen input point $\mathbf{z}$, we write $\mathbf{z}$ instead of $\mathbf{x}$ to distinguish it from the matrix of training inputs $\mathbf{X}$,

$$f(\mathbf{z}) = \langle \mathbf{w}, \mathbf{z} \rangle = \mathbf{w}^T \mathbf{z} = \boldsymbol{\alpha}^T \Phi(\mathbf{X}) \Phi(\mathbf{z}). \tag{44}$$

The output $f(\mathbf{z})$ can express as a summation

$$f(\mathbf{z}) = \sum_{i=1}^{m} \boldsymbol{\alpha_i} \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{z}) \rangle = \sum_{i=1}^{m} \boldsymbol{\alpha_i} k(\mathbf{x_i}, \mathbf{z}). \tag{45}$$

The inner-product in the expression above can be replaced by the kernel function $k(\mathbf{x}_i, \mathbf{z}) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{z}) \rangle$. By doing this replacement the space of candidate solutions of (35), that originally was the space of linear functions $\mathbf{y} = f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} = \langle \mathbf{w}, \mathbf{x} \rangle$, will now be all functions $f(\mathbf{x}) \in \mathcal{H}$, where $\mathcal{H}$ is the RKHS associated with the chosen p.d. kernel. Remember that the inner-products in the matrix $\mathbf{K}$ used to compute $\boldsymbol{\alpha}$ also have to be replaced by the same kernel function used in (45).

This example demonstrates precisely the framework presented in the Section 4.3. By simply expressing the problem as inner-products of the training data and replacing it with a kernel function, we were able to search for a solution in a high-dimension space without ever explicitly constructing any vector $\Phi(\mathbf{x})$.

The same approach used in this simple example will be carried on to more advanced methods described further in this document.

5 SPARSE DICTIONARY LEARNING

In the last chapter, it was presented the kernel ridge regression example. By the representer theorem, the problem:

$$\arg\min_{\mathbf{W}} \|\mathbf{Y} - \mathbf{W}k(\mathbf{X}, \mathbf{X})\|_F + \lambda R(f), \tag{46}$$

has a closed form solution

$$f(x) = \sum_{i=1}^{m} \alpha_i k(x_i, x), \tag{47}$$

with

$$\alpha = (\mathbf{K} - \lambda \mathbf{I}_m)^{-1}\mathbf{Y}. \tag{48}$$

There are a few practical and theoretical problems with this solution. The first one is due to numerical instability. Although the kernel matrix has full rank, it presents a large condition number. That is, it is very sensitive to perturbations in the input data. The condition number of the pseudoinverse will also be large.

By the choice of the kernel function choice, the Reproducing Kernel Hilbert Space (RKHS) may be a very rich space of functions. Depending on the data set, this may lead to over-fitting the data even in the presence of regularization.

Perhaps the most important issue is with respect to the computational complexity. In system identification one usually needs a large number of data point for two reasons. First, if the derivatives are computed numerically, their accuracy will depend on the sample period. Second, for nonlinear systems we typically aim to capture the different types of nonlinearities, this evolves exciting the system in different states. These two reasons result in a large data set.

For a large number $m \gg 1$ of data points, constructing the pseudoiverse $k(\mathbf{X}, \mathbf{X})^{\dagger}$ requires $O(m^3)$ operations to construct and $O(m^2)$ space in memory, witch may scale to a prohibitive scenario. Besides the training, using the model for prediction requires the multiplication of the $m \times n$ weight matrix $\boldsymbol{\alpha}$ by the m-vector of kernel evaluations $k(\mathbf{X}, x)$.

To solve these problems, the authors in (ENGEL; MANNOR; MEIR, 2004) developed the method KRLS. The aim of the method is to build a sparse dictionary $\tilde{\mathbf{X}}$ as a subset of the original data set $\mathbf{X}$. The key idea is to use only the necessary data point that capture most of the underlying dynamics, thereby mitigating the practical problems cited above.

To built the dictionary, each new data point is compared against the current dictionary and is determined by some criteria whether or not it should enter the dictionary. The criteria used is the ALD test. It checks if the current data point can be expressed as a linear combination of the current dictionary, if not, it its added. This method typically results in a very reduced sparse dictionary, as it will be seen in the applications chapter.

5.1 MATHEMATICAL FORMULATION

Let us define the dictionary $D_t = \{\tilde{x}_j : j = 1, ..., \tilde{m}_t\}$ at time t as a collection of vectors $\tilde{x}_j$, which is initialized a time $t = 1$ as $D_1 = \{x_1\}$. The matrices of snapshots up to time t are

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ x_1 & x_2 & \dots & x_t \\ | & | & & | \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} | & | & & | \\ y_1 & y_2 & \dots & y_t \\ | & | & & | \end{bmatrix}. \tag{49}$$

The current dictionary is represented as

$$\tilde{\mathbf{X}}_t = \begin{bmatrix} | & | & & | \\ \tilde{x}_1 & \tilde{x}_2 & \dots & \tilde{x}_t \\ | & | & & | \end{bmatrix} \tag{50}$$

and in the feature space

$$\mathbf{\Phi}_t = \begin{bmatrix} | & | & & | \\ \mathbf{\Phi}(\tilde{x}_1) & \mathbf{\Phi}(\tilde{x}_2) & \dots & \mathbf{\Phi}(\tilde{x}_t) \\ | & | & & | \end{bmatrix}. \tag{51}$$

To determine if a new sample x_t is added, or not, to the dictionary, we perform the ALD test. The test consists in comparing the minimum square distance of the sample and the current dictionary span.

$$\delta_t = \min_{\boldsymbol{\pi}_t} \|\mathbf{\Phi}(x_t) - \mathbf{\Phi}_{t-1}\boldsymbol{\pi}_t\|_2^2. \tag{52}$$

The scalar δ_t is the minimum distance, and $\boldsymbol{\pi}_t$ is the linear combination of the dictionary $\mathbf{\Phi}_{t-1}$ that minimizes δ_t. The next step is to compare δ_t against an arbitrary threshold ν. If $\delta_t > \nu$ the sample brings new information and should be included in the dictionary, otherwise not. As discussed in Chapter 4, we can express the problem (52) in terms of inner-products, never explicitly computing $\mathbf{\Phi}(x)$. Let $\boldsymbol{\pi}_t = (a_1, a_2, ...a_{m_{t-1}})$, then

$$\delta_t = \min_{\boldsymbol{\pi}_t} \left(\sum_{i,j=1}^{m_{t-1}} a_i a_j \left\langle \mathbf{\Phi}(\tilde{x}_i), \mathbf{\Phi}(\tilde{x}_j) \right\rangle - 2 \sum_{j=1}^{m_{t-1}} a_j \left\langle \mathbf{\Phi}(\tilde{x}_j), \mathbf{\Phi}(x_t) \right\rangle + \left\langle \mathbf{\Phi}(x_t), \mathbf{\Phi}(x_t) \right\rangle \right). \tag{53}$$

By replacing the inner-products with kernel functions we obtain

$$\begin{aligned} \delta_t = \min_{\boldsymbol{\pi}_t} \left(\sum_{i,j=1}^{m_{t-1}} a_i a_j k(\tilde{x}_i, \tilde{x}_j) - 2 \sum_{j=1}^{m_{t-1}} a_j k(\tilde{x}_j, x_t) + k(x_t, x_t) \right) \\ = \min_{\boldsymbol{\pi}_t} \left(\boldsymbol{\pi}_t^T \tilde{\mathbf{K}}_{t-1} \boldsymbol{\pi}_t - 2\boldsymbol{\pi}_t^T \tilde{\mathbf{k}}_{t-1}(x_t) + k_{tt} \right), \end{aligned} \tag{54}$$

where

$$[\tilde{\mathbf{K}}_{t-1}]_{i,j} = k(\tilde{x}_i, \tilde{x}_j) \tag{55}$$

$$(\tilde{\mathbf{k}}_{t-1}(x_t))_i = k(\tilde{x}_i, x_t) \tag{56}$$

$$k_{tt} = k(x_t, x_t) \tag{57}$$

with $i, j = 1, 2, ..., m_{t-1}$. This is a quadratic programming problem, which admits an analytical solution

$$\boldsymbol{\pi}_t = \tilde{\mathbf{K}}_{t-1}^{-1} \tilde{\mathbf{k}}_{t-1}(x_t), \tag{58}$$

$$\delta_t = k_{tt} - \tilde{\mathbf{k}}_{t-1}^{*}(x_t)\boldsymbol{\pi}_t. \tag{59}$$

Remember that for positive definite kernels, $\tilde{\mathbf{k}}_{t-1}$ always has an inverse. Batch computing $\tilde{\mathbf{K}}$ and $\tilde{\mathbf{K}}^{-1}$ for every new sample is not computationally efficient. This matrices can be recursively updated as

$$\tilde{\mathbf{K}}_t = \begin{bmatrix} \tilde{\mathbf{K}}_{t-1} & \tilde{\mathbf{k}}_{t-1} \\ \tilde{\mathbf{k}}_{t-1}^{*} & k_{tt} \end{bmatrix}, \quad \tilde{\mathbf{K}}_t^{-1} = \begin{bmatrix} \tilde{\mathbf{K}}_{t-1}^{-1} + \boldsymbol{\pi}_t\boldsymbol{\pi}_t^{*}/\delta_t & -\boldsymbol{\pi}_t/\delta_t \\ -\boldsymbol{\pi}_t^{*}/\delta_t & 1/\delta_t \end{bmatrix} \tag{60}$$

The formulations in (60) are mathematically correct for $\nu = 0$. For values of $\nu > 0$ there will be residuals in each iteration, that may accumulate over time. Although this issue is not relevant with Gaussian kernels, as discussed in (ENGEL; MANNOR; MEIR, 2004), for polynomial kernels, it may lead to numerical instability, due to the large condition number associated with the Gram matrix $\tilde{\mathbf{K}}_t$.

To overcome this instability issue, in (BADDOO et al., 2022) the authors propose the usage of a Cholesky decomposition of the matrices in (60). The matrix $\tilde{\mathbf{K}}_t$ is positive semidefinite by definition, but in practice it is usually positive definite. All positive definite matrices have a Cholesky decomposition, which is written into the product of a lower triangular matrix and its conjugate transpose.

$$\mathbf{A} = \mathbf{L}\mathbf{L}^{*}. \tag{61}$$

In fact, the decomposition also exists for positive semidefinite matrices, but it holds slightly different properties. With that we can write $\tilde{\mathbf{K}}_t = \mathbf{C}_t\mathbf{C}_t^{*}$, where $\mathbf{C}_t$ is the lower triangular matrix with dimensions $\tilde{m}_t \times \tilde{m}_t$.

Instead of updating $\tilde{\mathbf{K}}_t$ recursively as shown in (60), we update its decomposition

$$\mathbf{C}_t = \begin{bmatrix} \mathbf{C}_{t-1} & \mathbf{0} \\ \mathbf{s}_t^{*} & c_t \end{bmatrix}. \tag{62}$$

The initial value is defined as $\mathbf{C}_1 = \sqrt{k_{11}}$, the term $\mathbf{s}_t = \mathbf{C}_{t-1}^{-1}\tilde{\mathbf{k}}_t$ takes $O(\tilde{m}_t^2)$ operations to compute, and $c_t = \sqrt{k_{tt} - \mathbf{s}_t^{*}\mathbf{s}_t}$. It is also important to avoid imaginary values of c_t that may appear due to rounding error. This is done by using the maximum between c_t and 0, $c_t = \max(0, \sqrt{k_{tt} - \mathbf{s}_t^{*}\mathbf{s}_t})$.

The most important feature of this method is that, at any point, we explicitly compute the matrices $\tilde{\mathbf{K}}_t$ and $\tilde{\mathbf{K}}_t^{-1}$. The complete dictionary learning algorithm takes $O(m\tilde{m}^2 + nm\tilde{m})$ operations to complete, where m is the total number of data points, and $\tilde{m}$ is the final number of data points in the dictionary. The complete algorithm follows.

Algorithm 2: Sparse ALD dictionary learning with Cholesky updates (BADDOO et al., 2022).

Input: data matrix $\mathbf{X}$, kernel k, sparsification tolerance ν.
for $t = 1 \rightarrow m$ **do**
  Select new sample x_t;
  Compute $\tilde{\mathbf{k}}_{t-1}(x_t)$ with (56) ;
  Compute π_t with backsubstitution (58) ;
  Compute δ_t using (59) ;
  if $\delta_t \leq \nu$ **then**
    Maintain the dictionary: $D_t = D_{t-1}$;
  else if $\delta_t > \nu$ **then**
    Update the dictionary $D_t = D_{t-1} \cap \{x_t\}$;
    Update the Cholesky factor $\mathbf{C}_t$ with (62)
end

Although not efficient, the problem (52) can always be batch computed at each new data point, this would be the ground truth. In Figure 10 the Cholesky update method is compared against the original KRLS formulation, and the ground truth. The data set used is from a discretized version of the viscous Burgers' equation

$$u_t = \nu u_{xx} - uu_x, \tag{63}$$

on 1024 grid points.

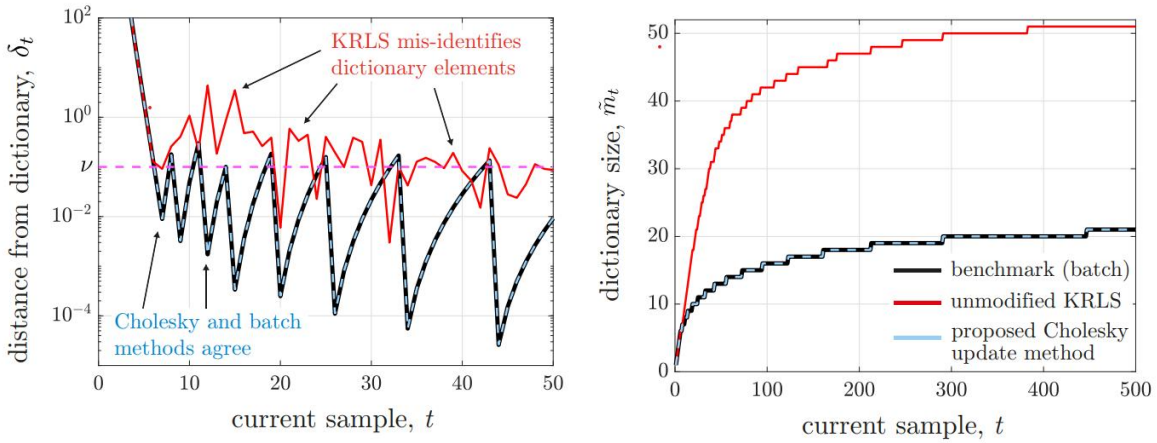

Figure 10 – Comparison of the ALD dictionary computed by the original KRLS, the Cholesky updates, and batch computing. All methods are using a quadratic kernel $k = (1 + x^T x)^2$ and a sparsity parameter $\nu = 0.1$. On the left, the distance δ_t at the current sample, and the threshold for adding, or not, to the dictionary. On the right, the current dictionary size. Source: (BADDOO et al., 2022).

It is clear from Figure 10 that the approach with Cholesky updates has better performance. The original KRLS method misidentifies the data point, while the Cholesky results in the same identification as the batch method. This is one instance of application, results may not follow for different data sets.

In the next chapter, this procedure will be used in combination with Kernel Ridge Regression to identify dynamical systems, with and without control inputs.

6 APPLICATIONS

In this chapter, we will present applications of the combined KRR and SDL framework. The framework will be applied to two instances, a simple nonlinear system and the three-phase oil reservoir SP1.

6.1 NUMERICAL DIFFERENTIATION

Numerical differentiation is the process of approximating the derivative of a function at a given point using numerical methods. One common method is Finite Difference, which involves calculating the slope of the function using the values of the function at nearby points. Another method is called symbolic differentiation, in which the derivative is represented using mathematical symbols rather than approximating it using numerical values. These methods are commonly used in Calculus and Numerical Analysis.

From Calculus the definition of the derivative is given by

$$\frac{\mathrm{d}f(t)}{\mathrm{d}t} = \lim_{\Delta t \to 0} \frac{f(t+\Delta t) - f(t)}{\Delta t}. \tag{64}$$

The derivative is the slope of the function at that point. The idea is that as Δt goes to zero, the right-hand side of the equation goes to the instantaneous slope. In practice, with Δt sufficiently small, the approximation becomes fairly accurate. The error associated with approximating the derivative can be quantified using Taylor series expansions.

$$f(t+\Delta t) = f(t) + \Delta t \frac{\mathrm{d}f(t)}{\mathrm{d}t} + \frac{\Delta t^2}{2!}\frac{\mathrm{d}^2 f(t)}{\mathrm{d}t^2} + \frac{\Delta t^3}{3!}\frac{\mathrm{d}^3 f(t)}{\mathrm{d}t^3} + \cdots \tag{65}$$

$$f(t-\Delta t) = f(t) - \Delta t \frac{\mathrm{d}f(t)}{\mathrm{d}t} + \frac{\Delta t^2}{2!}\frac{\mathrm{d}^2 f(t)}{\mathrm{d}t^2} - \frac{\Delta t^3}{3!}\frac{\mathrm{d}^3 f(t)}{\mathrm{d}t^3} + \cdots. \tag{66}$$

The simplest numerical differentiation method is the Forward Difference, where (64) is approximated by

$$\frac{\mathrm{d}f(t)}{\mathrm{d}t} \approx \frac{f(t+\Delta t) - f(t)}{\Delta t}. \tag{67}$$

If we substitute $f(t+\Delta t)$ in (67) by the expression in (65), the resulting expression is

$$\frac{\mathrm{d}f(t)}{\mathrm{d}t} \approx \frac{\mathrm{d}f(t)}{\mathrm{d}t} + \frac{\Delta t}{2!}\frac{\mathrm{d}^2 f(t)}{\mathrm{d}t^2} + \frac{\Delta t^2}{3!}\frac{\mathrm{d}^3 f(t)}{\mathrm{d}t^3} + \cdots \tag{68}$$

By the resulting expression, we can see that the error associated with the Forward Difference approximation is dominated by Δt, since it is a very small number, the higher-order terms can be neglected. In this case, the error scales linearly with Δt.

A second method is called Central Difference, which is defined as

$$\frac{\mathrm{d}f(t)}{\mathrm{d}t} \approx \frac{f(t+\Delta t) - f(t-\Delta t)}{2\Delta t}. \tag{69}$$

Doing the same analysis with the Taylor series expansion, the error associated with this method can be expressed as

$$\frac{\mathrm{d}f(t)}{\mathrm{d}t} \approx \frac{\mathrm{d}f(t)}{\mathrm{d}t} + \frac{\Delta t^2}{3!}\frac{\mathrm{d}^3 f(t)}{\mathrm{d}t^3} + \frac{\Delta t^4}{5!}\frac{\mathrm{d}^5 f(t)}{\mathrm{d}t^5} \cdots \tag{70}$$

In this case, the error is dominated by the Δt^2 term, which means that the error is reduced at a faster rate as Δt is decreased when compared to the Forward Difference method.

When dealing with time-series data, or online data, the Central Differences method can not be used to calculate the derivative of the first sample, the past value of $f(t - \Delta t)$ is not known, and the same happens with the last sample. At those points, Forward Difference and Backward Difference must be used.

In this work, we will be using Central Differences for estimating all the derivatives. Since all data will be provided from noiseless experiments, the results are adequate.

6.2 NONLINEAR SYSTEM WITH SINGLE FIXED POINT

The first application is a simple nonlinear autonomous dynamical system:

$$\begin{aligned} \dot{x_1} &= \mu x_1 \\ \dot{x_2} &= \lambda(x_2 - x_1^2). \end{aligned} \tag{71}$$

As seen in Figure 11, the system has a single equilibrium point at $x_2 = x_1^2$ and $x_1 = 0$, for $\lambda < \mu < 0$.

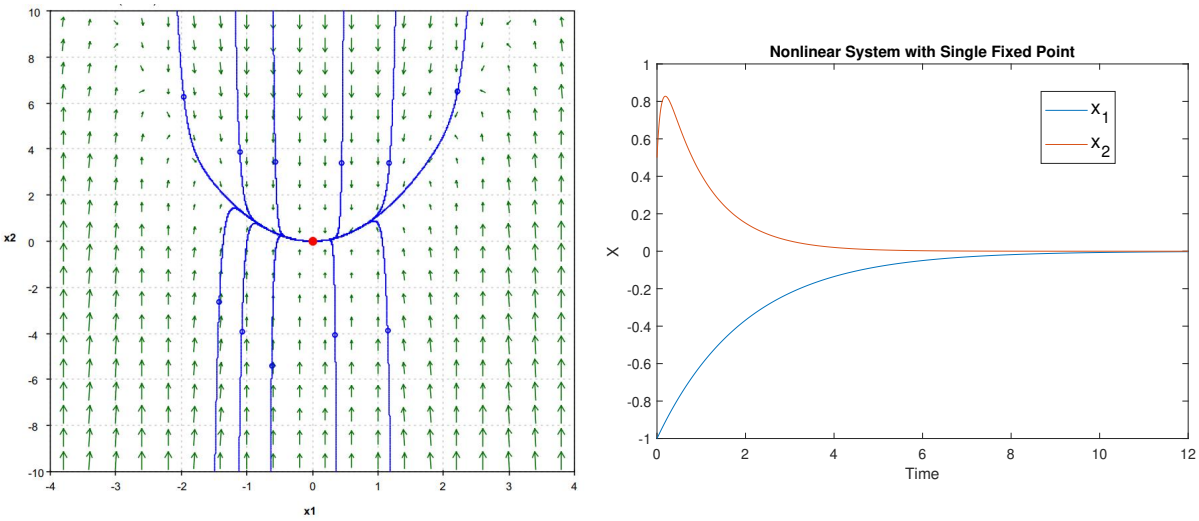

Figure 11 – Phase plane and simulation of the system described in (71), for parameters $\mu = -0.5$ and $\lambda = -10$, and initial condition $(x_1, x_2) = (-1, 0.5)$ The system has a single stable equilibrium point. Source: Author.

By using a fixed time-step of 0.001 s, and a total simulation time of 12 s, we collect the data corresponding to the evolution of the states over time. The Central

Table 2 – The number of samples selected by SDL with different values of ν. Source: Author.

ν	Number of samples selected
10^{2}	1
10^{1}	1
10^{0}	2
10^{-1}	3
10^{-2}	5
10^{-4}	6
10^{-10}	6

Difference method discussed above is applied to numerically estimate the derivatives, the result is seen in Figure 12.

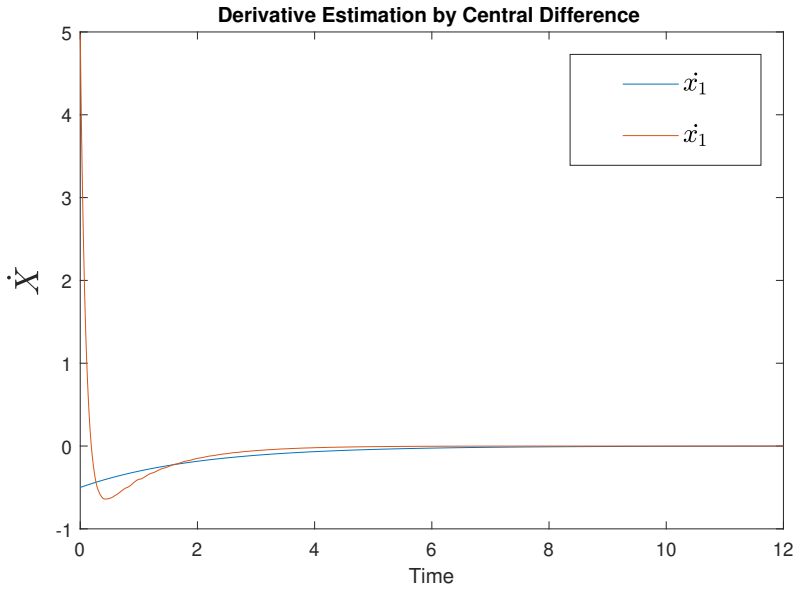

Figure 12 – Time derivatives of states x_1 and x_2 over time. Source: Author.

Using the Mean Absolute Percentage Error (MAPE) metric

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n} \left|\frac{y_i - \hat{y}_i}{y_i} * 100\right|, \tag{72}$$

the numerical differentiation results in an error of 0.0236%. The simulation resulted in 12000 samples. Using the SDL method, we will try to reduce this number, selecting only the most relevant data points.

For this system, the quadratic kernel $k = (x^T x + c)^2$ will be used, with $c = 1$. In Table 2 we can see the number of samples selected, for different values of ν (threshold for selecting a sample). For values of $\nu > 10^{-4}$, the number of selected samples stays at six, and those six will be the ones that we will use for the identification.

With the reduced dictionary, we will use those samples to perform a KRR, according to Section 4.4, to identify a non-parametric model in the form:

$$f(x) = \sum_{i=1}^{6} a_i k(x_i, x) = \sum_{i=1}^{6} a_i (x_i^T x + 1)^2. \tag{73}$$

Notice that the sum is up to $i = 6$ because we choose 6 samples. Here we can see the relevance of the SDL. If it was not used, the sum would be up to $i = 12000$, that is the initial number of samples from the experiment with 12 s and 0.001 s time-step.

In Figure 13 we can see the identified model applied to a different initial condition at $(x_1, x_2) = (1, 0.5)$. Using the numerical derivatives results in a MAPE error of 0.2552%, it can be seen at the top-right corner that the identified model is slightly different from the truth. In a scenario where the true derivatives are available, the model results in a MAPE error of $7.0595.10^{-4}$%, as seen in the bottom-right of Figure 13.

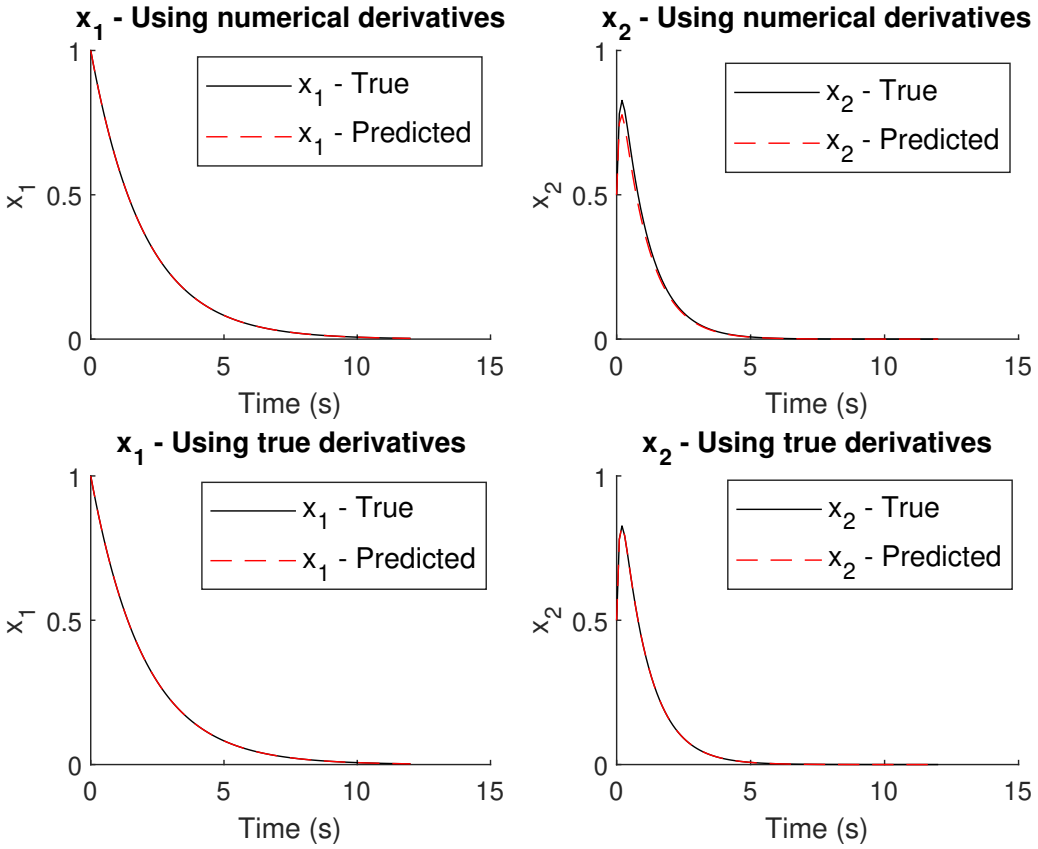

Figure 13 – Comparing the true data, in black, with the model prediction, in red. In the first row, the model is trained using numerical derivatives, estimated from data, and in the second row, using the true derivatives. Source: Author.

For this specific case, no regularization was used ($\lambda = 0$). To understand this,

let's analyze the associated RKHS of the quadratic kernel:

$$\Phi(x) = \left\langle x_2^2, x_1^2, \sqrt{2}x_2x_1, \sqrt{2c}x_2, \sqrt{2c}x_1, c \right\rangle \tag{74}$$

$$\langle \Phi(x), \Phi(x) \rangle = x_2^4 + x_1^4 + 2x_1^2x_2^2 + 2cx_2^2 + 2cx_1^2 + c^2 = (x^Tx + c)^2 = k(x, x). \tag{75}$$

Functions in RKHS contain the terms present in the true dynamics (71). In other words, it has the degrees of freedom to reproduce the training data with almost zero error. In this case, the SDL acts as a natural regularizer. In Figure 14 we see the case where the first 100 samples were used for training, not using the SDL approach, with $\lambda = 10^{-3}$. The regression fails to capture the true dynamics even in the presence of regularization, the same happens for different values of λ.

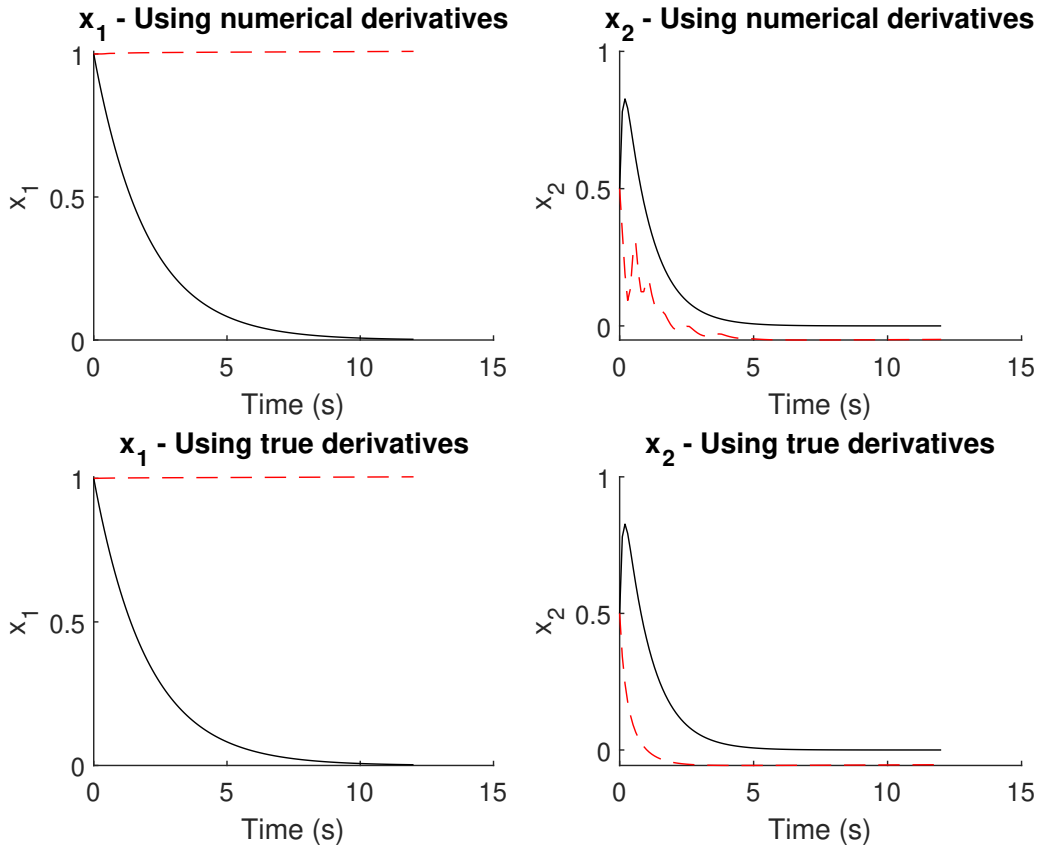

Figure 14 – Model training with the first 100 samples, not using SDL, and $\lambda = 10^{-3}$. In black, the true data, and in red the prediction. Source: Author.

The running time of the model can be compared to a standard solver. The data set was generated by solving the system (71) using the *ode45* solver, which implements a fifth-order Runge Kutta method. The solver took $0.0507s$ to solve the system for a 12 s horizon, with a $0.001s$ time-step. The model trained took $0.0057s$, this is a considerable speedup. Although a significant result, this is still a very simple example, the simulation speedup will be further discussed in the next applications.

6.3 SPE1 RESERVOIR - THE ODEH BENCHMARK

The SPE1 reservoir is a benchmark model proposed in (ODEH, 1981) to serve as a comparison between seven different commercial simulators. It is a three-dimensional black-oil small reservoir, consisting of a $10 \times 10 \times 3$ grid, with 1 injector and 1 producer. In Figure 15 the permeability fields with isotropic values 500, 50, and 200 md in the three layers with thicknesses 20, 30, and 50 ft.

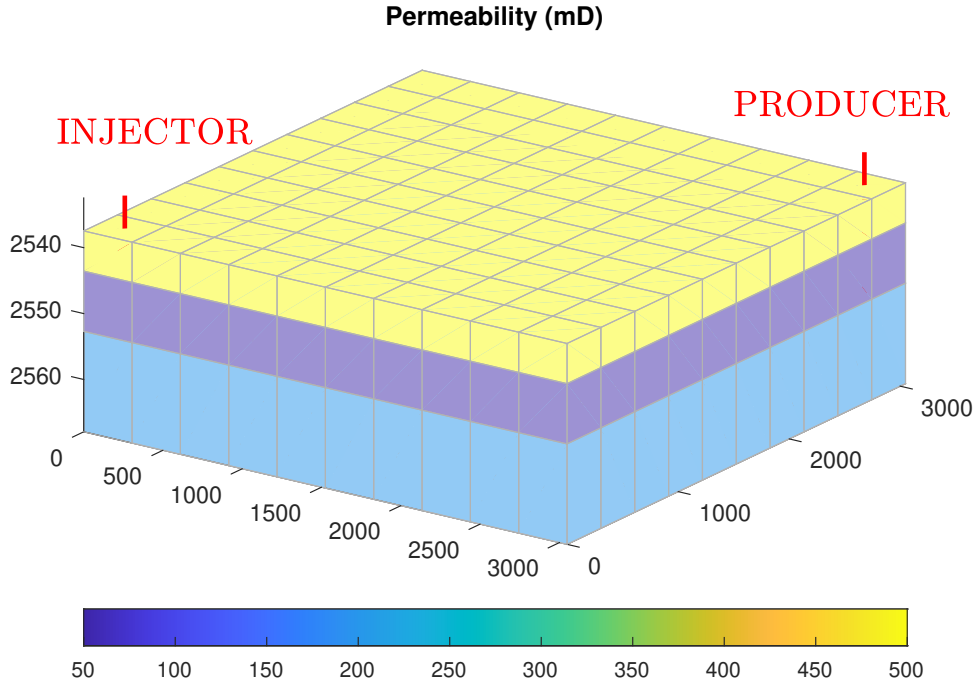

Figure 15 – SPE1 reservoir. A 300 cell reservoir with 1 producer, 1 injector, and a three layer permeability field. Source: (LIE, 2019).

The reservoir is initially under-saturated with a pressure field that is constant in each layer, a uniform mixture of water (Sw = 0.12) and oil (So = 0.88) with no initial free gas (Sg = 0.0) and a constant dissolved gas-oil ratio (Rs) throughout the model (LIE, 2019).

In Table 3 the wells manipulated and observed variables are defined. The injector is controlled by the rate of injection, and the producer by the BHP.

Table 3

	Type	Name	Manipulated Variable	Observed Variable
1	Injector	INJ-1	Gas injection rate [m^3/day]	Bottom Hole Pressure (BHP) [Pa]
2	Producer	PROD-1	Bottom Hole Pressure (BHP) [Pa]	Oil, gas, and water production [m^3]

6.3.1 Data Acquisition

Once the wells inputs are set, a complex nonlinear dynamical system is defined. To acquire data in order to train a proxy model, we will simulate the SPE1 reservoir, using the high-fidelity open-source simulator MRST.

The simulation takes the following steps:

1. In MRST, define the 3D grid, rock parameters, fluids, and wells.
2. A schedule of controls (injections rates and BHPs) and their duration is set up in a *schedule* file.
3. The simulation is run and at the end of it, the output data (oil, water, and gas productions) are stored, as well as the system states throughout the simulation, saturations, and pressures at all the reservoir cells.

An initial sample time step is defined, but the simulator has the freedom by default to cut steps to ensure good accuracy. This can be disabled but typically convergence issues arise from that The choice of this time step directly affects the volume of the data set to be stored. As discussed previously, a small sampling time step contributes to a better derivative estimation. In this work, we will work with different time steps and compare them further.

6.3.2 Simulation Scenarios

A simulation policy needs to be adjusted according to the objectives of the work. In this case, we want to test the capability of the system identification method using the KRR + SDL framework. To do this we need to generate a simulation data set that exploits the reservoir dynamics as much as possible, without extrapolating control values that are considered realistic.

If the scenarios are too similar, the capability of the method will not be tested correctly, since it will be too closely matched to the training data and will not have good generalization capability.

Given this, the simulation scenarios were defined in order to capture most of the dynamics present in the system. To do this we used a random variation signal, around a mean. The signal is the product of a fixed value, added to a variation defined by a uniform probability distribution. In this way, the simulations generated will have different input profiles.

An example of control trajectories can be seen in Figure 16, the controls of the injection well change randomly around a fixed point of 30 m^3/s.

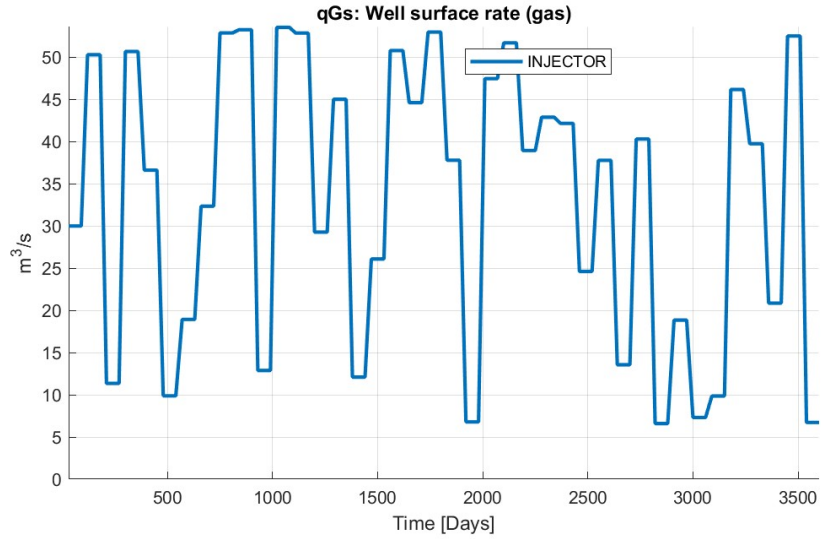

Figure 16 – Example of random injection control trajectory. Source: Author.

It is worth noting that although the BHP of producing well can be manipulated, in this work we restrict ourselves to only identifying models whose input is only the gas injection rates. With this, the BHP is kept constant at a value of 6.9×10^6 Pa.

Mathematically we can define the injection rate, given in cubic meters per second, of the well, at the k-th instant of time, by the function:

$$\begin{cases} \text{INJ}[k] = 30 & \text{if } k = 0 \\ \text{INJ}[k] = \text{INJ}[k-1] - 30 + 50 \cdot R[k] & \text{if } k > 1 \end{cases}, \tag{76}$$

where $R[k]$ defines a scalar randomly sampled from a uniform distribution on the interval $(0, 1)$.

Note that in this case, the controls are changing every 3 months, a time window that is considered adequate given the slow dynamics of the system.

With the control schedule defined, we can perform the simulation. Through the MRST software we obtain information on oil, gas, and water production, as well as the gas/oil rate (GOR), as illustrated in Figure 17. In addition, pressure and saturation information in all cells of the 3D grid is also obtained.

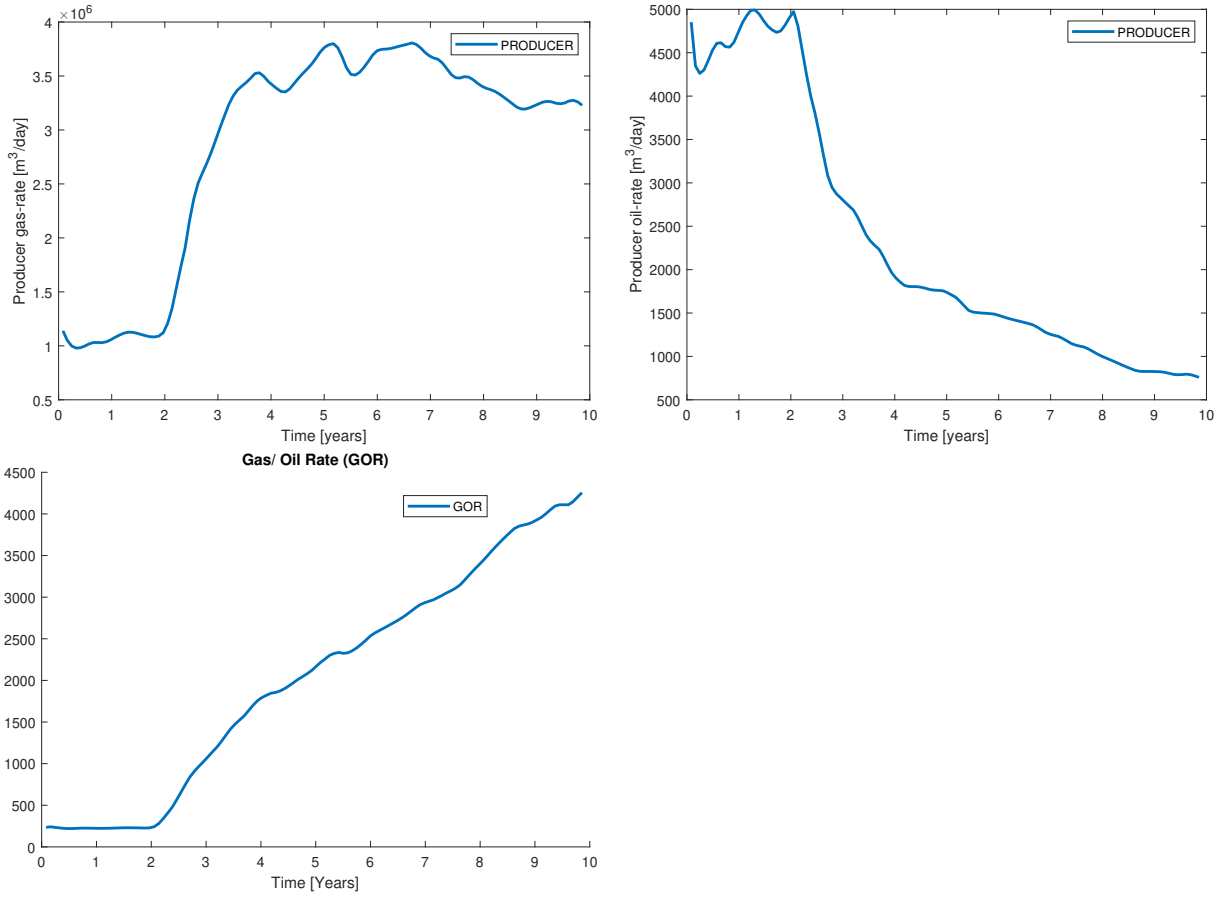

Figure 17 – Results of a SPE1 reservoir simulation. Source: Author.

6.3.3 Data Pre-processing

Since the data comes from a simulator, no noise and measurement uncertainties are present. Therefore, complex data cleaning and pre-processing processes are not necessary.

The control variables are normalized to the interval $(0, 1)$, according to the formula:

$$x_{normalized} = \frac{x - x_{min}}{x_{max} - x_{min}}, \tag{77}$$

and the pressures according to:

$$p_{normalized} = \frac{p}{p_{max}}. \tag{78}$$

The states will not be subtracted by the minimum, because there are no guarantees that the minimum of the data set will be the minimum for all scenarios. This may result in negative values for the pressures, outside the interval $(1, 0)$. The saturations are already normalized since they sum to one:

$$S_{water} + S_{oil} + S_{gas} = 1. \tag{79}$$

It is important for the controls to be in a realistic range, to prevent physical inconsistencies, such as negative flows, or BHPs outside the well capacity. In (HANSSEN;

FOSS, 2016), the authors conduct a study on the choice of control variables (BHPs or rates) in the presence of uncertainty in the reservoir parameters. The control values chosen for this work agree with the article's conclusion.

6.3.4 Model Overview

To synthesize the final model, a random simulation scenario is generated, following the pattern described in Figure 16, for model training. The next step is to estimate the variation, derivative with respect to time, of each state, pressures, and saturations at each cell. For this, we use the Central Differences method described in 6.1.

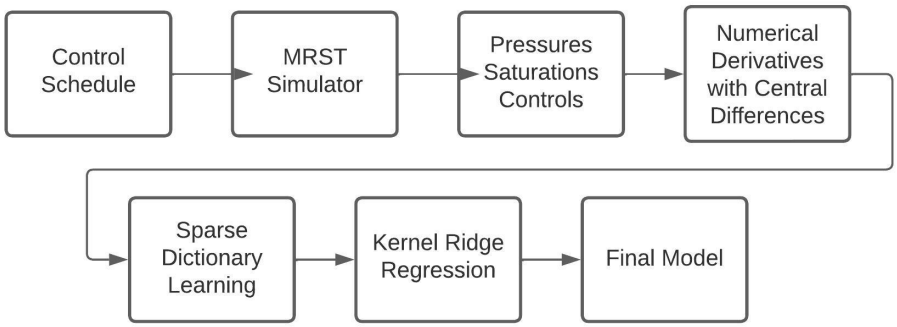

Figure 18 – Overview diagram for the model building, from collecting data to KRR. Source: Author.

The time derivatives, together with the states, form a data point pair $(\dot{\mathbf{x}}, \mathbf{x})$. Now we take only the states $\mathbf{x}$ and stack them into a data matrix $\mathbf{X}$:

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ x_1 & x_2 & \dots & x_t \\ | & | & & | \end{bmatrix}. \tag{80}$$

In order to reduce the number of data points, we use the SDL method, as described in Chapter 5. The method will result in a reduced number of samples $\tilde{m}$, according to the chosen parameter ν and the kernel function k.

The $\tilde{m}$ points $(\dot{\mathbf{x}}, \mathbf{x})$ are then used to perform a KRR. The points are stacked as row vectors, according to (36), with $\dot{\mathbf{x}} = \mathbf{y}$. By choosing an adequate regularization parameter λ, and the same kernel function k used to form the sparse dictionary, we can compute the weights vector:

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda \mathbf{I}_m)^{-1} \mathbf{y}. \tag{81}$$

The final model is then expressed as:

$$\hat{\dot{\mathbf{x}}} = f(\mathbf{x}) = \sum_{i=1}^{\tilde{m}} \alpha_i k(x_i, x), \tag{82}$$

with $\hat{\dot{\mathbf{x}}}$ being the prediction of $\dot{\mathbf{x}}$.

6.3.5 Model Training and Validation

For generating the training and test data, a two year simulation was run with a half an hour time-step. This resulted in a total of 34560 data points. The controls were defined according to (76), changing every two months.

The model generated will be in a days time scale, this means that the derivatives will be approximated by a $\Delta t = \frac{0.5\text{hour}}{day} = 0.0208$. In Figure 19 we can see an example of the time derivative for a given state.

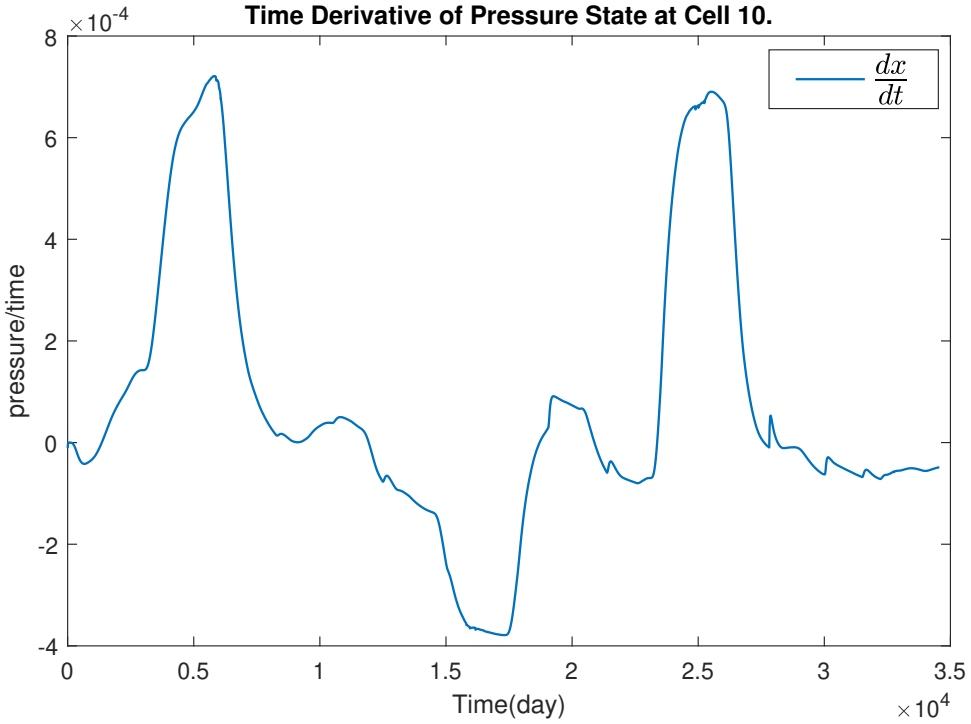

Figure 19 – Source: Author.

For training and validation, we choose a polynomial kernel $k = (\mathbf{x}^T\mathbf{x} + 1)^5$, other kernel functions may be used, but this one resulted in the best performance, when compared to the other ones tested. In Table 4 we can see the number of selected sample points for a few values of ν.

Table 4 – The number of samples selected by SDL with different values of ν. Source: Author.

ν	Number of samples selected
10^1	325
10^0	404
10^{-1}	503
10^{-2}	507
10^{-3}	513
10^{-4}	516

For the KRR, the key parameter is the regularization value λ. A small value may lead to over-fitting the training data and even unstable models. High values may penalize too harshly the model complexity, being unable to capture the correct dynamics.

In this work, the parameter λ will be chosen based on the holdout method. The holdout method consists in splitting the data into training and testing sets, here we use a two year simulation for each. The chosen λ will be the one that performs best at the test set, according to the MAPE error criteria.

In Table 4 we can see that the number of samples starts to increase very slowly after $\nu = 10^{-1}$, indicating we reach an adequate number of samples. For the next experiments, we will fix the value of $\nu = 10^{-2}$, using the selected 507 data points.

We used the two-year simulation to train the model. Now we will test it in a different data set, another two-year simulation, and select the best λ parameter.

Table 5 – Corresponding MAPE errors to a few values of λ. Source: Author.

λ	MAPE
10^{14}	7.0882%
10^{13}	2.4561%
10^{12}	1.3169%
10^{11}	1.486%
10^{10}	1.8418%
10^{9}	1.548%
10^{8}	Unstable

In Table 5 we can see the MAPE errors for a few values of λ. The value of $\lambda = 10^{12}$ delivered the lowest error. Values of $\lambda < 10^{8}$ resulted in unstable models, since the optimization problem has more liberty to select functions (f) with greater norm ($\|f\|_{\mathcal{H}}$), it ends up adapting too much to the training set.

In Figures 20 and 21, we can see the results comparing the model with the real data for the states in two of the three hundred cells. The proxy model does not predict perfectly the state trajectory, but it is able to capture the trend with some oscillation.

With respect to the computational cost, the gain is very significant. While a two year simulation of the high-fidelity model, with a one-day time-step, takes on average, 66.8 s, the proxy model only takes 0.1206 s.

When we raise the prediction horizon to four years, the proxy model loses accuracy, as seen in Figure 22. The MAPE error rises to approximately 26%, and the model became unusable in practice. In this case, we are using the same model trained with the two year simulation data set.

A common issue in machine learning is that most methods are very data intensive, that is, require a large data set for training. This work is no different, the two year simulation for training is a 4 Gb data set, and due to hardware limitations, this is the

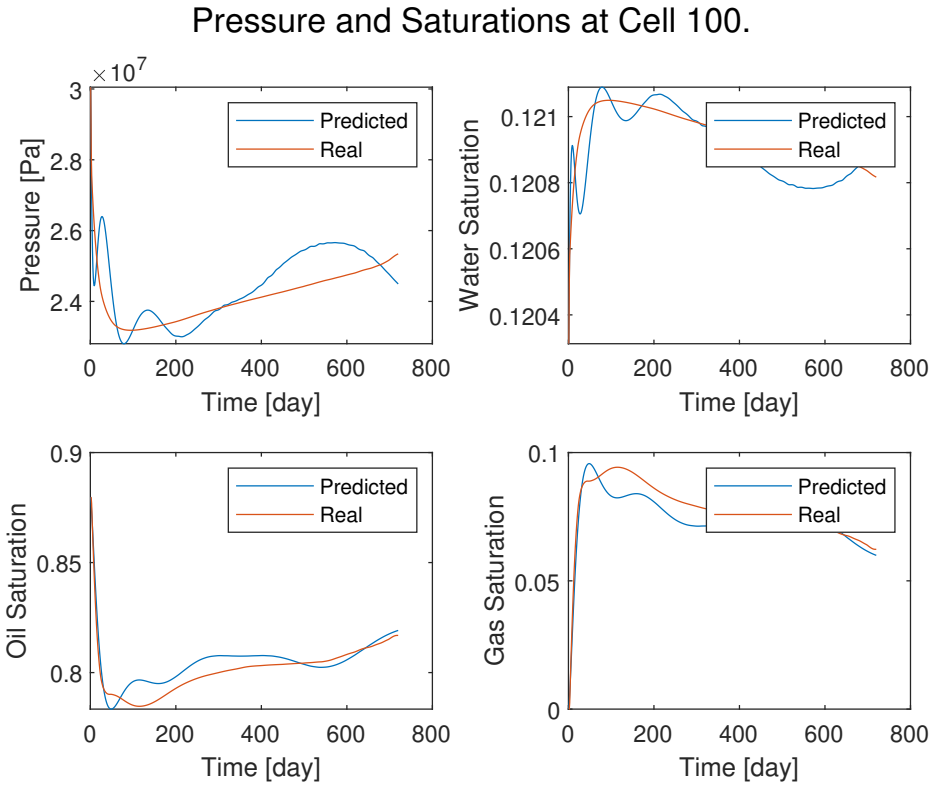

Figure 20 – Comparison between the trained model and the real data in cell 100, with $\nu = 10^{-2}$, and $\lambda = 10^{12}$. Source: Author.

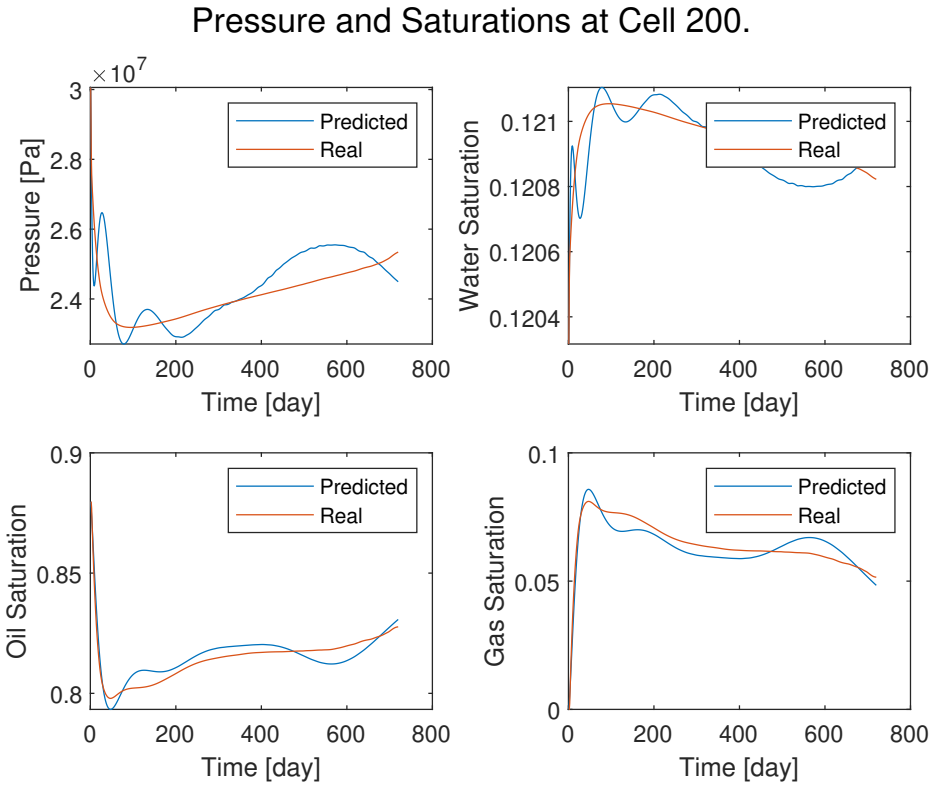

Figure 21 – Comparison between the trained model and the real data in cell 200, with $\nu = 10^{-2}$, and $\lambda = 10^{12}$. Source: Author.

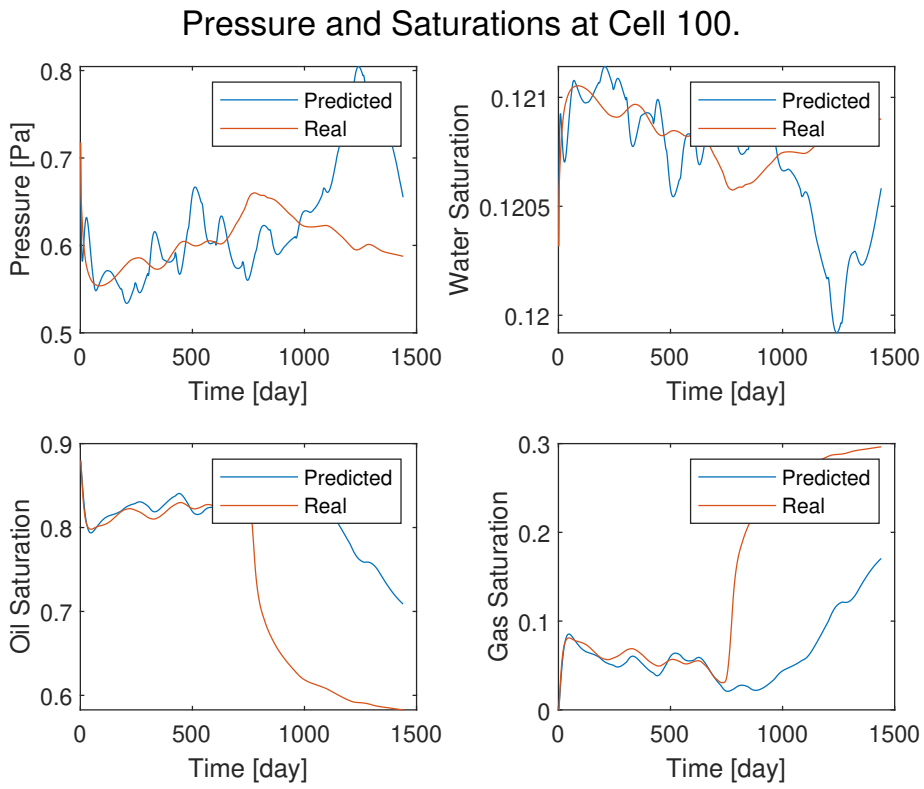

Figure 22 – Comparison between the trained model and the real data for a four year horizon in cell 200, with $\nu = 10^{-2}$, and $\lambda = 10^{12}$. Source: Author.

largest that could be used.

We have seen that the quality of the models is tightly related to the quality of the numerical derivatives, which increase in quality with a small time step. A small time step leads to a large data set because if we would cut it in half, and still maintain the same simulation horizon of two years, the data set would double in size.

To conclude, the complete framework KRR and SDL seems to be more suited for identifying models when the true dynamics actually correspond to terms in the RKHS, as discussed in (75). It is important to remember that the most important feature of the KRR is that at any point, the feature space must be constructed explicitly, rather we access functions in the RKHS by using the kernel trick.

Although the SPE1 is a simple synthetic reservoir, it presents very complex behaviors, that are difficult to capture and represent with a proxy model. This said the error results between 1% and 2% can be considered relevant. This is supported by the fact that here we try to reproduce the reservoir simulation solution by modeling a simplified formulation of the partial differential equations, not only predicting well production.

7 CONCLUSION

We have presented a data-driven combined framework of kernel methods and sparse dictionary learning to identify nonlinear dynamical systems. The framework was applied to a simple nonlinear system and the benchmark SPE1 petroleum reservoir.

For the reservoir, the method resulted in an average prediction error between 1% and 2% for saturations and pressures. The possibility or not of using these models for optimization and control purposes will depend on the flexibility regarding the prediction error of the application in question. Furthermore, the choice of the time window in which the model will be trained and validated plays an important role, as discussed in the previous section.

Regarding computational time, the proxy model achieves acceleration rates of about 10x for the simple dynamical system case, and of about 550x for the SPE1 reservoir. Typically, in the literature, the acceleration rate increases as the system complexity rises, the same behavior appears in this work. With this speedup, a whole range of iterative optimization and control methods may be used, which were previously prohibitive due to time complexity.

To know if this trade-off between prediction error and computational cost is valid, further research is needed, comparing classical reservoir optimization methods with methods using proxy models developed according to this work. This work presents an initial approach with a simple reservoir, with a small time window. To truly test the method capacity, it must be tested in more complex, maybe real, reservoirs, with a larger time window.

Further research may involve using partial knowledge of system physics to design kernels. The chosen kernel function k dictates the structure of the RKHS, for polynomial kernels we have (75). Kernel functions can be combined using properties for which they are closed, as conic combination ($k = \alpha_1 k_1 + \alpha_2 k_2$, for $\alpha_1, \alpha 2 \geq 0$) and point-wise product ($k = k_1(x, x')k_2(x, x')$). These combinations can be used to form a variety of RKHS, along with a series of positive definite kernels that were not explored in this work.

REFERENCES

BADDOO, Peter J.; HERRMANN, Benjamin; MCKEON, Beverley J.; BRUNTON, Steven L. Kernel learning for robust dynamic mode decomposition: linear and nonlinear disambiguation optimization. **Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences**, v. 478, n. 2260, p. 20210830, 2022. DOI: 10.1098/rspa.2021.0830. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rspa.2021.0830. Available from: https://royalsocietypublishing.org/doi/abs/10.1098/rspa.2021.0830.

BRUNTON, Steven L.; KUTZ, J. Nathan. **Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control**. [S.l.]: Cambridge University Press, 2019. DOI: 10.1017/9781108380690.

BRUNTON, Steven L.; PROCTOR, Joshua L.; KUTZ, J. Nathan. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. **Proceedings of the National Academy of Sciences**, Proceedings of the National Academy of Sciences, v. 113, n. 15, p. 3932–3937, Mar. 2016. DOI: 10.1073/pnas.1517384113. Available from: https://doi.org/10.1073%2Fpnas.1517384113.

COSSÉ. **Basics of Reservoir Engineering**. [S.l.]: Editions Technip, 1993. (Institut Français du Pétrole Publications: École Nationale Supérieure du Pétrole et des Moteurs). ISBN 9782710806301. Available from: https://books.google.com.br/books?id=RKECngEACAAJ.

CRAFT, B.C.; HAWKINS, M.F.; TERRY, R.E. **Applied Petroleum Reservoir Engineering**. [S.l.]: Prentice Hall, 1991. ISBN 9780130398840. Available from: https://books.google.com.br/books?id=uDFQAQAAIAAJ.

ENGEL, Y.; MANNOR, S.; MEIR, R. The kernel recursive least-squares algorithm. **IEEE Transactions on Signal Processing**, v. 52, n. 8, p. 2275–2285, 2004. DOI: 10.1109/TSP.2004.830985.

FASEL, Urban; KAISER, Eurika; KUTZ, J. Nathan; BRUNTON, Bingni W.; BRUNTON, Steven L. **SINDy with Control: A Tutorial**. [S.l.]: arXiv, 2021. DOI: 10.48550/ARXIV.2108.13404. Available from: https://arxiv.org/abs/2108.13404.

HANSSEN, Kristian G.; FOSS, Bjarne. On selection of controlled variables for robust reservoir management. **Journal of Petroleum Science and Engineering**, v. 147, p. 504–514, 2016. ISSN 0920-4105. DOI: https://doi.org/10.1016/j.petrol.2016.08.027. Available from: https://www.sciencedirect.com/science/article/pii/S0920410516303448.

HART, Allen; HOOK, James; DAWES, Jonathan. Embedding and Approximation Theorems for Echo State Networks. **Neural Networks**, May 2020. DOI: 10.1016/j.neunet.2020.05.013.

LIE, Knut-Andreas. Introduction. In: AN Introduction to Reservoir Simulation Using MATLAB/GNU Octave: User Guide for the MATLAB Reservoir Simulation Toolbox (MRST). [S.l.]: Cambridge University Press, 2019. P. 1–18. DOI: 10.1017/9781108591416.002.

[S.l.]. **An End-to-End Deep Sequential Surrogate Model for High Performance Reservoir Modeling: Enabling New Workflows**. [S.l.: s.n.], Oct. 2020. Day 4 Thu, October 29, 2020. D041S046R006. DOI: 10.2118/201775-MS. eprint: https://onepetro.org/SPEATCE/proceedings-pdf/20ATCE/4-20ATCE/D041S046R006/2363095/spe-201775-ms.pdf. Available from: https://doi.org/10.2118/201775-MS.

ODEH, Aziz S. Comparison of Solutions to a Three-Dimensional Black-Oil Reservoir Simulation Problem (includes associated paper 9741). **Journal of Petroleum Technology**, v. 33, n. 01, p. 13–25, Jan. 1981. ISSN 0149-2136. DOI: 10.2118/9723-PA. eprint: https://onepetro.org/JPT/article-pdf/33/01/13/2229415/spe-9723-pa.pdf. Available from: https://doi.org/10.2118/9723-PA.

PILLONETTO, Gianluigi; DE NICOLAO, Giuseppe. A new kernel-based approach for linear system identification. **Automatica**, v. 46, n. 1, p. 81–93, 2010. ISSN 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2009.10.031. Available from: https://www.sciencedirect.com/science/article/pii/S0005109809004920.

SAGHEER, Alaa; KOTB, Mostafa. Time series forecasting of petroleum production using deep LSTM recurrent networks. **Neurocomputing**, v. 323, p. 203–213, 2019. ISSN 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2018.09.082. Available from: https://www.sciencedirect.com/science/article/pii/S0925231218311639.

SCHIOZER, Denis J.; SANTOS, Susana M.; SANTOS, Antonio Alberto S.; HOHENDORFF FILHO, João C. von. Model-Based Life-Cycle Optimization for Field Development and Management Integrated with Production Facilities. Day 2 Tue, June 07, 2022, June 2022. D021S003R004. DOI: 10.2118/209681-MS. eprint: https://onepetro.org/SPEEURO/proceedings-pdf/22EURO/2-22EURO/D021S003R004/2690433/spe-209681-ms.pdf. Available from: https://doi.org/10.2118/209681-MS.

SCHÖLKOPF, Bernhard; SMOLA, Alexander J. **Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond**. [S.l.]: The MIT Press, June 2018. ISBN 9780262256933. DOI: 10.7551/mitpress/4175.001.0001. Available from: https://doi.org/10.7551/mitpress/4175.001.0001.

SHAWE-TAYLOR, John; CRISTIANINI, Nello. **Kernel Methods for Pattern Analysis**. [S.l.]: Cambridge University Press, 2004. DOI: 10.1017/CBO9780511809682.

TOMPSON, Jonathan; SCHLACHTER, Kristofer; SPRECHMANN, Pablo; PERLIN, Ken. **Accelerating Eulerian Fluid Simulation With Convolutional Networks**. [S.l.]: arXiv, 2016. DOI: 10.48550/ARXIV.1607.03597. Available from: https://arxiv.org/abs/1607.03597.

ZHANG, Linan; SCHAEFFER, Hayden. **On the Convergence of the SINDy Algorithm**. [S.l.]: arXiv, 2018. DOI: 10.48550/ARXIV.1805.06445. Available from: https://arxiv.org/abs/1805.06445.