TÉCNICO LISBOA

Data-driven discovery of the mechanism of the Belousov-Zhabotinsky reaction

Mariana Serra dos Reis Vieira da Mota

Thesis to obtain the Master of Science Degree in

Engineering Physics

Supervisor(s): Prof. Rui Manuel Agostinho Dilão

Examination Committee

Chairperson: Prof. Ilídio Pereira Lopes
Supervisor: Prof. Rui Manuel Agostinho Dilão
Member of the Committee: Dr. Madalena Chaves

December 2020

To my dad.

Acknowledgments

First and foremost, I would like to express my profound gratitude to my supervisor Prof. Rui Dilão for his valuable guidance, advice and support throughout this work. Without his constant help this thesis would never have been complete. Thank you for suggesting such an interesting topic for me to develop on this dissertation, I couldn't imagine any better.

I want to thank all my friends for helping me get through the hardest year of my life. I would have never been able to push through and finish this degree without you. To my friends from IST, thank you for the most amazing journey throughout these five years. Even in the most stressful and difficult times, your friendship made it all worth it.

I would also like to thank my family from the bottom of my heart for always believing in my abilities and always pushing me towards bigger and better things. Without their constant support I would not be where I am today.

Finally and most importantly, to my dad, whose absence in my life left a void which is impossible to fill. I am extremely grateful for everything you have taught me. Thank you for providing me the happiest of childhoods and for always being such a light in my life. I am so grateful I still got to share with you the topic of this thesis, I know you would have been so proud to see me finishing this degree. Your brightness, intelligence and sense of humour will forever live in me.

Resumo

O objetivo principal deste trabalho é deduzir o mecanismo cinético associado às ondas de reação-difusão observadas na reação de Belousov-Zhabotinsky, que é simulada através do modelo do Brusselator, um sistema de reações químicas oscilante virtual. Isto é concretizado usando um algoritmo desenvolvido recentemente que visa derivar equações governantes de dados de séries temporais.

O sistema Brusselator é estudado, definindo as suas equações e analisando a sua estabilidade, para melhor compreender a dinâmica do modelo. O seu espaço de fases é reconstruído usando os dados de evolução no tempo de apenas uma das duas variáveis do sistema. Na reação de Belousov-Zhabotinsky, apenas uma variável é observada experimentalmente, daí a importância da validade desta reconstrução. Os padrões típicos de reação-difusão são simulados em 2D com o sistema Brusselator.

O algoritmo de *data mining*, PDE-FIND, é descrito e testado para dois sistemas diferentes, sendo um deles também um modelo de reação-difusão, que se mostrou bem sucedido. O modelo foi então aplicado aos dados computados do Brusselator, inferindo-se o sistema de reação-difusão que melhor se ajusta a estes.

O algoritmo PDE-FIND mostrou ser capaz de identificar corretamente o sistema de equações diferenciais parciais, o que mostra que é possível obter informações importantes da dinâmica local de um sistema a partir de dados, e assim inferir os seus mecanismos cinéticos subjacentes.

Palavras-chave: Belousov-Zhabotinsky, reação-difusão, Brusselator, algoritmo, *data mining*

Abstract

The main goal of this work is to deduce the kinetic mechanism associated with reaction-diffusion waves observed in the Belousov-Zhabotinsky reaction, which is simulated by the Brusselator model, a virtual oscillating chemical reaction system. This is done using a recently developed algorithm that aims to derive governing equations from time-series data.

The Brusselator system is thoroughly studied, by defining its equations and analysing its stability, to better understand the dynamics of the model. Its phase space is reconstructed using the time evolution data of only one of the two variables of the system. In the Belousov-Zhabotinsky reaction, only one variable is experimentally observed, hence the importance of the validity of this reconstruction. Typical reaction-diffusion patterns are simulated in 2D with the Brusselator system.

The data mining algorithm, PDE-FIND, is described and tested for two different systems, one of them also being a reaction-diffusion model, which showed to be successful. The model was then applied to the computed data of the Brusselator, inferring the reaction-diffusion system that best fits it.

The PDE-FIND algorithm showed to be able to correctly identify the system of partial differential equations, which shows that it is possible to obtain valuable information of the local dynamics of a system from data, and thus to infer its underlying kinetic mechanisms.

Keywords: Belousov-Zhabotinsky, reaction-diffusion, Brusselator, algorithm, data mining

Contents

List of Tables

List of Figures

Nomenclature

Ag^{+}	Silver Cation
Br^{-}	Bromide
Ce^{3+}	Cerous Ions
Ce^{4+}	Ceric Ions
Ce	Cerium
HBrO_2	Bromous Acid
HBrO_3	Bromate
Mn	Manganese
BMA	Bromalonic Acid
BZ	Belousov-Zhabotinsky
KdV	Korteweg–de Vries
LASSO	Least Absolute Shrinkage and Selection Operator
MA	Malonic Acid
PDE	Partial Differencial Equation
PDE-FIND	PDE functional identification of nonlinear dynamics
R-D	Reaction-diffusion
STLS	Sequentially Thresholded Least Squares
STRidge	Sequential Threshold Ridge regression

Chapter 1

Introduction

Discovering the underlying dynamics of a system by extracting its governing equations from temporal and spatial data is a central challenge in diverse areas of science and engineering. Data-driven discovery methods have been made possible in the last decade by the rapid decrease of the cost of sensors, data storage, and computational resources. Machine learning and data science advances have made it possible to extract patterns from vast multimodal data, a breakthrough in the analysis and understanding of complex data. Traditionally, physical systems' underlying partial differential equations (PDEs) are derived from conservation laws, physical principles, and phenomenological behaviour. However, there are still complex systems that escape from quantitative analytic descriptions and where a new method for deriving underlying PDEs of dynamic processes from big data is essential. Many of these systems have large time-series data which has given rise to the new paradigm of data-driven model discovery. Recently, S. Rudy, S. Brunton, J. Proctor and J. Kutz [1] have developed deep learning techniques to fit observed data, with models based on time series measurements in the spatial domain.

1.1 Reaction-Diffusion systems

In general, the shape or pattern of a living being, or any other natural system, results from the symmetry or regularity it presents, as well as the frequency with which it is observed in nature. Thus, it is important to find the mechanisms that generate the biological patterns, as well as the reason why some shapes are more abundant than others [2]. It was Turing [3], in 1952, who proposed that these real systems present self-organizing properties which appear as emerging coherent patterns or structures in spatially extended media, generated by the coupling of reactive substances and diffusive processes. In purely diffusive media, spatial homogeneity is asymptotically achieved, corresponding to the state of maximum disorganization. Any fluctuation in the concentration of a chemical substance is obliterated by the random motion of molecules, contrary to the formation of stable gradients. In the presence of reactive processes, the effect of diffusion could be compensated by local chemical processes, and the reaction between two molecules could amplify local fluctuations to a macroscopic scale, leading to the formation of coherent structures or patterns. This way, properties of reaction-diffusion systems depend

on the balance between chemical and diffusive processes. This behaviour is experimentally observable in oscillating reactions, the most impactful and investigated one being the Belousov-Zhabotinsky reaction.

In general, reaction-diffusion (R-D) systems include two types of processes: reactive processes, which correspond to chemical reactions which can be mathematically represented by velocity equations derived from the mass action law of chemical kinetics; and diffusive processes, which are mathematically expressed by Fick's second law. Thus, a reaction-diffusion system can be defined as a system of partial differential equations of the form:

$$\frac{\partial \varphi}{\partial t} = f(\varphi) + D \cdot \nabla^2 \varphi \tag{1.1}$$

where $\varphi = (\varphi_1, ..., \varphi_m)$ is a m-dimensional vector representing the concentration of m chemical species, $f(\varphi)$ is a m-dimensional vector field representing the local kinetic mechanism and D is the diffusion matrix. For each one of the dynamical variables of a model, an equation of the type 1.1 is defined.

In a region of an approximately homogeneous medium, the emergence of a pattern or structure comes from the existence of a process which, once initiated, tends to grow by an activation mechanism. However, there needs to simultaneously be an inhibitor to keep a pattern from propagating to all regions of the medium, preventing similar structures from emerging in nearby regions and maintaining the active region localized. The inhibitor is produced at a similar rate as the activator, but is more quickly diffused. This way, if the activator's diffusion coefficient D is lower than that of the inhibitor, the action of the inhibitor does not prevent the increase in the local concentration of the activator. The richer models in terms of pattern variability assume an antagonist mechanism between two variables, where one behaves as an activator and the other as an inhibitor [4]. The activation effect is generally produced by autocatalytic reactions, which are reactions where one of the reactants regenerates as a product, with a stoichiometric relation higher than one:

$$\mathrm{A} + \mathrm{X} \xrightarrow{k} 2\,\mathrm{X} \tag{1.1.2a}$$

The need for an inhibitory effect in order to create stable patterns becomes intuitive by taking into account the fact that reaction (1.1.2a) is divergent for non-limiting concentrations of reactant A. For limiting concentrations, this variable has an inhibitory effect in the reaction progression [5].

1.2 Belousov-Zhabotinsky reaction

The Belousov-Zhabotinsky (BZ) reaction is named after B. P. Belousov who discovered the reaction and A. M. Zhabotinsky who continued Belousov's early work [6]. This discovery marked the beginning of modern non-linear chemical dynamics [7], and the reaction serves as a classical example of non-equilibrium thermodynamics. It remarkably maintains a prolonged state of non-equilibrium that leads to macroscopic temporal oscillations and spatial pattern formation that is very life-like. The reaction makes

it possible to observe development of complex patterns in time and space by naked eye, in a time scale of dozens of seconds and space scale of several millimetres [8]. The BZ reaction allows the study of chemical waves and patterns without constant replenishment of reactants, by generating up to several thousand oscillatory cycles in a closed system. It has been the most investigated experimental model regarding the emergence of complex patterns. It is easily implemented in the laboratory, without needing sophisticated instrumental methods.

1.2.1 History

Belousov first stumbled upon an oscillating system [9] when he observed a mixture of citric acid, bromate, and cerium catalyst in a sulfuric acid solution undergoing periodic colour changes between colourless and yellow. These colour changes indicated the cyclic formation and depletion of differently oxidized cerium species, and he found that the frequency of oscillations increased with rise of temperature [8]. Ten years after Belousov's works, Zhabotinsky reproduced his results and replaced citric acid with malonic acid (MA) as a reductant. He has shown that oscillations in concentration of ceric ions (Ce^{4+}) lead to the oscillations in the solution's colour [10]. These oscillation are represented in Figure 1.1. The yellow colour was found to be due to the preponderance of Ce^{4+} ions while the colourless state is due to the cerous ions (Ce^{3+}). This behaviour of the system is called a chemical clock, and is often studied as a prototype of a simple biological system.

Zhabotinsky also found that oxidation of Ce^{3+} by bromate (HBrO_3) was an autocatalytic reaction and self-sustained oscillations of Ce^{4+} concentration arose after accumulation of bromomalonic acid (BMA), and that bromide (Br^-) ion was an inhibitor of the autocatalytic oxidation of Ce^{3+} [8]. He proposed that the BZ reaction consists of two main parts: the autocatalytic oxidation of Ce^{3+} ions by HBrO_3 and the reduction of Ce^{4+} ions by malonic acid, which were produced during the overall reaction [10]. The Ce^{4+} reduction is accompanied by the production of Br^- from the bromoderatives of malonic acid. Br^- is a strong inhibitor of the autocatalytic oxidation of Ce^{3+} because of its rapid reaction with the autocatalyst, which is presumably bromous acid (HBrO_2) or some oxybromine free radical.

An oscillatory cycle can be described in the following way. Supposing that a sufficiently high concentration of Ce^{4+} is present in the system, Br^- will be produced rapidly, and its concentration will also be high. This results in the autocatalytic oxidation of Ce^{3+} being completely inhibited, and the Ce^{4+} concentration decreases due to its reduction by MA and BMA. The Br^- concentration decreases along with that of Ce^{4+}. The bromide ion concentration drops abruptly when Ce^{4+} reaches its lower threshold. Then, the rapid autocatalytic oxidation starts and raises Ce^{4+}. Finally, when this concentration reaches its higher threshold, Br^- increases sharply, completely inhibiting the autocatalytic oxidation of Ce^{3+}. The cycle then repeats. The phase resetting experiments in Figure 1.2 validate this scheme. Examining this figure, one can see that pulse injections of Br^- or Ce^{4+} during the rising of Ce^{4+} produces a switch to the phase of the Ce^{4+} decrease. An injection of silver cation (Ag^+) switches the system from a declining to an increasing Ce^{4+} phase. The simplest mechanism of the autocatalytic oxidation of Ce^{3+} or ferroin by bromate and its inhibition by bromide ion [8] is as follows:

(a) Small concentric rings and spirals start appearing.

(b) Circles start expanding and mutual annihilation takes place.

(c) System keeps expanding.

(d) Strong annihilation results in the loss of circular waves.

Figure 1.1: Target patterns in the BZ reaction, formed by point pacemakers. The waves emerge from a background of a reduced state (red) in which concentration waves of the autocatalytic species of the reaction are defined, whose production is coupled with the oxidation of ferroin in ferrin (blue).

$$HBrO_3 + HBrO_2 \longrightarrow 2\,BrO_2^{\bullet} + H_2O$$

$$H^+ + BrO_2^{\bullet} + Fe(phen)_3^{2+} \longrightarrow Fe(phen)_3^{3+} + HBrO_2$$

$$HBrO_2 + H^+ + Br^- \longrightarrow 2\,HOBr$$

Many variants of the reaction exist, the only key chemical being the bromate oxidizer. The catalyst ion is usually cerium (Ce), but it can also be manganese (Mn) or complexes of Fe, Ru, Co, Cu, Cr, Ag, Ni, and Os. Various different reductants give rise to oscillations [8]. However, the kinetic mechanisms just schematized are kinetically unrealistic due to the fact that they involve triple collisions.

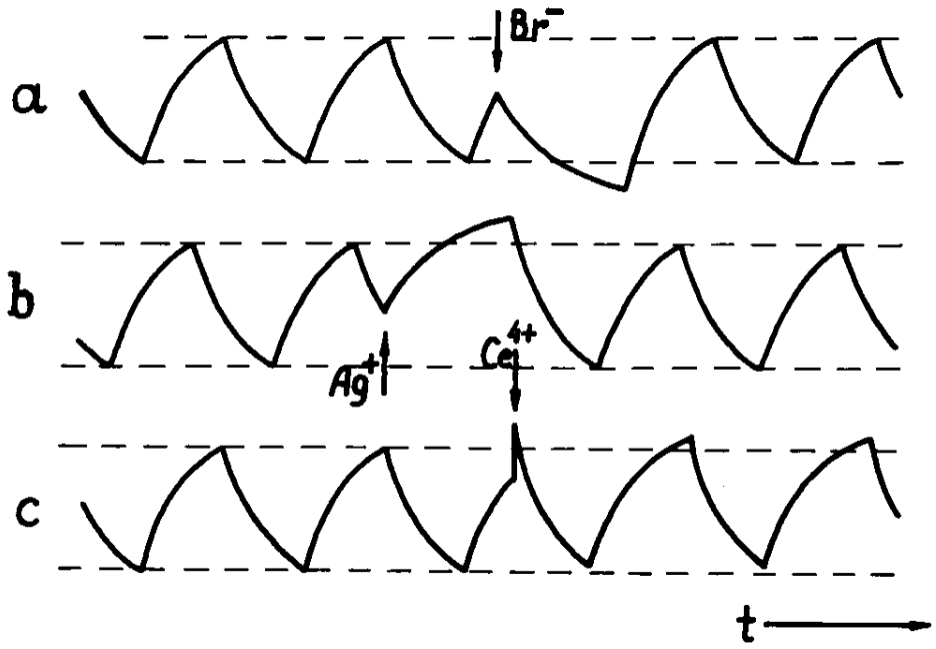


Figure 1.2: Oscillations of concentration of Ce^{4+} and phase resetting in the BZ reaction caused by pulse injection of (a) Br^{-}, (b) Ag^{+}, and (c) Ce^{4+}. Solutions of indicated ions are injected at the times shown by the arrows.

1.2.2 Chemical Waves and Patterns

In homogeneous media, the main attribute of the BZ reaction is the induction of periodic fluctuations in the concentrations of the intermediates. In non-homogeneous conditions, this is, in media with no agitation, local fluctuations in the concentration of reagents are transmitted to the reactor region, giving rise to wave fronts that propagate and interact with each other. These wave fronts are dynamic patterns with spatial periodicity that depends on the reaction kinetics and on the diffusion constants of the various substances involved in the process [2].

There are two types of chemical waves present in the BZ reaction: phases waves and relaxation or excitation waves. Phase waves propagate throughout space, and result from heterogeneities that form in non-homogeneous system, producing global effects throughout self-excitable media. The propagation of these waves is independent of diffusion and only appear in oscillatory states. These are quick waves and their propagation speed is unpredictable, but the frequency they pass a certain reference point is related to the oscillatory frequency of the homogeneous system [5].

Relaxation waves are local perturbations that propagate throughout space from emitting points, or *pacemakers*, that work as local organizers. Their propagation depends on diffusion and, as long as the system is excitable, it does not need to be in an oscillatory state. The two types of chemical waves can coexist, but when a phase waves supervenes there is an annihilation of the most external wavefront.

Zhabotinsky described the propagation of waves of oxidation in thin unstirred layers of BZ reagent, organized as concentric rings that expanded away from a central zone of periodic initiation [6]. These concentric chemical waves generated by point pacemakers formed target patterns, which vary in temporal period, and are composed of pulses of excitation followed by refractory zones. Collisions of the

waves lead to mutual annihilation due to the presence of non-excitable refractory zones [8]. If these waves break, the excitation front curls around their refractory tails and form spiral waves. The formed target patterns can be seen in Figure 1.1.

These waves emerge from a background of a reduced state (red) in which concentration waves of $\mathrm{HBrO_2}$, the autocatalytic species of the reaction, are defined, whose production is coupled with the oxidation of ferroin in ferrin (blue). They propagate at a constant speed, which depends on the temperature and the kinetic and diffusive parameters of the chemical species that regulate the mechanism [5]. This is specific for plane waves, since the propagation speed of circular waves depends on their curvature, according to the equation:

$$v = v_0 - \frac{D}{r} \tag{1.2.1}$$

where v_0 is the plane wave speed, D is the diffusion coefficient of the autocatalytic species and r is the radius of the circular wave [5].

1.3 Objectives

The main goal of this project is to deduce the kinetic mechanism associated with reaction-diffusion waves observed in the Belousov-Zhabotinsky reaction, virtually simulated and described by the Brusselator model: instead of fitting data to a particular model, we do an inverse process, inferring the R-D system that best fits the data. Resorting to the algorithm developed by Rudy and co-authors [1], the collected data will be fitted to a reaction-diffusion model and the corresponding kinetic equations, testing the accuracy and applicability of the model.

1.4 Thesis Outline

This dissertation gives an in-depth study of one of the most researched reaction-diffusion systems, the Brusselator, and a thorough analysis of a recently developed algorithm that aims to derive partial differential equations from a given dataset, the PDE-FIND algorithm. Chapter 1 gives an introduction about reaction-diffusion models and how they are mathematically described. It also contains an analysis of the Belousov-Zhabotinsky reaction and its historical background. Chapter 2 contains a deep study of the Brusselator system, a virtual model of the Belousov-Zhabotinsky reaction, and its dynamics. The Takens' theorem is applied on the phase space of our system, and 2D reaction-diffusion patterns are simulated. In Chapter 3, the PDE-FIND algorithm is described, and two examples of its application are shown: the KdV equation and a λ-ω reaction-diffusion system. The results of the application of this algorithm on the Brusselator system are presented in Chapter 4. Chapter 5 contains the final remarks of this dissertation and a description of future experimental work to be performed.

Chapter 2

Brusselator Model

In this section, a model used to virtually describe the Belousov-Zhabotinsky reaction is studied, known as the Brusselator model. Its equations are defined, and a thorough analysis of its stability is done, to better understand the dynamics behind the model. The phase space of the system is reconstructed following the Takens' delay embedding theorem [11]. Also, typical reaction-diffusion patterns are simulated in 2D with the Brusselator system.

The Brusselator is a virtual oscillating chemical reaction system and serves as a prototype of an activation–inhibition mechanism in chemistry. It has been used to describe the spatial dynamics of the Belousov-Zhabotinsky reaction. The model was proposed by Prigogine and Lefever (1968) and was given its name as a reference to its birthplace (Université Libre de Bruxelles). It is widely used due to its theoretical simplicity while retaining the functional form of more complex reaction networks.

2.1 Equations and Stability Analysis

The Brusselator reaction consists of four steps:

$$\mathrm{A} \xrightarrow{k_1} \mathrm{X} \tag{2.1.1a}$$

$$\mathrm{B} + \mathrm{X} \xrightarrow{k_2} \mathrm{Y} + \mathrm{D} \tag{2.1.1b}$$

$$2\,\mathrm{X} + \mathrm{Y} \xrightarrow{k_3} 3\,\mathrm{X} \tag{2.1.1c}$$

$$\mathrm{X} \xrightarrow{k_4} \mathrm{E} \tag{2.1.1d}$$

where X and Y represent the dynamic variables of the system, A and B are control variables (which are kept constant) and $k_i, i = 1, 2, 3, 4$ represent the reaction rates. The third step is autocatalytic since two X molecules make three, and also has an inhibiting factor because Y is used in this process while it is necessary to make the reaction. The reaction scheme is physically unrealistic because of the trimolecular third step, since this reaction is statistically unlikely [12]. However, systems with two dynamical variables can only show limit cycle variations if the kinetic mechanism includes a trimolecular term.

This model can be taken as an open system since the control variables A and B are input variables and assumed constant. Since the reactions are all irreversible, the output variables are irrelevant for the mechanism. A simple schematic of the mechanism of the Brusselator is shown in Figure 2.1.

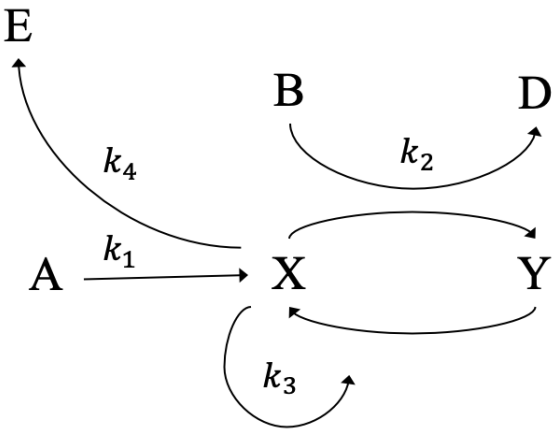


Figure 2.1: Schematic of the mechanism of the Brusselator model. The dynamic variables of the system are represented by X and Y, and A and B are the control variables. $k_i, i = 1, 2, 3, 4$ are the reaction rates of the system. There is an autocatalytic step with rate k_3 where two X molecules make three, and there is also an inhibiting factor Y. This step makes the reaction scheme unrealistic, since this is statistically unlikely.

In order to derive the rate equations for the production of X and Y, we have to follow the empirical rule: when two substances react to produce a third, the reaction rate is proportional to the product of the concentrations of the two substances [13]. In the third reaction (2.1.1c), $2X$ is treated as $X + X$, which will give a X^2 term is the rate equation for X. Hence, we get the following system of differential equations, for a two dimensional media:

$$\begin{cases} \frac{\partial X}{\partial t} = k_1 A - k_2 B X + k_3 X^2 Y - k_4 X + D_X \left(\frac{\partial^2 X}{\partial x^2} + \frac{\partial^2 X}{\partial y^2} \right) \\ \frac{\partial Y}{\partial t} = k_2 B X - k_3 X^2 Y + D_Y \left(\frac{\partial^2 Y}{\partial x^2} + \frac{\partial^2 Y}{\partial y^2} \right) \end{cases} \tag{2.1.2}$$

The steady state conditions are defined by the solutions of the following system:

$$\begin{cases} k_1 A - k_2 BX + k_3 X^2 Y - k_4 X = 0 \\ k_2 BX - k_3 X^2 Y = 0 \end{cases} \tag{2.1.3}$$

which gives us:

$$\begin{cases} X* = A\frac{k_1}{k_4} \\ Y* = \frac{B}{A}\frac{k_2 k_4}{k_1 k_3} \end{cases} \tag{2.1.4}$$

For the steady state, the Jacobian matrix is given by:

$$J = \begin{bmatrix} 2k_3 XY - k_2 B - k_4 & k_3 X^2 \\ -2k_3 XY + k_2 B & -k_3 X^2 \end{bmatrix} \tag{2.1.5}$$

By entering the solutions given in (2.1.4), the Jacobian around the zero fixed point is obtained:

$$\mathbf{A} = J|_{(X*,Y*)} = \begin{bmatrix} k_2 B - k_4 & A^2 \frac{k_1^1 k_3}{k_4^2} \\ -k_2 B & -A^2 \frac{k_1^1 k_3}{k_4^2} \end{bmatrix} \tag{2.1.6}$$

Letting λ_1 and λ_2 be the eigenvalues of **A**, we have that $\lambda_1 + \lambda_2 =$Trace$\mathbf{A}$ and $\lambda_1 \lambda_2 =$det$\mathbf{A}$. Matrix $\mathbf{A}$ is characterized by:

$$Trace\mathbf{A} = k_2 B - k_4 - A^2 \frac{k_1^2 k_3}{k_4^2} \tag{2.1.7}$$

$$det\mathbf{A} = A^2 \frac{k_1^2 k_3}{k_4} \tag{2.1.8}$$

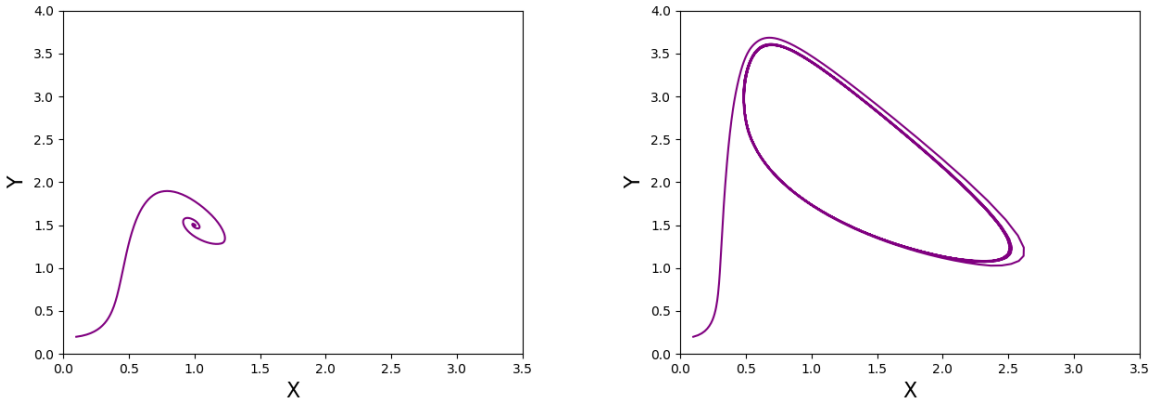
Since $\lambda_1 \lambda_2 =$det**A**> 0, the fixed node can only be of node or focus types, and its stability is determined by the trace of **A**. If Trace**A**> 0, $(X*, Y*)$ is an unstable focus or node. If Trace**A**< 0, $(X*, Y*)$ is a stable focus or node. From (2.1.7), Trace**A** can take either a positive or a negative value depending on the values given to the kinetic constants, therefore there is a supercritical Hopf bifurcation for Trace**A**$= 0$. Solving this last condition in order to the parameter B, the Hopf bifurcation occurs for

$$B = \frac{k_4}{k_2} + A^2 \frac{k_1^2 k_3}{k_2 k_4^2} \tag{2.1.9}$$

To better understand the dynamics of the Brusselator model, let's start by analysing the temporal evolution of the system (2.1.2) with suppressed diffusion, $D_X = D_Y = 0$. This way, the system of partial differential equations is reduced to a system of ordinary differential equations and it is possible to follow the time evolution of X and Y in the concentration space. For an appropriate choice of the kinetic parameters, the main feature of this system lies in the existence of an unstable steady state. This means that there is a value for X and Y concentrations, which do not evolve over time, but any small perturbation moves them away from the steady state. When the initial concentrations at a given time $t = 0$ do not correspond specifically to the values of the stationary concentrations, the system quickly

reaches an oscillatory regime. Since this regime is represented by a closed curve in the concentration space, which acts as an attractor of all other trajectories, it is said that the system has a limit cycle [2].

Choosing the parameters $k_1 = k_2 = k_3 = k_4 = 1$ and $A = 1.0$, the Hopf bifurcation in the Brusselator model occurs for $B = 2.0$, according to (2.1.9). Thus, for certain parameter values, there is a phase space limit cycle and the dynamics are asymptotically periodic.



(a) B = 1.5 (b) B = 2.5

Figure 2.2: Phase space orbit of equation (2.1.2) for two values of B, with fixed parameters $k_1 = k_2 = k_3 = k_4 = 1$, $A = 1.0$ and $D_X = D_Y = 0.0$. In b), the phase space orbit converges to a limit cycle since the value of B is greater than the one for which there is an Hopf bifurcation for these conditions.
.

In Figure 2.2 a), the fixed point is a stable focus and in Figure 2.2 b), it is an unstable focus. For values of B greater than the value for which there occurs an Hopf bifurcation, that is, for $B > 2.0$, all orbits in phase space converge to the limit cycle, with the exception of the zero orbit.

The reference conditions of this work, according to (2.1.9), are: $k_1 = k_2 = k_3 = k_4 = 1$, $A = 1.0$, $B = 2.3$. For an non-diffusive media, the Brusselator's model phase space is shown in Figure 2.3. Since $B > 2.0$, the fixed point is an unstable focus.

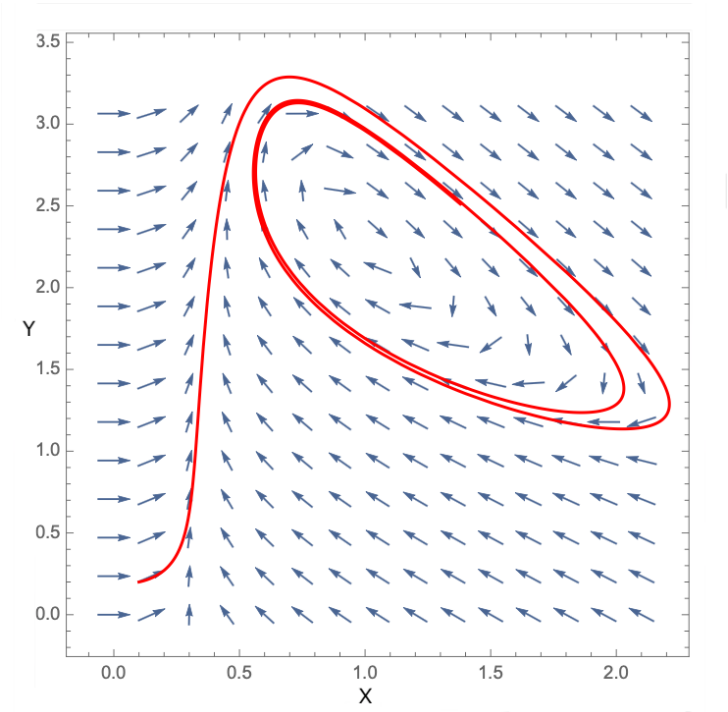


Figure 2.3: Brusselator's phase space on a non-diffusive media for reference conditions of this work: $A = 1.0$, $B = 2.3$ and $k_i = 1.0$. The phase space orbit converges to a limit cycle, and the fixed point is an unstable focus since $B > 2.0$.

2.2 Phase-Space Reconstruction

The modelling of a dynamical system relies on the knowledge of the phase space, which is a collection of possible system states. The system state at time t consists of all information needed to determine the future system states for times $\geq t$. For a mathematically modelled system, like the Brusselator, the phase space is known from the equations of motion [14]. However, for experimental and naturally occurring chaotic dynamical systems, the phase space and a mathematical description of the system are often unknown. In practice, the number of dynamical variables available from a given system is restricted, and in the limiting case, it is one [15]. In fact, the whole dynamic information of a system might be contained in a limited number of variables, if this dynamic system has a strange attractor or a periodic orbit. This observation led Takens to introduce an algorithm in order to reconstruct an attractor with the information of only one of the state variables, based on the empirical observation that, along the attractor, each of the coordinates taken at consecutive time intervals can generically define an immersion of the attractor in a Euclidean space [16].

For example, if we have the time sampled measurement of a single variable of a dynamic system as $x(t) = x(t_0 + n\tau_s)$, where t_0 is the initial condition and τ_s is the sampling time, then it is possible, according to the Takens' time delay embedding theorem, to construct a higher dimensional phase space from that single univariate observation [15]. The purpose of time delay embedding is to unfold the projection back to a multivariate space representative of the original system, this is, a process of one-to-one mapping

of points on the attractor from its original dimensional space to the attractor in the reconstructed space [17].

According to Takens delay embedding theorem [11], from a single coordinate of a dynamic system in N dimension, for instance, $x(t) \in \mathbb{R}$, which leads to an observable time series $[x_1, x_2, x_3, \ldots, x_n]$ measured at a certain value of the control parameters, the signal can be embedded into a higher-dimensional phase space if we select an appropriate value of time delay (Δt) and embedding dimension (D). With this, a time series $\{X_i\}_{i=1}^{n}$ can be defined:

$$X_i = (x(i), x(i + \Delta t), x(i + 2\Delta t), \ldots, x(i + (D - 1)\Delta t)) \tag{2.2.1}$$

which represents the constructed delay-coordinate vectors, and where $i = 1, 2, 3, \ldots, N - (D - 1)\Delta t$. Given a dynamic system where only one time series from a sample of a single state variable is known, it is possible to obtain an approximate image of the geometry of the attractor for each value of D, when this is an integer. The plot between the elements of the vectors (2.2.1) shows the evolution of the dynamics of the system in D-dimensional phase space.

In order to reconstruct a system's phase space, one must properly select the reconstruction parameters D and Δt. Only for some pairs of these parameters there will be clear results. Selecting an appropriate value for the time delay Δt is not crucial for infinite noise-free data, since any value of the successive measurement can be chosen as an ideal time delay, according to Takens embedding theorem. However, the same does not apply for real data with finite length, lack of precision and presence of noise. The choice of time delay should be such that it will produce independent delayed coordinates in their reconstructed phase space, as every acquired signal contains new information in successive measurements of time [15]. This delay cannot be too small, as this will produce highly correlated delayed vectors; and it cannot be too large, as this will create completely uncorrelated delayed vectors and the reconstructed phase space will not accurately represent the true dynamics of the system.

One of the methods for finding the optimum time delay is the Average Mutual Information (AMI). The concept of AMI is based on information theory, and it measures the extent to which the delayed time series ($x(t + \Delta t)$) is related to the non-delayed one ($x(t)$) for a given Δt. The first local minima of AMI is chosen as the optimum time delay, effectively identifying a value of Δt for which the two time series share the least information.

The choice of a proper embedding dimension D is important in the reconstruction of phase space, as the dimension in which the attractor is unfolded needs to be sufficiently large. This ensures the presence of true neighbours for every trajectory in the reconstructed phase space, which means that, for an attractor that is unfolded in the d-dimensional space, the neighbours of each trajectory of the reconstructed attractor have the same neighbours as that observed in $d - 1$ phase space [15]. When, for a sufficiently large value of embedding dimension D, the immersion of the attractor in D-dimensional phase space does not present intersections, it can be said that the attractor's dynamic is described by a finite number of state variables. For the Brusselator system, a dimension of $D = 2$ was chosen.

The Brusselator system has two dynamic variables: an autocatalytic variable X and an inhibiting

variable Y. Using the Takens' technique, we intend to reconstruct the phase space of the original system in Figure 2.3, with only the information of the X variable. For our Brusselator system in (2.1.2), the set parameters were: $A = 1.0$, $B = 2.3$, $k_i = 1.0$ and, for simplicity, no diffusion was considered. These were the same conditions used to obtain the phase space in Figure 2.3. For these conditions, and selecting a time delay of $\Delta t = 7$, the phase space reconstruction of Figure 2.4 was obtained.

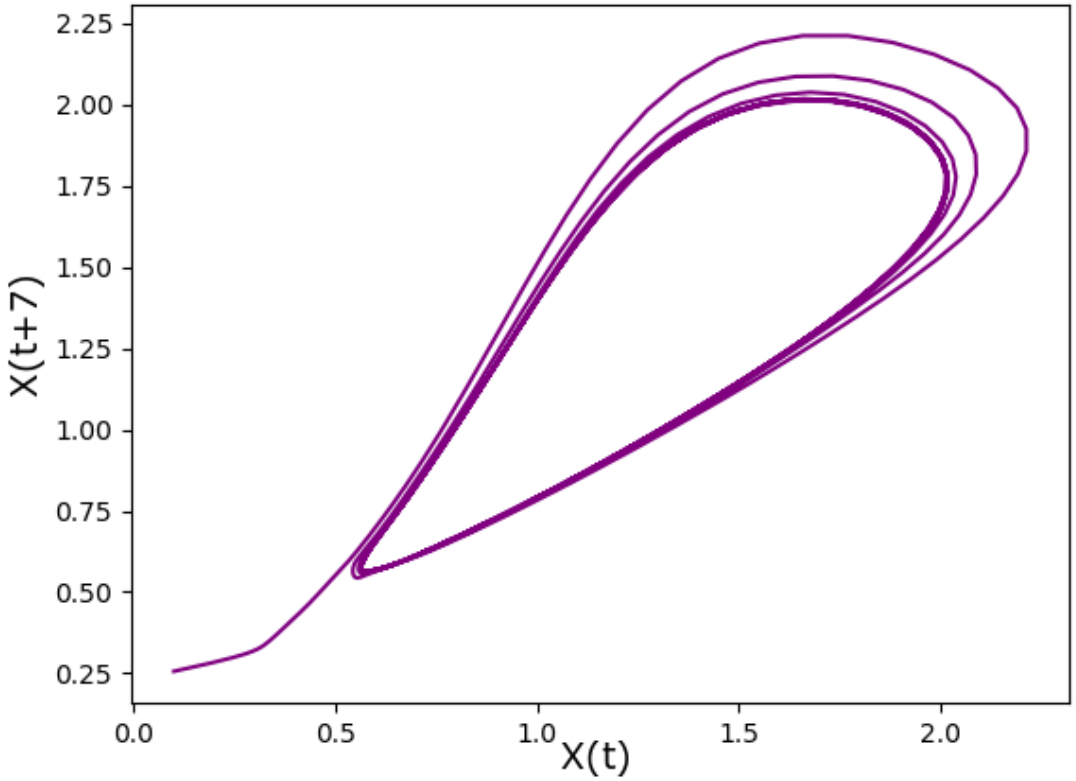


Figure 2.4: Brusselator's phase space reconstruction with the Takens' technique for reference conditions of this work: $A = 1.0$, $B = 2.3$ and $k_i = 1.0$, and reconstruction conditions of $D = 2$ and $\Delta t = 7$. The phase space of the original system is reconstructed with only the information of the autocatalytic variable X. It shows the same dynamic behaviour and follows the same shape as the original Brusselator phase space portrait.

Analysing Figure 2.4, we can see that the reconstructed phase space has the same shape (that is, the same qualitative properties) and shows the same dynamic behaviour as the original phase space. Experimentally, in the BZ reaction only a single diffusive variable is observed (the autocatalytic variable X), while variable Y is, in principle, not diffusive, but a control variable.

2.3 Reaction-Diffusion Pattern Simulation

In this section, reaction-diffusion patterns generated in diffusive media will be computationally simulated, using the two-dimensional Brusselator model. The main goal is to simulate patterns that appear experimentally through the Belousov-Zhabotinsky reaction. Although the BZ reaction leads to the formation of both concentric rings and spiral waves, in this work we will only be focusing on simulating the circular waves.

2.3.1 Numerical Integration of Reaction-Diffusion Systems

The existence of solid numerical algorithms gives the possibility to calibrate and validate a physical or chemical model for the emergence of a given coherent pattern. Model parameters can be calibrated with experimental data if simulation models show the same qualitative behaviour as their respective real systems. If this is the case for the calibrated model, it is validated, and its properties are characterized by the association of specific parameter values to the real system. Reaction-diffusion systems are usually non-linear, and their calibration and validation process bases on the comparison between data obtained through numerical solutions of R-D equations, and experimentally obtained data. This raises the problem of comparing numerical and analytical solutions of non-linear equations [18].

While linear equations that describe one dimensional reaction-diffusion systems are integrable, two dimensional models generically described by a system of equations:

$$\frac{\partial X}{\partial t} = f(X,Y) + D_X \nabla^2 X \tag{2.3.1a}$$

$$\frac{\partial Y}{\partial t} = g(X,Y) + D_Y \nabla^2 Y \tag{2.3.1b}$$

are not generally analytically integrable, making it necessary to use numerical methods to discretize the equations. The calculations are done using difference equations, which result from the application of discretization methods. Taking the unidimensional case into account, when applying the finite difference method on equation of the form (1.1) one obtains:

$$C_i^{k+1} = C_i^k + f(C_i^k) \cdot \Delta t + D_C \frac{\Delta t}{\Delta x^2}(C_{i-1}^k + C_{i+1}^k - 2 \cdot C_i^k) \tag{2.3.2}$$

where $i = 0, 1, \ldots, N-1$ and $k = 0, 1, \ldots$ represent the space and time coordinates of discrete space, respectively. Corresponding the continuous coordinates gives $x = i \cdot \Delta x$ for space, and $t = k \cdot \Delta t$ for time, where Δx and Δt represent the space and time discretization steps, respectively. The fact that space and time scales are not independent in diffusion, that is, different patterns are obtained in numerical simulations when Δx and Δt are varied independently, makes it necessary for a system to undergo a calibration process that aims to optimize the spatial and temporal discretization parameters as a way

of validating the applicability of a numerical method. In fact, there is a scale relation for the reaction-diffusion equations for which the error between the continuous equation solution and the discrete solution is minimal, independent of the chosen dimension. Numerical solutions approach the exact solution of the continuous equation for finite Δx and Δt, if the parameter $\gamma = \frac{D \cdot \Delta t}{\Delta x^2}$ assumes a fixed constant value, $\gamma = \frac{1}{6}$. Applying this parametrization to finite difference equations (2.3.2), the minimization of the global error shows geometrically through the patterns' symmetry properties preservation.

Considering the two-dimensional R-D model, the finite difference method applied to the numerical integration of (2.3.1) gives the system of difference equations:

$$X_{i,j}^{t+\Delta t} = \Delta t \cdot f(X_{i,j}^t, Y_{i,j}^t) + \frac{D_X \Delta t}{(\Delta x)^2}(X_{i-1,j}^t + X_{i+1,j}^t + X_{i,j-1}^t + X_{i,j+1}^t - 4X_{i,j}^t) \tag{2.3.3a}$$

$$Y_{i,j}^{t+\Delta t} = \Delta t \cdot g(X_{i,j}^t, Y_{i,j}^t) + \frac{D_Y \Delta t}{(\Delta x)^2}(Y_{i-1,j}^t + Y_{i+1,j}^t + Y_{i,j-1}^t + Y_{i,j+1}^t - 4Y_{i,j}^t) \tag{2.3.3b}$$

where i, j represent the coordinates of the vertices of a squared, symmetric lattice. Iterating through given initial conditions $X(x, y, 0) = f_1(x, y)$ and $Y(x, y, 0) = f_2(x, y)$, $X_{i,j}^0 = f_1(i\Delta x, j\Delta x)$ and $Y_{i,j}^0 = f_2(i\Delta x, j\Delta x)$, the solutions at time t are obtained. The system (2.3.3) is stable if $\gamma = \frac{max(D_X, D_Y) \cdot \Delta t}{\Delta x^2} \leq \frac{1}{4}$ [18].

2.3.2 General Simulation Conditions

The Brusselator model was studied and simulated, using the finite difference method described in the previous subsection. The 2D simulations were done in Python programming language, resorting to *SciPy*, *NumPy* and *Matplotlib* packages. Zero flux boundary conditions were used in the simulations, as periodic boundary conditions have specific symmetry properties that reflect on the patterns' symmetry properties. The chosen parametrization for the Brusselator reflected on simplicity of the algebraic equations and computational efficiency. Therefore, the values for the velocity constants of the model (2.1.2) were set to $k_1 = k_2 = k_3 = k_4 = 1.0$, and the control variables took the numerical values $A = 1.0$ and $B = 2.3$. The diffusion coefficients D_X and D_Y introduce diffusive instability which generates R-D patterns. Only the autocatalytic variable's diffusion coefficient was taken into account ($D_X = 1.0$), and the inhibiting variable is studied with no associated diffusion ($D_Y = 0.0$).

The wave phase propagates centrifugally for values of the autocatalytic variable's diffusion coefficient D_X greater than that of the inhibiting variable D_Y, which will be the only case studied. This happens because the diffusion will induce an advance of the phase space of the wave of concentration X, in a way that the phase relations change and the X wave appears before the Y wave, in spatial terms. The variable X propagates to regions where the concentration of Y is maximum, since the production of X depends on the concentration of Y, hence the centrifugal propagation. In this situation, the *pacemaker* works as a wave emitter. The phase space points, when leaving the steady state, evolve with a clockwise

curvature, according to the direction of the limit cycle's rotation in a homogeneous phase.

The initial distribution of the chemical species concentrations is given by the steady state condition of the model:

$$X^* = 1.0 \quad \text{and} \quad Y^* = 2.3 \tag{2.3.4}$$

Since biological systems usually assume periodic dynamics, in the form of repetitive oscillations or cycles (both spatially and temporally), an oscillating mathematical model is studied. However, one must keep in mind that pattern formation can occur in other dynamic situations, and what matters is the balance between reactive and diffusive processes, where the reactive processes counter the homogenizing effect of diffusion [5].

2.3.3 Dynamic Patterns

The most frequent pattern in the BZ reaction is the circular wave pattern, which is characterized by a constant propagation speed, which will be studied ahead. Using a thin layer of unstirred solution with the BZ reaction, point pacemakers start generating concentric chemical waves which are spontaneously formed, or punctually perturbed. Each oscillatory cycle creates a new wave front, forming target patterns [8].

To simulate circular patterns, one can simply apply an infinitesimal perturbation to a point in a still, homogeneous media, defined by steady state conditions. Since the steady state is unstable, the perturbed site evolves to an oscillating state whose effects propagate to the revolving space [5]. Any perturbation magnitude is efficient as long as there is not an escape from the limit cycle's domain of stability. For these simulations, the waves were initiated with a simple perturbation around the steady state in the central point of the square grid of size $M \times M$:

$$X[M/2][M/2] = X^*_{ss} + 1.0 \tag{2.3.5a}$$

$$Y[M/2][M/2] = Y^*_{ss} + 1.0 \tag{2.3.5b}$$

The use of different diffusion coefficients in the same diffusive space, discretized according to the explicit finite differences formulas, leads to the creation of non completely controllable effects. This is why, whenever possible, only the autocatalytic variable's diffusion coefficient is considered, and the inhibiting variable's diffusion is set to zero ($D_X = 1.0$, $D_Y = 0.0$).

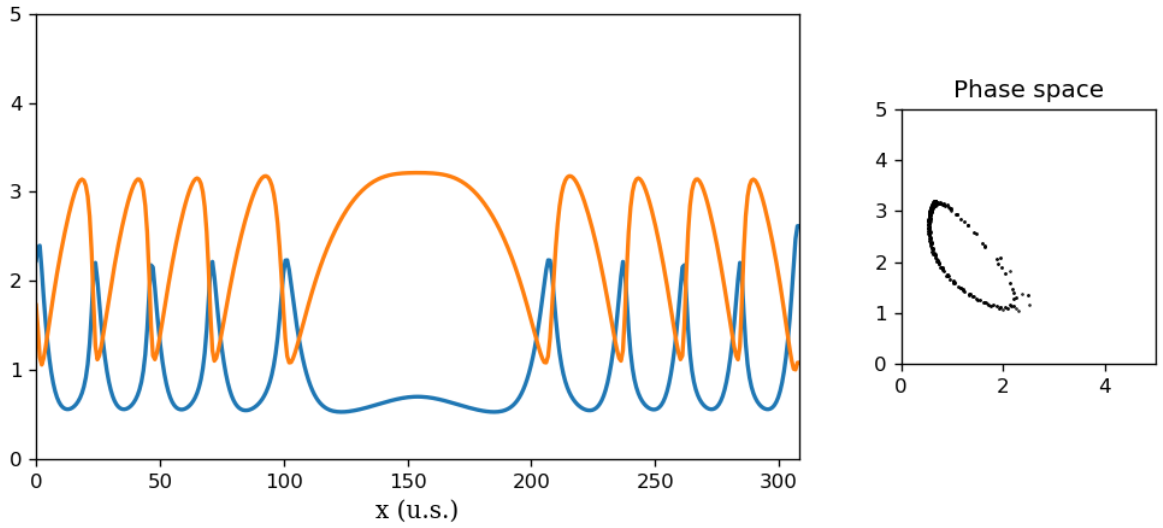


Figure 2.5: 1D dynamical patterns for a time of evolution of *200 u.t.*, with diffusion only on the autocatalytic variable ($D_X = 1.0$, $D_Y = 0.0$) and for reference conditions of this work. Periodic patterns are formed when the simulation starts with a symmetric perturbation. In the phase space, the limit cycle has a similar trajectory to that of the homogeneous system.

For these conditions, periodic patterns are formed when the simulation begins with a symmetric perturbation, as shown in Figure 2.5. The resulting pattern generates, in the phase space, a similar trajectory to the limit cycle of the homogeneous system.

It is in two dimensional spaces that reaction-diffusion patterns are better experimentally documented. In our work, we will be focusing on concentric waves.

The way the simulations are presented depends on the chosen colour code. The evolution of the X substance is documented with a Red-White colour code, and the Y substance is shown in a Blue-White colour code. To simulate the experimental formation of these waves, both canals are merged, forming a Red-Blue output (Figure 2.6). Mathematically, this means that both canals are intersected in order to obtain the output result.

2.3.4 Characteristic Curves

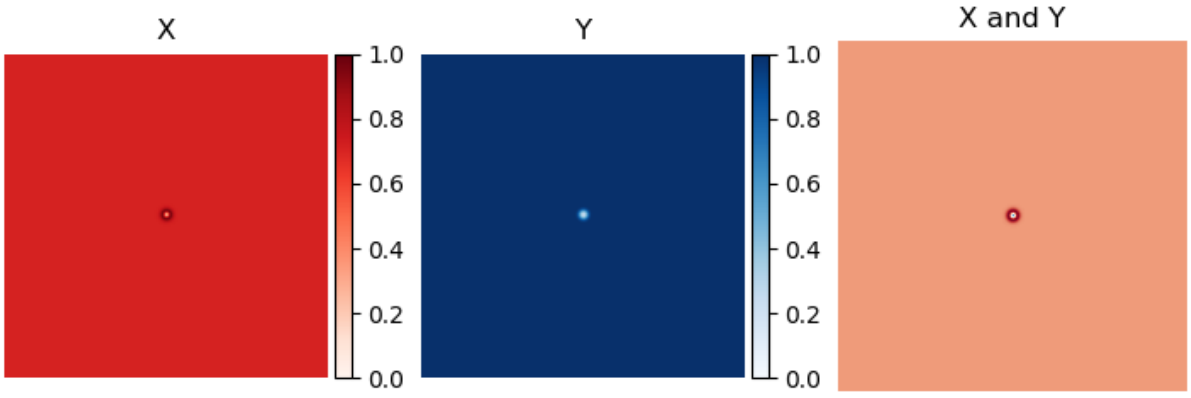
To better explain the definition of a characteristic curve, we begin by discussing a simple first-order partial differential equation in one dimension:

$$\frac{\partial \omega}{\partial t} + c\frac{\partial \omega}{\partial x} = 0 \tag{2.3.6}$$

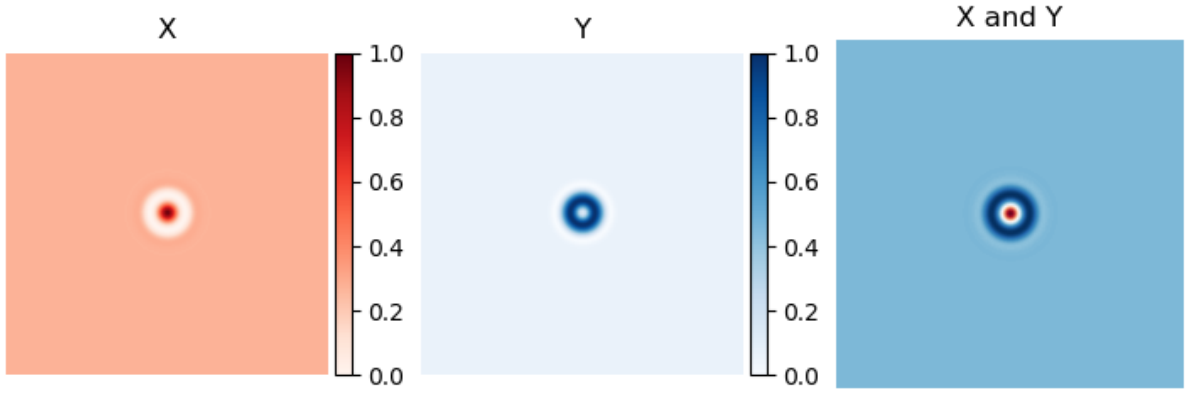
We consider the rate of change of $\omega(x(t), t)$ as measured by a moving observer, $x = x(t)$. The chain rules implies:

$$\frac{d}{dt}\omega(x(t), t) = \frac{\partial \omega}{\partial t} + \frac{dx}{dt}\frac{\partial \omega}{\partial x} \tag{2.3.7}$$

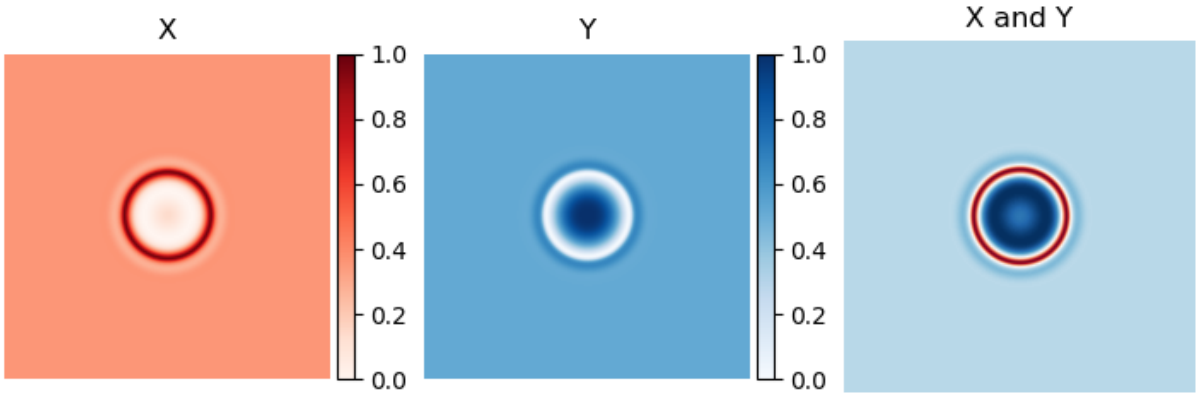
where the term $\partial\omega/\partial t$ represents the change in ω at the fixed position, and (dx/dt) $(\partial\omega/\partial x)$ represents



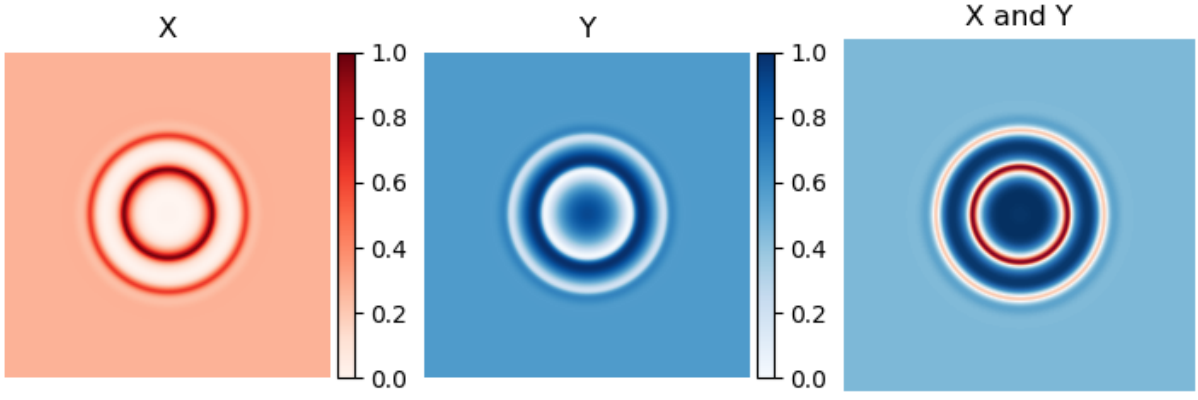
t = 1 *u.t.*



t = 25 *u.t.*



t = 50 *u.t.*



t = 75 *u.t.*

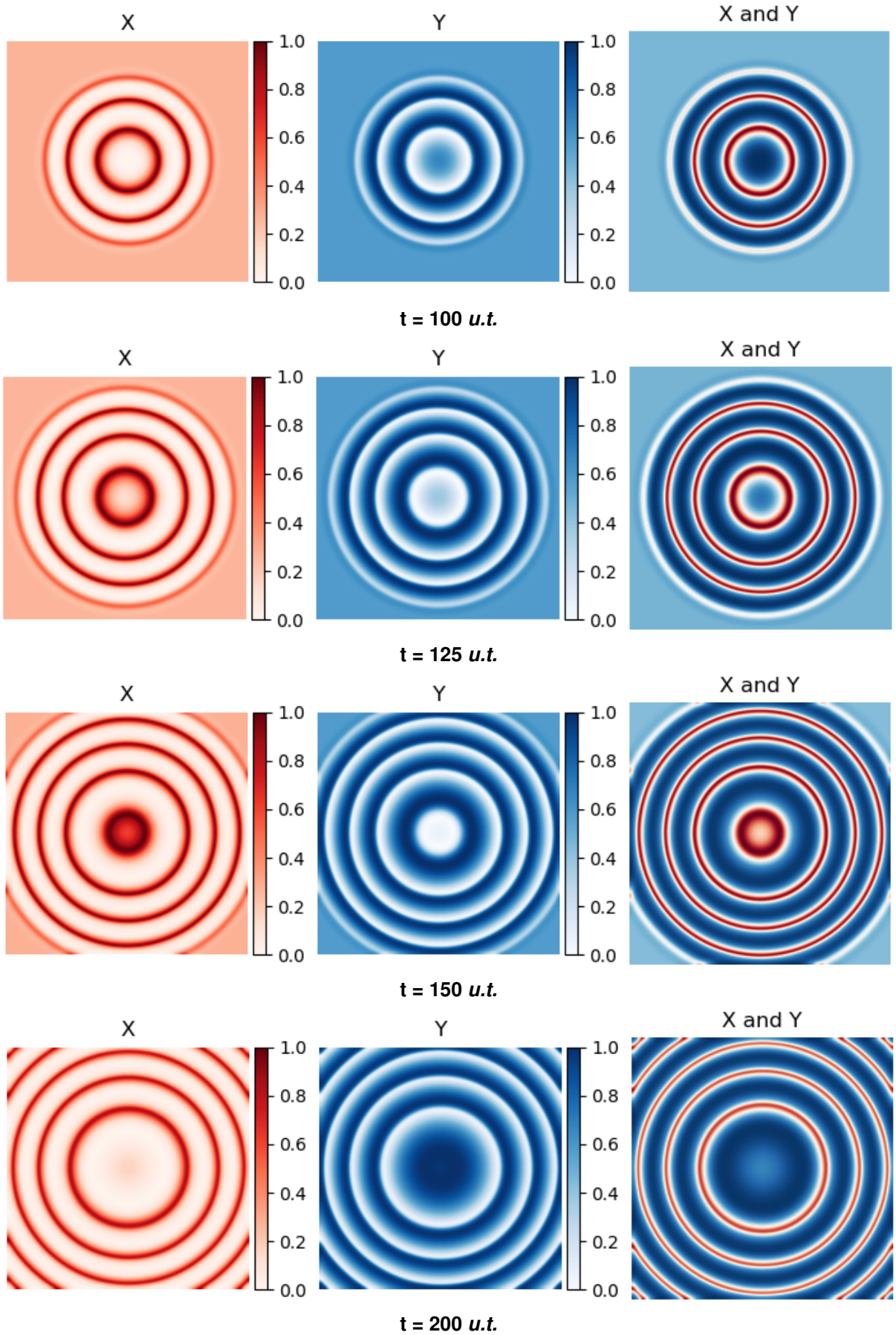


Figure 2.6: Target patterns in the Brusselator system, showing the time evolution of concentric waves over 200 u.t..

the change due to the fact that the observer moves into a region of possibly different ω.

Comparing the above two equations, and considering that the observer moves with velocity c such that

$$c = \frac{dx}{dt} \tag{2.3.8}$$

, then,

$$\frac{d\omega}{dt} = 0 \tag{2.3.9}$$

which means that ω is constant, that is, an observer moving at speed c would measure no changes in ω.

This way, the partial differential equation (2.3.6) has been replaced with the two ordinary differential equations above. Integrating (2.3.8) leads to the equation of characteristic curves:

$$x = ct + x_0 \tag{2.3.10}$$

When partial differential equations are the subject of study, and their solutions are waves that propagate throughout space, their velocities are studied with the method of characteristics. This means that the velocity c is measured according to the parameters that appear in the studied equations, since there is usually a dependence between the propagation speed and these parameters. For our study of the Brusselator, the parameters $A = 1.0$, $B = 2.3$ and $k_i = 1$ were kept fixed, as well as the autocatalytic variable's diffusion coefficient $D_X = 1.0$, while D_Y was varied between 0.0 and 0.5, in 0.1 steps.

We begin by making a spatial cut on the variables' concentrations in the middle of the y-axis, on a 100 $\times$ 100 grid. This will give us the reaction-diffusion waves along the x-axis, for $y = 50$. For $D_Y = 0$, a single wave-front is represented throughout space in Figure 2.7.

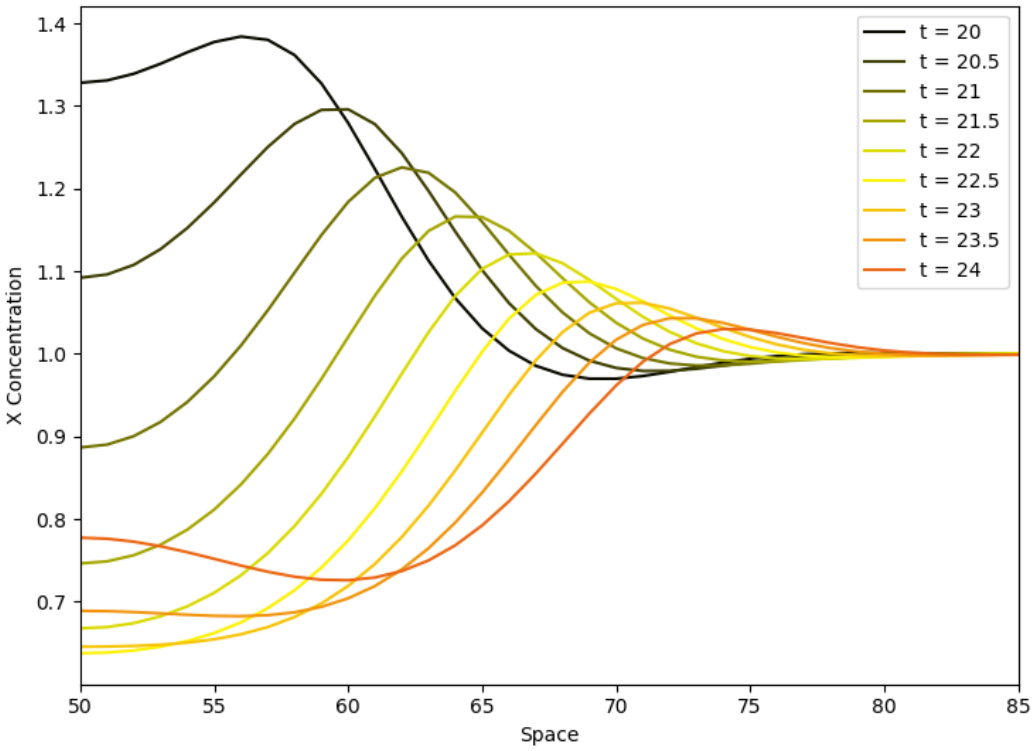
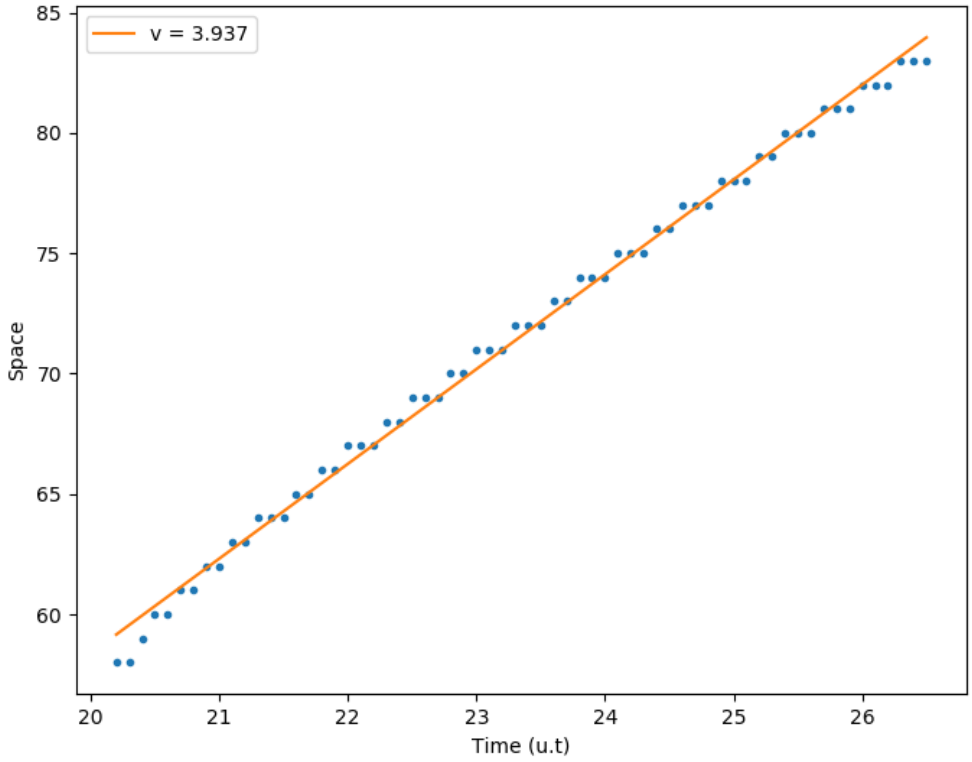
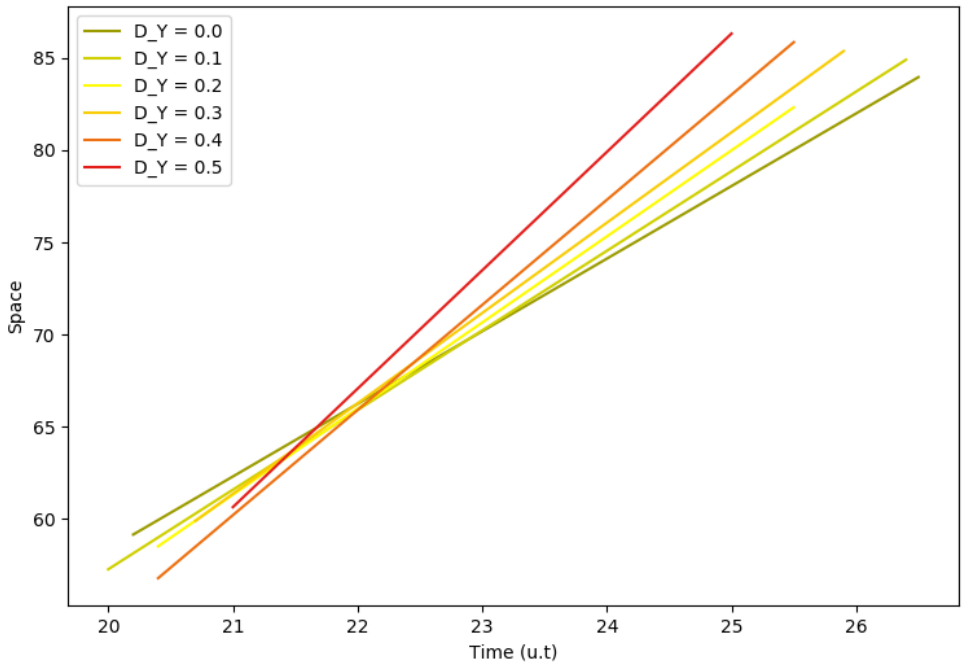


Figure 2.7: Time evolution of reaction-diffusion waves on $y = 50$, for $D_Y = 0.0$, between $t = 20$ u.t. and $t = 24$ u.t. with a 0.5 time step, for visual simplicity. A single wave-front is shown evolving in time along the x-axis.

For one single wave-front, we take its space and time coordinates for which there is a maximum concentration value. The slope of the line formed by these coordinates (Figure 2.8) gives the propagation speed of the waves, which can be approximately constant or, for some cases, slightly variable.



(a) The time and space coordinates were linearly fitted to find the propagation speed of the waves, for each value of D_Y.



(b) The propagation speed increases with D_Y.

Figure 2.8: Study of the propagation speed of concentric waves with parameters: $A = 1.0$, $B = 2.3$, $k_i = 1.0$, $D_X = 1.0$, and varying D_Y between 0.0 and 0.5. The slope of the line formed by the space and time coordinates for which there is a maximum concentration value for a single wave-front gives the propagation speed of the waves.

Analysing the propagation speed for each value of D_Y, it shows an non-linear increase according to the increase in D_Y.

Propagation Speed	D_Y = 0.0	D_Y = 0.1	D_Y = 0.2	D_Y = 0.3	D_Y = 0.4	D_Y = 0.5
v	3.937	4.317	4.667	4.898	5.698	6.418

Table 2.1: Results for the propagation speeds of the wave fronts for varying values of D_Y.

2.3.5 Reconstruction from a Single Spatial Point

Finally, the Brusselator model's phase space was reconstructed, following the time evolution of the concentration on a single point in a 100×100 grid. The data collected from the time evolution of the concentrations of X and Y at the point $x = y = 50$ was plotted to obtain the reconstruction in Figure 2.9.

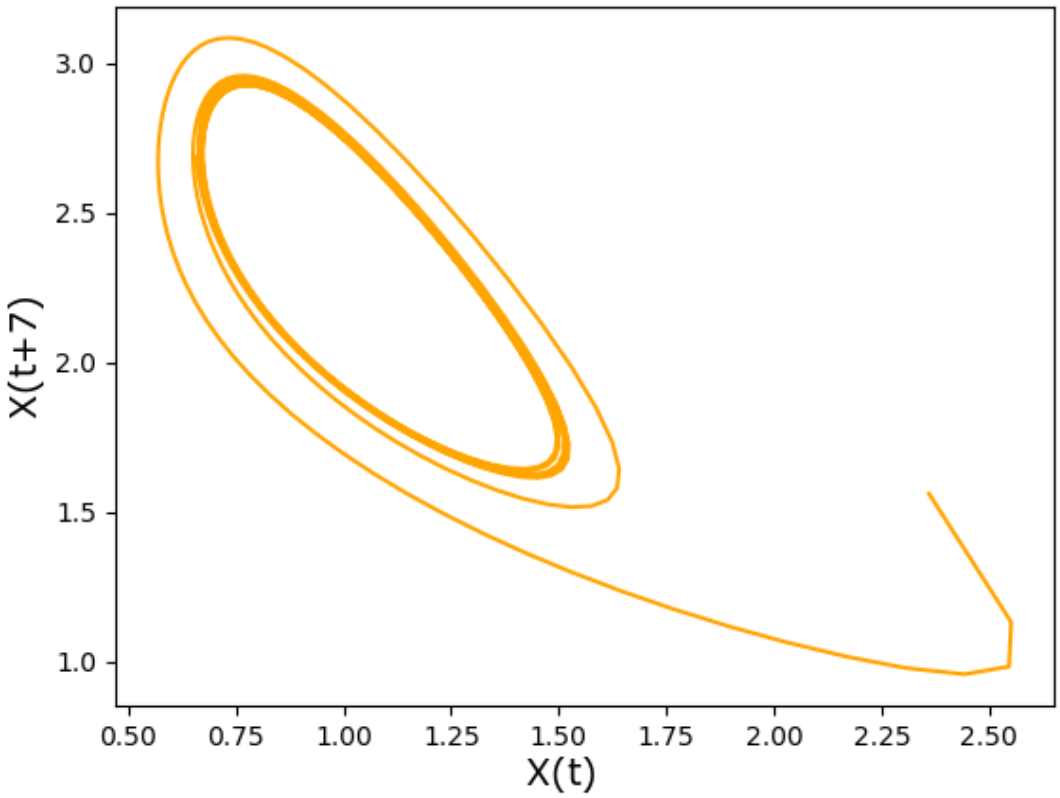


Figure 2.9: Phase space reconstruction from a single point in a 100×100 grid, in position $x = y = 50$, for parameters: $A = 1.0$, $B = 2.3$, $k_i = 1.0$, $D_X = 1.0$ and $D_Y = 0.0$. The reconstructed portrait has a similar dynamic behaviour and shape as the Brusselator phase space from Figure 2.3, constructed from every point in the system.

The same reconstruction was also performed with the Takens' technique described in Section 2.2, following the time evolution of only the concentration of the X variable taken from the spatial point $x =$

$y = 50$, with the same reconstruction parameters used in Figure 2.4: an embedding dimension of $D = 2$ and a time delay $\Delta t = 7$ (Figure 2.10).

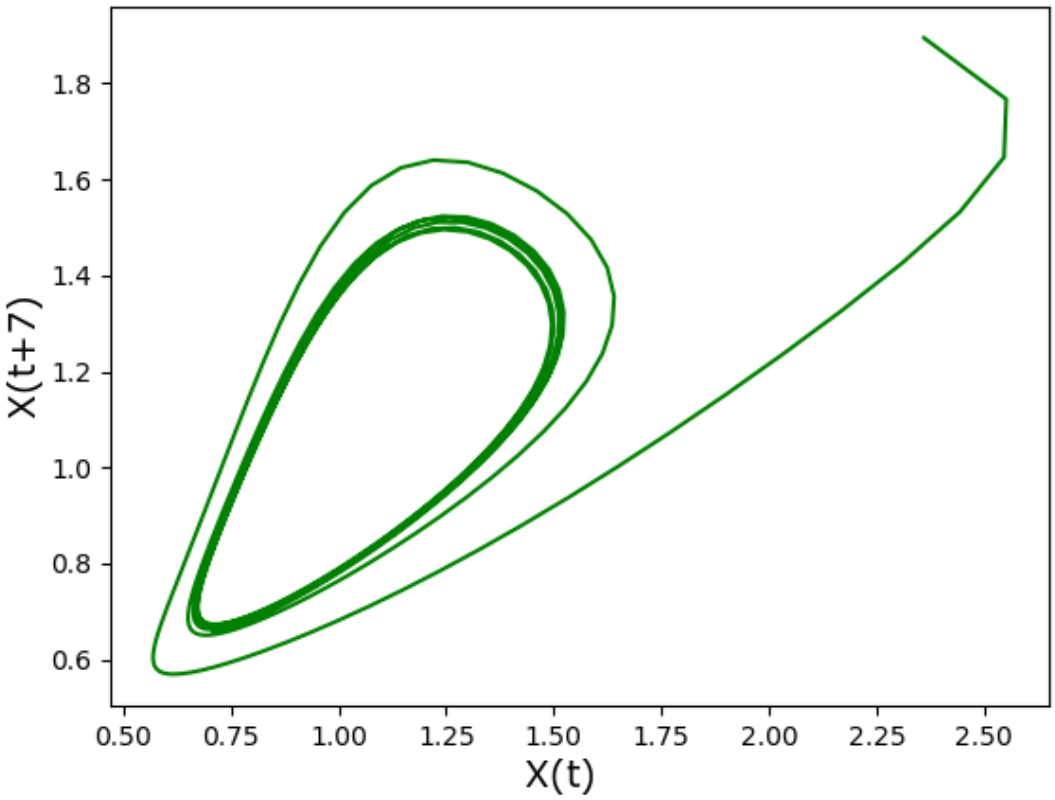


Figure 2.10: Brusselator's phase space reconstruction from a single point in position $x = y = 50$, with the Takens' technique ($D = 2$ and $\Delta t = 7$), for parameters: $A = 1.0$, $B = 2.3$, $k_i = 1.0$, $D_X = 1.0$ and $D_Y = 0.0$. The phase space of the system in Figure 2.9 is reconstructed with only the information of the autocatalytic variable X.

The point here is that the observation of a time series at a single point of the spatial domain indicates the existence of a limit cycle in the underlying local finite dimensional dynamics. The Takens technique has been shown for finite dimensional systems and it has not been tested for infinite dimensional dynamical systems such as the R-D equations. We propose to reconstruct the local dynamics from the experimental data. This shows that this technique is adequate.

Chapter 3

The Data Mining Algorithm

A data mining method that aims to derive governing equations from time series data collected at a fixed number of spatial locations will be implemented. The terms of the governing PDE that will represent the data with the greatest accuracy from a large library of possible governing PDEs will be a result of sparse regression techniques. Advances in sparse regression are allowing scientists and engineers to identify the structure and parameters of a non-linear system from data. In the last decade, a seminal breakthrough with an evolutionary algorithm [19, 20] and symbolic regression have made this possible. This method balances complexity and accuracy of the model. However, symbolic regression is expensive, and was found to not scale well to large systems and may be prone to overfitting if parsimonious models are not selected. More recently, sparse regression [21] has been used for this purpose [22, 23], which avoids overfitting by selecting the parsimonious models that manage to balance complexity of the model, measured in the number of terms, and model accuracy via Pareto analysis [20]. The method allows for time-series measurements to be made in a fixed frame, where the sensors are fixed spatially (Eulerian framework). This algorithm, PDE functional identification of non-linear dynamics (PDE-FIND), has proved to be able to select the correct linear, non-linear, and spatial derivative terms from a large library, resulting in the accurate identification of PDEs from data.

3.1 PDE-FIND Algorithm

The PDE-FIND algorithm aims to discover the governing equation for a discretized dataset, which is assumed to be the solution of a parameterized and non-linear PDE of the general form:

$$u_t = N(u, u_x, u_{xx}, ..., x, t, \mu) \tag{3.1.1}$$

where the subscripts represent partial differentiation in time or space, μ denotes parameters in the system, and N is an unknown right-hand side that is usually a non-linear function of $u(x,t)$, its derivatives, and μ parameters. The main objective of the algorithm is to construct N from time series measurements of the system at a fixed number of spatial locations in x. It is assumed that the function N may be expressed as a sum of a small number of terms, which is the case for the PDEs which are widely used in

practice, and which makes the space of possible contributing terms very large compared to the sparse functional form.

Upon discretization, we denote **U** to be a matrix containing the values of u, and hence the right hand side of equation (3.1.1) can be expressed as a function of **U**, and **Q** is a matrix containing additional information about the system that may be relevant.

PDE-FIND begins by first collecting all the spatial time series data and combining it into a single column vector $\mathbf{U} \in \mathbb{C}^{n \times m}$, which represents data collected over m time points and n spatial locations. The additional input is also considered in a column vector $\mathbf{Q} \in \mathbb{C}^{n \times m}$. Then, the algorithm creates a large library $\mathbf{\Theta}(\mathbf{U}, \mathbf{Q}) \in \mathbb{C}^{nm \times D}$ of D candidate terms that may appear in N, including linear and non-linear terms, and partial derivatives, and then selects a sparse subset of active terms from this list. The candidate terms are then combined into a matrix $\mathbf{\Theta}(\mathbf{U}, \mathbf{Q})$:

$$\mathbf{\Theta}(\mathbf{U}, \mathbf{Q}) = \begin{bmatrix} 1 & \mathbf{U} & \mathbf{U}^2 & \dots & \mathbf{Q} & \dots & \mathbf{U}_x & \mathbf{U}\mathbf{U}_x & \dots & \mathbf{Q}^2\mathbf{U}^3\mathbf{U}_{xxx} \end{bmatrix} \tag{3.1.2}$$

Each column of $\mathbf{\Theta}$ contains all the values for a candidate function across all the grid points on which data was collected. This means that, if data is combined on an $n \times m$ grid (e.g. a 256×100 grid represents 256 spatial measurements at 100 time points), and there are 50 candidate terms in the PDE, then $\mathbf{\Theta}(\mathbf{U}, \mathbf{Q}) \in \mathbb{C}^{256 \cdot 100 \times 50}$ [1]. The time derivative is also taken to compute $\mathbf{U}_t$ and it is then reshaped into a column vector, just like the columns of $\mathbf{\Theta}$. The PDE evolution can be represented by the linear equation:

$$\mathbf{U}_t = \mathbf{\Theta}(\mathbf{U}, \mathbf{Q})\xi \tag{3.1.3}$$

$$\begin{bmatrix} u_t(x_0, t_0) \\ u_t(x_1, t_0) \\ u_t(x_2, t_0) \\ \vdots \\ u_t(x_{n-1}, t_m) \\ u_t(x_n, t_m) \end{bmatrix} = \begin{bmatrix} 1 & u(x_0, t_0) & u_x(x_0, t_0) & \dots & u^5 u_{xxx}(x_0, t_0) \\ 1 & u(x_1, t_0) & u_x(x_1, t_0) & \dots & u^5 u_{xxx}(x_1, t_0) \\ 1 & u(x_2, t_0) & u_x(x_2, t_0) & \dots & u^5 u_{xxx}(x_2, t_0) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & u(x_{n-1}, t_m) & u_x(x_{n-1}, t_m) & \dots & u^5 u_{xxx}(x_{n-1}, t_m) \\ 1 & u(x_n, t_m) & u_x(x_n, t_m) & \dots & u^5 u_{xxx}(x_n, t_m) \end{bmatrix} \begin{bmatrix} \xi \end{bmatrix} \tag{3.1.4}$$

In a number of examples, the number of rows in a linear system is equal to the total number of data points, which calls for a very large system. We should be able to represent the PDE with a sparse vector of coefficients ξ if Θ is assumed to be an over complete library, meaning it has a sufficiently rich column space that the dynamics will be in its range. This amounts to picking enough candidate functions that the full PDE can be written as a weighted sum of library terms. In this linear system, each row represents an observation of the dynamics at some point in time and space:

[1] Note that $\mathbf{\Theta}(\mathbf{U}, \mathbf{Q})$ is not a square matrix, implying that the solutions of (3.1.3) are not unique. This follows from the fact that non square systems of linear equations have an infinite set of right and left inverses — Moore-Penrose inverses.

$$u_t(x,t) = \sum_j \Theta_j(u(x,t), q(x,t))\xi_j \tag{3.1.5}$$

One could think to simply solve the least squares problem for ξ in order to get a representation of the dynamics. However, the least-square problem is poorly conditioned for regression problems similar to (3.1.5). ξ would mostly have non-zero values, suggesting a PDE with every functional form contained in the library. This way, if least squares was to be used, it could radically change the qualitative nature of the deduced dynamics.

3.2 Sparse Regression

Regular linear regression models tend to perform poorly in situations with large multivariate datasets, so penalized sparse regression becomes a better alternative. Penalized regression allows to create a linear regression model that is penalized for having too many variables in the model, by adding a constraint in the equation and, this way, reducing the coefficient values towards zero. In general, the sparsest vector ξ that satisfies equation (3.1.3) with a small residual is required. Sparse regression is used to approximate a solution of

$$\xi = argmin_{\hat{\xi}} \left\| \mathbf{\Theta}\hat{\xi} - \mathbf{U}_t \right\|_2^2 + \lambda \left\| \hat{\xi} \right\|_0 \tag{3.2.1}$$

where ξ represents the true (unknown) parameter value that generated the data and $\hat{\xi}$ is its estimate, and $\lambda \left\| \hat{\xi} \right\|_0$ is a penalty term ($\lambda > 0$ represents how much is penalized). The term $\left\| \hat{\xi} \right\|_0$ is the ℓ_0 pseudo norm and represents the number of non zero components of ξ. We intend to minimize the residual sum of squares denoted in the first term of the right-hand side of equation (3.2.1), since the value $\hat{\xi}$ that minimizes the function is usually a good estimator for ξ.

A few regression methods were tested in order to see which one gave the most accurate results. The first one, a common technique called least absolute shrinkage and selection operator (LASSO), was to relax the problem to a convex ℓ_1 regularized least squares [21], which has a norm function $\left\| \hat{\xi} \right\|_1 = \sum_{j=1}^{p} |\hat{\xi}_j|$, changing the penalty term in (3.2.1). This model has the effect of forcing some of the coefficient estimates, with a minor contribution to the model, to be equal to zero. The convex relaxation of the ℓ_0 optimization problem in (3.2.1) is given by:

$$\xi = argmin_{\hat{\xi}} \left\| \mathbf{\Theta}\hat{\xi} - \mathbf{U}_t \right\|_2^2 + \lambda \left\| \hat{\xi} \right\|_1 \tag{3.2.2}$$

However, this technique was found to have difficulty finding a sparse basis when the data matrix Θ has high correlations between columns, which is the case for many dynamical systems.

Then, a second, alternative, method for sparse regression was tested, which is called sequentially thresholded least squares (STLS) [22]. In STLS, a least squares predictor is obtained and then all coefficients that are smaller than some cut-off value λ are thresholded. Once the indices of the remaining non-zero coefficients are identified, a new least squares predictor is obtained onto the remaining indices.

These new coefficients are again thresholded with respect to λ, and the process continues until the non-zero coefficients converge.

This algorithm is described below, in Matlab:

Sparse representation algorithm in Matlab

```
%% compute Sparse Regression : Sequential Least Squares
Xi = Theta\dXdt; % initial guess : Least Squares

% lambda is our sparsification knob
for k=1:10
smallinds = (abs(Xi)<lambda); % find small coefficients
Xi(smallinds)=0;              % and threshold
for ind = 1:n                 % n is state dimension
biginds = ~smallinds(:,ind);
% Regress dynamics onto remaining terms to find sparse Xi
Xi(biginds,ind) = Theta(:,biginds)\dXdt(:,ind);
end
end
```

This method, although it demonstrated to be highly efficient when applied to the algorithm of Brunton and his co-authors [22], did not give outstanding results when applied to the PDE-FIND algorithm. Although it seemed to outperform LASSO in some cases, it did not avoid the challenge of correlation in the data, and so a regularizer for the least squares problem was implemented: ridge regression. This is an ℓ_2 regularized variation of the least squares problem, with an ℓ_2 norm that corresponds to the sum of the squared coefficients, which corresponds to shrinking the regression coefficients so that variables with minor contribution to the outcome have their coefficients close to zero. It is defined by:

$$\begin{aligned} \hat{\xi} &= argmin_{\xi} \left\| \boldsymbol{\Theta}\hat{\xi} - \mathbf{U}_t \right\|_2^2 + \lambda \left\| \xi \right\|_2^2 \\ &= (\boldsymbol{\Theta}^T\boldsymbol{\Theta} + \lambda I)^{-1}\boldsymbol{\Theta}^T\mathbf{U}_t \end{aligned} \tag{3.2.3}$$

In this algorithm, least squares from STLS is substituted by ridge regression, and so this regression method was given the name Sequential Threshold Ridge regression (STRidge), outlined in Algorithm 1. Note that this algorithm reduces to STLS when $\lambda = 0$.

Algorithm 1 STRidge($\boldsymbol{\Theta}$, $\mathbf{U}_t$, λ, tol, iters)

$\hat{\xi} = argmin_{\xi} \left\| \boldsymbol{\Theta}\hat{\xi} - \mathbf{U}_t \right\|_2^2 + \lambda \left\| \xi \right\|_2^2$ ▷ ridge regression
bigcoeffs = $\{j : |\hat{\xi}_j| \geq tol\}$ ▷ select large coefficients
$\hat{\xi}$[$\sim$ bigcoeffs] = 0 ▷ apply hard threshold
$\hat{\xi}$[bigcoeffs] = STRidge($\boldsymbol{\Theta}$, [:,bigcoeffs], $\mathbf{U}_t$, tol, iters - 1) ▷ recursive call with fewer coefficients
return $\hat{\xi}$

A method for finding the best threshold tolerance was also implemented, since different values of tolerance will give a different level of sparsity in the final solution. This way, predictors are trained at varying tolerances and their performance considering an ℓ_0 penalty is used to find the best one. The algorithm for this method is described in Appendix A.

Each column of Θ is normalized to unit variance as a preprocessing step, which is useful so that higher powers are neither very large or small. A final prediction of ξ is then obtained by regressing the non-normalized data onto the identified terms.

3.2.1 Subsampling Data

As mentioned before, the PDE-FIND algorithm can be used on subsampled data when the datasets are large, such as those with more than one spatial dimension. In the linear system in (3.1.3), a fraction of the rows is ignored: a set of spatial points is randomly selected and the data is evenly sampled in time at a lower frequency than data is collected, which results in using only a fraction of the dataset. The subsampling method is illustrated below:

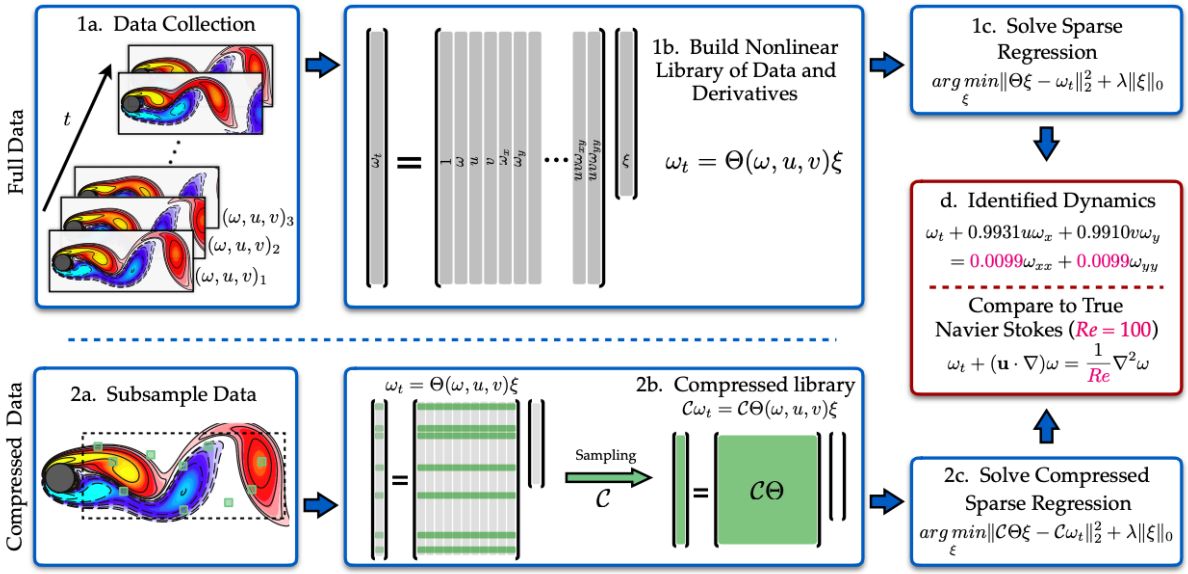


Figure 3.1: Steps in the PDE-FIND algorithm, applied to infer the Navier-Stokes equation from data, for a full dataset and for a compressed dataset. When datasets are very large, the algorithm can be used on subsampled data without losing accuracy.

The steps in Figure 3.1 can be described in the following way, given that the steps in the panels 1a - 1c are taken for a full dataset, and steps in 2a - 2c are taken for only a fraction of a dataset: starting from data (**1a.**), a large linear system is formed to represent the PDE, by taking the numerical derivatives and compiling the data into a large matrix Θ, where every column is a possible non-linear function of the data (**1b.**). Sparse regression is then used (**1c.**) to identify active terms in the PDE (**d.**) . For large datasets, we can identify the same dynamics by using a small subset of the data points, so sparse sampling may be used to reduce the size of the problem (**2a.**), which is equivalent to taking a subset of rows from the

linear system (**2b.**). An identical sparse regression problem is formed but with fewer rows (**2c.**), and finally the active terms in ξ are synthesized into a PDE (**d.**).

3.2.2 Numerical Evaluation of Derivatives

The biggest challenge of the PDE-FIND method is the numerical evaluation of derivatives, and it is also the most important task for its success [1]. The most reliable method for computing derivatives was found to be polynomial interpolation. For each point where there is a derivative being computed, a polynomial of degree P is fit to greater than P points, and derivatives of the polynomial are taken to be approximate to those of the numerical data. It is difficult to fit a polynomial near the boundaries, so the data points close to them are not used in the regression. This data is difficult to differentiate, and it was found that this strongly influences the results and accuracy of PDE-FIND.

3.3 Code

This section demonstrates the code where the PDE-FIND algorithm is implemented on the Brusselator system. The code aims to derive the PDEs for both quantities X and Y (which, for coherence with the PDE-FIND algorithm code, will be from now on called u and v), where they have a dependence on one another. We expect the algorithm to converge to a system of equations of the form:

$$\frac{\partial u}{\partial t} = A - (B+1)u + u^2 v + D_u\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{3.3.1a}$$

$$\frac{\partial v}{\partial t} = Bu - u^2 v + D_v\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{3.3.1b}$$

with $A = 1$ and $B = 2.3$ for every studied case.

First, we import the data file and specify the variables in the Matlab file. The n is then set to the length of x (or y) which is the number of spatial points of each axis on the grid, and identically for the temporal points.

```
data = sio.loadmat('brusselator.mat')


t = data['t'][:,0]
x = data['x'][0,:]
y = data['y'][0,:]
U = data['u']
V = data['v']


n = len(x) # same as y
steps = len(t)
dx = x[2]-x[1]
```

```
dy = y[2]-y[1]
dt = t[2]-t[1]
```

To confirm the data file is being well read, we can display a figure of the state of the system at a given point in time.

```
fig = plt.figure(figsize=(12,6))
xx, yy = np.meshgrid(
   np.arange(n)*dx,
   np.arange(n)*dy)
plt.subplot(1,2,1)
plt.pcolor(xx,yy,U[:,:,50],cmap='coolwarm')
plt.title('U', fontsize = 20)
plt.xlabel('x', fontsize = 16)
plt.ylabel('y', fontsize = 16)
plt.subplot(1,2,2)
plt.pcolor(xx,yy,V[:,:,50],cmap='coolwarm')
plt.title('V', fontsize = 20)
plt.xlabel('x', fontsize = 16)
plt.ylabel('y', fontsize = 16)

plt.show()
```

Next, the subsampling task is done by setting a value for both the number of spatial and temporal points desired to use in the fit. Ideally, every value should be used, but this would require an excessively long computing time and memory usage. The number of spatial points chosen should be large in order to work with noise, if that is the case for the system being studied.

The code for selecting points in time is so that they are evenly spaced in time. In this case, 60 out of 200 temporal points are sampled and these are taken in steps of 3. The sampled points are distributed evenly between the ends of the simulation, with a boundary of 10 points on either side.

```
num_xy = 5000
num_t = 60
num_points = num_xy * num_t
boundary = 5
points = {}
count = 0

for p in range(num_xy):
   x = np.random.choice(np.arange(boundary,n-boundary),1)[0]
```

```
y = np.random.choice(np.arange(boundary,n-boundary),1)[0]
for t in range(num_t):
   points[count] = [x,y,3*t+10]
   count = count + 1
```

To construct Θ and compute U_t, we take the derivatives around each one of the points and combine these with the candidate functions of u and v for the PDE.

```
u = np.zeros((num_points,1))
v = np.zeros((num_points,1))
ut = np.zeros((num_points,1))
vt = np.zeros((num_points,1))
ux = np.zeros((num_points,1))
uy = np.zeros((num_points,1))
uxx = np.zeros((num_points,1))
uxy = np.zeros((num_points,1))
uyy = np.zeros((num_points,1))
vx = np.zeros((num_points,1))
vy = np.zeros((num_points,1))
vxx = np.zeros((num_points,1))
vxy = np.zeros((num_points,1))
vyy = np.zeros((num_points,1))

N = 2*boundary-1 # number of points to use in fitting
Nt = N
deg = 4 # degree of polynomial to use

for p in points.keys():

   [x,y,t] = points[p]

   # value of function
   u[p] = U[x,y,t]
   v[p] = V[x,y,t]

   # time derivatives
   ut[p] = PolyDiffPoint(U[x,y,t-(Nt-1)/2:t+(Nt+1)/2], np.arange(Nt)*dt, deg, 1)[0]
   vt[p] = PolyDiffPoint(V[x,y,t-(Nt-1)/2:t+(Nt+1)/2], np.arange(Nt)*dt, deg, 1)[0]

   # spatial derivatives
```

```
    ux_diff = PolyDiffPoint(U[x-(N-1)/2:x+(N+1)/2,y,t], np.arange(N)*dx, deg, 2)
    uy_diff = PolyDiffPoint(U[x,y-(N-1)/2:y+(N+1)/2,t], np.arange(N)*dy, deg, 2)
    vx_diff = PolyDiffPoint(V[x-(N-1)/2:x+(N+1)/2,y,t], np.arange(N)*dx, deg, 2)
    vy_diff = PolyDiffPoint(V[x,y-(N-1)/2:y+(N+1)/2,t], np.arange(N)*dy, deg, 2)
    ux_diff_yp = PolyDiffPoint(U[x-(N-1)/2:x+(N+1)/2,y+1,t], np.arange(N)*dx, deg, 2)
    ux_diff_ym = PolyDiffPoint(U[x-(N-1)/2:x+(N+1)/2,y-1,t], np.arange(N)*dx, deg, 2)
    vx_diff_yp = PolyDiffPoint(V[x-(N-1)/2:x+(N+1)/2,y+1,t], np.arange(N)*dx, deg, 2)
    vx_diff_ym = PolyDiffPoint(V[x-(N-1)/2:x+(N+1)/2,y-1,t], np.arange(N)*dx, deg, 2)

    ux[p] = ux_diff[0]
    uy[p] = uy_diff[0]
    uxx[p] = ux_diff[1]
    uxy[p] = (ux_diff_yp[0]-ux_diff_ym[0])/(2*dy)
    uyy[p] = uy_diff[1]

    vx[p] = vx_diff[0]
    vy[p] = vy_diff[0]
    vxx[p] = vx_diff[1]
    vxy[p] = (vx_diff_yp[0]-vx_diff_ym[0])/(2*dy)
    vyy[p] = vy_diff[1]

# Form a huge matrix using up to quadratic polynomials in all variables.
X_data = np.hstack([u,v])
X_ders = np.hstack([np.ones((num_points,1)), ux, uy, uxx, uxy, uyy, vx, vy, vxx,
    vxy, vyy])
X_ders_descr = ['','u_{x}', 'u_{y}','u_{xx}','u_{xy}','u_{yy}','v_{x}',
    'v_{y}','v_{xx}','v_{xy}','v_{yy}']
X, description = build_Theta(X_data, X_ders, X_ders_descr, 3, data_description =
    ['u','v'])
['1'] + description[1:]
```

Finally, we solve for ξ with the TrainSTRidge function of the algorithm (Appendix A.1). This function splits the data up into 80% training and 20% validation, then searches over multiples values of tolerance in the STRidge algorithm and finds the one with the best performance and accuracy on the validation set.

```
c = TrainSTRidge(X,ut,10**-5,5)
print_pde(c, description)

c = TrainSTRidge(X,vt,10**-5,5)
```

```
print_pde(c, description, ut = 'v_t')
```

3.4 Examples

In order to test and better understand the algorithm, two examples were implemented: the simplest one, a simple solution to the Korteweg–de Vries (KdV) equation, and a reaction-diffusion system, similar to the one being studied. This helped to understand how the Brusselator data would have to be adapted in order to obtain the most accurate results, and how the algorithm responds to changes in different parameters (number of spatial points/timesteps, grid size, etc.).

3.4.1 KdV

The first implemented example was a simple solution to the KdV equation having two non-interacting traveling waves with different amplitudes. The KdV equation [24] is an asymptotic simplification of Euler equations, used to model waves in shallow water. The reason why a two-wave system was tested is because single travelling waves in a solution to the KdV equation behave linearly. Solutions with waves at multiple amplitudes exhibit nonlinear behaviour due to the amplitude dependence of wave speed.

Given the dataset in the *Github* page of the PDE-FIND algorithm for the two soliton interaction, the KdV equation was created from 512 spatial points and 200 timesteps. The inital condition is a superposition of functions of the form:

$$u(x,t) = \frac{c}{2} sech^2\left(\frac{\sqrt{c}}{2}(x - ct - a)\right) \tag{3.4.1}$$

with different amplitudes and offset centers, where c is the travelling speed of a single soliton. This solution is shown in Figure 3.2:

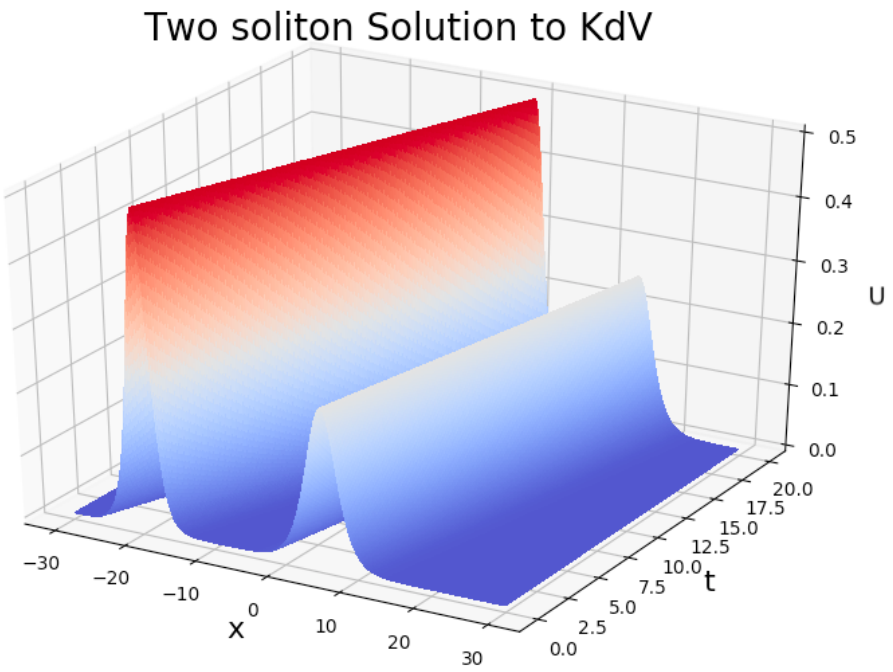


Figure 3.2: Numerical solution to the KdV equation.

Then, specifying the highest derivative and the highest degree polynomial of u to appear in Θ, a function of the algorithm builds a linear system with candidate functions for the PDE. Finally, sparse regression is applied and the final derived PDE is printed. The results are summarized for clean data and for data added 1% noise in Table 3.1.

Correct PDE	$u_t + 6uu_x + u_{xxx} = 0$
Identified PDE (clean data)	$u_t + 5.956uu_x + 0.988u_{xxx} = 0$
Identified PDE (1% noise)	$u_t + 6.152uu_x + 1.124u_{xxx} = 0$

Table 3.1: Identification of the KdV equation with PDE-FIND.

As expected, the algorithm produces the equation with great accuracy, with an error of $(1 \pm 0.2)\%$ for clean data and $(7 \pm 5)\%$ for data with added 1% noise.

3.4.2 Reaction-Diffusion system

The second implemented example was a reaction-diffusion system, in this case a lambda-omega reaction-diffusion system. This system has the form:

$$\frac{\partial u}{\partial t} = \lambda(A)u - \omega(A)v + D_u\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{3.4.2a}$$

$$\frac{\partial v}{\partial t} = \omega(A)u - \lambda(A)v + D_v\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{3.4.2b}$$

$$A^2 = X^2 + Y^2, \omega = -\beta A^2, \lambda = 1 - A^2 \tag{3.4.2c}$$

where, in this example, $D_u = D_v = 0.1$ and $\beta = 1.0$.

Model (3.4.2) is called the "λ - ω system" and was introduced by Kopell and Howard. Although it is claimed that the system does not correspond to any specific physical situation, the λ - ω system is still meaningful and is discussed in various sources [25].

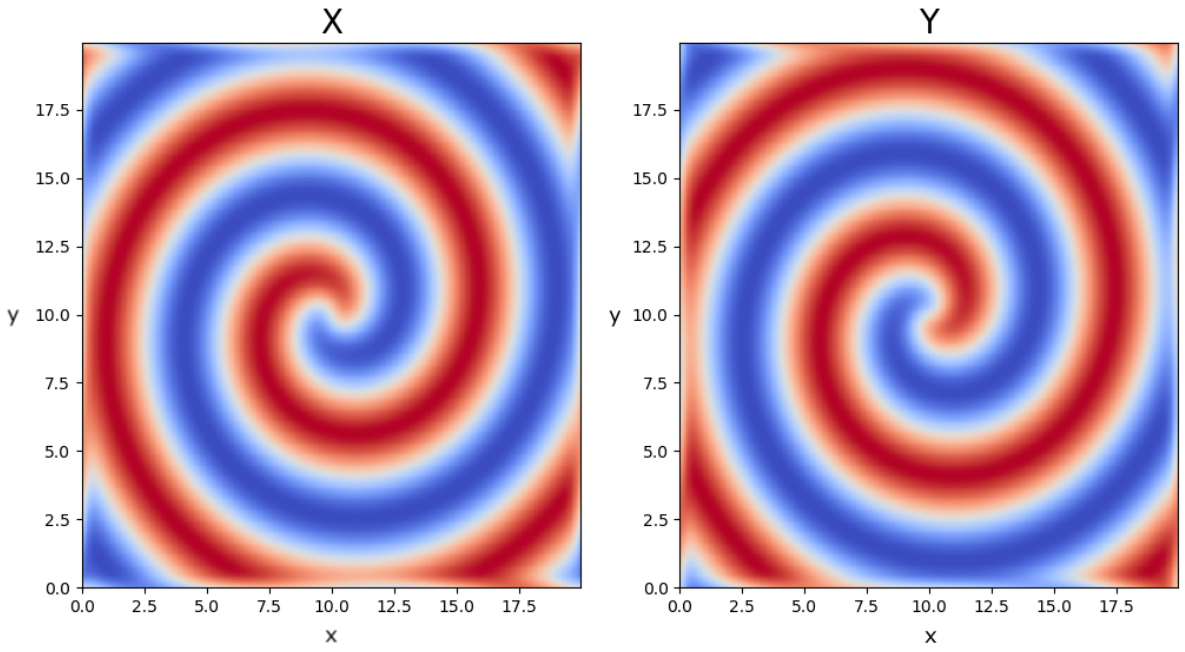


Figure 3.3: Representation of the numerical solution to the λ - ω system of equations, plotted in space-time, for the timeframe of $t = 10$ u.t.. The red colour represents the X variable (u) and the blue colour represents the Y variable (v).

This system exhibits spiral waves on a 2D domain with periodic boundaries, as shown in Figure 3.3. In this identification, 5000 spatial points were sampled with 30 time points each, which results in $\approx 1.14\%$ of the dataset. Applying the algorithm after specifying the highest derivative and highest degree polynomial of X and Y gave the results shown in Table 3.2.

This gives an error of $(0.004 \pm 0.005)\%$ for the clean data and an error of for the data with 0.5% noise.

The validation of this system is extremely relevant since it helps us to predict a satisfactory performance of the algorithm in the Brusselator system, given the similarities between the two.

Correct PDE	$u_t = 0.1u_{xx} + 0.1u_{yy} - uv^2 - u^3 + v^3 + u^2v + u$ $v_t = 0.1v_{xx} + 0.1v_{yy} + v - uv^2 - u^3 - v^3 - u^2v$
Identified PDE (clean data)	$u_t = 0.100013u_{xx} + 0.100010u_{yy} - 1.000007uv^2 - 1.000008u^3 +$ $+0.999995v^3 + 0.999995u^2v + 1.000018u$ $v_t = 0.100010v_{xx} + 0.100012v_{yy} + 1.000033v - 0.999996uv^2 -$ $-0.999996u^3 - 1.000024v^3 - 1.000024u^2v$
Identified PDE (0.5% noise)	$u_t = 0.0.95u_{xx} + 0.095u_{yy} - 0.945uv^2 - 0.945u^3 +$ $+1.000v^3 + 1.000u^2v + 0.945u$ $v_t = 0.095v_{xx} + 0.095v_{yy} + 0.946v - 1.000uv^2 -$ $-1.000u^3 - 0.946v^3 - 0.946u^2v$

Table 3.2: Identification of the λ - ω reaction-diffusion equation system with PDE-FIND.

3.5 Limitations

There are a few known situations in which the PDE-FIND algorithm may not converge to the desired solutions. One of them is when there is an incomplete library of terms. When applying the algorithm to a dataset, it is assumed that the column space of Θ is sufficiently rich to have a sparse representation of its time dynamics. However, when the algorithm is applied to a dataset where the dynamics are unknown, it might happen that the column space of Θ is insufficient. When this is the case, the PDE-FIND algorithm will usually not converge to the real dynamics of a system. One thing that might happen, for instance, in a system which involves a fourth spatial derivative where we don't allow for fourth order derivatives, is that an expected u_{xxxx} term will be replaced by something like uu_{xxx}.

In datasets with high levels of noise, identifying governing equations is made challenging due to the problem of numerically differentiating noisy data. It is expected that, by increasing the level of noise, the PDE-FIND algorithm is able to identify the correct terms with increasing error until the noise level reaches a value for which more terms than necessary are added in the equations. While this value shows to be around 10% for the systems studied in [1], the reaction-diffusion systems seemed to be much less robust to noise, with misidentified terms at just 1% noise.

For the algorithm to be effective, we need to have far more data points then library functions, which is usually the case. Using a large number of points not only helps supply sufficient data for the regression,

but most importantly it makes numerical evaluation of derivatives possible. Solutions computed on courser grids, this is, grids with less spatial and temporal points, show a decline in accuracy when compared to solutions computed from finer grids. In Chapter 4, the PDE-FIND method will be tested in a number of discretizations of the Brusselator solutions, showing its accuracy with various grid sizes and sampling times.

Chapter 4

Results

In this chapter, the results for the application of the PDE-FIND algorithm to the Brusselator system are presented. The methods and code from Chapther 3 are used, inferring the R-D system that best fits the computed data.

If the model is successfully able to identify the correct system of partial differential equations of the Brusselator system, we show that it is possible to obtain valuable information of the local dynamics of a system from data, and therefore to infer its underlying kinetic mechanisms.

4.1 PDE-FIND algorithm on the Brusselator system

After carefully analysing the PDE-FIND algorithm, deeply learning how it works and testing its accuracy with the examples mentioned in Chapter 3, it was finally tested on a dataset of the 2D Brusselator system. This dataset was computed in Matlab, setting the initial conditions and the equations for the evolution of the system, its parameters, grid size and data acquisition time. The Brusselator parameters were set to: $A = 1.0$, $B = 2.3$, $k_i = 1.0$, $D_X = 1.0$ and $D_Y = 0.0$. The variables were then saved on a Matlab file format to be read by the code in Section 3.3, which applies the PDE-FIND model to the computed data. For all the tested cases, the algorithm constructed a matrix Θ with 11 candidate functions.

To ensure that the data acquisition was successful and the dataset accurately represented the Brusselator system, a snapshot of its evolution at a certain time point is shown in Figure 4.1 for the first tested case: circular waves.

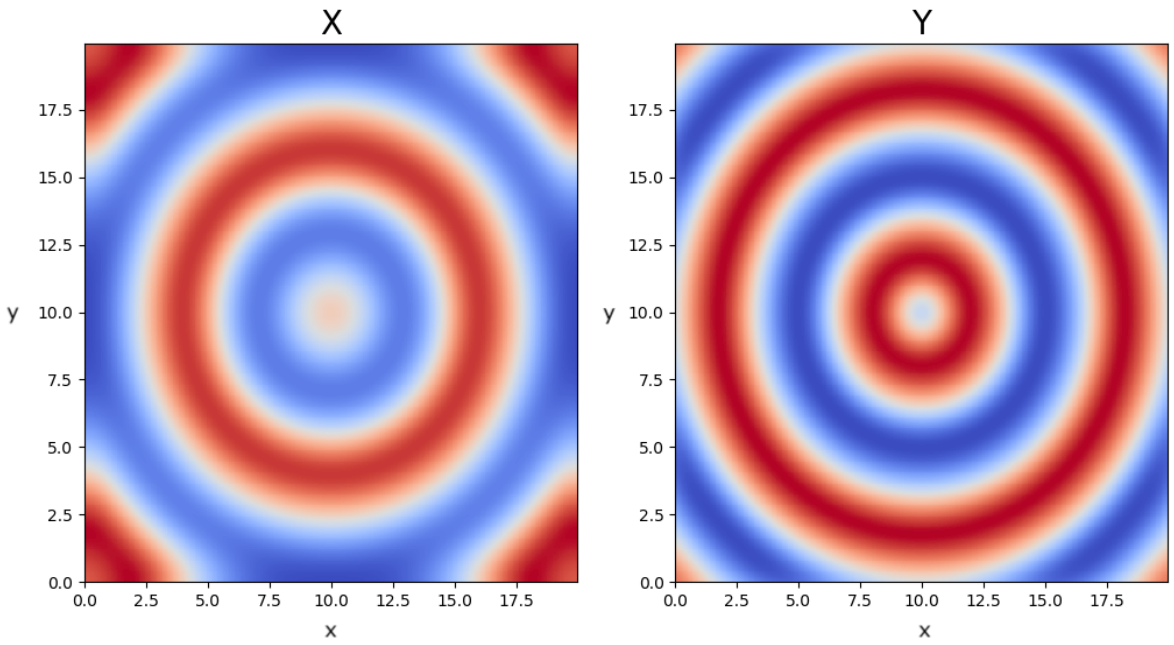


Figure 4.1: Numerical solution to the Brusselator system of equations, plotted in space-time for $t = 10$ u.t.. Red colour represents the autocatalytic X substance and blue represents the Y substance.

In order to find the optimal conditions for applying the algorithm, a lot of tests were made to see which combination of number of sampling spatial and temporal points would give the best results, while still maintaining a decently low running time. The conditions which can be varied according to how many sampling points we want in our simulation, as well as how fine or coarse we choose our grid of data points to be, are: the number of spatial points in the grid $m \times m$, the number of temporal points in the simulation n as well as the time step dt, the number of spatial (num_{xy}) and temporal (num_t) points to be actually used in the simulation, and the step taken by the algorithm when it is searching for the optimal tolerance d_{tol} . As such, the chosen testing conditions were:

$$n = 512, m = 201, dt = 0.05 \tag{4.1.1}$$

$$num_{xy} = 50 \tag{4.1.2}$$

$$num_t = 85 \tag{4.1.3}$$

$$d_{tol} = 1 \tag{4.1.4}$$

Results obtained with a lower number of sampling points resulted in a less accurate convergence, which will be shown more in depth in Section 4.2. Running the code with the above mentioned testing conditions, the algorithm converged to equations for u and v very satisfactorily close to the real equations, and the results are presented in Table 4.1.

This result shows that the algorithm was able to accurately identify the PDE with an error of $(0.04 \pm 0.01)\%$ when compared to the original set of equations.

Correct PDE	$u_t = 1.0 + u_{xx} + u_{yy} - 3.3u + u^2v$ $v_t = 2.3u - u^2v$
Identified PDE	$u_t = 0.999538 + 0.999808u_{xx} + 0.999375u_{yy} - 3.298330u - 0.999497u^2v$ $v_t = 2.299332u - 0.999729u^2v$

Table 4.1: Identification of the Brusselator reaction-diffusion equation system with PDE-FIND.

In order to see if the method would be equally successful for different parameter values, in this case with the addition of diffusion on the inhibiting variable, two other values of D_Y were tested. The results for $D_Y = 0.5$ and $D_Y = 1.0$ are shown in Tables 4.2 and 4.3, respectively. Although the algorithm was able to converge, a larger number of sampling points ($num_{xy} = 10000$) had to be used for both cases, otherwise the method would be unsuccessful. The remaining testing conditions 4.1.1 were maintained.

Correct PDE	$u_t = 1.0 + u_{xx} + u_{yy} - 3.3u + u^2v$ $v_t = 0.5v_{xx} + 0.5v_{yy} + 2.3u - u^2v$
Identified PDE	$u_t = 0.998089 + 1.047309u_{xx} + 0.996114u_{yy} - 3.290348u - 0.995294u^2v$ $v_t = 0.500109v_{xx} + 0.500111v_{yy} + 2.294337u - 0.994988u^2v$

Table 4.2: Identification of the Brusselator reaction-diffusion equation system with PDE-FIND ($D_Y = 0.5$).

Correct PDE	$u_t = 1.0 + u_{xx} + u_{yy} - 3.3u + u^2v$ $v_t = v_{xx} + v_{yy} + 2.3u - u^2v$
Identified PDE	$u_t = 0.996333 + 1.001942u_{xx} + 1.001798u_{yy} - 3.283741u - 0.997793u^2v$ $v_t = 1.000130v_{xx} + 1.000112v_{yy} + 2.299436u - 0.999773u^2v$

Table 4.3: Identification of the Brusselator reaction-diffusion equation system with PDE-FIND ($D_Y = 1.0$).

The algorithm was able to identify the PDE's for both cases with errors of $(0.78 \pm 1.41)\%$ for $D_Y = 0.5$ and $(0.17 \pm 0.16)\%$ for $D_Y = 1.0$.

Finally, the algorithm was also tested on spiral waves, shown in Figure 4.2. Maintaining the same parameters as the original system ($A = 1.0$, $B = 2.3$, $k_i = 1.0$, $D_X = 1.0$ and $D_Y = 0.0$) and testing conditions 4.1.1, the spiral waves are obtained by changing the initial conditions. Table 4.4 shows how the algorithm converged for this case.

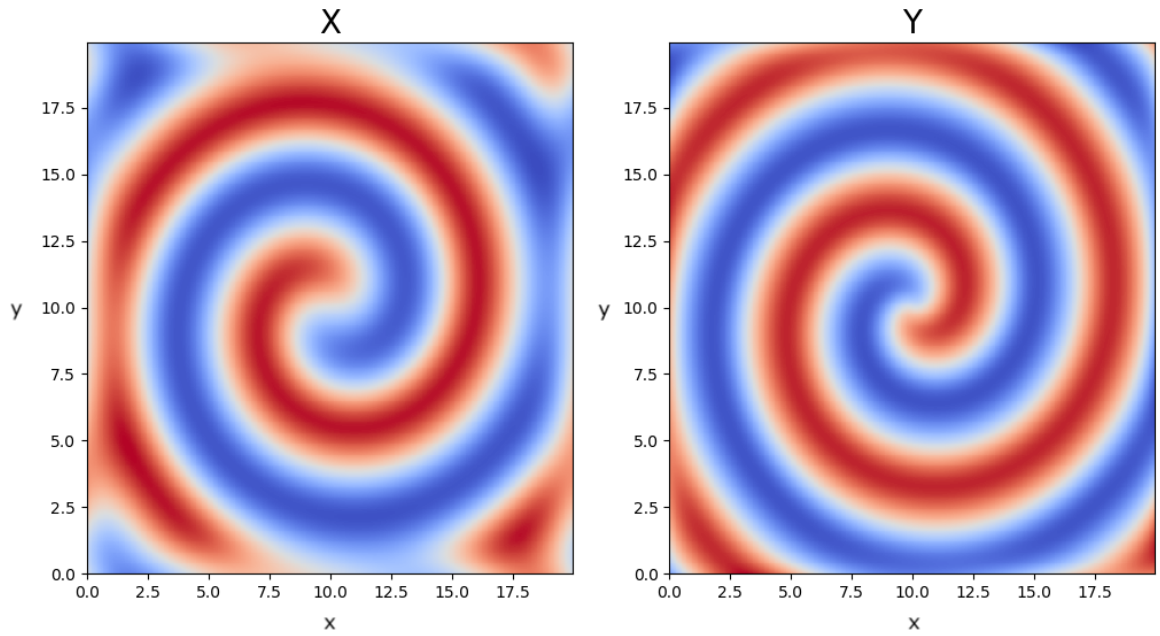


Figure 4.2: Numerical solution to the Brusselator system of equations, exhibiting spiral waves, for $t = 10$ u.t..

Correct PDE	$u_t = 1.0 + u_{xx} + u_{yy} - 3.3u + u^2v$ $v_t = 2.3u - u^2v$
Identified PDE	$u_t = 0.999620 + 0.999541u_{xx} + 0.999577u_{yy} - 3.298610u - 0.999578u^2v$ $v_t = 2.299442u - 0.999773u^2v$

Table 4.4: Identification of the Brusselator reaction-diffusion equation system with PDE-FIND, for spiral waves.

For a Brusselator system exhibiting spiral waves, the method was once again able to accurately converge to the real system of PDE's with an error of $(0.04 \pm 0.01)\%$.

4.2 Limited Data

As mentioned in Section 3.5, one of the cases for which the PDE-FIND algorithm may not converge to the desired solution is when there is not a rich enough dataset with a sufficient number of points to successfully numerically evaluate the derivatives. In Table 4.5, the algorithm is tested on a number of discretizations of the Brusselator equations. Initially, the system was computed on a finer grid over 201 temporal points, and successively computed on to coarser grids over shorter sampling times, in order to evaluate the method with courser sampling.

		Spatial Points (n)				
		512	256	128	64	32
Temporal Points (m)	201	0.041	0.041	0.049	0.28	
	101	0.77	0.71			
	51					

Table 4.5: Accuracy of the PDE-FIND algorithm with various grids sizes on the Brusselator system. Red table entries denote a misidentification of the sparsity pattern either due to the inclusion of extra terms which are not present in the real PDE, or missing any term of either the PDE for u_t or v_t. Numbers shown represent the average parameter error in percentage.

Finally, a similar analysis was done, now varying the number of spatial sampling points num_{xy} on a grid of size 512, over an interval of 201 u.t. , where only 85 sampling temporal points are chosen. This was done to test the lower limit from which the algorithm fails to correctly converge.

Number of sampling points	5000	1000	100	50	25	15	10
Error in percentage	0.041	0.039	0.036	0.041			

Table 4.6: Accuracy of the PDE-FIND algorithm with different numbers of spatial sampling points num_{xy} on the Brusselator system. The numbers shown represent the average parameter error in percentage. Red table entries denote a misidentification of the sparsity pattern due to missing any term of either the PDE for u_t or v_t.

Analysing Table 4.6, we can see that there is a lower threshold of 25 sampling spatial points from which the PDE stops being correctly identified. This is important to keep in mind when performing the upcoming experimental work, described in Section 5.2.

Chapter 5

Conclusions

5.1 Achievements

In this work, we have analysed the possibility of calibrating a R-D partial differential equation exhibiting travelling waves transient solutions with the PDE-FIND algorithm. With this, we have shown the possibility of introducing the BZ data to obtain information about the local dynamics and therefore for underlying kinetic mechanisms of the BZ reaction. Our ultimate goal would be to find the optimal R-D system that best fits the BZ reaction data.

5.2 Future Work

Due to the pandemic, we were unable to reach the ultimate goal of applying the algorithm to collected data of the BZ reaction, which would be obtained experimentally in a lab. After having successfully calibrated the model, the next step would be to capture and analyse images of reaction-diffusion travelling waves of the Belousov-Zhabotinsky reaction in order to create a dataset to be read by the algorithm. To do this, we would start by creating the reaction in the lab, by mixing sulphuric acid, ferroin solution, sodium bromate, potassium bromide and sodium malonate. Then, resorting to suitable material and software, video footage of the evolution of the reaction would be taken and analysed. For this work, only circular waves would be studied. The different colour channels (red and blue) would be separated to represent the X and Y variables, respectively. Then, a dataset containing these variables would be created by scanning images of the travelling waves during successive time instants, which would transform the visual data into numerical data points. Finally, the algorithm would read this dataset and deduce the kinetic mechanism associated with reaction-diffusion waves observed in the Belousov-Zhabotinsky reaction.

Bibliography

[1] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 2017. 3:e1602614.

[2] J. Sainhas and R. Dilão. Morfogénese em sistemas de reacção-difusão. *Ciência*, 9.VI(12), 1998.

[3] A. M. Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 237(641):37–72, 1952.

[4] H. Meinhardt. *Models of Biological Pattern Formation*. Academic Press, 05 1982.

[5] J. Sainhas. *Morfogénese em Sistemas de Reacção-Difusão*. PhD thesis, Instituto Superior Técnico, 1999.

[6] J. Tyson. What Everyone Should Know About the Belousov-Zhabotinsky Reaction. *Frontiers in Mathematical Biology*, 100:569–587, 1994.

[7] I. R. Epstein and J. A. Pojman. *An Introduction to Nonlinear Chemical Dynamics*. Oxford University Press, 1998.

[8] A. M. Zhabotinsky. Belousov-Zhabotinsky reaction. *Scholarpedia*, 2(9):1435, 2007. doi:10.4249/scholarpedia.1435.

[9] S. Ault and E. Holmgreen. Dynamics of the Brusselator. 2003.

[10] A. M. Zhabotinsky. A history of chemical oscillations and waves. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 1(4):379–386, 1991. doi:10.1063/1.165848.

[11] F. Takens. *Detecting Strange Attractors in Turbulence.*, volume 898. 11 2006. ISBN 978-3-540-11171-9. doi: 10.1007/BFb0091924.

[12] I. Prigogine and R. Lefever. Symmetry breaking instabilities in dissipative systems. ii. *Chemical Physics - CHEM PHYS.*, 48(4):1695–1700, 1968. doi:10.1063/1.1668896.

[13] R. H. Enns and G. McGuire. *Nonlinear Physics with Mathematica for Scientists and Engineers*. Birkhauser, 2001.

[14] T. D. Sauer. Attractor reconstruction. *Scholarpedia*, 1(10):1727, 2006. doi:10.4249/scholarpedia.1727.

[15] R. Sujith. Tutorial 2: Tools from nonlinear dynamics. 2019.

[16] R. Dilão. *Uma Introdução à Teoria dos Sistemas Dinâmicos e do Caos*. 2018.

[17] M. B. Kennel, R. Brown, and H. D. Abarbanel. Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Physical review A*, 45(6):3403, 1992.

[18] R. Dilão and J. Sainhas. Validation and calibration of models for reaction-diffusion systems. *International Journal of Bifurcation and Chaos*, 8(6):1163—-1182, 1998.

[19] J. Bongard and H. Lipson. Automated reverse engineering of nonlinear dynamical systems. *Proc. Natl. Acad. Sci. U.S.A.*, 104:9943–9948, 2007.

[20] M. Schimidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324: 81—-85, 2009.

[21] R. Tibshirani. Regression shrinkage and selection via the lasso. *J. R. Stat. Soc. B*, 58:267—-288, 1996.

[22] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proc. Natl. Acad. Sci. U.S.A.*, 113:3932–3937, 2016.

[23] K. Champion, B. Lusch, J. N. Kutz, and S. L. Brunton. Data-driven discovery of coordinates and governing equations. *Proc. Natl. Acad. Sci. U.S.A.*, 116:22445–22451, 2019.

[24] T. Xiang. A summary of the korteweg-de vries equation. 2015.

[25] J. J. Lin and S. Y. Huang. Explicit periodic travelling waves for a discrete lambda–omega reaction–diffusion system. *Journal of Difference Equations and Applications*, 20, 09 2014. doi: 10.1080/10236198.2014.903935.

Appendix A

Supplementary Code

In this section, some code and algorithms to better understand how the PDE-FIND method works and how it was used in this work are provided. While the main piece of code is in its entirety in Section 3.3, a few functions from PDE-FIND are called and it is important to understand what these do.

A.1 TrainSTRidge Algorithm

As previously mentioned in Section 3.2, the method used in PDE-FIND to perform sparse regression is called STRidge, whose algorithm is written in Chapter 3. This method chooses the active terms in the equations and sets to zero the coefficients of variables whose contribution to the outcome is minor, preventing the outcome from having too many terms which might not be relevant to the dynamics of the system.

A method to find the best threshold tolerance was also implemented, since this influences the level of sparsity in the final solution. The best tolerance is found by taking into account the performance of predictors which are trained at varying tolerances. The ℓ_0 penalty is set to be proportional to the condition number of the matrix Θ, which enforces sparsity for high correlated data. The condition number measures the sensitivity of a function to changes or errors in the input, and how this influences the error in the output. The search algorithm receives as arguments Θ, U_t, λ, d_{tol}, tol_{iters} and STR_{iters}, as seen on Algorithm 2. The last three are passed directly to STRidge. The size of the step taken while looking for the optimal tolerance is told to the search algorithm by d_{tol}, and tol_{iters} indicates how many times the algorithm refines its guess at the best tolerance.

A.2 PDE-FIND Functions

The code in Section 3.3 calls three functions from the PDE-FIND algorithm: PolyDiffPoint, build_theta and TrainSTRidge, whose algorithm was described in Section A.1. In this section, these three functions are presented, to better understand how the method works.

Algorithm 2 TrainSTRidge($\mathbf{\Theta}$, $\mathbf{U}_t$, λ, d_{tol}, tol_{iters}, STR_{iters})

Split the data into training and testing sets with an 80/20 split
$\mathbf{\Theta} \rightarrow [\mathbf{\Theta}^{train}, \mathbf{\Theta}^{test}]$
$\mathbf{U}_t \rightarrow [\mathbf{U}_t^{train}, \mathbf{U}_t^{test}]$

Set an appropriate ℓ_0 penalty. A 10^{-3} multiplier was chosen
$\eta = 10^{-3}\kappa(\mathbf{\Theta})$

Get a baseline predictor
$\xi_{best} = (\mathbf{\Theta}^{train})^{-1}\mathbf{U}_t^{train}$
error$_{best}$ = $\|\mathbf{\Theta}^{test}\xi_{best} - \mathbf{U}_t^{test}\|_2^2 + \eta \|\xi_{best}\|_0$

# Search through values of tolerance to find the best predictor
tol = d_{tol}
for $iter$ = 1, ..., tol_{iters} :

    # Train and evaluate performance
    ξ = STRidge($\mathbf{\Theta}^{train}$, $\mathbf{U}_t^{train}$, λ, tol, STR_{iters})
    error = $\|\mathbf{\Theta}^{test}\xi - \mathbf{U}_t^{test}\|_2^2 + \eta \|\xi\|_0$

    # Is the error still dropping?
    if error $\leq$ error$_{best}$:
        error$_{best}$ = error
        ξ_{best} = ξ
        tol = tol + d_{tol}

        # Or is tolerance too high?
    else:
        tol = max($[0, tol - 2d_{tol}]$)
        $d_{tol} = \frac{2d_{tol}}{tol_{iters} - iter}$
        $tol = tol + d_{tol}$
return ξ_{best}

The PolyDiffPoint function aims to compute derivatives looking at a single point. It discards the data close to the edges of the grid since the polynomial derivative only works well when we do this.

```
def PolyDiffPoint(u, x, deg=3, diff=1, index=None):

"""
u = values of some function
x = x-coordinates where values are known
deg = degree of polynomial to use
diff = maximum order derivative we want
"""

n = len(x)
if index == None: index = int((n - 1) / 2)

# Fit to a Chebyshev polynomial
# better conditioned than normal polynomials
poly = np.polynomial.chebyshev.Chebyshev.fit(x, u, deg)

# Take derivatives
derivatives = []
for d in range(1, diff + 1):
   derivatives.append(poly.deriv(m=d)(x[index]))

return derivatives
```

The build_theta function builds a matrix with columns representing polynomials up to degree P of all variables. It is used when we perform subsampling and take all the derivatives point by point, or if there is an extra input to put in.

```
def build_Theta(data, derivatives, derivatives_description, P, data_description=None):
"""
input:
data: column 0 is U, and columns 1:end are Q
derivatives: derivatives of U and maybe Q, should start with a column of ones
derivatives_description: description of what derivatives have been passed in
P: max power of polynomial function of U to be included in Theta
returns:
Theta = Theta(U,Q)
descr = description of what all the columns in Theta are
```

```
"""

n, d = data.shape
m, d2 = derivatives.shape
if n != m: raise Exception('dimension error')
if data_description is not None:
   if len(data_description) != d: raise Exception('data descrption error')

# Create a list of all polynomials in d variables up to degree P
rhs_functions = {}
f = lambda x, y: np.prod(np.power(list(x), list(y)))
powers = []
for p in range(1, P + 1):
   size = d + p - 1
   for indices in itertools.combinations(range(size), d - 1):
      starts = [0] + [index + 1 for index in indices]
      stops = indices + (size,)
      powers.append(tuple(map(operator.sub, stops, starts)))
for power in powers: rhs_functions[power] = [lambda x, y=power: f(x, y), power]

# First column of Theta is just ones.
Theta = np.ones((n, 1), dtype=np.complex64)
descr = ['']

# Add the derivatives onto Theta
for D in range(1, derivatives.shape[1]):
   Theta = np.hstack([Theta, derivatives[:, D].reshape(n, 1)])
   descr.append(derivatives_description[D])

# Add on derivatives times polynomials
for D in range(derivatives.shape[1]):
   for k in rhs_functions.keys():
      func = rhs_functions[k][0]
      new_column = np.zeros((n, 1), dtype=np.complex64)
      for i in range(n):
         new_column[i] = func(data[i, :]) * derivatives[i, D]
      Theta = np.hstack([Theta, new_column])
      if data_description is None:
         descr.append(str(rhs_functions[k][1]) + derivatives_description[D])
```

```
        else:
            function_description = ''
            for j in range(d):
                if rhs_functions[k][1][j] != 0:
                    if rhs_functions[k][1][j] == 1:
                        function_description = function_description + data_description[j]
                    else:
                        function_description = function_description + data_description[j] +
                            '^' + str(
                        rhs_functions[k][1][j])
            descr.append(function_description + derivatives_description[D])

    return Theta, descr
```

Finally, the TrainSTRidge function trains a predictor using the STRidge algorithm (Algorithm 1). It runs over different values of tolerance and trains predictors on a training set, then evaluates them using a loss function.

```
def TrainSTRidge(R, Ut, lam, d_tol, maxit=25, STR_iters=10, l0_penalty=None,
    normalize=2, split=0.8,
print_best_tol=False):

# Split data into 80% training and 20% test, then search for the best tolerance.
np.random.seed(0) # for consistancy
n, _ = R.shape
train = np.random.choice(n, int(n * split), replace=False)
test = [i for i in np.arange(n) if i not in train]
TrainR = R[train, :]
TestR = R[test, :]
TrainY = Ut[train, :]
TestY = Ut[test, :]
D = TrainR.shape[1]

# Set up the initial tolerance and l0 penalty
d_tol = float(d_tol)
tol = d_tol
if l0_penalty == None: l0_penalty = 0.001 * np.linalg.cond(R)

# Get the standard least squares estimator
w = np.zeros((D, 1))
```

```
w_best = np.linalg.lstsq(TrainR, TrainY)[0]
err_best = np.linalg.norm(TestY - TestR.dot(w_best), 2) + l0_penalty *
    np.count_nonzero(w_best)
tol_best = 0

# Now increase tolerance until test performance decreases
for iter in range(maxit):

   # Get a set of coefficients and error
   w = STRidge(R, Ut, lam, STR_iters, tol, normalize=normalize)
   err = np.linalg.norm(TestY - TestR.dot(w), 2) + l0_penalty * np.count_nonzero(w)

   # Has the accuracy improved?
   if err <= err_best:
      err_best = err
      w_best = w
      tol_best = tol
      tol = tol + d_tol

   else:
      tol = max([0, tol - 2 * d_tol])
      d_tol = 2 * d_tol / (maxit - iter)
      tol = tol + d_tol

if print_best_tol: print
"Optimal tolerance:", tol_best

return w_best
```