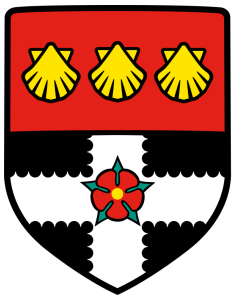
University of Reading

Department of Mathematics and Statistics



MODEL RECONSTRUCTION FOR DYNAMICAL SYSTEMS

Abdulrafiu Babatunde Odunuga

A thesis submitted for the degree of Doctor of Philosophy

January 2022
Revision Date: April 2023

Abstract

In numerical weather prediction (NWP), determining states and reconstructing model parameters or the underlying structural functions of dynamical models are essential and crucial components of the modelling and simulation process. In addition, it is gaining prominence in emerging application areas. In light of this, we examine the estimate of an unknown model M or the F function that determines the rate of change of x, which represents the dynamical system model of the type $\dot{x} = F(x)$ using a high-dimensional nonlinear approach based on Ansatz functions such as Polynomial or Gaussian functions.

The focus of this thesis is to propose specific iterative learning methods developed for estimating a dynamical system of interest using data assimilation (DA) techniques, as opposed to the traditional approaches typically used for parameter estimation, reduced order model approximation and the current approaches of machine learning (ML) and artificial intelligence (AI) techniques in general.

This is initially evaluated using two models of dynamical systems with increasing complexity: Lorenz '63 and Lorenz '96. Then, we examine the reconstruction strategy based on a variety of basis functions, including Polynomial and Radial Basis Functions (RBF). As generic application issues, we examine the reconstruction of the dynamics of Lorenz '63 with implicitly applied RBF under the assumption that the L^2 metric in coefficient space corresponds to a Gaussian prior in coefficient space. In addition, we employ the Amari Neural Field model for kernel reconstruction as a simulation test case for brain neural activity.

Using the Lorenz '96 model, we examine a Taylor series technique to express the forcing function $F(x)$ with regard to the state variables. This was utilised for the rebuilding of models via ensemble data assimilation. Using the variational data assimilation method and the Kalman filter technique, our

primary objective is to study a general method for resolving this problem with little or some specific understanding of the underlying dynamical system.

The models are then supplemented with a reaction-diffusion system. We demonstrate that learning a reaction-diffusion model's fundamental partial differential equation is doable and produces good results when the learnt model is utilised as a propagator.

Thus, we notice that the general iterative model reconstruction is competitive for the specific inverse issue under study for a broad range of initial conditions. Included are numerical examples demonstrating the practicability of the method.

Dedication

To my late dad and mum (Alhaji & Alhaja S. A Odunuga) for their love, care, and inspiration from the beginning to this day. I pray that the Almighty Allah grant them Aljanat Firdaus (Aameen).

...This is a promise kept.

Declaration

I declare that this thesis is my original authorial work. All sources, references, and pieces of literature used are correctly cited and listed in complete reference.

A B Odunuga

Acknowledgements

My reserved and heartfelt appreciation goes to Prof. Dr. Roland Potthast for his painstaking efforts, guidance, and supervision throughout this PhD. I am immensely grateful for his extensive feedback and encouragement during challenging times. My interactions with him in the last few years have been enriching. I have been motivated to continue seeking further knowledge because of his unwavering interest in science, which has profoundly impacted me as I advance in my research journey.

I also want to show my appreciation to Prof Doug Saddy, whose analogy of a 'learner to expert driver' interpretation of the challenges we seek to address from the first time I met him stuck with me throughout this research. I am deeply grateful for your contribution and helpful comments.

My appreciation also goes to Prof Amos Lawless, Dr Fazil Baksh, and Dr Danica Greetham. I have benefited greatly from your comments during and after the committee meetings, not forgetting Prof. Dan Crisan (external supervisor). The feedback has proved valuable and helped reshape my views on what I need to improve to fulfil the tasks ahead of me. Once again, I thank you all for your support and patience during these periods.

I have had the privilege of meeting people and making new friends, all of whom I cannot name here, including past and present PhD colleagues, staff, and all those who have been helpful in one way or another throughout my journey here at the University of Reading.

Lastly, to my incredibly supportive family, Ajoke, Adisa, and Ajike, for their love, understanding, and steadfast support all the time.

Contents

List of Figures

Chapter 1

Introduction

The primary objective of this study is to create *novel learning methods* based on *data assimilation* (DA) techniques for estimating a dynamical system of interest. The development of machine learning approaches for models with a range of neural networks is increasing; we shall examine some of these techniques below. The estimations of the *states* are often from observations using data assimilation techniques. In this instance, these approaches were used to recreate the *model*, not simply its states.

We again study dynamical system representations to construct innovative data assimilation strategies to model learning. In the midst of accomplishing this, we investigate:

a) the **Lorenz '63** model, which is prevalent in low-dimensional chaotic dynamical model.

b) the **Lorenz '96** model, which is a potentially higher-dimensional chaotic dynamical model.

c) the **Amari Neural Field Model**, which is a high-dimensional fundamental model from neuroscience; and

d) the **reaction-diffusion system model** as a full-grown weather model used by the reaction-diffusion system and COSMO consortia[1], where we restrict our attention to some specific sets of atmospheric variables, in particular, temperature over central Europe.

[1] http://cosmo-model.org

The algorithmic approach we investigate here starts with employing well-known data assimilation techniques to estimate models dynamically. To our knowledge, this has yet to be previously applied to model learning or explored in the field of neuroscience research in this way.

(VAR) As a first generic approach, we develop and investigate **variational data assimilation** to estimate or reconstruct a model. Such approaches have already been used for *adaptive or variational bias correction* (VAR) in satellite data assimilation [34], [125]. Here, we will develop the ideas into a more generic tool that can be applied in different setups. This will be worked out in detail in Section 3.1.

(KF) An immensely popular data assimilation method is the **Kalman filter** (KF). We will formulate and investigate the Kalman filter for learning the model in Section 3.2. It can be viewed as an extension to the above variational approach, where in this case, the *background uncertainty covariance matrix* B is updated in each assimilation step.

The dynamical system described in this research is not explicitly dependent on time (t) for emphasis and clarity. In most instances in this study, its dependence is on the fields, and it is an implicit dependence, and the coefficients are not time dependent. However, the forcing term F is dependent on time for its prediction.

Dynamical systems are usually written in the form of primitive equations

$$\dot{x} = F(x) \tag{1.1}$$

with a forcing term $F : \mathbb{R}^n \to \mathbb{R}^n$ mapping the state space $\mathbb{R}^n$ of dimension $n \in \mathbb{N}$ of the system into itself. In the case of the Amari model [5], the forcing term has a particular form

$$\tau \frac{du}{dt}(x,t) = -u(z,t) + \int_D w(z,y) f(u(y,t))\, dy \tag{1.2}$$

such that for the state u which is a function in $L^2(D) \times C^1([0,T])$ with some domain $D \in \mathbb{R}^d$, d=2,3 and a time interval [0,T],

$$F(u)(z,t) = -u(z,t) + \int_D w(z,y) f(u(y,t))\, dy, \tag{1.3}$$

where the function $w(z,y)$ for $z, y \in D$ with some domain D in space is called the *neural kernel*. In that case, the forcing term estimation can be reformulated as a kernel reconstruction or kernel learning task.

- We will describe the use of the learning methods VAR and KF to **Forcing Term Estimation** in Section 3.3. It also applies to the reconstruction of dynamics of the Lorenz models as particular cases. We will show the results in Section 4.1 and Section 4.2. The application to some subdynamics of the reaction-diffusion system weather model is conducted in Section 4.4.
- We will investigate the application of VAR and KF to the **Kernel Learning Problem** in Section 3.4, compared to the traditional kernel reconstruction and numerical results shown in Section 4.3.

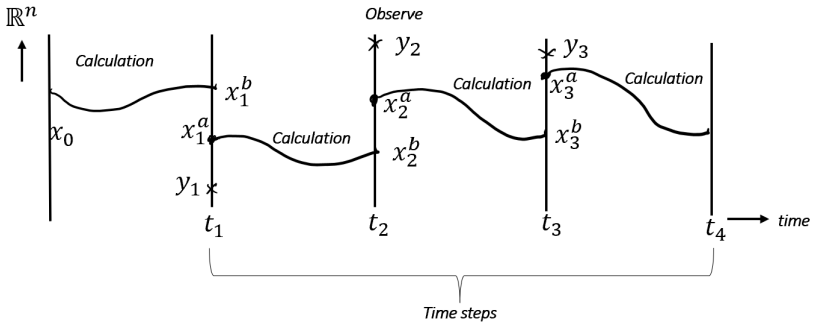
Data assimilation combines data from several sources, such as observations and numerical models, to estimate a complex system's state. Specifically, it is commonly used to estimate the state of a dynamical system, which is a system that changes over time based on a set of rules or equations.

By merging information from observations and a mathematical model, estimating the state of a dynamical system at a particular moment can be done by applying the data assimilation technique. Given the available data, it requires solving an optimisation problem to get the most accurate approximation of the state. The estimated state can then predict the system's future behaviour.

Several domains, including meteorology (especially for weather forecasts), oceanography, and geophysics, utilise data assimilation to enhance weather forecasting and climate modelling. It is also utilised in engineering and finance to evaluate the current state of complicated systems and anticipate their future behaviour. Data assimilation is also the method of combining observations and models to estimate the state of a dynamic system.

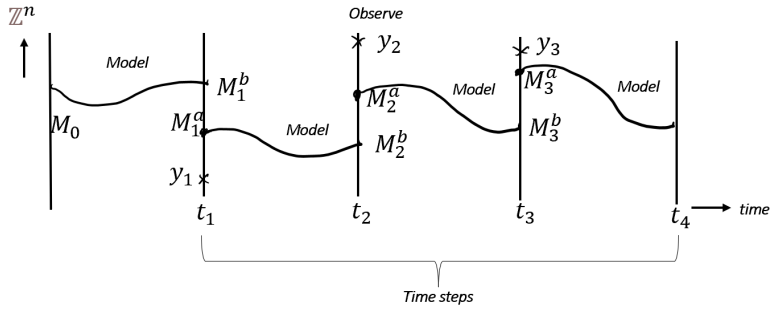
A dynamical system is a mathematical model that represents the time-dependent behaviour of a system [106]. It consists of a collection of equations that regulate the deterministic or stochastic behaviour of the system. Usually, the future state of a deterministic dynamical system is defined or determined by its current state and the governing equations. Random variables impact the future state of a stochastic dynamical system.

A diagrammatic representation of the technique in its simplest form is presented in Figure 1.1 for the classical data assimilation approach, and Figure 1.2 depicts the simplest diagrammatic representation of the model learning ansatz.



(a)

Figure 1.1: The figure shows a one-dimension state-space illustration of the trajectories of the data assimilation approach used in this thesis at each assimilation step. We consider the state-space $X = \mathbb{R}^n$, i.e., our states $x \in \mathbb{R}^n$ are real numbers. At time t_1, we have calculated $x_1^b \in \mathbb{R}^1$; Similarly, we measure $y_1 \in \mathbb{R}^1$. We then estimate x_1 at time t_1, the result is the analysis at $x_1^a \in \mathbb{R}^1$. The process is repeated at each successive time step t_2, t_3, t_4,...,t_n.



(b)

Figure 1.2: Depicts the **model learning** strategy included in a classical data assimilation approach at each assimilation phase employed in this thesis. The M_0, M_1^b, M_2^b,...,M_n^b (model error background at every iteration stage), while M_1^a, M_2^a,...,M_n^a represents the model analysis at each propagation stage. The variational or Kalman filter method updates the **Model** at each time step t_k in the model-space $M \in \mathbb{Z}^n$.

By learning models in the *neuroscience* framework, one of this thesis's long-term applications is to find new mathematical and computational models that can support and lead neuroscience research concerned with collective neural activities and the large-scale complex processes targeted by non-invasive brain imaging methods. The methods developed here offer a path towards new predictive analytical approaches and could offer some solutions to the long-standing contentious debates about using predictive models to obtain insights from large-scale neural dynamics within the neuroscience research communities and other fields of study.

We will start our presentation with an *overview section* (1.1) that examines earlier and current research studies in this field. In addition to the description of the mathematical approaches formulated, it lends credence in support of the argument to establish a vital *requirement* for a model reconstruction approach in neuroscience and other application areas. It emphasizes the use of data assimilation as a method of choice for this study, provides an overview of related approaches such as *reduced order modelling*, reviewed surveys on modern learning approaches (machine learning and artificial intelligence) used in data science and completes it with a brief introduction to *parameter estimation* techniques.

1.1 An Overview

The study of dynamical systems has grown in popularity over the last century, according to [16]. Its applications span various subjects and sub-disciplines, including mechanics, biomedical engineering, medical physics, biology, and other non-technical topics (like economics and social sciences). Over the previous three decades, the rise of big data has presented significant obstacles to data sourcing, handled, analysed, and efficient use for decision-making, diagnostics, and forecasting.

Dynamical systems are usually classified into two phases: the phase state or *state space*, which contains states that develop over a particular period and usually contain the full description of the real-life occurrence we are attempting to simulate. The other is *system dynamics*, which describes the rules guiding the development of our dynamic system of interest with time, with time evolution being either discrete or continuous [50]. This study's modelling technique considers both discrete and continuous dynamical systems.

One of the primary issues for many application areas in the modelling and simulation of dynamical systems is how to grasp the changes that have occurred through time from real-world phenomena in estimating a system's future state. In their descriptions of Gaussian Process Dynamical Models (GPDM),[52], [83] and [119] stressed the difficulties of capturing the nonlinearities of the data without overfitting the model. This frequently leads to the additional challenge of determining an appropriate mathematical modelling approach or an extraction algorithm that determines the states and reconstructs the model parameters of our dynamical system, especially when we have little prior knowledge of the system without distorting the inherent characteristics or properties of the system.

This could also require providing a part or the complete description of a system with a detailed understanding of the *hidden or underlying structural functions* of the dynamical model of interest. *On the other hand, the goal of this research is to examine and create a generic technique that may be utilised to solve these issues with little or specialised knowledge of the underlying dynamical system.*

In doing this, we have developed two mathematical approaches below describing the steps we have taken to deal with these cases.

i. **(POL)** We describe the model dynamics by a *polynomial approximation*, which may be considered a Taylor function approach. The coefficients of a polynomial basis function representation parametrise the unknown model.

ii. **(RBF)** We investigate a *radial basis function approximation* with nodes and coefficients used to estimate the forcing term or the integrated model application $M(x)$.

The RBF approach is similar to the reconstruction of a flow field along the known model trajectories, where the direction or movement of the fluid can be observed at specific locations within a state space as time passes. This is the reference approach to be investigated for this study.

In both cases, we describe a *generic variational approach* based on the basic approach of the *Kalman filter* to estimate the model coefficients. It follows the idea of *variational data assimilation*, with updates of the covariance matrix B_c of the coefficient,s which parametrise the model dynamics by the standard Kalman update formula.

$$B^{(a)} = (I - KH)B^{(b)} \tag{1.4}$$

Where $K = BH^T(R + HBH^T)^{-1}$ is the Kalman matrix for model coefficient updates, H is the model's observation operator, and $B = B^{(b)}$ is the model coefficient uncertainty covariance matrix before the current update step. A detailed explanation of this method is described in Section 3.2.

First, we describe a generic but naïve approach to **model reconstruction by variational data assimilation** in model space. However, this approach requires a model covariance matrix in this case. In the particular case where the covariance is Gaussian, we see that the model is represented as a sum of Gaussians around the current analysis states

$$x_k^{(a)}, \quad k = 1, 2, 3, ... \tag{1.5}$$

One major drawback of the approach described above is that, over some time, it accumulates a considerable sum of Gaussians that will need to be stored in each time step for cases where the dynamical system is highly nonlinear and high-dimensional, i.e., there is the need to store a growing number of the entire model states, and this naive application of 3D-VAR to model reconstruction may be computationally inefficient.

To avoid adding large sums of Gaussians, we need to keep the dimension of the model approximation space limited. *The essential task for a particular system is to design appropriate ansatz functions which can be used to approximate the real-world model under consideration.* As a result, we study two particular approaches for this thesis, the *polynomial* or *Taylor* approach and the *radial basis function* (RBF) *approach* to help address these challenges.

1.2 Model Reconstruction in Neuroscience

In this section, it is essential to reiterate that, to our knowledge, there is no known model reconstruction strategy in the area of neuroscience or elsewhere where the approaches utilised in this thesis have been previously examined or implemented in this manner. Thus, this is a groundbreaking study for future research in this field.

Model reconstruction is a crucial method in neuroscience that tries to comprehend the brain's complicated processes by developing computational models

that replicate the behaviour of neural networks. The data from neurophysiological tests, such as electrophysiological recordings, imaging methods, and behavioural research, are used to develop these models. A good example was provided by these researchers [75] where they reconstructed a large-scale model of the neocortical microcircuitry using anatomical and physiological data from multiple sources.

The primary objective of model reconstruction is to increase knowledge of the processes underlying brain function and behaviour. Researchers may examine the impact of various changes on brain activity and behaviour by creating computer models that represent the critical characteristics of neural systems.

An essential advantage of model reconstruction is that it enables researchers to test hypotheses and make experimentally testable brain function and behaviour predictions. For instance, researchers can use models to anticipate how a specific medicine or intervention would impact brain activity and behaviour and verify these predictions using animal models or people.

The human brain is an enormously complex and diverse organ, with more than 100 billion nerves and neurons interacting with trillions of synapses. Hence, it is challenging to generalise from a confined examination or area of the brain to the rest of the brain [83], [87]. This analysis can only show or provide insights into the brain region from where the data extraction occurred. As a result, it is not always feasible to extract as much data as we may need from the brain using an invasive extraction technique, especially when combined with the challenge of identifying dependencies from a system that correlate to the process that would have formed such a system.

In recent times, the vast amount of the data [37], [69] generated within neuroinformatics has significantly been influenced by advances in information technology (IT) and decades of expanding neuroscientific research at all levels [66], [97]. [111] investigated constructing a neural network model for computational neuroscience consisting of constituent neural units. This has also led to the proliferation of multiple databases where a large amount of neuroscience "big data" can be stored to enhance more research projects and develop strategies for future systematic collection while analysing these data remains a crucial challenge.

The authors [31], [81] also emphasised the need to deal with the staggering amount of data sets produced in a field of scientific investigation such as

medical imaging, including functional magnetic resonance imaging (fMRI) or computerised tomography (CT) brain scans, Neuroimaging, and other sources of data generated from the brain in the neurosciences. Lichtman et al. [69] acknowledged the need for a breakthrough algorithm and new computational framework to address the inherent difficulty in mapping synaptic network connections within the brain and the sheer complexity of the amount of big data generated due to the interconnectedness of the structure of the nervous system as a result of the emergence and advancement of connectomics.

Two main big data projects are at the frontiers of these recent innovations. The *Brain Initiative* and the *Open Connectome Project*, while the former aims to develop advanced technological tools that can be used to deepen our understanding of the inner workings of the neural activity in the brain [3], and the latter aims to produce a wiring map of the brain to understand connections between neurons better [37].There are other known research projects in the field of Neuroscience, [91] explored the use of a mathematical model to determine the impact of conductances by two chemical components in the nervous system by reconstructing an electrical signal conducted along axons (or muscle fibres) by which information is conveyed from one place to another known as the action potential. An equally notable project is The *Brainbox Initiative*, designed for non-invasive simulations and imaging of the brain [17].

Similarly, there are other comparable projects with this research; notable amongst them is the construction of predictive neuron models through large-scale data assimilation of electrophysiological data [82], but the direction of this thesis follows a slightly different layer of focus dealing with simpler systems. Again, [82] investigated the characterization of diverse neuronal dynamics in sensory circuits by estimating the dynamical properties of neuron networks. Furthermore, [20] and [73] used very well-known statistical methods to make different types of statistical inferences using the Bayesian Model to investigate some established hypotheses within the neuroscience community on the dependences of the temporal structure of sequences of events as an essential part of the decision-making process in the human brain.

Another equally relevant study is that provided by [74], he presented a relatively simple mathematical approach that fits our perspective, i.e., $\dot{x} = f(x,t)$ with observations $y = h(x)$. He estimated the model error using a dynamical elastical net with L^2 and L^1 regularization on some term w, which is added to the dynamics in the form $\dot{x} = f(x,t) + w(x)$. The key point is learning model features from the data, which could be described as a weak

constraint 4D-VAR and [84] examined the delay coordinates map, which maps the original state space into a reconstruction state space and the local inverse of this map.

There are other well-known technical challenging areas of interest within neuroscience, some of which includes *understanding the connections or mapping* of the various cells that go on in the brain during blood flows, provision of valuable insights on the neural activities within the brain, assimilating and evaluating the critical stages of learning and skills acquisition processes and other medical or diagnostic interventions beneficial for patients with debilitating health conditions that require regular monitoring in order to manage them. There is a big analytical challenge in this field, and current research efforts are yet to address this seeming problem of *integrating or synthesizing all the available data* into a more coherent and cohesive format for better analysis and interpretation.

These are some of the fundamental driving forces behind this research which is to develop a mathematical model that can be used to characterize the data, help detect unknown signal sources, connect and track network activities from the brain and provide adequate knowledge of the model and make the prediction task more effective and efficient.

1.3 Parameter Estimation in Data Assimilation

Parameter estimation, model order reduction, and machine learning are three key topics in computational science and engineering. This literature study will investigate how these domains are interrelated and how they might be utilised together to address complicated challenges. This section examines several works of literature on parameter estimate strategies in data assimilation. On this issue, historical and contemporary literature outlines the numerous methods and approaches of parameter estimation, some closely resembling the methods utilised in this work. Parameter estimation is concerned with estimating the values of model parameters [80].

The process of guessing the values of model parameters based on available data is known as parameter estimation. In statistical modelling, model parameters are estimated values based on observable data. The most frequent approach for parameter estimation is maximum likelihood estimation (MLE),

which requires selecting the set of parameter values that maximises the chance of observing the data supplied in the model. Based on the data, bayesian estimation requires assigning prior distributions to the parameters and updating them using Bayes' theorem.

The authors [10] and [11] provided a comprehensive and detailed overview of the theory and practice of parameter estimation techniques, including Bayesian methods, variational methods, ensemble Kalman filter, particle filters, maximum likelihood estimation, and least squares methods, and their application to a variety of scientific fields, such as geophysics, weather forecasting, ecology, and economics.

[76] outlined an equation learning strategy that aims to draw conclusions from observation data and aid in filtering out the noise that might influence the structure and values of the parameters learnt via uncertainty quantifications. In their research on linking a groundwater flow model with a polluted transport model, the authors [46] also introduced a dual-state ensemble Kalman filter (EnKF) technique that allows them to estimate the states of flow and pollutants simultaneously, provides a sequential updating scheme between models, simplifies the implementation of the filtering system, and yields more stable and accurate solutions than the standard approach.

The preponderance of the publications of these authors [105], [27, 104], [65] emphasise the primacy of data assimilation for state estimation and its utility in estimating uncertainty in model parameters inside the model state. In their numerous research works, they have combined these data assimilation approaches to estimate state parameters and estimation precision.

Parameter identifiability [94] is an additional important factor to consider when determining how well given model parameters are adequately described by the quality of available data. Parameter estimation and identifiability are fundamental concepts in statistical modelling and data analysis. Despite their relationship, these phrases refer to different aspects of the modelling process.

Identifiability of parameters refers to the extent to which the values of model parameters can be uniquely derived from the data [72] without equifinality presenting obstacles. Identifiable parameters are those for which a singular set of parameter values can adequately explain the observed data. Even with an infinite number of data, it is impossible to accurately predict the value of an unknown parameter [123] and [7], and estimating unknown parameters from data relies on statistical models.

Nevertheless, it is possible that not all model parameters can be identified, posing difficulties for parameter estimation and model interpretation. Non-identifiability occurs when different parameter value combinations generate similar model outputs, making it impossible to determine the parameter values using the available data uniquely [9]. [35] presented two statistical computations to compare the ability of some parameters to be uniquely estimated before and after calibrations based on the changes in their datasets.

In essence, parameter identifiability investigates whether parameters can be precisely estimated from the data. Extensive research has been conducted on parameter identifiability in systems biology, mathematical modelling, and statistics. Numerous techniques and methods have been developed to evaluate and improve the identifiability of model parameters. [118], [45] and [51] are some examples including sensitivity analysis, identifiability analysis, experimental design, regularisation techniques, and Bayesian inference methods.

1.4 Reduced Order Model Literature

In most highly nonlinear and high-dimensional dynamical systems, for example, the brain system, it is often tricky and computationally challenging to use the entire brain model for simulations; this will require exceedingly high computing power to be able to execute, and as a result, we need to adopt the use of an equivalent or replica system on a small scale that can offer an approximate description of the processes of the system of interest in a reduced set up while also preserving the integrity of its internal structures in the process.

This research also acknowledges using a distinct technique called Model Order Reduction (MOR). It is a popular statistical and mathematical strategy for lowering the computational complexity of mathematical models that are frequently employed in numerical simulations. Reduced Order Models (ROMs) are miniature or reduced versions of high-dimensional and complicated models, as their name indicates. Whilst this technique is not utilised in this thesis, we foresee a future link or expansion of the methodology employed in this work to the MOR. In light of this, this part introduces the general framework utilised for the model order reduction technique compared to other methods, as well as some of the assumptions underlying their respective selections.

MOR techniques have a long-standing history with the success of subspace projection methods for solving large linear systems and matrix eigenvalue

problems. One of the most well-known approaches was proposed by Krylov in 1931 [99] for the explicit construction of the characteristic polynomial of a matrix which eigenvalues can then be calculated as the roots of that polynomial. Amsallem et al. [6] proposed a model reduction method that approximates solutions of global basis vectors into lower-dimensional subspace generated by the most appropriate local basis vectors. Concisely, MOR methods search for a correlated set of governance equations in a subspace significantly lower in dimension than the central system of interest.

Similarly, [41] Festjens et al. described using ROMs as essential catalysts for improving the computational efficiency of simulations in processes with a high value of the degrees of freedom (DoF), especially in an optimization framework. [33] Corigliano et al. equally proposed a new strategy formulated as a MOR technique; the basic idea employs an everyday use of Domain Decomposition (DD) with a popular version of the Proper Orthogonal Decomposition techniques, which are usually used to extract a reduced basis from a set of snapshots in nonlinear problems. Other methods include the use of t-Distributed Stochastic Neighbour Embedding (t-SNE) by Cieslak et al. [28] and [114], Uniform Manifold Approximation and Projection (UMAP) [78] and Linear Discriminant Analysis (LDA) [112].

Therefore, MOR could also be described as a pseudo or surrogate modelling method. It is a widely used technique for reducing large-scale order to low-dimensional order models, and its use in this format spans several different science and engineering domains. They are also called metamodels and emulators due to their use as decoys to reduce the computational burden in many application areas [47]. Koo et al. [64] proposed a proper orthogonality decomposition (POD) with a robust point interpolation method (RPIM) approach for predicting the temperature and sensor placement for a cylindrical steam reformer, a similar approach was used by [117] as a parameter MOR (pMOR) method. Another MOR technique used in [124] reducing the sizeable computational complexity based on tensor decomposition and matrix product was the use of an equivalent transformation of the main quadratic-bilinear (QB) systems from their non-linear input-output systems.

In summary, reduced-order models are good enablers of rapid prediction, inversion, and design and help in quantifying uncertainties of large-scale scientific and engineering systems [120]. However, using ROMs also comes with challenges related to finding or deciding which ROMs are the most efficient and suitable solutions for the system of interest. In this study, we have compared

the models' errors and their convergence by investigating the error evolution of the reconstructed model with the original dynamics.

There are several works of literature with examples of MOR applications, one of which includes its application in Cosmology [86], where it was used to infer or reconstruct the underlying model from observational data obtained from gravitational waves using polynomial regression and Gaussian Process without depending on too many highly intricate model assumptions. Another approach is the temperature prediction and sensor placement in cylindrical steam reformer [64], an analytical framework for control systems [14], and for simulations in engineering sciences and modern ICT technologies [42] amongst others. The authors [26] investigated a model which combines the Reduced Order Model (ROM) with Data Assimilation (DA) to enhance the precision of simulations of computational fluid dynamics.

In this thesis, the focus is on learning models, not on the construction of appropriate MOR approaches. Our approach could be applied to any MOR ansatz. We have based our technique on the use of a 3D-VAR data assimilation approach or on the localized ensemble transform Kalman filter (LETKF) to model reconstruction developing new techniques for learning the model.

1.5 Survey on Modern Learning Techniques

The goal of this thesis is to employ data assimilation techniques for model learning or model reconstruction, respectively. Of course, today there is a rapidly expanding field of *machine learning*, which usually determines neural network connectivity to adapt the neural network as a model to given sets of observations. Our approach is a different learning method, and we apply it to a wide range of different approximate models.

To be able to discuss similarities and differences of our data assimilation approach to machine learning, in this section, we present a literature survey of the recent and modern modelling techniques used in machine learning (ML), deep learning (DL), which is a branch of artificial intelligence (AI), and computer science and other scientific computing community. In practice, the application of ML (both supervised and unsupervised) has evolved over the last decade with the proliferation of big data and increasing computing power. In addition, data assimilation and machine learning models [19] are now being combined to simulate and train ML-based parametrization using data with noisy and sparse

observations. The link of this section with the thesis is to provide a narrative and an exploration of some pieces of literature with recent applications of advanced machine learning and other analytical methods to provide solutions to issues in new or existing fields of research areas, which this thesis as well aims to achieve in the field of neuroscience as an example.

The urge mainly drives them for the enormous demands and consumption of new insights by consumers, governments, businesses (including social media platforms -the likes of Google, Facebook, YouTube, Twitter, Amazon, Netflix, Microsoft, etc.), and more recently, its use on a large scale for providing insightful analysis on the Covid-19 pandemic responses across multiple disciplines, and other emerging artificial intelligence (AI) technologies and platforms to name a few [1, 2, 44, 54, 79, 95, 107].

Another relevant learning method is reinforcement learning (RL). In its simplest form, it could be described as a *give-and-take* learning technique due to the interactive or sequential approach of the learning method. The authors in [116], [62], [13], have all described the concept of RL from the perspective of their different application areas ranging from engineering, neuroscience, and psychology, etc. However, they are all unanimous in their conclusions of the adaptive nature of the learning method to its environment with limited knowledge of such and uses the limited feedback provided to improve the quality of the decisions. Supervised ML techniques are used to develop predictive models based on the input and output data.

In contrast, unsupervised techniques on the other hand, can be used to group and interpret data based only on input data. The formulated mathematical approaches used in this research fall into both techniques. As a field of research, ML primarily focuses on the theory, performance, and properties of learning systems and the algorithms used [92].

Several recent research studies focus on the evolution and use of machine learning that transcends many disciplines. This thesis could be developed for further research into some of the new and emerging trends in the use of ML and AI. Machine Learning can be described as using data and algorithms to copy or emulate how humans learn [36], [77]. The information known by the algorithms is then adapted by improving the performance of the knowledge gained as additional samples or real-world data becomes more available. These researchers presented a modern approach [23] to integrating DA and ML models to increase prediction reliability.

Recent advances in Information Technology, big data processing, and open-source software have brought considerably new insights into how machine learning algorithms and artificial intelligence are used in shaping our understanding of the world we live in today. Some of the novel machine learning techniques include classification models, clustering and retrieval, kernel-based learning, dimensionality reduction methods, recommender systems, and deep learning (Mostly implemented using neural network architecture), amongst others, which are now widely applied to provide new valuable insights including visualisations into real-world phenomenon using data (including images, texts, and sounds) derived from such systems.

Current ML and AI applications include advanced learning algorithms to aid climate change research and preparedness [57]. ML algorithms have been successfully applied to high-dimensional input data in numerous fields for playing games, web searches, fraud detection, the spam filtering in emails, credit score ratings, and many more. [100]. Other emerging and future uses of ML applications are expected in the face and voice recognition, self-driving vehicles, driver assistance systems, space technology, and many more.

However, the use of ML algorithms and their application has its challenges. These algorithms exist as black boxes due to the lack of unique laws governing the understanding, knowledge, and, sometimes, the interpretation of results from their use. They can be attributed to the multidisciplinary nature of ML, which spans computer science, statistics, mathematics, engineering, cognitive science, and various other scientific and socioeconomic fields. It is also important to note that other learning techniques like regression (Linear and Logistics), Support Vector Machines (SVM), Random Forest, Classification, Bayesian Learning, and Decision-Tree modelling are all algorithms embedded in ML techniques.

In conclusion, parameter estimation, model order reduction, parameter identifiability and machine learning are all interrelated topics with strong links to data assimilation. They may be employed in tandem to tackle difficult issues. The combination of these strategies can result in more accurate and efficient models, as well as a substantial decrease in processing costs. Beyond data assimilation, there are connections between these domains that can lead to additional advancements in parameter estimation and model order reduction in future research.

1.6 Data Assimilation Techniques

Data Assimilation is a well-known mathematical modelling technique used in numerical weather prediction (NWP) models and many other application areas. The flexibility of data assimilation as a technique makes it adaptable for application in various fields apart from in numerical weather predictions (NWP), where it was first used for planetary weather analysis and more recently in the field of biological cells [52] and many other emerging application areas. However, weather forecasting is still predominantly the key driver of many recent theoretical and practical applications of the algorithm due to the deluge of available data and the short turnaround time used in dealing with such systems [85].

The authors in [87], [30], [81], [108] described on several occasions in their numerous publications that the basic idea of the data assimilation technique is to enable forecasts by *combining information from prior knowledge with new observed information* from the system to obtain the best description of the system of interest [85], [52].

Data Assimilation can be seen as a recursive *Bayesian inference technique*. Data assimilation combines measurements with models as its basic idea using interpolation and filtering methods. The use of data assimilation in estimating the true state of dynamical systems has been well documented in several research works of literature and journals and one such by [98] was an attempt to use data assimilation in hydrological models to improve model state and estimate streamflow.

The traditional data assimilation approach aims to determine a state x which is close to the background state x^b containing knowledge from the past to the observations y which also contains knowledge about the current state of the dynamical system of study. Apart from its daily use in numerical weather predictions (NWP), it is now also widely applied in many fields. Here, we briefly describe the main approach of data assimilation with much emphasis on how it was applied to the model reconstruction approach used in this thesis within the model space $M \in Z$.

There are two basic steps in data assimilation applications:

Definition 1.6.1. **(Propagation Step)** *Mapping* $x_{k-1}^{(a)} \mapsto x_k^{(b)} := M(x_k^{(a)})$, *with time index* $k = 0, 1, 2, 3, ...$

This step involves the application of a defined or known model M which maps a state $x_{k-1}^{(a)}$ at time t_{k-1} (previous time or period) into the current state $x_k^{(b)} = M(x_{k-1}^{(a)})$ at time t_k (current period).

DEFINITION 1.6.2. **(Estimation Step)** *Mapping* $(y_k, x_k^{(b)}) \to x_k^{(a)}$, *where* y_k *denote observations at time* t_k *and* $k = 0, 1, 2, 3, ...$

The estimation step uses the measurement y_k at the current time t_k. The *first guess* or *background* $x_k^{(b)}$ derived from the propagation step above to calculate an analysis $xk(a)$, which is the solution to the minimisation problem for some cost functions and can be used to calculate the difference between the observed and predicted values in a model, which will be described below.

The recurrence of these two phases is known as the data assimilation cycle or cycling. This technique is especially significant for variational data assimilation approaches in which the algorithms iteratively alter the model parameters to minimise the cost function until a satisfying solution is reached, as demonstrated in the following parts of the thesis.

The state space x, background state $x^{(b)}$ and the observations y are employed to define the cost function:

$$J(x) = \frac{1}{2}(x - x^{(b)})^T B^{-1}(x - x^{(b)}) + \frac{1}{2}(H(x) - y)^T R^{-1}(H(x) - y), \qquad (1.6)$$

where

x = State space, $x \in \mathbb{R}^n$, i.e., x is in the state space $\mathbb{R}^n$
x^b = Background state, $x^b \in \mathbb{R}^n$,
B = Background error covariance matrix, $B \in \mathbb{R}^{n \times n}$,
R = Observation error covariance matrix, $R \in \mathbb{R}^{m \times m}$,
H = Observation Operator,
y = Observation, $y \in \mathbb{R}^m$

If H is linear, we have $H \in \mathbb{R}^{m \times n}$ but in general,

$$H := \mathbb{R}^n \to \mathbb{R}^m, \quad x \mapsto H(x),$$

The minimizer of the functional (1.6) is called the analysis x^a. For linear H, the minimizer can be explicitly calculated, it is given by

$$x^{(a)} = x^b + BH^T(HBH^T + R)^{-1}(y - H(x^b)), \qquad (1.7)$$

where the so-called *analysis* is a state $x^a \in \mathbb{R}^n$. Using functional analytic notation, the cost functional can also be written in the form

$$J(x) = \|x - x^b\|^2_{B^{-1}} + \|y - Hx\|^2_{R^{-1}}. \tag{1.8}$$

In this case, and for H linear, the *normal equations* can be written as

$$x^a = x^b + H^*(I + HH^*)^{-1}(y - Hx^b), \tag{1.9}$$

where H^* denotes the *adjoint* operator in the space X with some scalar product $\langle \cdot, \cdot \rangle$ which defines the norm $|| \cdot ||$ by $||x||^2 = \langle x, x \rangle$, where

$$|| \cdot ||_A = \sqrt{(x^T A x)} \tag{1.10}$$

It is used to access the size or magnitude of the vector as shown in 1.8.

In the classical data assimilation approach, the equation in (1.9) above can be written as:

$$\begin{aligned} x^{(a)} &= x^{(b)} + \underbrace{BH^T(HBH^T + R)^{-1}}_{=:K}(y - Hx^{(b)}) \\ &= x^{(b)} + K(y - Hx^{(b)}), \end{aligned} \tag{1.11}$$

where $K = BH^T(HBH^T + R)^{-1}$ is known as the *Kalman gain matrix*. This matrix is a central tool for data assimilation, and it will be the main tool for all of our model reconstruction algorithms.

In investigating this technique for model reconstruction, we consider a dynamical system of the form $\dot{x} = F(x)$ by a high-dimensional nonlinear approach based on Gaussian basis function.

Therefore, we propose two generic approaches for reconstructing the model dynamics of a dynamical system by estimating the model given by the forcing term F using a radial basis function approximation to estimate the forcing term which is the trajectory or path of the dynamical system. The technique used in this approach is analogous to the reconstruction of a flow field, where the path or movement of the fluid can be observed at specific locations or points within a state space as time passes.

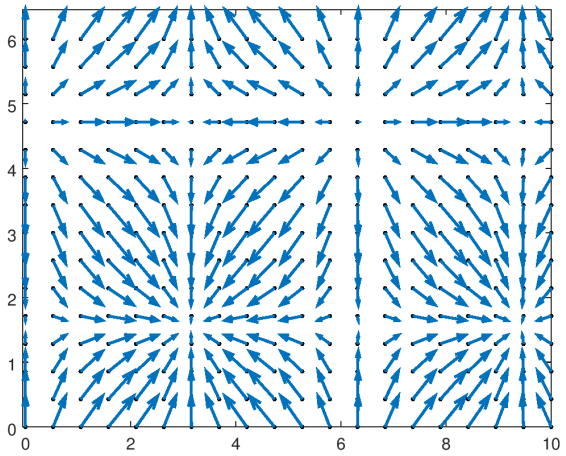


Figure 1.3: In this figure, we show a flow field description of a typical dynamical system trajectory estimation

1.7 The Approach of Parameter Estimation

This section highlights the role of parameter estimation as a key component used in the making of statistical inference. There are many different parameter estimation techniques usually called estimators, which are often applied to data to obtain an estimate. An estimator is a rule that is followed to estimate a parameter while the numerical value that is derived from a particular sample is known as the estimate. In this study, however, the definitions of the attributes or qualities of a good estimator are not covered.

Also in this section, we highlight under which side conditions should a *generic inverse approach* be undertaken, when is *parameter estimation* a good choice, and what *more generic model reconstruction method* which we will develop in the upcoming chapters is, and what are its advantages.

As a description, suppose we have variables X_1, X_2,...,X_n which are random samples drawn from a population with a parameter θ, then $\hat{\theta}$ is a statistic that can be used as an estimator of θ if

$$\hat{\theta} = \hat{\theta}(X_1, X_2, ..., X_n) \tag{1.12}$$

As a result, for an equation of the form $\dot{x} = F(x)$ described in this study, we assume that we have measurements of the dynamics at points in time

$t_1, t_2, ..., t_n$. Now, assume that the F is given as a system of polynomials in the variable of the state space, as shown in the example for the Lorenz '63 system where

$$\dot{x} = \sigma(y - x) \tag{1.13}$$

$$\dot{y} = x(\rho - z) - y \tag{1.14}$$

$$\dot{z} = xy - \beta z, \tag{1.15}$$

we obtain

$$F(x) = \begin{pmatrix} \sigma y - \sigma x \\ \rho x - xz - y \\ xy - \beta z \end{pmatrix}. \tag{1.16}$$

for a state space with variables $(x, y, z) \in \mathbb{R}^3$. The classical idea [52], [39] and [40, 68] is to use an *Ansatz* of the form

$$cF_i(x) = c_i + \sum_{j=1}^{3} a_{i,j} x_j + \sum_{j,\ell=1}^{3} b_{i,j,\ell} x_j x_\ell \tag{1.17}$$

for $i = 1, ..., 3$. where $(x, y, andz)$ are the system's state variables and $(\sigma, \rho, and\beta)$ are the parameters that govern the system's behaviour, and

1) σ(**sigma**): The Prandtl number quantifies the ratio between momentum and heat diffusivity. This parameter determines the rate at which fluid motion smooths out temperature differences.

2) ρ(**rho**): The Rayleigh number quantifies the proportion of buoyant forces to viscous forces. This parameter controls whether or not the system will experience convection.

3) β(**beta**): A parameter that affects the rate of heat transfer between the system's top and bottom. This parameter influences the pace at which the system's temperature gradient will equilibrate.

Therefore, the method to find F by estimating the coefficients in (1.17) is a parameter estimation problem, which now has much fewer degrees of freedom and all the standard data assimilation methods such as the Kalman filter, and parameter identifiability techniques in 1.3 can all be employed to solve this parameter estimation problem.

For the neural field equation or other very large-scale problems as well, the approach of trying to find unknown parameters or parameter functions leads

us to an example of the neural kernel reconstruction problem, which is also a reference point for our investigation.

Equally, parameter estimation and pattern recognition or classification approach are similar since they both use measurements to describe an object or system but the former use real-valued scalar or vector to describe an object while the latter description is based on a selected class or category from a finite number of attributes.

In general, [12], [113] described the process of attributing a parametric description to an object, a physical process, or an event based on measurements obtained from the same object or system as parameter estimation. In contrast to kernel estimation which is more generic in approach since we do not prescribe a special form of the kernel, but just look for the connectivity between two points. We could, however, transform kernel estimation into parameter estimation when we base it on some form as in (3.40).

1.8 Thesis Outline

The thesis is structured into five chapters except for the appendix part. The introductory Chapter 1 provides a well-rounded view of the main ideas underpinning this research, and the detailed outline of the dynamical systems representations including the mathematical expressions used to denote them in the thesis are in Chapters 2 and 3. Chapter 4 provide the mathematical details and outputs of the model reconstruction approaches. The results of the statistical analysis of the sensitivity analysis of the models used in this thesis were presented in Chapter 5, while the conclusions and summary of the findings and perspectives for future research on the thesis were highlighted in Chapter 6.

We developed two model reconstruction approaches using variational data assimilation and Kalman filter methods using a radial basis function to efficiently approximate nonlinear models.

1. Chapter 1 is the introduction to the applications and techniques of this thesis, including an overview of the history of related pieces of research in the field of neuroscience 1.2, its biological and mathematical aspects, followed by a brief overview of existing pieces of literature on reduced order models and their applications in Section 1.4, a survey on machine learning

in Section 1.5, and other modern algorithms used in the data science and scientific computing communities. In addition, a brief description of parameter estimation and parameter identification in Sections 1.3. It also contains a review of the past and current works of literature on the fundamental data assimilation techniques used in the thesis 1.6, coupled with a brief explanation of the parameter estimation approach in the concluding section 1.7.

2. In Chapter 2, we highlight the model systems used as a testbed for this research - Lorenz '63 and Lorenz '96 model equations, particularly in Sections 2.1 and 2.2 respectively. The background, simulation setup for both models and a brief overview of the Amari Neural Field model 2.3.1. A complimentary weather forecasting model using data from the reaction-diffusion system Atmospheric model and its background in Section 2.4 and 2.4.1 respectively completes this chapter.

3. In Chapter 3, we show an efficient approach for model reconstruction. First, we describe a variational approach to data assimilation in a model reconstruction setup, which is essentially a three-dimensional variational assimilation 3D-VAR equipped with a Gaussian covariance matrix. A brief introduction to the Kalman filter method for model learning and its applications is in Section 3.2; the expansion coefficients described here have the potential to achieve full model reconstruction, the forcing term estimation, and polynomial estimations in Sections 3.3 and 3.3.1. This is then completed with the radial basis function in Section 3.3.2 and kernel estimations in Subsections 3.4.1 and 3.4.2.

4. Chapter 4 provides a detailed description of the algorithm developed for large-scale and highly nonlinear dynamical systems using the variational data assimilation approach. We show our learning results in *numerical examples* developed for Lorenz '63 and '96 equations, its state estimation, and the kernel reconstruction problem and carry out an application to weather forecasting. The statistical analysis of the numerical examples in the following Subsections 4.1.3, 4.2.1 and 4.4.1 describes the sensitivity analysis and error evolution of the models used in the study. Those examples were built on the theoretical discussions from the main Chapters in 1, 2 and 3, and 4.

5. In Chapter 5, we present the results of the statistical analysis of the evaluations for the L63 model in Section 5.1, for the L96 model in Section

5.2 and for the Amari Neural field model in Section 5.3. The evaluations highlighted the analysis of the error evolution considering changes to the input parameters, while documenting the learning performances of the models mentioned above at varying time steps or experiences.

6. In Chapter 6, we provide our conclusions and further possible studies building on our work and results. In Section 6.1, we provide an evaluation of results and summarise our thesis and provide the conclusion and further possible studies building on the knowledge and understanding gained from this research. In addition, a new Section 6.2 has been added to discuss the limitations of the thesis with a further concluding overview on the perspective of the thesis in Section 6.3.

7. Finally, the concluding chapter of this thesis contains a detailed description of all the codes (model learning) used in this research as an attachment in the appendix 7.1 and list of figures.

Chapter 2

Model Systems

This chapter highlights the dynamical system's representatives of all the models used as test beds for the model reconstruction techniques developed in this thesis.

The technique developed in [88] and [15] serves as a point of reference for our method, which in the case of the neural field equation is an innovative sequential kernel reconstruction approach, whose goal is to determine the strength or shape of the connectivity between the various types of neurons and collections of neurons in the brain.

First, we study the popular Lorenz 63 and 96 dynamical systems. Section 2.1 highlights the use of the Lorenz '63 model in dealing with chaotic systems. Similarly, Section 2.2 describes the use of the Lorenz '96 equation as a testbed for systems with an independent external driving force and a damping term. Then, we treat the neural field equation of computational neuroscience in Section 2.3.

The technique developed in [88] and [15] serves as a point of reference for our method, which in the case of the neural field equation is an innovative sequential kernel reconstruction approach, whose goal is to determine the strength or shape of the connectivity between the various types of neurons and collections of neurons in the brain. Finally, we investigate a reaction-diffusion system model in Subsection 2.4.

2.1 Lorenz '63

We have chosen to employ some popular basic model systems to test our model reconstruction approach. This section describes the systems which we

use for evaluating the theoretical and practical applications of our algorithm in real-life scenarios.

The reference to both theory and applications in dealing with nonlinear dynamical systems stem largely from its use in Numerical Weather Prediction (NWP) to Neuroscience including other well-known and emerging application areas. Sections 2.3 and then further in Section 3.4 discusses the use of the Amari Neural Field equation and a kernel reconstruction approach by regularized inversion as a reference point for our method in the case of the Amari model setup.

2.1.1 Simulation Setup, Techniques, and Deterministic Chaos

Over five decades ago, Lorenz [70], a meteorologist, published in the Journal of the Atmospheric Sciences [Vol 20] his findings that certain nonconservative hydrodynamical systems (with both viscous and thermal dissipations) exhibited varying patterns when subjected to different initial conditions while studying the behaviours of natural systems. In his work, Lorenz showed that the properties of nonperiodic solutions of finite systems are deterministic in nature and are designed to represent the forced dissipative hydrodynamical systems.

In his conclusion, he also posed pertinent questions to researchers regarding how long a forecast for numerical weather predictions should be. As a solution, he suggested that this could evolve either through a comparison of pairs of numerical solutions with identically initialised conditions or through the existence of an analogue if the former approach does not produce the desired result while also acknowledging the fact that significantly modified initial conditions coexist with identical numerical solutions.

The Lorenz '63 equations as displayed in Figure 2.1 are a widely used scientific and mathematical model for simulating and differentiating dynamical systems and it is also mostly acknowledged as a case study prototype for data assimilation techniques even when the available measurement data is limited, [81]. [121] described the model as a significant breakthrough in the study of chaoticity, which was considered one of the groundbreaking discoveries of the 20th century after relativity, quantum mechanics and cosmology in the field of theoretical physics.

It is a system of three coupled non-linear ordinary differential equations now known as the Lorenz equation as shown in the previous chapter above: (1.13),

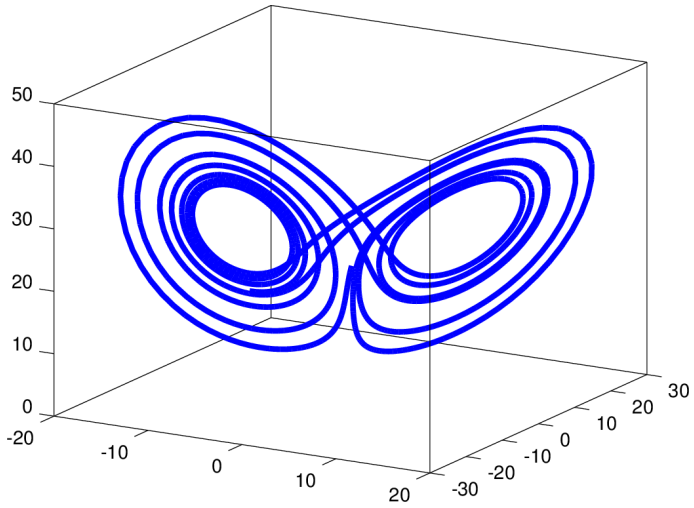


Figure 2.1: The simulation of the Lorenz '63 so-called *butterfly* equations.

(1.14) and (1.15), where $x(t) = (x_1(t), x_2(t), x_3(t))^T \in \mathbb{R}^3$ are the dynamical states. The constants σ, ρ, β are the model parameters known as *Prandtl number*, the *Rayleigh number* and a non-dimensional *wave number* respectively and τ is a temporal scaling factor.

Gianfelice [48] also examined similar variations of the Lorenz '63 models which they mapped to an Ordinary Differential Equation (ODE) system through the change of variables using the original Lorenz '63 equations. There are an appreciable amount of different kinds of literature on Lorenz '63 and one such is a more recent look at the new and emerging uses of the model explored by [115].

We also note that in the above equations (1.13)-(1.15), we have a steady-state solution when $\tau \frac{dx_1}{dt} = \tau \frac{dx_2}{dt} = \tau \frac{dx_3}{dt} = 0$, i.e., there is no convection in the state.

In general, for certain choices of the parameters σ, ρ and β we obtain chaotic behaviour of the model dynamics. For the goal of this study, we use the standard numerical values for the constants $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$.

The use of this model for this study may be justified by the arguments mentioned above. As a result, we have found its usage beneficial in reconstructing the forcing term of an interesting dynamical system. The Lorenz 63 model

and the synapses and neuronal activity within the brain comprise complicated, non-linear designs.

In addition, the simplified nature of Lorenz '63 equations has been one of the major attractions for its use by many researchers. The results of further simulations carried out using this model for the reconstruction of the forcing term and the data assimilation techniques used in this study are discussed and displayed in subsequent sections of the thesis below.

2.2 Lorenz '96

The Lorenz '96 model was suggested by E. Lorenz in a seminar on predictability at ECMWF in late 1995. It has since been used as a case study for simulating non-linear and high-dimensional chaotic systems. It is continuous in time and discrete in space model for dynamical systems used in numerical weather prediction to study the key aspects related to forecasting of spatially extended chaotic systems. It is commonly used as a testbed example system to assess basic ideas of data assimilation and forecasting in a high-dimensional set-up [61].

2.2.1 The Lorenz '96 Model System and its Background

The use of the Lorenz '96 equation as a case study has been well documented in many research areas for parametrizing highly nonlinear dynamical systems. [18] used it to implement the convergence of a novel method using numerical experiments by combining data assimilation and machine learning techniques.[25] used the prism of Lyapunov analysis to investigate the geometrical structure of instabilities in the two-scale (2.2) and ([8]) used the model as a testbed for a stochastic parametrization scheme in numerical weather simulations.

The Lorenz '96 model in its simplest form can be mathematically represented by the following linear equation below where the dynamics of the kth variable is given by:

$$\frac{dX_k}{dt} = \underbrace{-X_{k-1}(X_{k-2} - X_{k+1})}_{\text{Advection}} - \underbrace{X_k}_{\text{Diffusion}} + \underbrace{F}_{\text{Forcing}} \tag{2.1}$$

where $X = (X_1, ..., X_n)^T \in \mathbb{R}^n$ and time index $k = 1, ..., N$, with constant F which represents the magnitude of an external driving force, and it is

independent of k, n is the dimension or size of the system and $-X_k$ is a damping term.

The equation described in (2.1) above, can be extended to use the two-scale version of the Lorenz '96 model. In this case, we introduce another periodic variable Y with its own set of ODEs. Both X and Y ODEs are correlated or linked through the process called coupling.

$$\frac{dX_k}{dt} = \underbrace{-X_{k-1}(X_{k-2} - X_{k+1})}_{\text{Advection}} - \underbrace{X_k}_{\text{Diffusion}} + \underbrace{F}_{\text{Forcing}} - \underbrace{hc\bar{Y}_k}_{\text{Coupling}}, \tag{2.2}$$

$$\frac{dY_{j,k}}{dt} = \underbrace{-cbY_{j+1,k}(Y_{j-1,k} - Y_{j+2,k})}_{\text{Advection}} - \underbrace{cY_{j,k}}_{\text{Diffusion}} + \underbrace{\frac{hc}{b}Y_k}_{\text{Coupling}}, \tag{2.3}$$

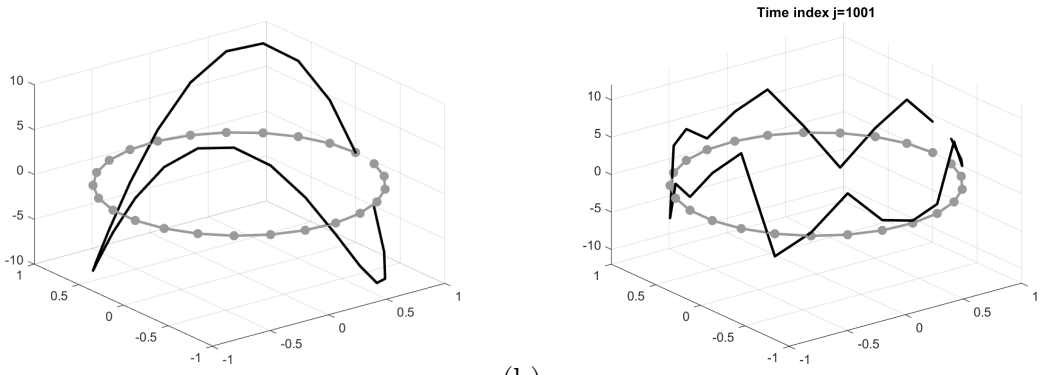
where X_k and $Y_{j,k}$ are assumed to be periodic variables denoting atmospheric quantities discretized into K and $K * J$ sectors respectively along the latitude circle [102]. The main driver for the two model variables (2.2) and (2.3) is the quadratic nonlinear modeling advection, constant forcing, linear damping and coupling between both models in each of the sectors where they operate, while the b, c, and h are constant parameters representing the spatial and temporal scale ratios and coefficients of the coupling respectively.

However, for the purpose of this thesis, the use of the extended version of this model in 2.2 and 2.3 above is beyond the scope of what is being used in this research, but it has been referenced for completeness in the description of the 2.2 model.

2.2.2 Simulation Setup, Techniques, and Visualization

The Lorenz '96 model is a simplified model for studying atmospheric dynamics (a dynamical system used to research meteorological and oceanic processes). [71], [58] and [122]. In the subsection above 2.2.1, the Lorenz '96 described by equation 2.1 gives a simple mathematical representation of the model showing its key components. The Lorenz '96 model has been extensively researched, and it has been demonstrated to display a range of complicated dynamical behaviours, including chaos, turbulence, and pattern generation. The authors [93] in their review paper, analysed links between the L96 model and specific features of brain dynamics but cautions that the L96 model is not intended to emulate neural dynamics, and its applicability to the neural world is more metaphorical than literal.

As a first example in the setup for the Lorenz '96 system, let us consider that there are N nodes located on a circle. At each node, we have some excitation modelled by $x(j)$, j = 1,...,N and the setup code is defined in the Matlab code in the Appendix Section 7.2.1. As a result, we then visualize the location of the nodes and the values of the excitation given by the code above. This is just the initial value of the excitation for $N = 9$ nodes.



(a) (b)

Figure 2.2: (a) Displays the visualization of the nodes and activity function of Lorenz '96 and (b) shows the final view when (time index=1001) of the changes in the excitations for the model at each time step around the rings.

Furthermore, we display the trajectories for all nodes individually with the time shift and integration step size in Runge-Kutta which are generated by the code defined in 7.2.1.
This leads us to the image shown below in Figure 2.3.

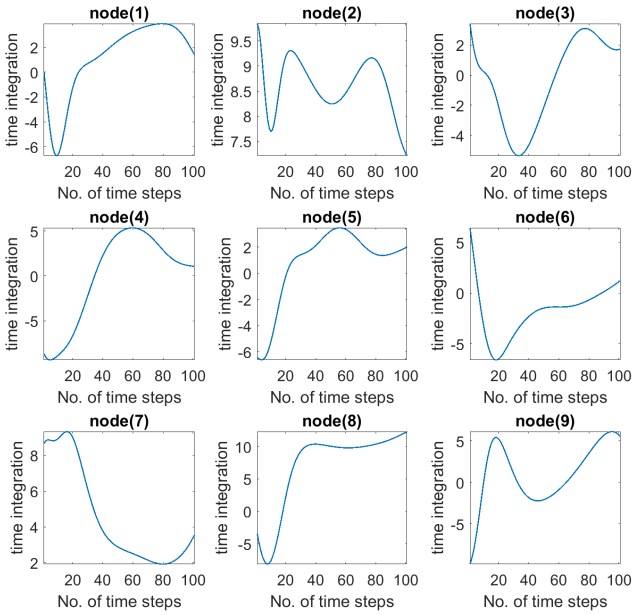


Figure 2.3: Visualization of the trajectories of the Lorenz '96 system with $N = 9$ nodes and $nsteps = 100$ time-steps with their respective time integrations

.

Lastly, we investigated the model time-integration using the `4th` order Runge-Kutta method in more detail by displaying the result of the change of excitation equations of the Lorenz '96 system defined below in the Appendix Section 7.2.1.

In this case, we define a jth time index for all the number of time steps (1,000) and monitor the changes or evolution of the excitations $x's$ for our Lorenz '96 at each time step around the defined nodal points around the circle, see 2.2 (b) above.

In Figures 2.2 and 2.3 we see the forward evolution of the model for the excitations at 1000 timesteps as shown in the setup above. The descriptions of these steps are highlighted below:

- We construct a set of nodes located on a circle, as shown above, these are the grey points and each grey dot is likened to a neuron, now placed on a circle.
- At each node, we have an excitation, given by a value $x(j)$ where j is the number of the node and $x(j) \in \mathbb{R}$ is the set of all excitation values in the state, a vector $x \in \mathbb{R}^n$, with n nodes (or neurons).
- Also, the excitation can be positive or negative, this is generally more of an oscillator model.
- We then display the excitation values by plotting the black curve as shown in both figures above.
- The neuron is fixed in space. The excitation value changes at every time index, this excitation might oscillate, i.e., it tends to go up and down.

2.3 Amari Neural Field Model

Unlike the Lorenz models, which are standard dynamical systems used in both neuroscience and meteorology as a simple test case for innovative ideas, the neural field model is more complicated and more specific towards particular applications of neuroscience. The neural field equation is also high-dimensional and has a completely different forcing term compared to the Lorenz models, where we can evaluate different approaches.

2.3.1 The Amari Model and its Background

The brain is a complex system. The Neural field theory [29] is a population-level approach to modelling the non-linear dynamics of a large population of neurons while maintaining a degree of mathematical tractability. Coombes [32] described the Neural field models as the coarse-grained activity of populations of interacting neurons. In large neural networks with complex topology, analysing and simulating networks of such high magnitude and complexity is often challenging and this is due to the nonlinearity of the activation functions of the substantial number of synaptic weights, [15].

The *Amari neural field* equation (NFE) or the *Cowan-Wilson* neural field models are used as a generic tool of study for more complex neural mass approaches in *cognitive neurodynamics*. In this study, we have used the Amari equation to reconstruct a connectivity kernel described in the inverse neural problem below:

Definition 2.3.1 (Inverse Neural Problem). *Given the dynamics $u(x,t)$, calculate a connectivity kernel $w(x,y)$ for $x,y \in D$ such that u satisfies the corresponding Amari equation, i.e.*

$$\tau \frac{du}{dt}(x,t) = -u(x,t) + \int_D w(x,y) f(u(y,t))\, dy, \tag{2.4}$$

for $x \in D, t > 0$ with initial condition

$$u(x,0) = u_0(x), \quad x \in D, \tag{2.5}$$

where D ($D \in \mathbb{R}^d$, d=2,3 and a time interval [0,T]) is assumed to be a brain area with some neuronal activity, τ is the membrane's time constant, $u(x,t)$ represents the activation of neurons at position x and time t, $w(x,y)$ is the connectivity kernel that determines the strength of connections between neurons. $f(u)$ depicts the connection between the input current and output firing rate of a neuron, whereas $f(y,t)$ is the external input. The citations are [22], [21], [53] and [110].

The Amari neural field equation is a simple model used for simulating neural activities in the brain. The basic idea that connects the use of this study object with our research is that it builds on our knowledge and understanding of the network or *wiring* connecting one neuron or a group of neurons within the brain.

Apart from building on our knowledge and understanding of this wiring connecting a neuron or group of neurons, the Amari NFE can be used to simulate the strength and or shape of the connectivity between the various types and collections of neurons in the brain.

The Amari neural field equation was used as a basis for the neural kernel reconstruction technique in this study. The inverse techniques introduced in [15] serve as a reference point for the model reconstruction approach used in the thesis.

2.4 Weather Forecasting

Weather forecasting is an age-long practice, with activities ranging from its early beginnings in the early 20th century, its first successful attempts after the second world war to modern weather models, e.g. [90].

2.4.1 Setup for Reaction-Diffusion Equation

Weather forecasting is based on partial differential equations, which control the evolution of some fields in three-dimensional space. The dynamical core of such models simulate the fluid-dimensional parts of field dynamics. Here, we use a simple two-dimensional reaction-diffusion model as a learning space for model reconstruction. The equation is given by

$$\frac{du}{dt}(x,t) = p(x) \cdot \nabla u(x,t) + c_{Diff} \triangle u(x,t), \tag{2.6}$$

for $x \in [0, a_1] \times [0, a_2]$, $t \in [0, T]$ and $a_1, a_2, T \in \mathbb{R}$. Initial conditions are prescribed by

$$u(x,0) = u_0(x), \quad x \in [0, a_1] \times [0, a_2] \tag{2.7}$$

with some function u_0. The parameter field p is chosen to be

$$p(x) := \frac{1}{\|p_{tmp}(x)\|} p_{tmp}(x), \quad x \in [0, a_1] \times [0, a_2] \tag{2.8}$$

based on

$$p_{tmp}(x) := (\frac{x}{\|x\|})^{\perp} + 0.2 * (\frac{x}{\|x\|}) \tag{2.9}$$

for $x \in [0, a_1] \times [0, a_2]$.

We employ a fourth-order Runge-Kutta scheme for integrating (2.6). To enhance stability, we use a filter $\mathcal{F}$ which removes high-order noise. The filter is based on a 2d Fourier transform F and its inverse F^{-1} in the form

$$\mathcal{F}(u) := F^{-1}(c_f \cdot F(u)) \tag{2.10}$$

with function c_f defined by

$$c_r(x) := \begin{cases} 1, & \|x\| \leq r \\ 0, & \|x\| > r. \end{cases} \tag{2.11}$$

An example with $T = 20$, $a_1 = a_2 = 5$ is shown in Figure 2.4.

The discretized version of equation (2.6) can be written in the form

$$u_{k+1,j} = \sum_{\xi} c_{j\xi} u_{k,\xi}, \quad j = 1, ..., n \tag{2.12}$$

with time index $k = 1, 2, 3, ...$ and spatial index $j = 1, ..., n$, where n is the total number of spatial grid points. The constants $c_{j\xi}$ depend on the vector field $p(x_j)$ and constants given by the finite approximation of the operators

$$\nabla := \begin{pmatrix} \frac{\partial}{\partial x_1} \\ \frac{\partial}{\partial x_2} \end{pmatrix} \tag{2.13}$$

and

$$\triangle = \frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_1^2}. \tag{2.14}$$

In a discretized setting, learning the model can be accomplished by learning the coefficients $c_{j\xi}$ for $j, \xi = 1, ..., n$. The Kalman Filter (KF) gives an estimate of the system's status based on measurements, which may be used to alter the system's output and ensure it remains within acceptable limits. However, a mismatch between the system's input and output can lead to instability, which might manifest as oscillation, overshoot, or loss of control, and a detailed description the Kalman filter approach to this task is given in Section 4.4.

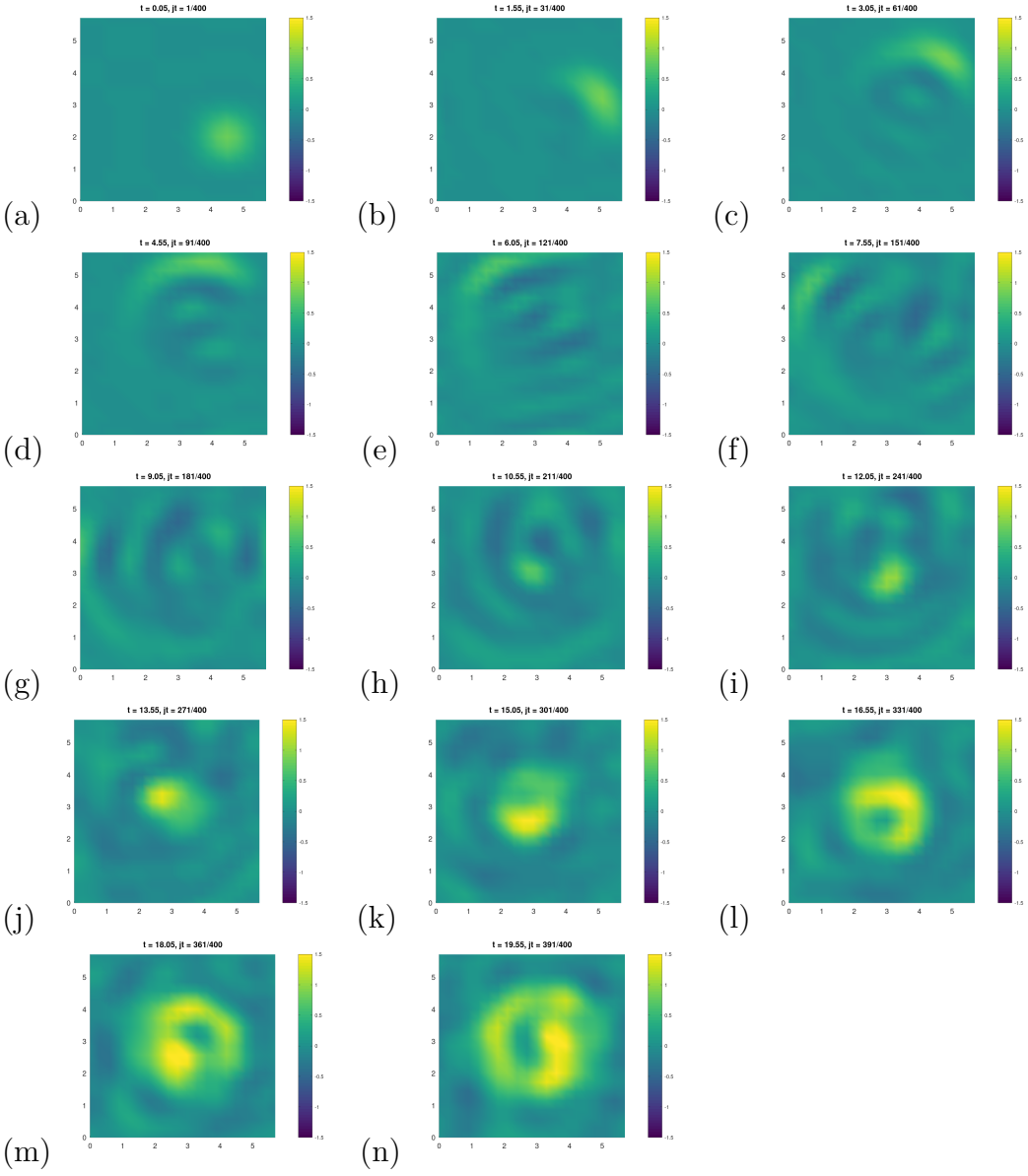


Figure 2.4: We show the time series of the integration of equation (2.6) with every $30th$ step between $T = 0$ and $T = 20$, where $h_t = 0.05$, t is the time taken and jt is the time index for the integration.

Chapter 3

Data Assimilation for Learning Models

This chapter describes two approaches to model reconstruction. First, in Section 3.1 a generic *variational data assimilation approach* for model learning is developed. In Section 3.2 a *Kalman Filter* for model construction is described, which can be understood as an extension of the variational method, where the covariance matrix is updated in each step.

We describe the general mathematical approach, which is then worked out for different low and high-dimensional dynamical systems in the upcoming sections. In particular, when applied to *forcing term estimation* in Section 3.3 we also provide some basic convergence results. Comparison to traditional *kernel reconstruction* for the *neural field equation* is carried out in Section 3.4.

Numerical results for Lorenz 63 (L63), Lorenz 96 (L96), Amari Neural Field Equation (NFE) and the reaction-diffusion system NWP model will be presented in the subsequent Chapter 4.

3.1 A Variational Approach to Model Reconstruction

The goal of this section is to develop an iterative *variational approach* to model reconstruction. Variational methods have a long tradition for the solution of inverse problems [38, 49, 60, 67, 81] and in data assimilation [59, 81, 101].

In addition, we have formulated a variational method for model reconstruction by employing the ideas of both *inverse problems* and *data assimilation*

as presented in [24]. This is done by using the 3D-VAR approach to estimate nonlinear model dynamics M which is then tested for the case of the systems L63, L96, NFE and the reaction-diffusion system. In contrast to classical data assimilation where the method is used for state estimation, in this thesis, we apply it on the second level of abstraction to reconstruct the model instead 3.9 in the model space $M \in Z$.

Following the notation of [81], the *variational approach* aims to determine some quantity $x \in X$ in a *state space* X from measurements $y \in Y$ in some *observation space* Y by minimizing the functional

$$J(x) := \alpha||x - x^{(b)}||^2 + ||y - Hx||^2, \tag{3.1}$$

where $H : X \to Y$ is the *observation operator* which maps a state x onto the simulated observation $Hx \in Y$, $\alpha > 0$ is known as *the regularization parameter* and where $x^{(b)}$ is a so-called *first guess* or *background* for the solution and the minimization of (3.1) contains both a fit to the data y and tries to keep the distance to the background $x^{(b)}$ as small as possible. It is well-known in the description by [43, 49] that the background term $||x - x^{(b)}||$ corresponds to the *Tikhonov regularization* around the first guess $x^{(b)}$.

In the examples described from the various publications by [49, 67, 81] (see also (1.7)), the minimizer of the quadratic functional (3.1) is given by

$$x^{(a)} = x^{(b)} + H^*(\alpha I + HH^*)^{-1}(y - Hx^{(b)}). \tag{3.2}$$

Here, H^* denotes the *adjoint* operator in the space X with some scalar product $\langle \cdot, \cdot \rangle$ which defines the norm $|| \cdot ||$ by $||x||^2 = \langle x, x \rangle$.

We now apply the variational approach to state estimation to the task of *model reconstruction* in the framework of the *data-based model reconstruction problem*. The task is to find the model M or its model dynamics F such that the solution x_k at times $t_k = k \cdot h_t$ with

$$h_t = \frac{T}{n_t} \tag{3.3}$$

for $k = 0, 1, 2, ..., n_t$ with the number of time steps n_t in the time interval $[0, T]$, $T > 0$, to

$$\dot{x} = F(x) \tag{3.4}$$

with H taken to be a linear operator, and initial state $x(0) = x_0$ satisfies

$$H(x_k) = y_k, \quad k = 0, 1, 2, ... \tag{3.5}$$

In the presence of observation error, however, the measurement y_k may not accurately reflect the actual state of the system. To take care of this, the observation equation above 3.5 can be adjusted to add an observation error term, e:

$$H(x_k) = y_k + e, \quad k = 0, 1, 2, ... \tag{3.6}$$

where, e accounts for the uncertainty of the measured data and the increase in the true estimation of the state using the equation of the kalman filter. In summary, it is the difference between the measured value,y_k, and the true value of the system's state.

However, we need to take care of two layers of our problem.

(L1) The *first* layer is the original state space which we use for the system and its dynamics. This is the space X on which model M acts.

(L2) The *second* layer is the space of possible models. We call it Z and assume that Z describes some adequate framework for the model M or its forcing function F.

Here, we assume that Z is a Hilbert space of model M under consideration, where the models $M : X \to X$ are *non-linear* functions describing the state evolution on layer L1 of our problem.

We can now define some dynamics on our model space, i.e. the model M_k at time t_k is mapped into the model M_{k+1} at time t_{k+1} by

$$M_{k+1} = \mathcal{M}(M_k), \quad k = 1, 2, 3, ... \tag{3.7}$$

We choose $\mathcal{M}$ to be the *constant* dynamics, i.e. we define

$$\mathcal{M}(M) = M, \quad M \in Z, \tag{3.8}$$

i.e. we only treat the case where the model under consideration is not dependent on time t. We are now prepared to define the observation operator acting on the model space by

$$\mathcal{H}_k(M) := HM(x_{k-1}), \quad k = 1, 2, 3, ... \tag{3.9}$$

with $\mathcal{H} : Z \to Y$. In this study, we have assumed that H is a linear observation operator (which contains observation error of variance epsilon taken into account in all parts of our theory and numerical simulations), then $\mathcal{H}_k$ is linear on Z

as well. In addition, M is assumed to be any model or vector valued function, and its linearity is not a requirement, since we have

$$\begin{aligned} \mathcal{H}_k(M_1 + M_2) &= H(M_1 + M_2)(x_{k-1}) \\ &= HM_1(x_{k-1}) + HM_2(x_{k-1}) \\ &= \mathcal{H}_k(M_1) + \mathcal{H}_k(M_2) \end{aligned} \tag{3.10}$$

and for $s \in \mathbb{R}$ we calculate

$$\begin{aligned} \mathcal{H}_k(sM) &= H(sM)(x_{k-1}) \\ &= H(sM(x_{k-1})) \\ &= s\mathcal{H}_k(M). \end{aligned} \tag{3.11}$$

The adjoint of the operator $\mathcal{H}_k$ is the operator $\mathcal{H}_k^*$ which satisfies

$$\langle \mathcal{H}_k(M), y \rangle_Y = \langle M, \mathcal{H}_k^* y \rangle_Z . \tag{3.12}$$

Using (3.9) we transform the left-hand side to obtain

$$\langle HM(x_{k-1}), y \rangle_Y = \langle M(x_{k-1}), H^* y \rangle_X \tag{3.13}$$

leading to

$$\langle M, \mathcal{H}_k^* y \rangle_Z = \langle M(x_{k-1}), H^* y \rangle_X . \tag{3.14}$$

Now assume that we are given some *first guess* $M^{(b)}$ for the model. For clarity, if basis functions represent the model in equation 3.15 below, it can be a function depending on its coefficient vector. We give a one-dimensional example in Section 4.1.2 below. Then, we can apply the *variational state estimation* to the model space Z at each time step t_k by

$$M_k^{(a)} = M_k^{(b)} + \mathcal{K}_k(y_k - \mathcal{H}(M_k^{(b)})), \tag{3.15}$$

$$M_{k+1}^{(b)} := \mathcal{M}(M_k^{(a)}) = M_k^{(a)} \tag{3.16}$$

for $k = 1, 2, 3, ...$ with the *model Kalman operator*

$$\mathcal{K}_k := \mathcal{H}^*(\alpha I + \mathcal{H}^*\mathcal{H})^{-1}, \quad k = 1, 2, 3, ... \tag{3.17}$$

with some regularization parameter $\alpha > 0$. Here, we remark that we have argued in a general Hilbert space environment. When we choose classical

ℓ^2-spaces with weights given by the covariance $\mathcal{B}$ on Z and the error covariance $\mathcal{R}$ on Y, as worked out in [24] or in [81], equations (5.2.11) and (5.2.14), the Kalman operator (3.17) takes the form

$$\mathcal{K}_k := \mathcal{B}\mathcal{H}'(\alpha\mathcal{R} + \mathcal{H}\mathcal{B}\mathcal{H}')^{-1}, \quad k = 1, 2, 3, ..., \tag{3.18}$$

where now the adjoints $\mathcal{H}^*$ have been replaced by $\mathcal{H}^* = \mathcal{B}\mathcal{H}'\mathcal{R}^{-1}$. Equations (3.15), (3.16) and (3.18) describe the use of *3D-variational data assimilation* to the determination of the model M.

The following sections will apply this rather generic abstract setup to the estimation of a general nonlinear model on a low or high-dimensional state space. Here, as a first step (we call it the naïve step), we remark that $\mathcal{R}$ is an operator or matrix on observation space Y. The operator $\mathcal{H}'$ maps the observation space $\mathbb{R}^m$ into the model space Z and $\mathcal{B}$ is an operator or a matrix on Z. We further note that

- The observation operator is evaluated by

$$\mathcal{H}(M^b) = HM^{(b)}(x_{k-1}^{(a)}) = Hx_k^{(b)}, \tag{3.19}$$

  which corresponds to $\mathcal{H} = H\delta_{x_{k-1}}$, using x_{k-1} as a short notation for $x_{k-1}^{(a)}$, where the δ is the delta function, i.e., the mapping of a function onto its value at a given point. δ_x in model space is defined by $\delta_x\ (M) := M(x)$, therefore, $H\delta_x(M) = HM(x)$.

- In the simplest case, using $\mathcal{H}' = \delta_{x_{k-1}}H'$, we can derive (see [83])

$$\begin{aligned}
&\mathcal{H}'(\mathcal{R} + \mathcal{H}\mathcal{B}\mathcal{H}')^{-1}(y_k - \mathcal{H}(M_k^{(b)})) \\
&= \delta_{x_{k-1}}H'(R + \mathcal{B}(x_{k-1}, x_{k-1})HH')^{-1}(y_k - Hx_k^{(b)}) \\
&= \delta_{x_{k-1}}H'(R + HH')^{-1}(y_k - Hx_k^{(b)}) \\
&= \delta_{x_{k-1}}x_k^{(a)}.
\end{aligned} \tag{3.20}$$

Finally, the increment is given by

$$\begin{aligned}
\mathcal{B}\mathcal{H}'(\mathcal{R} + \mathcal{H}\mathcal{B}\mathcal{H}')^{-1}(y_k - \mathcal{H}(M_k^{(b)})) &= \mathcal{B}\delta_{x_{k-1}}x_k^{(a)} \\
&= \mathcal{B}(x, x_{k-1})x_k^{(a)}.
\end{aligned} \tag{3.21}$$

If $\mathcal{B}(x, \tilde{x})$ is a Gaussian function of x with centre $\tilde{x}$, then the model increment in each step is a Gaussian RBF function centred at $x_k^{(a)}$, compare Odunuga et al. [83].

3.2 Kalman Filter for Model Learning

The Kalman filter (KF) is an important method in geosciences, for *numerical weather prediction* (NWP) and other related fields of study. It has a very strong background in optimal estimation field theory [63]. Particle filtering is particularly a useful technique for inference in state-space dynamic models [109].

Here, we describe the Kalman filter in the above generic framework for model reconstruction. As an application, it can be used for estimating unknown parameters (i.e., finding the right parameter updates) from known measurements observed over a given period.

The algorithmic approach follows the basic ideas of Section 3.1, but the Kalman Filter adapts the covariance matrix in each assimilation step. This means we complement equation (3.15) with the updated equation

$$\mathcal{B}^{(a)} = (I - \mathcal{K}\mathcal{H})\mathcal{B}^{(b)} \tag{3.22}$$

where the *first guess* covariance matrix $\mathcal{B}^{(b)}$ is employed at time step t_k, and $\mathcal{B}^{(a)}$ is the *analysis* covariance matrix which reflects the posterior uncertainty in model space after the assimilation of the observations based on the observation operator $\mathcal{H}$. Altogether, we obtain the Kalman equations

$$\begin{aligned} M_k^{(a)} &= M_k^{(b)} + \mathcal{K}_k(y_k - \mathcal{H}(M_k^{(b)})), && (3.23)\\ M_{k+1}^{(b)} &:= \mathcal{M}(M_k^{(a)}) = M_k^{(a)} && (3.24)\\ \mathcal{B}_{k+1} &= (I - \mathcal{K}\mathcal{H})\mathcal{B}_k && (3.25) \end{aligned}$$

for $k = 1, 2, 3, ...$ with the *model Kalman operator*

$$\mathcal{K}_k := \mathcal{B}_k\mathcal{H}'(\alpha\mathcal{R} + \mathcal{H}\mathcal{B}_k\mathcal{H}')^{-1}, \quad k = 1, 2, 3, ... \tag{3.26}$$

To gain a deeper understanding, let us describe the above equations in a concrete setting. Using an *expansion* of the model M with respect to some basis functions

$$\varphi_\xi : X \to X \tag{3.27}$$

for $\xi = 1, ..., N$ acting on the state space, we describe $M = M[c]$ with some *coefficient vector* $c \in Z = \mathbb{R}^N$ of dimension $N \in \mathbb{N}$. We can now formulate the Kalman filter approach with respect to the expansion coefficients. We note

that the functions can be non-linear, but the expansion depends linearly on the coefficients c. In the case where $X = \mathbb{R}^n$, this leads to a linear form

$$M[c] = Ac \tag{3.28}$$

with a matrix

$$A := \left(\varphi_\xi(x_{k-1}^{(a)})\right)_{\xi=1,\ldots,N} \in \mathbb{R}^{n\times N} \tag{3.29}$$

which consists of the basis functions $\varphi_\xi(x)$ evaluated at the current background state. With more detail this reads

$$M[c](x_{k-1}^{(a)}) = \sum_{\xi=1}^{N} \varphi_\xi(x_{k-1}^{(a)})c_\xi, \tag{3.30}$$

We now obtain the observation operator $\mathcal{H}$ applied to $M[c]$ to be given by

$$\begin{aligned} H_c \cdot c &= H \sum_{\xi=1}^{N} \varphi_\xi(x_{k-1}^{(a)})c_\xi \\ &= HAc. \end{aligned} \tag{3.31}$$

Clearly, from (3.30), (3.31) and (3.29) we see that H_c is given by

$$H_c := HA \in \mathbb{R}^{m\times N}. \tag{3.32}$$

Here, we use the notation H_c in general and `Hc` in Octave or Matlab code.

We are now prepared to carry out the data assimilation step for model learning with the Kalman filter. Let B_c be the covariance matrix in coefficient space, i.e. $B_c \in \mathbb{R}^{z\times z}$. Then, we solve

$$H_c \cdot c = Hx_k^{(a)} \tag{3.33}$$

by standard data assimilation to calculate an update of the model coefficient c, i.e. we calculate

$$c_k^{(a)} = c_k^{(b)} + B_c H_c^T (R + H_c B_c H_c^T)^{-1}(y_k - H_c c_k^{(b)}) \tag{3.34}$$

for $k = 1, 2, 3, \ldots$. Further, we update the covariance matrix B_c based on the Kalman filter equations, i.e.

$$B_c^{(a)} = (I - K_c H_c)B_c^{(b)} \tag{3.35}$$

in each step of the filtering, where

$$K_c = B_c^{(b)} H_c^T (R + H_c B_c^{(b)} H_c^T)^{-1}. \tag{3.36}$$

In equation 3.31 above, the c coefficient indicates the connection between the input and output variables in the model, i.e., it quantifies the effect of a certain input variable on its corresponding output variable. Thus, the reliance of the linear operator H on c is vital for comprehending and evaluating the model's outcomes, and it underscores the need of selecting and estimating the proper coefficient c for a specific model to ensure correct and dependable results. Therefore, any changes in the value of the coefficients can drastically impact the model's behaviour and performance.

3.3 Forcing Term Estimation (FTE)

In general, models for dynamical systems are complex codes based on partial differential and integral equations modeling the underlying dynamics, physics, biology or chemistry. Usually, the time derivative $\dot{x}$ of the state variables x is given by a forcing term $F(x)$ in the form

$$\dot{x} = F(x), \tag{3.37}$$

where $F : X \mapsto X$ with state space X is a function of the state variables.

In this section, we focus on the formulated technique used for estimating the forcing term F which controls the dynamical system. For small time steps, we can employ Euler's method for integration, leading to

$$x(t_{k+1}) \approx x(t_k) + \dot{x}(t_k) \cdot \tau = x(t_k) + F(x(t_k)) \cdot \tau \tag{3.38}$$

with some small time interval τ and $t_{k+1} = t_k + \tau$. We rearrange the terms into

$$F(x(t_k)) \approx \frac{1}{\tau}\Big(x(t_{k+1}) - x(t_k)\Big). \tag{3.39}$$

We approximate F at $x(t_k)$ based on the trajectory of our dynamical systems at the point in time t_k and t_{k+1}. Now, using expansions of the forcing term $F(x)$ in the form

$$F(x) = \sum_{\xi=1}^{N} \varphi_\xi(x) c_\xi \tag{3.40}$$

with basis functions φ_ξ and coefficients $c_\xi \in \mathbb{R}$ for $\xi = 1, ..., N$, we can work out reconstructions of F based on the variational algorithm (VAR) of Section 3.1 or the Kalman Filter (KF) of Section 3.2, reconstructing the coefficient vector $c \in \mathbb{R}^N$.

The above task is, of course, a very broad and generic problem. We will see that in the simplest possible case of $X = \mathbb{R}$ it leads to classical interpolation or splines. In the subsequent Section 3.3.1, we will employ *polynomial expansions* for the forcing term. The approach of *radial basis functions* will be described in Section 3.3.2.

Giles et al. [55] in their conclusion, reiterated the importance of the estimation of forcing functions as a diagnostic tool for nonlinear dynamics. He also posited that the lack of fit of models is adversely affected when they contain unmeasured components and further recommended developing tests for independence to evaluate the synergy between forcing functions and model trajectory. This approach is relevant to the methods formulated in this thesis.

3.3.1 Forcing Term Estimation based on Polynomials (POL)

Here, we approach the reconstruction of the dynamical model propagation from t_{k-1} to t_k based on the *Taylor series approximation* of the forcing function $F(x)$ with respect to the state variables $x_j, j = 1, ..., n$. The

- convergence of Taylor's expansion or its *polynomial approximation* to approximate the forcing term F together with
- the convergence of Euler's method when time steps are chosen smaller and smaller

will lead to the *convergence of model reconstruction* on the manifold covered by the dynamical system trajectories. Here, we work out these arguments in adequate detail.

We assume that we work in state space $X = \mathbb{R}^n$, such that F maps $\mathbb{R}^n$ into itself. In a first step, we approximate F by its Taylor series around $x = 0 \in \mathbb{R}^n$,

which using the standard definitions for vectors $x, \beta \in \mathbb{R}^n$

$$\begin{aligned} |\beta| &:= \beta_1 + ... + \beta_n, \\ \beta! &:= \beta_1! \beta_2! \cdots \beta_n! \\ x^\beta &:= x_1^{\beta_1} x_2^{\beta_2} \cdots x_n^{\beta_n} \\ d^\beta &= d_1^{\beta_1} d_2^{\beta_2} \cdots d_n^{\beta_n} = \frac{\partial^{|\beta|}}{\partial x_1^{\beta_1} \cdots \partial x_n^{\beta_n}}. \end{aligned} \tag{3.41}$$

can be written in the form

$$(F(x))_j = \sum_{|\beta| \leq \ell} \frac{d^\beta F}{\beta!}\Big|_{x=0} x^\beta + R_{\ell+1,j}(x), \quad j = 1, ..., n, \tag{3.42}$$

with remainder terms $R_{\ell+1,j}(x), x \in \mathbb{R}^n$, $j = 1, ..., n$. The polynomial forcing term is then given by

$$F_\ell(x) = \left(\sum_{|\beta| \leq \ell} a_\beta^{(j)} x^\beta \right)_{j=1,...,n} \tag{3.43}$$

for $x \in \mathbb{R}^n$. It can be used to approximate the general term $F(x)$. We note that in (3.43) the model for each component $(F_\ell(x))_j$, $j = 1, ..., n$, of $F_\ell(x)$ is given by the coefficients $a_\beta^{(j)}$ for all $|\beta| \leq \ell$. The number of these terms is given by the sum of the partition functions $p(\xi)$ for $\xi = 0, 1, 2, ..., \ell$.

Here, we simply estimate the total number of degrees of freedom d_{free} to be bounded by

$$d_{free} = n \cdot (\ell + 1)^n \tag{3.44}$$

degrees of freedom when we employ the above equation, counting less than $\ell + 1$ choices for each power of the n variables, multiplied by n components of the function with values in $\mathbb{R}^n$. For a model in $\mathbb{R}^3$ with a polynomial degree for each variable x_1, ..., x_n up to $\ell = 1$, this would lead to $d_{free} = 3 \cdot 2^3 = 24$ possible degrees of freedom. So, if we know that the approximate model falls into the class given by (3.43), we know that we have to reconstruct at most 24 coefficients. Also, using further symmetries and constraints will restrict the number of degrees of freedom significantly.

As described in Section 1.6, the standard model of data assimilation is to receive measurements at times t_k, $k = 1, 2, 3, ...$ These are the times when we can use data to update our knowledge of the dynamical model as well. However,

the forcing term equation (3.37) describes the change of x at time t. The model dynamics $M(x)$ from t_{k-1} to t_k is obtained from the full integration of the equation (3.37). We will assume that the trajectory $x(t)$ remains in the ball B_r for all $t > 0$, i.e., that we have $||x(t)|| < r, \quad t \geq 0$. If $F(x)$ is Lipschitz continuous on $\overline{B_r}$ with Lipschitz constant C_L, we can estimate

$$||F(x) - F(x_{k-1})|| \leq C_L ||x - x_{k-1}|| \tag{3.45}$$

for the integration of $x(t)$ from t_{k-1} to t_k. Also, if F depends continuously on x, we can estimate

$$\begin{aligned} ||x(t) - x_{k-1}|| &= || \int_{t_{k-1}}^{t} \dot{x} \ dt \ || \\ &= || \int_{t_{k-1}}^{t} F(x(t)) \ dt \ || \\ &\leq C_0 |t - t_{k-1}| \end{aligned} \tag{3.46}$$

with

$$C_0 := \sup_{x \in \overline{B_r}} ||F(x)||. \tag{3.47}$$

Now, we calculate

$$\begin{aligned} x(t) - x_{k-1} &= \int_{t_{k-1}}^{t} \dot{x} \ dt \\ &= \int_{t_{k-1}}^{t} F(x(t)) \ dt \\ &= \int_{t_{k-1}}^{t} \Big[F(x_{k-1}) + \Big(F(x(t)) - F(x_{k-1}) \Big) \Big] \ dt \\ &= F(x_{k-1}) \cdot (t - t_{k-1}) + \int_{t_{k-1}}^{t} \Big(F(x(t)) - F(x_{k-1}) \Big) \ dt \end{aligned} \tag{3.48}$$

Using (3.45) and (3.46) we estimate the last term in (3.48) by

$$\begin{aligned} \Big|\Big| \int_{t_{k-1}}^{t} \Big(F(x(t)) - F(x_{k-1}) \Big) \ dt \Big|\Big| &\leq \int_{t_{k-1}}^{t} ||F(x(t)) - F(x_{k-1})|| \ dt \\ &\leq C_L \int_{t_{k-1}}^{t} ||x(t) - x_{k-1}|| \ dt \\ &\leq C_L \end{aligned} \tag{3.49}$$

to write (3.48) for time $t = t_k$ in the form

$$x_k - x_{k-1} = F(x_{k-1}) \cdot (t_k - t_{k-1}) + O(|t_k - t_{k-1}|^2). \tag{3.50}$$

The estimate (3.50) means that we can approximate the full model dynamics in the form

$$\begin{aligned} x_k - x_{k-1} &= M(x_{k-1}) - x_{k-1} \\ &= (M - I)(x_{k-1}) \\ &= F(x_{k-1}) \cdot (t_k - t_{k-1}) + O(|t_k - t_{k-1}|^2) \end{aligned} \tag{3.51}$$

with the components of F given by (3.42). Using standard convergence results for polynomial approximation of C^ℓ-functions (e.g. [96]), i.e., the estimates on the remainder R_ℓ of (3.42), we now obtain the following result.

LEMMA 3.3.1. *Assume that for a ball $B_r \subset \mathbb{R}^n$ of radius $r > 0$ and $\ell \in \mathbb{N}$ the forcing term $F(x), x \in \mathbb{R}^n$, of a dynamical system (3.37) satisfies $F \in C^\ell(\overline{B_r})$ and that the trajectory $x(t)$ remains in B_r for all times $t > 0$. Then, the model propagation $M : x(t_{k-1}) \mapsto x(t_k)$ satisfies*

$$\begin{aligned} &\left|\left| x(t_k) - x(t_{k-1}) - F_\ell(x(t_{k-1})) \cdot (t_k - t_{k-1}) \right|\right| \\ &\leq C_L C_\ell (t_k - t_{k-1})^2 \frac{|x_k - x_{k-1}|^\ell}{\ell!} \end{aligned} \tag{3.52}$$

for $k = 1, 2, 3, ...$ with Lipschitz constant C_L of F and C_ℓ given by the supremum of the ℓ-st derivative

$$C_\ell := \sup_{x \in \overline{B_{2r}}} \sum_{|\beta| = \ell + 1} \left| \frac{d^\ell F(x)}{dx_1^{\ell_1} \cdots dx_n^{\ell_n}} \right| \tag{3.53}$$

of the forcing term F.

Proof. The result is a consequence of equation (3.51) derived above. □

We use the notation $x_k = x(t_k)$ and $x_{k-1} = x(t_{k-1})$ and assume that the true model M is mapping x_{k-1} into x_k, i.e.,

$$x_k = M(x_{k-1}), \quad k = 1, 2, 3, ... \tag{3.54}$$

Then, the above theorem estimates the difference between the true model M applied to x_{k-1} and its approximation

$$M[a](x_{k-1}) = x_{k-1} + F_\ell(x(t_{k-1})) \cdot (t_k - t_{k-1}) \tag{3.55}$$

based on the polynomial forcing term F, i.e.,

$$\left|\left| M(x_k) - M[a](x_k) \right|\right| \leq C_L C_\ell (t_k - t_{k-1})^2 \frac{|x_k - x_{k-1}|^\ell}{\ell!} \tag{3.56}$$

The *algorithmic approach* now follows the basic ideas of Section 3.1 and Section 3.2, see equations (3.15) and (3.16) as well as (3.34) and (3.35). For the case of a polynomial expansion, it provides an estimate for $M[a]$ with some coefficient vector a.

We also note that any knowledge about the true polynomial representation of the model can be used here to *limit the number of terms* needed to represent the forcing term and make the whole approach efficient and sufficiently stable. We will apply this for example to the Lorenz 96 system in Section 4.2.

3.3.2 Forcing Term Estimation based on Radial Basis Functions (RBF)

We have studied a polynomial approximation of the forcing term in our previous Section 3.3.1. Of course, other sets of basis functions or splines can be tried and might be adequate depending on the particular dynamical system under consideration.

A natural approach is to search for an approximation of the function F by high-dimensional *Gaussian basis functions*, i.e., basis functions of the form

$$\varphi[\mu, q](x) := q \cdot e^{-(x-\mu)^T B^{-1}(x-\mu)}, \quad x \in X, \tag{3.57}$$

where $\mu \in X$ is the centre of the radial basis function, $q \in X$ is its target and B is its covariance. Here, we will study the case where we keep B constant and employ sets of basis functions defined by their nodes μ and expansion coefficients.

Let us assume we are given $N \in \mathbb{N}$ points p_ξ, $\xi = 1, ..., N$ close to the manifold covered by the system dynamics, target vectors q_ξ, $\xi = 1, ..., N$ and that we want to approximate F by $N \in \mathbb{N}$ radial basis functions with centre given by points $p_1, ..., p_N$ and target vectors q_ξ. We employ the notation

$$\varphi_\xi := \varphi[p_\xi, q_\xi]. \tag{3.58}$$

Then, solving equation (3.39) with the above basis functions, the classical RBF interpolation system leads to the equations

$$\sum_{\xi=1}^{N} \varphi_\xi(x_k) c_\xi = \rho_k, \quad k = 1, ..., K \tag{3.59}$$

for the unknown *coefficients* $c_\xi \in X$, $\xi = 1, ..., N$ and the vectors

$$\rho_k := \frac{1}{\tau}(x_{k+1} - x_k) \in X, \quad k = 1, 2, 3, ... \tag{3.60}$$

In the case where $X = \mathbb{R}^n$ we can write (3.59) in matrix equation form

$$\rho_k = A_k c \tag{3.61}$$

where $\rho_k \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times N}$ is the matrix with entries

$$A_k = \Big(\varphi[p_\xi, q_\xi](x_k)\Big)_{\xi=1,...,N} \tag{3.62}$$

for $k = 1, ..., K$. When solving the matrix equations (3.61) for $k = 1, ..., K$ based on the algorithms (VAR) and (KF), the dynamics F is approximated by

$$F_{appr}(x) = \sum_{\xi=1}^{N} \varphi_\xi(x) c_\xi, \quad x \in X. \tag{3.63}$$

3.4 Neural Kernel Estimation

We have introduced the *neural field model* in Section 2.3. Neural activity is described by some activity function u on a domain D modelling the brain, where u satisfies an equation of the form (3.37) with F given by

$$F(u, x) = -u(x, t) + \int_D w(x, y) f(u(y, t))\, dy, \tag{3.64}$$

where in traditional notation $x, y \in D$ are points in space $\mathbb{R}^d$ for $d = 2, 3$. The area of *kernel reconstruction*, compare [4, 15, 89], *estimates the neural connectivity* $w(x, y)$ from the given measurements. Usually, it is based on reconstructing the activity function $u(x, t)$ for $x \in D$ and $t \in [0, T]$ first and then *determining the neural dynamics* from the states, i.e., the underlying parameter functions as e.g. the *neural kernel* w driving these dynamics given some initial condition using the Amari neural field equation (2.4).

Our goal in the following sections is to compare sequential model reconstruction with classical kernel estimation. To this end, we briefly describe kernel estimation in Section 3.4.1. We will then describe how the variational algorithm (VAR) defined in (3.15) and (3.16) or Kalman Filter (KF) given by (3.34) and (3.35) can be used for sequential kernel estimation in Section 3.4.2. We will see

that the sequential methods can be equivalent to classical kernel estimation, but they are computationally much more efficient and exhibit further very useful properties.

In practice, computational efficiency is the capacity of a computer method or programme to do a particular job with minimal computing resources, such as memory, processing power, and time. An algorithm is deemed efficient if it uses fewer computational resources to solve a problem than other algorithms in a similar capacity. Efficiency is essential to software development since it may affect the user experience and overall system performance. When programmes are inefficient, they might take longer to execute, consume more memory, and need more processing power, resulting in decreased performance, higher energy usage, and higher expenses.

The algorithmic complexity of a programme, which refers to the number of processes it must complete to solve a problem, is one of the most important criteria affecting its computing efficiency. In general, algorithms with a lower computational complexity are more effective than those with a larger one. Other approaches to increase computational efficiency include optimising the algorithms used in the programme, decreasing the number of memory accesses, and lowering the quantity of data that must be processed. Moreover, techniques like caching, parallel processing, and data compression can increase productivity.

Computational efficiency is essential to the software development life cycle. Algorithms and programmes that are efficient can give faster, more dependable, robust and more cost-effective solutions to issues, whereas inefficient programmes can lead to sluggish performance, excessive energy consumption, and increased expenditures.

For our case, we compare the calculation of the full matrix W over a full collection of time steps, which needs to solve a problem of size $N = n^d * n^d$ with n being the number of neural variables in one space direction, d being the space dimension (for our example $d = 2$). The inverse problem is a minimization problem with N unknowns and $m * nk$ observations, with nk being the number of time steps (measurement times) under consideration and m the number of measurements per time step. Using a model approximation with Np degrees of freedom and the Kalman filter for learning a coefficient vector $c \in \mathbb{R}^{Np}$, we solve a problem with Np unknowns at each time step.

As a small example, with $n = 100$, $d = 2$ and $m = 100$ and $nk = 100$ the inversion problem solves for $n^4 = 10^8$ unknowns solving an equation system of

dimension $10^4 \times 10^8$. With an approximation of dimension $Np = 50$, Kalman filter needs to solve 100 problems of size 50×100 iteratively. The second task is fast on a standard PC, the first task is slow. When moving to dimension $d = 3$, the first task would be a system of size $10^4 \times 10^{12}$, not feasible on standard PCs at the moment.

3.4.1 Classical Kernel Estimation

The neural fields model is more complicated and specific towards a particular application area e.g., Neuroscience, unlike the Lorenz which is a standard dynamical system used in both neuroscience and meteorology as a simple test case for new ideas.

Under the condition that the forcing function $f()$ is known, the task to learn the neural model can be reduced to learning the connectivity kernel w. We test the technique by carrying out the following steps.

- We first choose some reconstructed (or prescribed) dynamical field $v(p,t)$. Here, one example is to employ some Gaussian-shaped pulse with a centre travelling along some parabolic curve through a rectangular domain D.
- We pick matching times $t_k = 1, 2, 3, ..., 25$ for which we fed the corresponding function $v(\cdot, t_k)$, $k = 1, ..., 25$ into the model reconstruction algorithms.
- For testing the result of the model reconstruction, we simulate the neural field equation according to equation (4.4) with initial state $v(\cdot, t_1)$ and function f based on the Eulerian finite difference method.

In more detail, given some dynamical field $u(p,t)$ for $p \in D$ and $t \in [0,T]$, we can rewrite equation (4.4) in the form

$$\tau \, \dot{u} \, (p,t) + u(p,t) = \int_D w(p,q) f(u(q,t)) \, dq, \tag{3.65}$$

for $p \in D$ and $t \in [0,T]$. In discretized form based on a quadrature formula for the integration, we obtain an equation of the form

$$\Psi = W\Phi \tag{3.66}$$

with matrices

$$\Psi_{jk} := \tau \, \dot{u} \, (p_j, t_k) + u(p_j, t_k), \tag{3.67}$$

$$\Phi_{\xi k} := f(u(q_\xi, t_k)) s_\xi \tag{3.68}$$

with quadrature weights s_ξ and

$$W_{j\xi} := w(p_j, q_\xi), \tag{3.69}$$

for $j, \xi = 1, ..., n$ and $k = 1, ..., n_t$. In the case of a rectangular rule, the quadrature weights s_ξ are given by $s_\xi = a_1/n_1 \cdot a_2/n_2$ on a domain $D = [0, a_1] \times [0, a_2]$ with n_1 discretization points in e_1 direction and n_2 discretization points in e_2 direction.

Equation (3.66) includes the knowledge about the field $u(p, t)$ at all time steps t_k for $k = 1, ..., n_t$. With this knowledge, we solve the equation for W by rewriting it into

$$\Psi^T = \Phi^T W^T. \tag{3.70}$$

The regularized solution of (3.70) based on Tikhonov regularization is given by

$$W^{(\alpha)} := (\alpha I + \Phi\Phi^T)^{-1}\Phi\Psi^T \tag{3.71}$$

3.4.2 A Kalman Filter for Kernel Estimation

As a follow-up from 3.4 and 3.4.1 above, in this subsection, we discuss the setup, field simulations and the model reconstruction approach used in the kernel estimation methods described above. We then compare it with the reconstructed dynamics generated. Here, we carry out learning of the model itself for the neural field dynamics based on the Kalman filter. Finally, we then display the results of the error for Kalman reconstruction in comparison with the full four-dimensional neural field reconstruction method of Section 3.4.1.

In more detail, we start as in the previous section with the equation

$$\tau\ \dot{u}\ (p, t) + u(p, t) = \int_D w(p, q) f(u(q, t))\ dq, \tag{3.72}$$

for $p \in D$ and $t \in [0, T]$. But now, we sequentially treat the times t. In discretized form based on a quadrature formula for the integration, we obtain an equation of the form

$$\Psi_k = W\Phi_k \tag{3.73}$$

with column vectors

$$\Psi_k := \Big(\tau\ \dot{u}\ (p_j, t_k) + u(p_j, t_k)\Big)_{j=1,...,n}, \tag{3.74}$$

$$\Phi_k := \Big(f(u(q_\xi, t_k))\Big)_{\xi=1,...,n} \tag{3.75}$$

and W defined as in (3.69).

The equation (3.73) below, represents a linear transformation or mapping between two vectors, Ψ_k^T and Φ_k^T, using the transpose of a matrix W

$$\Psi_k^T = \Phi_k^T W^T \tag{3.76}$$

where Ψ_k^T indicates the transposition of the row vector Ψ_k, Φ_k^T indicates the transposition of the row vector Φ_k, which also has k components, and W^T represents the transpose of the matrix W.

Clearly, one single equation is not sufficient for the reconstruction of W. But with the Kalman filter solving equations described in Section 3.2 for $k = 1, 2, ..., n_t$ we can iteratively obtain a reconstruction of W. To use (3.34) - (3.35) we need to bring (3.76) into the form

$$r = H_c \cdot c \tag{3.77}$$

with c being either a reordered version of W or the constants of an appropriate approximation Ansatz for W, $r = \Psi_k$ and H_c being the observation operator mapping c onto $(\Phi_k^T W^T)^T$.

We will describe a high-dimensional and a low-dimensional version of this type of Kalman learning of the neural kernel in Section 4.3.

Chapter 4

Low and High-Dimensional Applications

This chapter works out the applications of the algorithms introduced in Section 3 for sequential model reconstruction developed based on standard ensemble data assimilation. The aim is to provide an approach which takes care of the limitations of the algorithm in dealing with large-scale data assimilation frameworks for model reconstruction.

We will apply the variational scheme (3.15) and (3.16) to reconstruct the Lorenz 63 model in Section 4.1. For reconstructing the model in the case of Lorenz 96 we work out the Kalman filter approach (3.34) and (3.35) in Section 4.2. Reconstructions of the neural connectivity kernel from dynamics of the neural field equation based on the Kalman filter model reconstruction are shown in Section 4.3. Finally, we study reconstructions of a simplified version of temperature dynamics for the reaction-diffusion system numerical NWP model in Section 4.4.

4.1 Learning the three-dimensional Lorenz '63

In this section, our goal here is to present the result of the scheme for a low-dimensional model using the Lorenz '63 model, which is widely used as a study object for dynamical systems, compare for example [81]. It is a system of three non-linear ordinary differential equations (see 1.13, 1.14 and 1.15 above, with constants σ, ρ, β known as *Prandtl number*, the *Rayleigh number* and a nondimensional *wave number*. Here, for the constants we take the classical

values $\sigma = 10$, $\beta = 8/3$ and $\rho = 28$. The implementation of the system is straightforward.

As a first test case for model reconstruction, we test the reconstruction of its dynamics from a sequence of measurements y_k of the full state.

We use a straightforward implementation of equations (3.20) and (3.21). We first generate the measurement data by running a Lorenz 63 model, calculating the measurements $y(:,k)$ for $k = 1, 2, 3, ..., \text{Nnat}$. The true curve is stored in $\text{xv}(:,k)$ for $k = 1, 2, 3, ..., \text{Nnat}$.

In sub-section 4.1.2 below, we display the results of the convergence of Lorenz '63 model learning reconstruction technique in figure (4.1). Similar displays were shown in figures (4.2) and (4.3) for true and approximated model with different time steps.

A display of the original and reconstructed trajectory is found in Figure 4.6(b). The error evolution is shown in Figure 4.6(c). The Matlab/Octave codes used in both cases are in Appendix Section 7.1.

- In particular, the learning equations (3.15) - (3.17) are realized by line 14 of the first code example of Section 7.1.
- The approximate model is a sum of Gaussians with centres x_k and learned coefficients, shown by the second code example in Section 7.1.
- The data assimilation cycle in which the model learning is integrated is shown by the third code example of Section 7.1.

4.1.1 Variational Model Learning a 1d Scenario

In this subsection, we demonstrate the convergence of the model learning technique described above using a one-dimensional case, in addition, we show the results of the findings and their corresponding errors. We describe on an interval $I = [a, b]$, we then define a scalar function $M : x \mapsto M(x)$ with values $M(x) \in I$ for all $x \in I$. Then, M defines a dynamics with initial state $x_0 \in I$ by

$$x_k := M(x_{k-1}) \tag{4.1}$$

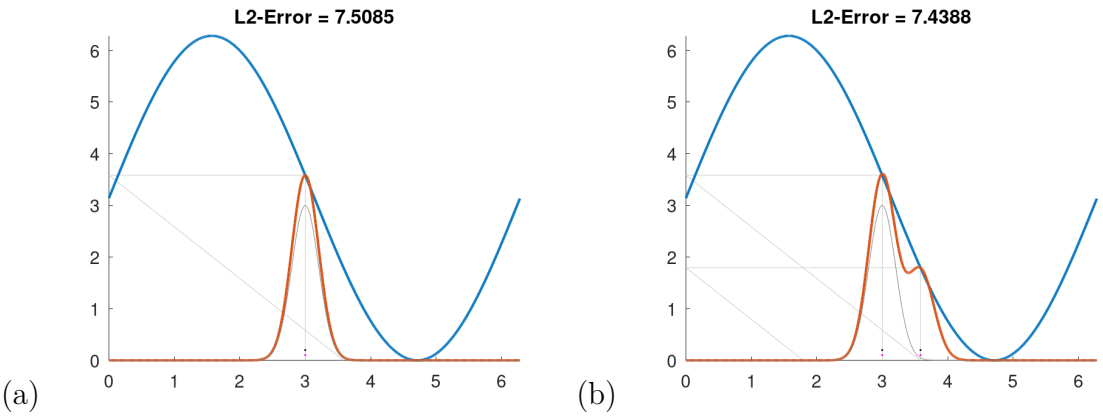
for $k = 1, 2, 3, ...$ Here, for our first example we have chosen

$$M(x) = \pi \cdot \sin(x) + \pi.$$

With a starting value of $x_0 = 3$ and interval $[a, b] = [0, 2\pi]$, we display the dynamics and first two steps of model reconstruction in Figure (4.1). Furthermore, we first show the *natural* approach where a radial basis function around each analysis state is constructed in each learning step.

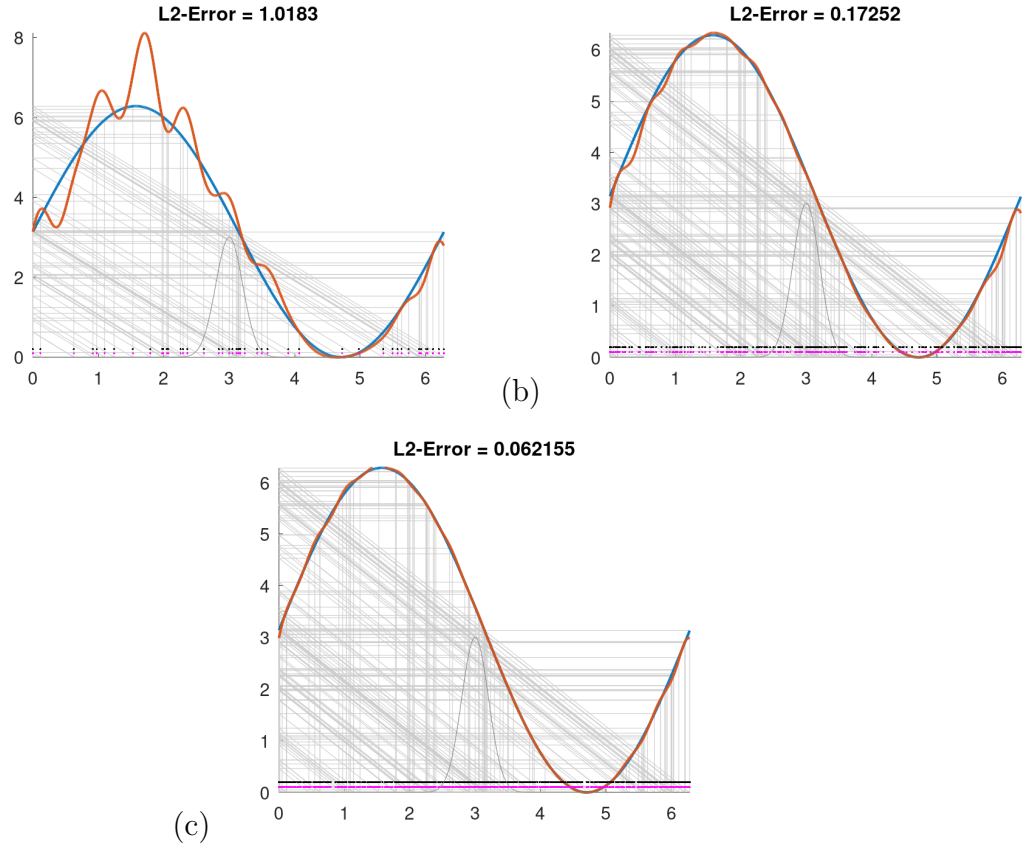
To explain the grey lines in Figure (4.1), starting with the value x_{k-1} we draw a grey line first to the function value $M(x_{k-1})$, which is a vertical line from the point $(x_{k-1}, 0)$ up from the x-axis to the point $(x_{k-1}, M(x_{k-1}))$. Then, the value $x_k = M(x_{k-1})$ is the next value, which can be read from the y-axis. We draw a grey line from this point $(x_{k-1}, M(x_{k-1}))$ to the point $(0, M(x_{k-1}))$ on the y axis, and then to the point $(M(x_{k-1}), 0) = (x_k, 0)$ on the x-axis. This is the third grey line which terminates at the new state x_{k-1} on the x-axis.

Figure (4.2) shows the result after time step $k = 50$ in (a), $k = 300$ in (b) and $k = 1000$ in (c). We observe good convergence of the reconstruction toward an L^2 error of 0.062155. The grid points used ($Ng = 150$) for the approximation is indicated by the black dots. We display the dynamics in grey lines up to the first 100 time steps.



(a) (b)

Figure 4.1: True and approximated model in time step $k = 1$ and $k = 2$, where the blue curve displays the true model, the orange curve the approximative model, the grey lines indicate the model dynamics (describes how the variables or components of the model change and interact with each other as time progresses), and the magenta points are the states of the model. We also display an example of a radial basis function around the initial state as a grey line. The grid points as centers of the RBF functions are shown as black dots, they coincide with the model states $x_{k-1}^{(a)}$ in this example.



(a) (b) (c)

Figure 4.2: True and approximated model in time step $k = 50$, $k = 300$ and $k = 1000$, where the blue curve displays the true model, the orange curve the approximative model, the grey lines indicate the model dynamics and the magenta points the states of the model. We also display an example of a radial basis function around the initial state as a grey line. The grid points as centers of the RBF functions are shown as black dots, they coincide with the model states $x_{k-1}^{(a)}$ in this example.

In a second stage, we demonstrate the outcomes of using a $N_g = 150$ point grid throughout the model-learning operation. Figure (4.3) depicts the findings and errors. As a result, the centres of the radial basis functions no longer correspond with the model trajectories, but are instead represented by the whole collection of $Ng = 150$ black dots seen in the figure. Currently, the approximation of the model has a fixed dimension specified by N_g; during the data assimilation cycle, the coefficients are estimated repeatedly.

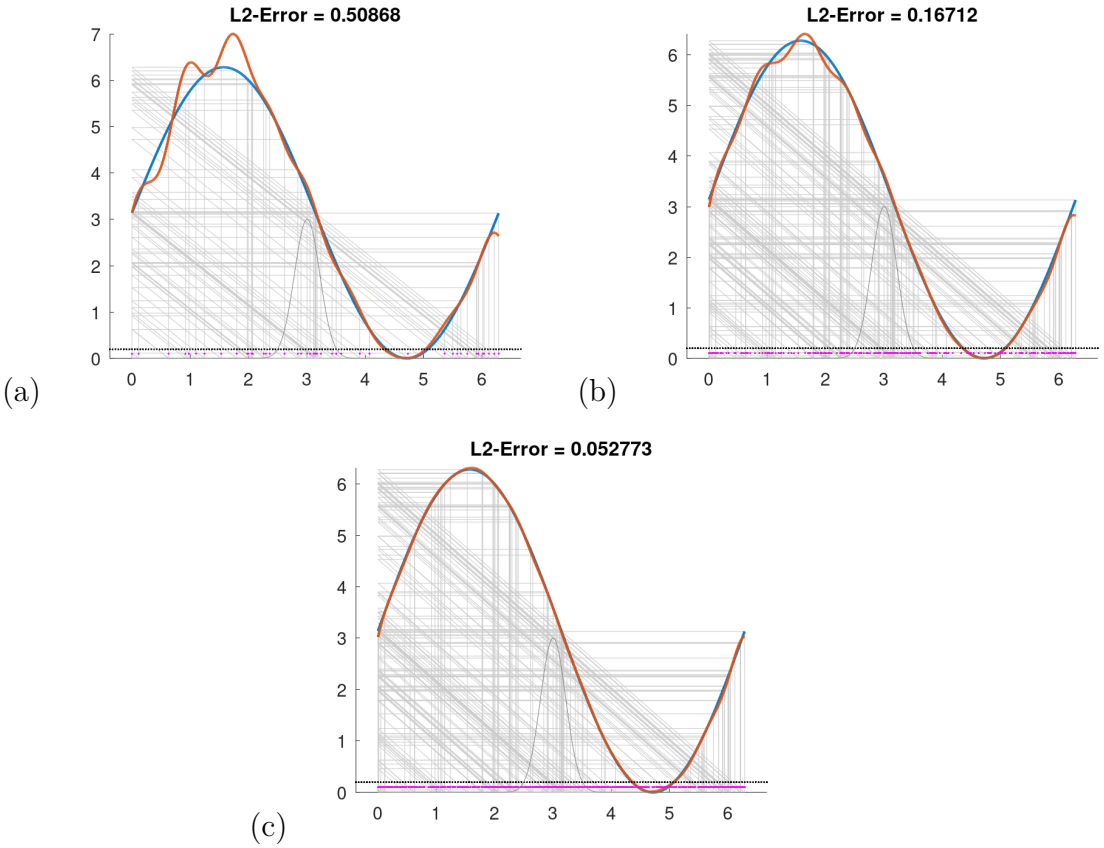


Figure 4.3: True and approximated model in time step $k = 50$, $k = 300$ and $k = 1000$, where the blue curve displays the true model, the orange curve the approximative model, the grey lines indicate the model dynamics and the magenta points the states of the model. We also display an example of a radial basis function around the initial state as a grey line. The grid points as centers of the RBF functions are shown as black dots, they do not coincide with the model states $x_{k-1}^{(a)}$ shown as magenta points in this example.

4.1.2 Variational Model Learning for L63

We now come to the Lorenz 63 model. In figure 4.4 below, we first select some true dynamics and generate observations y_k at points t_k for $k \in \mathbb{N}$. Next, we carry out the Lorenz '63 model reconstruction according to equations (3.15) and (3.16). Now, in the first step, we assume that we have perfect observations. Observations with some error $R = rI$ with $r > 0$ are treated in a second test scenario.

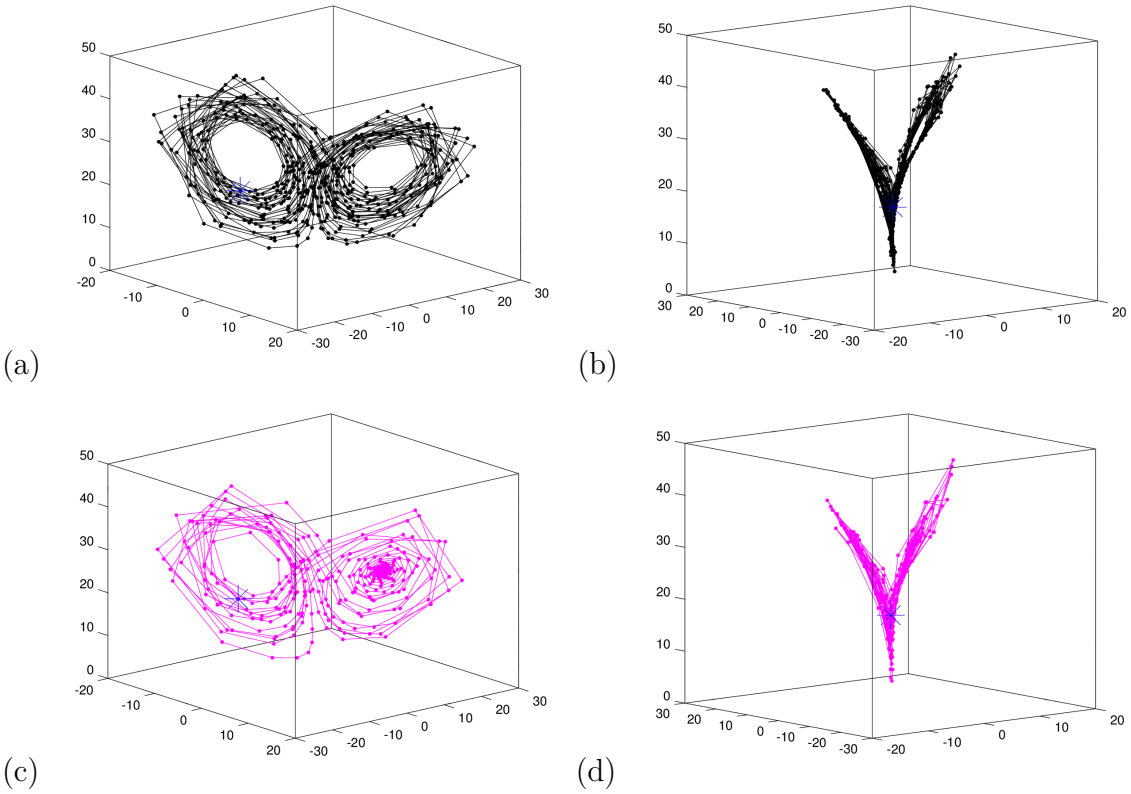
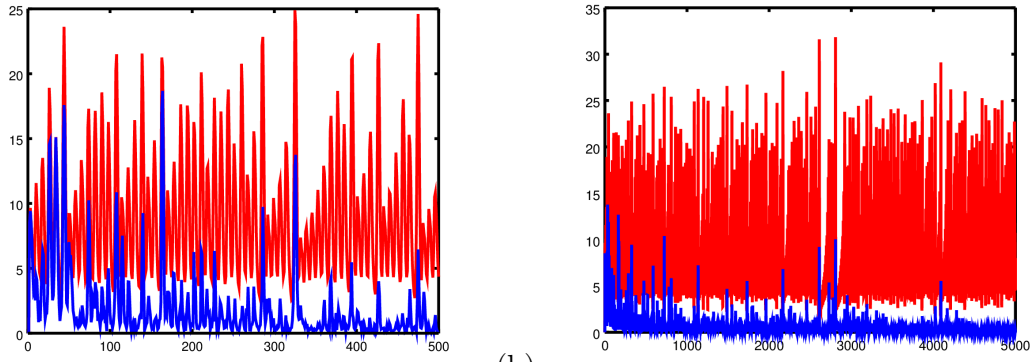


Figure 4.4: (a) and (b) depict the original dynamics from different perspectives, whereas (c) and (d) depict the rebuilt dynamics from the same perspectives, where the model has been reconstructed using equation (3.23). The blue star is the beginning state for the dynamics and the reconstructed trajectory to the rest of the neural patch.



(a) (b)

Figure 4.5: The error evolution of the first guess during the assimilation analysis cycle is shown in blue, in (a) we show the first 500 steps, and in (b) the evolution over an assimilation cycle of 5000 steps. The red curve shows the error of the constant model in each of the steps as a reference. The x-axis shows the number of time steps while the y-axis is the error evolution in both cases.

When we carry out the assimilation, the error of the current model can be estimated by calculating the difference between the measurement y_k and the first guess error shown below in figure 4.5, and given by the norm of

$$\begin{aligned} e_k &:= y_k - M_k^{(b)}(x_{k-1}) \\ &= M^{(true)}(x_k^{(true)}) - M_k^{(b)}(x_{k-1}). \end{aligned} \tag{4.2}$$

for $k \in \mathbb{N}$. This is not only reflecting the *model error* $M_k^{(true)} - M_k^{(b)}$, but it is also the sum

$$\begin{aligned} M^{(true)}(x_{k-1}^{(true)}) - M_k^{(b)}(x_{k-1}) &= M^{(true)}(x_{k-1}^{(true)}) - M_k^{(b)}(x_{k-1}^{(true)}) \\ &\quad + M_k^{(b)}(x_{k-1}^{(true)}) - M_k^{(b)}(x_{k-1}). \end{aligned} \tag{4.3}$$

of the model error on the true state plus the propagation of the error of the current state estimate. However, if the model $M_k^{(b)}$ becomes better, also the state estimate will become better. The norm $||e_k||$ of e_k is a reasonable score to measure the convergence of the model estimation.

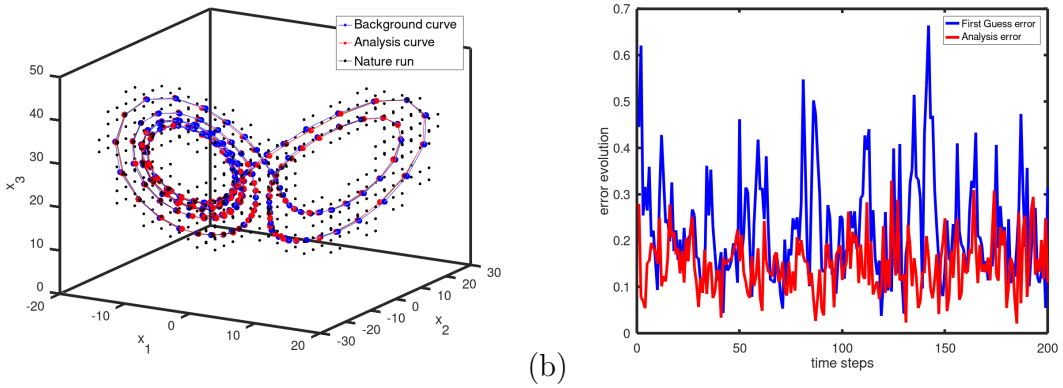
In Figure 4.6 we see the model trajectory, the approximation points are chosen by a selection algorithm around the actual model trajectory and also the first guess and analysis errors when a data assimilation plus model reconstruction algorithm is run. Similarly, figure 4.7 below, show the evolution of

the first guess and analysis error for the model reconstruction approach at 200 time steps.

For comparison, the first guess error with a constant model approximation, i.e., with $M(x) = x$ for all points, is shown in Figure 4.6 (c). Here, the total dimension of the model approximation was $N = 429$. The number N is fundamentally influenced by

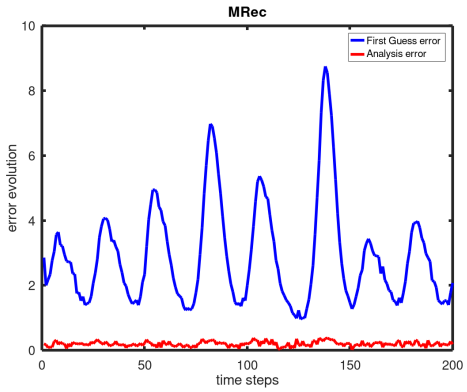
1. the grid size of the underlying regular grid. This grid size will also determine the approximation quality of the model. When the grid size is made smaller, the approximation will become better based on standard approximation results for radial basis function approximation.

2. The model dimension N is influenced by the area the trajectory of the underlying system covers. In the case of Lorenz '63, the trace of the trajectory is profoundly two-dimensional, such that only a part of the full state space is touched by the trajectory and needs to be taken into consideration.



(a) (b)

Figure 4.6: Figure (a) shows the points generated to approximate the dynamical model by radial basis functions chosen from some uniform grid and taking all points where the trajectory passes through in the enclosed cube. Figure (b) shows the first guess approximation achieved by this within 200-time steps.



(c)

Figure 4.7: The error evolution of the first guess error compared with that of the analysis error for the model reconstruction approach within 200 time steps.

4.1.3 Statistical Analysis of the Numerical Experiment Description: L63

In this section, we emphasise and describe the processes required to undertake a statistical analysis of the numerical experiment using the Lorenz '63 model learning approach. We do a sensitivity analysis to demonstrate how variable input changes affect the model's output. After finishing the Lorenz 63 model learning, this was accomplished by modifying the time steps and visualising the accurate and approximation trajectories at each stage.

For the Lorenz 63 system we test the reconstruction of the model dynamics based on the Kalman Filter for the Ansatz of a superposition of exponential functions on a grid. We will conduct experiments where

1. We let the exponential nodes be given by the list of analysis states $x_k^{(a)}$, $k = 1, 2, ...$ growing over time.

2. Limit the nodes of the exponentials to a fixed grid covering the area of the trajectory.

Further, we can learn the coefficients based on

1. A three-dimensional approach with fixed covariance matrix in coefficient space.

2. The Kalman Filter approach with covariance matrix in coefficient space changed in each step of the learning procedure.

3. A four-dimensional approach where the coefficients are calculated based on an interval of points.

By theory, for a linear observation operator the Kalman Filter is equivalent to the 4D approach [103], [56]. We have tested this on a limited interval.

Sensitivity Analysis: L63 Model Reconstruction

A sensitivity analysis is carried out for the following approaches below:

1. Testing the result of the model reconstruction for different parameter settings, displaying the resulting model trajectories in comparison for selected cases.

2. Testing a range of parameters and studying the histogram of resulting model errors.

For the model errors we can employ various metrics.

1. First guess error in the analysis cycle where the approximate model is growing step by step based on the model learning procedure.

2. Analysing the forecasting error for some fixed lead time.

3. Testing the length of a model trajectory which stays within some error bounds around the true model trajectory.

Sensitivity Experiment #1. We carried out experiments with different number of time step `Nnat`, ranging from 400 to 1150. Depending on the number of time steps, the original trajectory will cover more or less area in state space. This leads to a varying number of base points for the model approximation. When the approximate trajectory leaves the area where the model training worked well, it can happen that the increments tend to zero and the approximate trajectory is stuck somewhere. In the cases where the approximate model trajectory stays within the area where approximations work sufficiently well, the approximate trajectory follows the original trajectory oscillating between the two wings of the butterfly.

Choices of parameters for this experiment are:

```
x0 = [0;-12;21]    % initial point for trajectory
Nnat    = 300;     % steps for nature run
x       = x0;      % initial state for iteration
noise   = 0.00001;  % noise factor on measurements
dtime   = 0.03;    % time interval between measurements
sigma0  = 10;      % standard parameter in the Lorenz system
rho0    = 28;      % ~
beta0   = 8/3;     % ~
```

The result of these runs is reflected in Figure 4.8. We show the model learning outcome by carrying out a free run starting with x_0 based on a varying number of time steps of the nature run and corresponding training period. The results compares the true and approximate trajectory after learning the Lorenz '63 model. Even if the exact trajectory is met only for 200 or 300 time steps, in most tests there is a clear correlation in the patterns noticeable between the trajectories. In (f) and (g), though, the free run started tobe stuck in a local fixed point after about 100 time steps.

Sensitivity Experiment #2. We tested the whole model reconstruction with different seeds for the random error generator of the observation errors. Learning was based on a trajectory starting with some state x_0 given above.

The result of these runs is reflected in the histogram below in Figure 4.9 showing the distribution of the first guess errors over the full model trajectory. The experiments have been carried out with noise at $\epsilon = 10^{-5}$. The standard deviation of the distribution is quite small with values of 0.0002.

Sensitivity Experiment #3. We tested the whole model reconstruction with different initial conditions for the trajectory employed for training. We show a histogram of the first guess errors over the full trajectory during model learning.

The result of these runs is reflected in the histogram shown below in Figure 4.10, which shows the distribution of the first guess errors over the whole model trajectory when the initial point for the training trajectory was changed. In this case, we notice a negative or left-skewed distributed error. It suggests a clear indication of extremely low values in the standard deviation of the noise clustered towards the left.

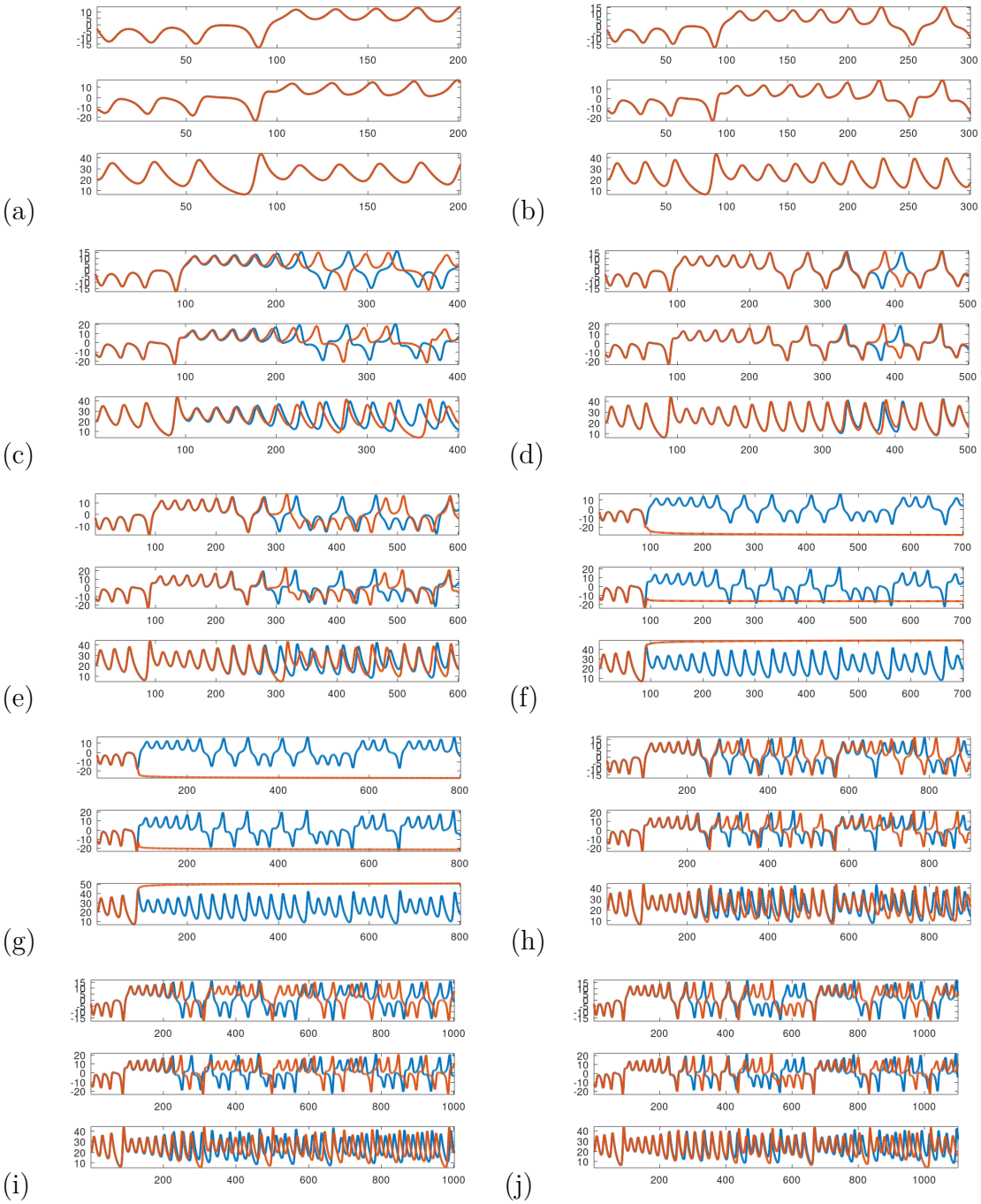


Figure 4.8: Experimenting with model learning with `Nnat`=200, 300, ..., 1100 steps. We display a visualization of the true and the approximate trajectory both run freely for `Nnat` steps after completing the learning for the Lorenz 63 model, starting with the original point $x0$.

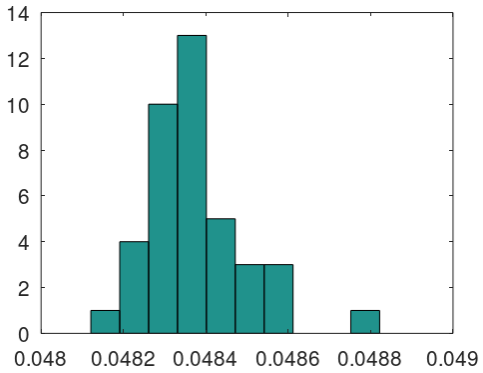


Figure 4.9: A histogram of the first guess errors over the full trajectory during model learning when the seed of the observation error random number generator was changed. The standard deviation of the noise was set to $\epsilon = 10^{-5}$.

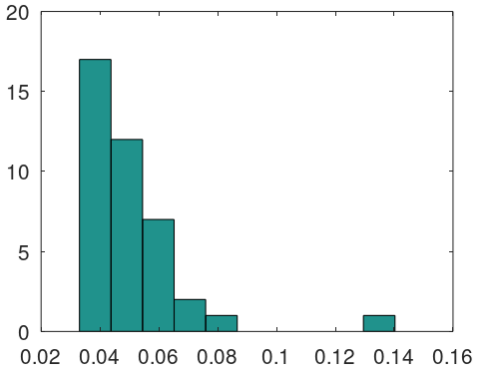


Figure 4.10: A histogram of the first guess errors over the full trajectory during model learning when the initial point for the training trajectory was changed. The standard deviation of the changes to x_0 in the form $x_0 = x_{00} + \sigma * \text{randn}(3, 1)$ was set to $\sigma = 0.2$.

4.2 Learning higher dimensional Lorenz 96

This section displays the results of the learning obtained through the application of Lorenz 96. This was applied in a higher dimensional environment to show its feasibility as a dimension reduction approach.

We follow through with the simulation of the Lorenz 96 model as described in Section 2.2, in particular equation (2.1). To set up our learning framework, we model the forcing term F at each node $j \in \{1, .., n\}$ as a polynomial function of the variables in an environment $j - dj, ..., j + dj$ with $dj \in \mathbb{N}$. The polynomial ansatz for the forcing term as given by (3.40) provides a framework for learning the L96 model. We can explicitly compare the 'true' coefficients c_ξ as given by the explicit $F(x)$ for $L96$ with the reconstructed coefficients c_ξ.

The coefficients c_ξ are successively reconstructed based on either a three-dimensional approach as described in Section 3.1 or the Kalman filter of Section 3.2 when the coefficient covariance matrix is updated in each step of the iteration.

Figure 4.11 displays the results of a nature run where we carry out a classical data assimilation cycle for L96. We then carry out the same data assimilation cycle without the knowledge of the L96 system, but with the model reconstruction for the coefficients c_ξ of the ODE system.

The iterative results of the coefficient reconstructions are shown in Figure 4.12 (d) over 40 steps, where we observe that most coefficients are readily reconstructed after about 10 steps. The first guess error of the model reconstruction is shown in Figure 4.12 (a) and (b). At the end of the 40 steps the original and reconstructed coefficients c_ξ are shown in Figure 4.12 (c). The algorithm was able to recover the coefficients of the system of ODEs in a quite reasonable way.

A targeted selection of the reconstruction code carrying out the reconstruction of the coefficients based on the observation operator H_c is shown in line 6 of the Matlab code No. 1 in Part B in Appendix 7.2.3 below. In particular,

- The operator H_c is defined in code example No. 3 of Section 7.2.3.
- The Kalman filter based learning of the coefficients is carried out on line 16 of the Matlab code No. 2 of Section 7.2.3 with an update of the model B_c matrix on line No. 17 of the Matlab code below.

Here we employed the Localized Ensemble Transform Kalman Filter (LETKF) following [56] for the data assimilation parts of the Lorenz 96 model. In this study, we have used the LETKF for our Lorenz '96 system as a data assimilation (DA) method, which is a well-known approach for incorporating observations into numerical models, and can be used with the Lorenz '96 system to estimate the system's state. In this case, we have used it for the reconstructing the model's state instead by using observations included in an ensemble to compare model reconstructed or predicted to true observations.

The numerical example from the following codes by Potthast and Schenk[1]. The basic cycling code is shown in Section 7.2.4, code No. 1.

[1]Taken from the Data Assimilation Coding Environment DACE of Deutscher Wetterdienst (DWD), specifically, the explorative coding parts `dace_play/octave`.

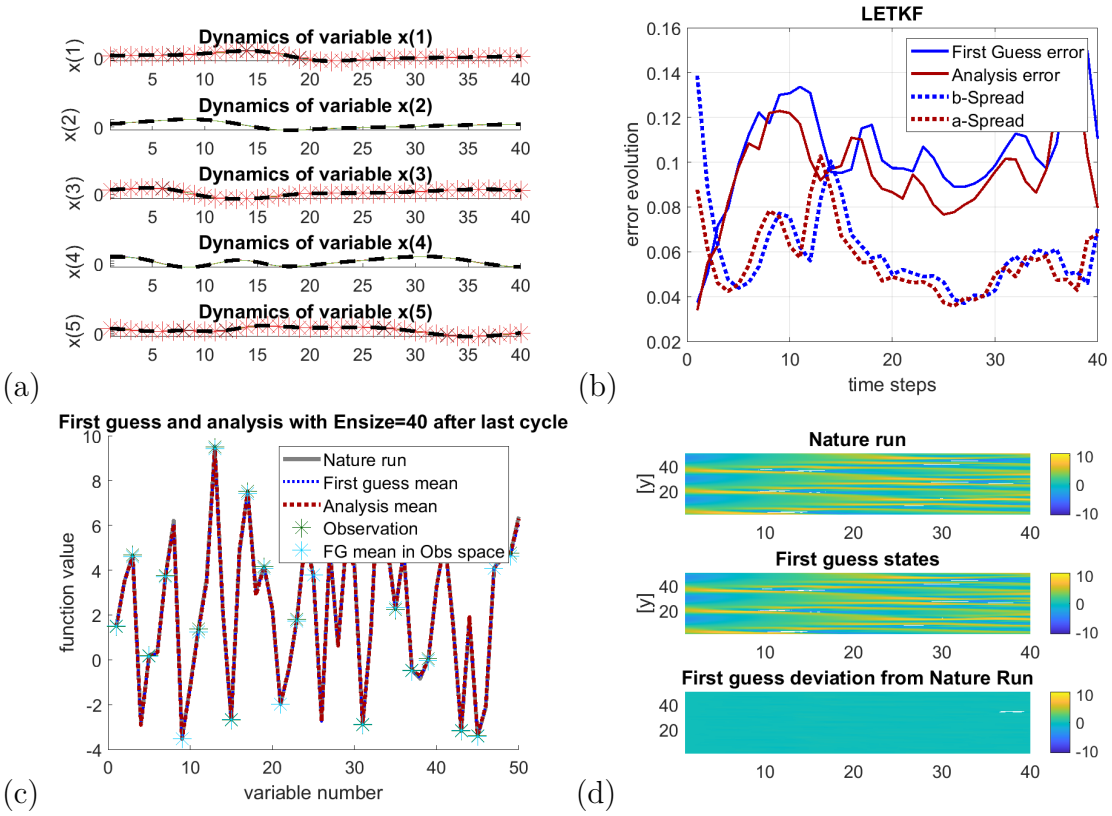


Figure 4.11: Results of the dynamics of some variables used in the simulation of the Lorenz '96 model in (a), (b) Displays Local Ensemble Transform Kalman Filter (LETKF) plot of the first guess and analysis evolution errors and the ensemble spread of the background and analysis errors (c) The nature run, first guess and analysis mean for each variable at the end of the last cycle is displayed with the first guess mean in observation space while (d) shows the nature run, the first guess states and the first guess deviation from nature run.

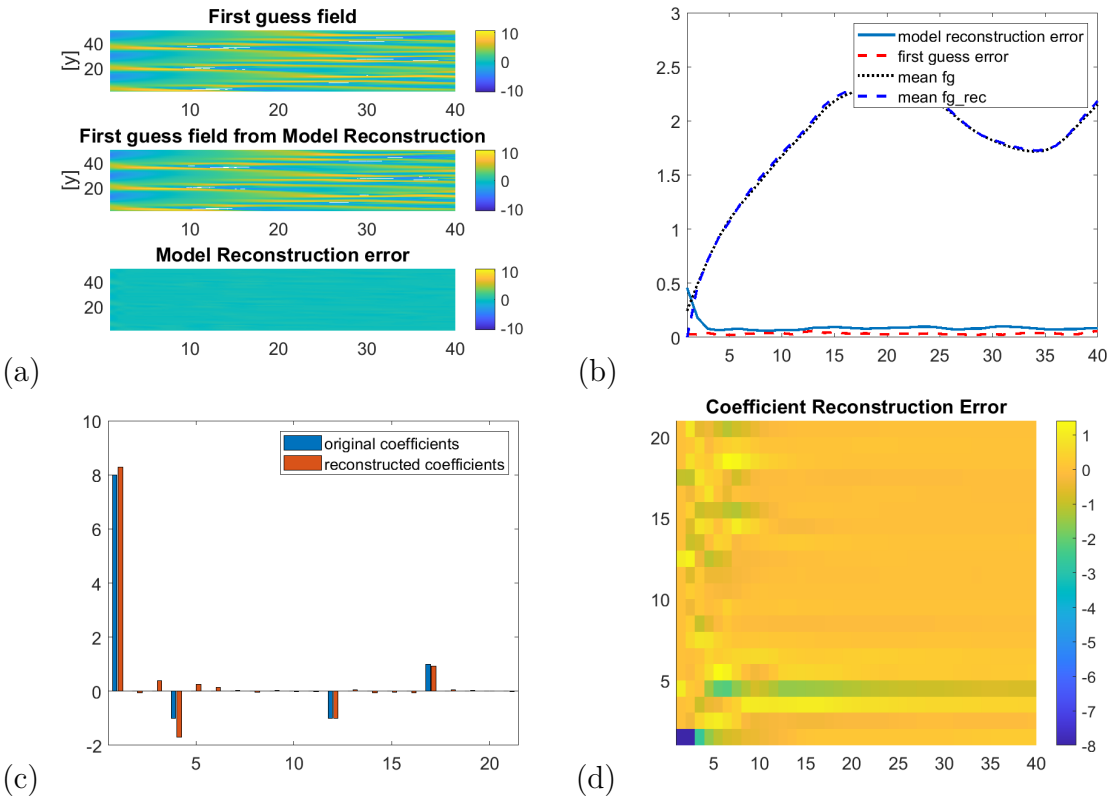


Figure 4.12: Further results of the simulation of the Lorenz '96 showing the first guess field from the original and model reconstruction and its errors in (a) Line plots of the model reconstruction and first guess errors and their corresponding means of the first guess and reconstructed model respectively in (b) and (c) shows the coefficients of the original and reconstructed model while (d) displays the coefficient reconstruction error.

4.2.1 Statistical Analysis of the Numerical Experiment Description: L96

This section shows the methods used to analyse the numerical experiment and their corresponding explanations for the Lorenz '96 model reconstruction approach. We conduct a sensitivity analysis to demonstrate the effect of variable input modifications on the model's output. We display the results of the learning process for the Lorenz 96 model, this was accomplished by modifying the time steps and, afterwards, visualising the true and approximated trajectories at each stage.

We carry out a run of the Lorenz '96 system with parameters described as follows:

```
Nnat = 40;      % number of cycles
N     = 50;      % model state dimension 50
J     = 2;       % neighbors needed for Lorenz 96
F1    = 8;       % forcing term for the true state
% Parameters for the numerical solution of the model
dtime = 0.05;    % steps of integration
h     = 0.05;
% Parameters for generating observations and in
noise = 0.02;   % noise factor on measurements  % standard deviation of observati
hH    = 2;      % every hH-th variable is observed
rho   = 4*pi/N; % localization radius with respect to differences of nodes
```

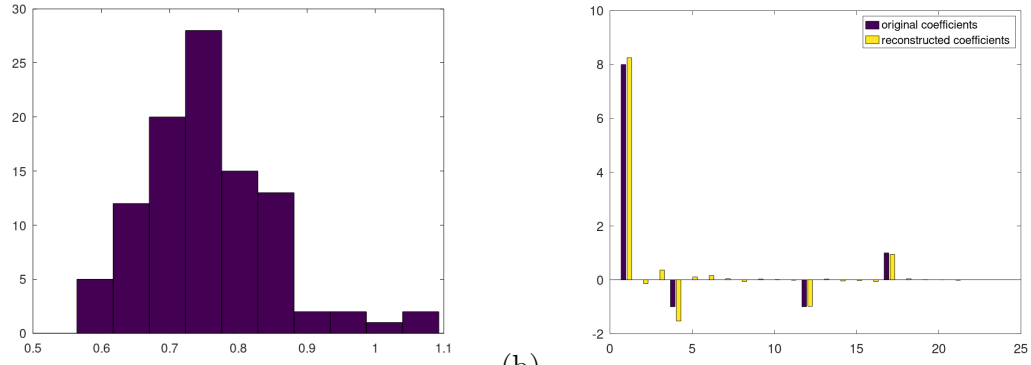
with some initial state `x0`. Then, we run the model reconstruction scheme reconstructing the coefficients in the forcing term.

Sensitivity Analysis: L96 Model Reconstruction

Sensitivity Experiment #1. With different initial states of L96 defined by

```
x0 = 5*sin(2*pi*3*(1:N)'/N) + 1*randn(N,1);
```

The dimension of the local coefficients of Lorenz '96 for reconstruction has been chosen to be $n_c = 21$.



(a) (b)

Figure 4.13: Figure (a) shows the distribution of the reconstruction error for $N = 100$ model reconstruction runs with x_0 taken from the random distribution described above. The coefficients and their reconstruction for the last sample are shown in Figure (b).

4.3 Neural Field Model Learning the Kernel

Here, we develop the model reconstruction techniques described in 3.4, i.e., the four-dimensional neural kernel reconstruction and the neural Kalman filter in Section 3.4.2 by applying it to the reconstruction of the dynamics of a *neural field* as described by the Amari neural field equation in equation 2.4

$$\tau \dot{u}(p,t) = -u(p,t) + \int_D w(p,q) f(u(q,t))\, dq, \tag{4.4}$$

for $p \in D$ with some domain $D \in \mathbb{R}^2$ and $t \in [0,T]$ with $T > 0$.

With the definition of Ψ and Φ given by (3.74) and (3.75), our observations of the model in each step are given by

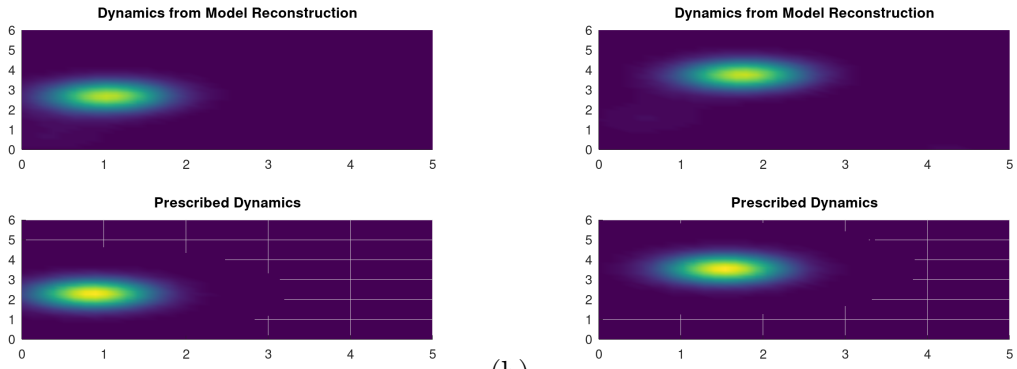
$$r = \Psi_k. \tag{4.5}$$

We define our set of parameters to reconstruct to be

$$c_\ell^{(b)} := W_{j\xi}, \quad \ell = n(j-1) + \xi \tag{4.6}$$

for $j, \xi = 1, ..., n$. In this case, the Kalman filter equations (3.34) - (3.35) are based on the operator $H_c = H_c^{(k)}$ defined by

$$(H_c^{(k)})_{j,n*(j-1)+\xi} = (\Phi_k^T)_\xi, \quad j, \xi = 1, ..., , n, \tag{4.7}$$



(a) (b)

Figure 4.14: We show two snapshots from the prescribed (bottom), and reconstructed dynamics based on the RBF nonlinear model reconstruction technique. The images show the times $t = 5$ for (a) and $t = 10$ for (b) with a simulation time-step of 0.9, where the input was given with times steps of size 1. The approximate dynamics can generate the movement of the pulse, though with a slight phase error.

and zero otherwise, such that we have

$$\begin{aligned} (H_c^{(k)} c^{(b)})_j &= r_j && (4.8)\\ &= (\Phi_k^T W^T)_j && (4.9)\\ &= \sum_\xi \Phi_{k,\xi} W_{j\xi} && (4.10) \end{aligned}$$

for $j = 1, ..., n$. This Kalman approach is, at the end of any given time window $[0, T]$, equivalent to the full reconstruction for the window $[0, T]$. Results are shown in Figure 4.16 as "Neuro Reconstruction".

The advantage of the Kalman approach for reconstructing W is that it iteratively solves an equation and does not need to take into account all time steps in one step. The disadvantage here is that the number of elements to reconstruct is n, and the covariance matrix B_c is of dimension n^2, and it needs an update of this matrix in each step of the algorithm.

Reducing the dimension of the model parameter space is an option to provide more efficient and stable reconstructions. Here, we might take the Ansatz to represent W as a superposition of Gaussian basis functions, i.e. in

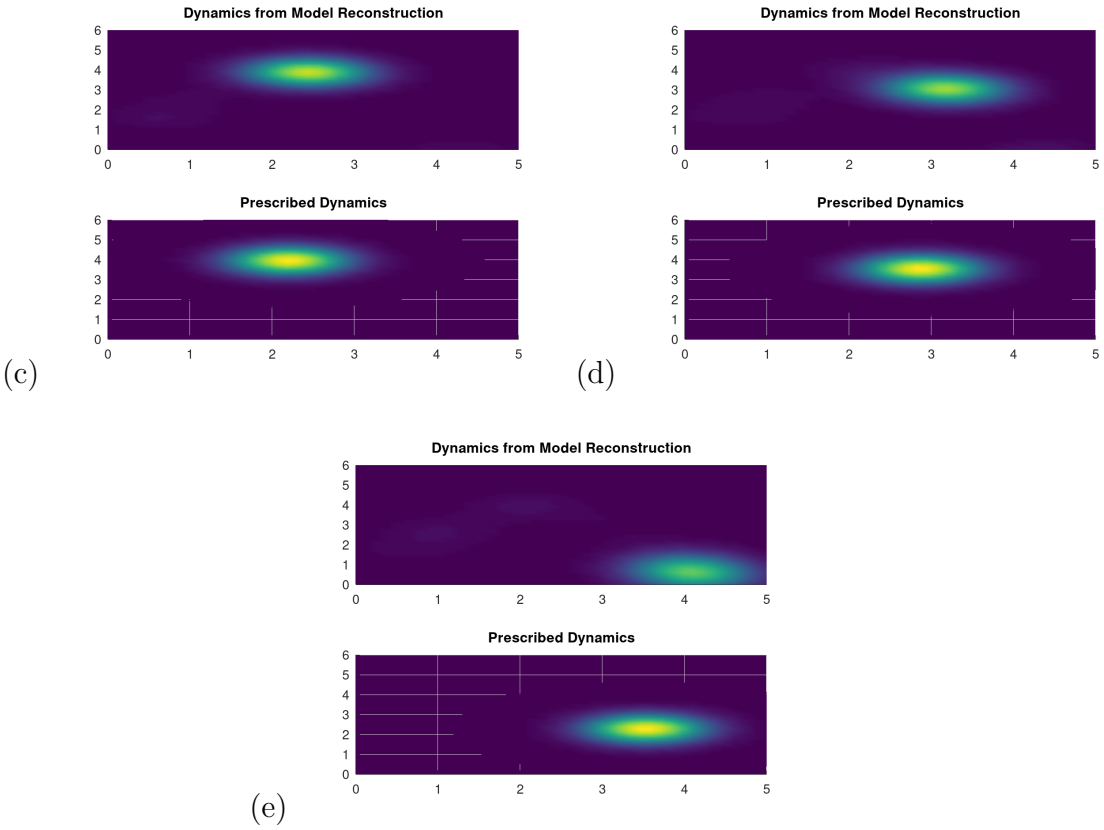


Figure 4.15: We show two snapshots from the prescribed (bottom), and reconstructed model (top) dynamics based on the RBF nonlinear model reconstruction technique. The images show the times $t = 15$ for (c) and $t = 20$ for (d) and $t = 25$ for (e) with a simulation time-step of 0.5, where the input was given with times steps of size 1. We observe a growing phase shift and a growing error in the excitation strength when we move towards smaller time steps for the simulation.

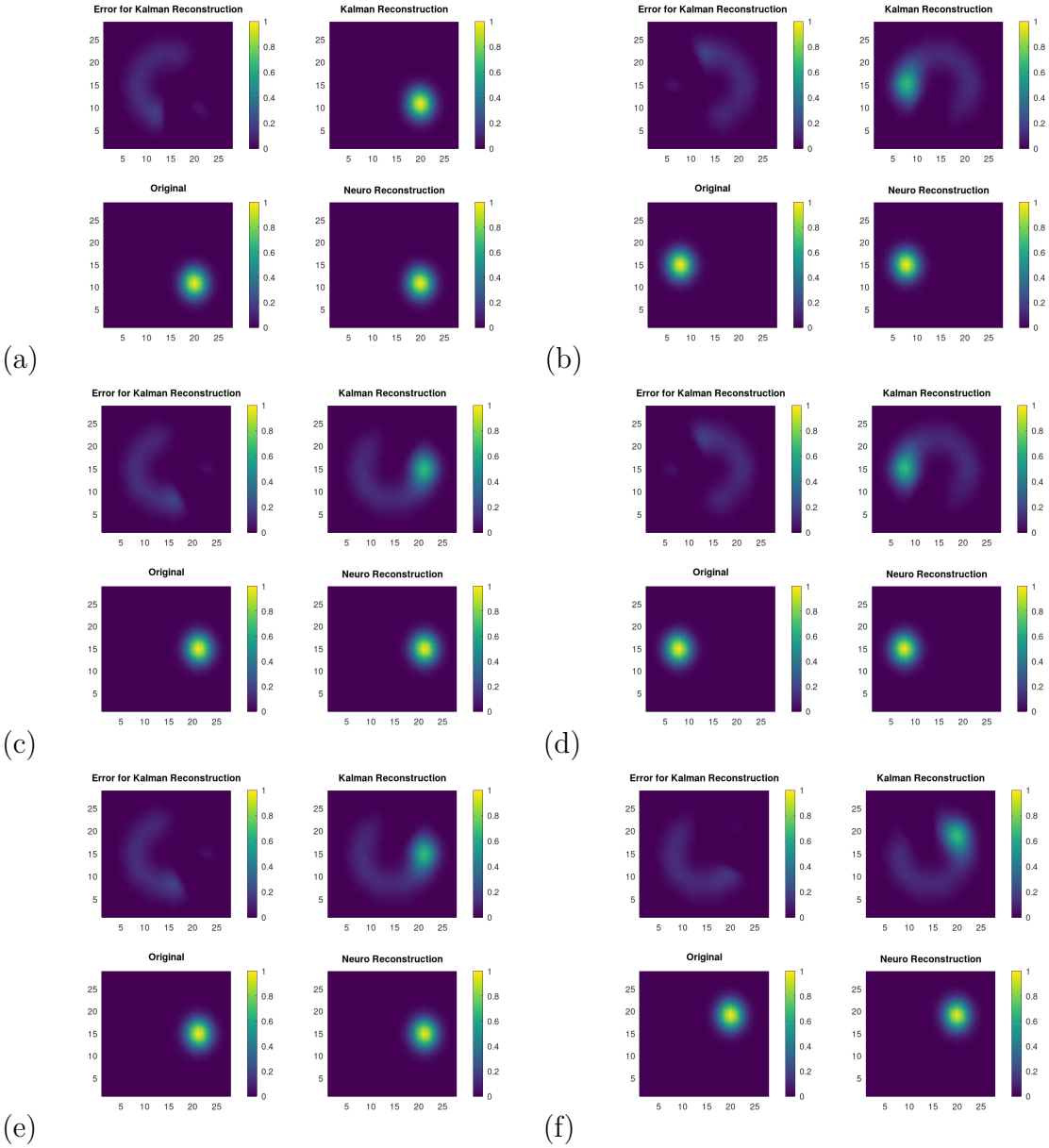


Figure 4.16: In Figure 4.16 (a)-(e) above, We show the snapshots of the different phases in the simulations of the original dynamics (bottom left) at different time steps in comparison with the Kalman and Kalman error reconstructed dynamics alongside the neuro reconstruction approach using the kalman filter kernel estimation technique.

the form

$$W = \sum_{\eta=1}^{N_c} c_\eta G_1^{(\eta)} (G_2^{(\eta)})^T \tag{4.11}$$

where for $\ell = 1, 2$ we employ

$$G_{\ell,\xi}^{(\eta)} \quad = \quad \exp(-\sigma_\ell \| p_\xi - p^{(\eta,\ell)} \|^2), \quad \xi = 1, ..., n \tag{4.12}$$

with constants σ_ℓ with some set of points

$$M = \left\{ p^{(\eta,\ell)}, \quad \eta = 1, ..., N_c, \ \ell = 1, 2 \right\}. \tag{4.13}$$

We note

$$G_\ell^{(\eta)} \in \mathbb{R}^n. \tag{4.14}$$

If σ_ℓ is sufficiently large, and the set of points covers the domain D, the ansatz (4.11) will be an approximation to the full matrix W. Equation (4.10) can now be written as

$$\begin{aligned}
(H_c^{(k)} c^{(b)})_j &= r_j && (4.15)\\
&= (\Phi_k^T W^T)_j && (4.16)\\
&= \sum_\xi \Phi_{k,\xi} W_{j\xi} && (4.17)\\
&= \sum_\xi \sum_\eta \sum_\rho \Phi_{k,\xi} c_\eta (G_1^{(\eta)})_{j\rho} (G_2^{(\eta)})_{\xi\rho} && (4.18)\\
&= \sum_\eta (G_1^{(\eta)} (G_s^{(\eta)})^T \Phi_k)_j c_\eta && (4.19)
\end{aligned}$$

for $j = 1, ..., n$. This leads to

$$H_c^{(k)} = \left(G_1^{(\eta)} (G_s^{(\eta)})^T \Phi_k \right)_{\eta=1,...,Nc}. \tag{4.20}$$

Results for model reconstruction or model learning, respectively, with the above RBF ansatz, are shown in Figure 4.16 as "Kalman Reconstruction". Here, we chose a selection of points $p^{(\eta,\ell)}$ along the path of the neural pulse propagation. Practically, such a choice could be made by an algorithm testing activity in space and putting the basis function into the area of activity for a given time slice.

For the kernel reconstruction by the Kalman filter with full matrix W, we obtain identical reconstruction kernels W_{approx} at the end of the time window

T to the original four-dimensional method. Results are shown in Figures 4.14 and 4.15.

In Figure 4.14 below, we show the original states and the states based on the reconstructed kernel at times $t = 5$ and $t = 10$. Time steps $t = 15, 20$ and $t = 25$ are shown in Figure 4.15. For all cases, we obtain reasonable approximations of the prescribed dynamical evolution of the travelling pulse, though with smaller time steps here we observed a growing phase shift, i.e. the pulse speed increased.

The second example with the Kalman is shown in Figure 4.16, where this time, the pulse is rotating around the centre of the neural tissue. Here, the strongly reduced dimensionality of the RBF approximation still yields reasonable dynamical reconstruction, but a larger error than the much higher dimensional full W reconstruction.

We remark that with the discretization of $n_1 = 28$ and $n_2 = 29$ points for the domain D, we have a dimension $n = 812$ of the neural states u at each point in time $t \in [0, T]$. The dimension of W is then $n^2 = 659344$. In contrast, for the RBF Ansatz, we only needed a dimension $n_c = 58$ of $c^{(b)}$, putting radial basis functions along the path of the neural pulse.

A selection of the core code for reconstructing the coefficients with a Gaussian RBF Ansatz for the neural field equation is shown in Section 7.3. In particular

- The reconstruction based on the Kalman filter with Gaussian basis functions as displayed in Figure 4.16 as "Kalman Reconstruction" is shown in code No. 1 of Section 7.3.
- The original neural reconstruction as displayed in Figure 4.16 as "Neuro Reconstruction" is shown in code No. 2 of Section 7.3.

4.4 Applications to Reaction-Diffusion System

This section displays the results in the framework of the reaction-diffusion system atmospheric model described in 2.4, and 2.4.1 above.

We start our reconstruction with the discretized version (2.12) of equation (2.6), i.e. with

$$u_{k+1,j} = \sum_{\xi} c_{j\xi} u_{k,\xi}, \quad j = 1, ..., n \tag{4.21}$$

with time index $k = 1, 2, 3, ...$ and spatial index $j = 1, ..., n$. Given the field values $u_{k+1,j}$ for a point p_j with index j and values $u_{k,\xi}$ located at points p_ξ in a neighbourhood of p_j. The model learning can take place locally and independently of each other for each j. This also means that we can work independently with the coefficient covariance matrices $B_c^{(j)}$.

Let M_j be the set of neighbourhood indices for p_j. Then we solve a family of equations

$$u_{k+1,j} = \sum_{\xi \in M_j} c_{j\xi} u_{k,\xi}, \quad j = 1, ..., n. \tag{4.22}$$

The operator H_c is given by the collection of field values $u_{k,\xi}$, such that

$$H_c^{(j)} c_j = \sum_{\xi \in M_j} c_{j\xi} u_{k,\xi}, \tag{4.23}$$

with a column vector of coefficients

$$c_j := \Big(c_{j,\xi} \Big)_{\xi \in M_j}, \tag{4.24}$$

which means that with the correct ordering $H_c^{(j)}$ consists of the values of $u_{k,\xi}$ in the neighbourhood of p_j.

We carried out a simulation of the partial differential equation (2.6) with the initial condition (2.7). We then applied the Kalman learning (3.34) - (3.35) to the sequence of values $u_{k,j}$ for time steps t_k, $k = 1, ..., n_t$ and for each point $j = 1, ..., n$. For testing the result of the model reconstruction, we then simulated the reconstructed model, with the identical setup including the filtering (2.10).

An example, where we start with $u(\cdot, t_k)$ at time t_k for $k = 100$ is displayed in Figure 4.17. The models show a good coincidence of the original and reconstructed field dynamics for about 150-time steps, $k = 100$ to $k = 250$.

The core code for reconstructing the coefficients of a reaction-diffusion partial differential equation is shown in Section 7.4. The update of the coefficient is done on line 41 of the code, and the update of the B_c matrix is in lines 42-46.

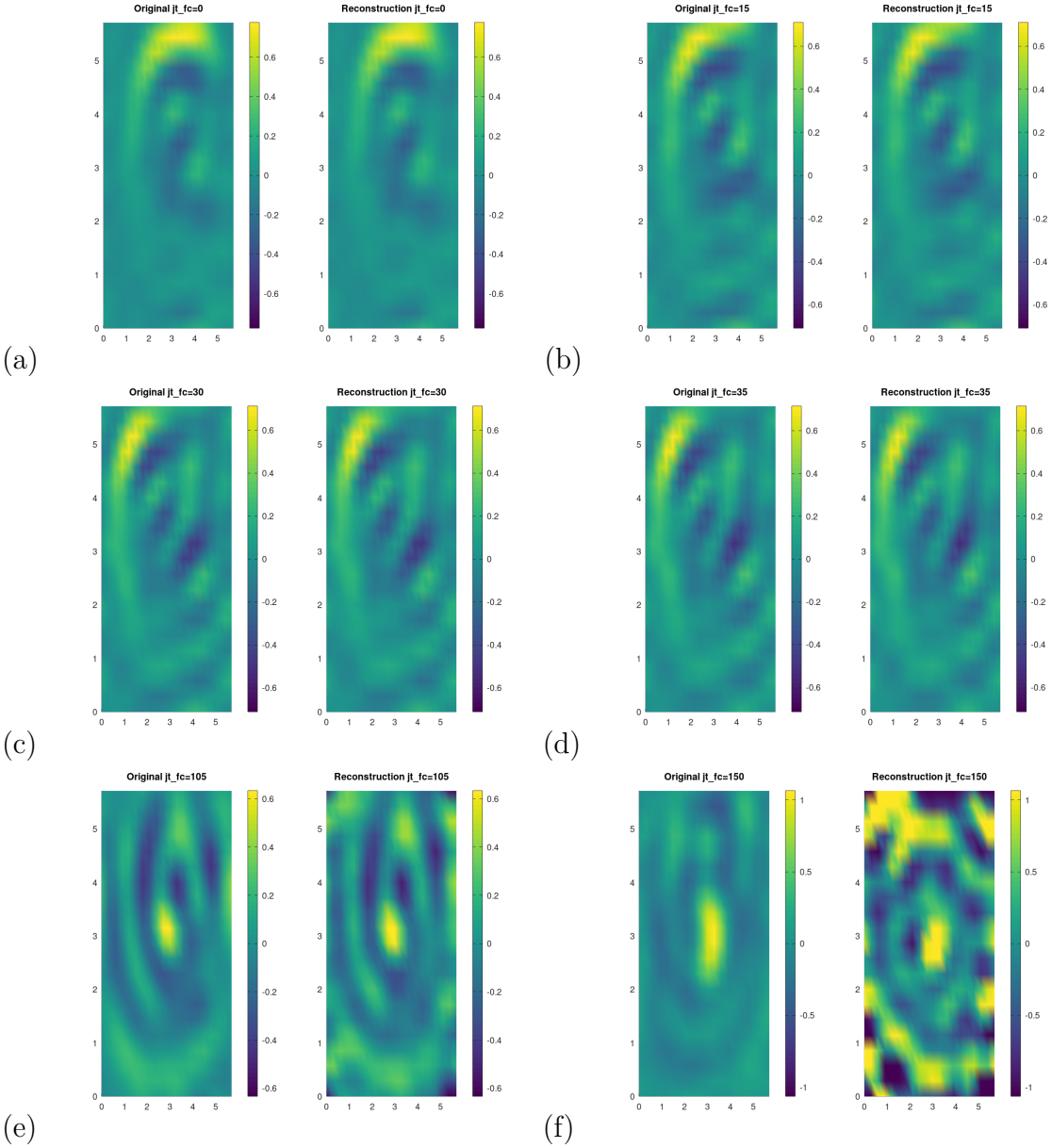


Figure 4.17: Results of the iterations at different point coordinates and estimated time steps with index $k = 100, 115, 135, 205, 250$.

4.4.1 Sensitivity Analysis of the Numerical Experiment Description: PDE

This section presents the findings of the numerical experiment for the reaction-diffusion system weather equation shown in the preceding section(4.4). We demonstrate that it is possible to learn the integral partial differential equation of the known model when utilised as a propagator. As a learning space, we employ a basic two-dimensional reaction-diffusion model for the model reconstruction method given below.

We carry out a run of the simulation with the reaction-diffusion system with parameters described as follows:

```
% two-dimensional domain given by [0 a1]x[0 a2]
xv1 = [0:h1:a1-h1];
xv2 = [0:h2:a2-h2];
% calculate points
[X1,X2] = meshgrid(xv1,xv2);
X1v = reshape(X1,n,1);
X2v = reshape(X2,n,1);
u0 = exp(-2*((X1v-4.5).^2 + (X2v-2).^2));  % initial field
```

with some initial state `u0`. Then, we run the model reconstruction scheme reconstructing the coefficients in the forcing term.

Sensitivity Analysis dependent on Gradient Direction

Sensitivity Experiment #1. With different directions of the gradient we test the reconstruction with different values of $p \in \mathcal{R}^2, \|p\| = 1$ and $c = 1^{-12}$.

$$\frac{du}{dt} = p \cdot \nabla u - c\triangle u = p_1 \frac{\partial u}{\partial x_1} + p_2 \frac{\partial u}{\partial x_2} - c(\frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2}). \tag{4.25}$$

```
a = pi/3;     % angle for calculating p; parameter choices for the PDE
ptmp = [cos(a)*ones(n1*n2,1) sin(a)*ones(n1*n2,1)];
p = ptmp./sqrt(ptmp(:,1).^2+ptmp(:,2).^2);
c = 1e-12;
```

See Figures 4.18 - 4.20 for three different choices and the corresponding comparision of reconstructions with the original trajectory.

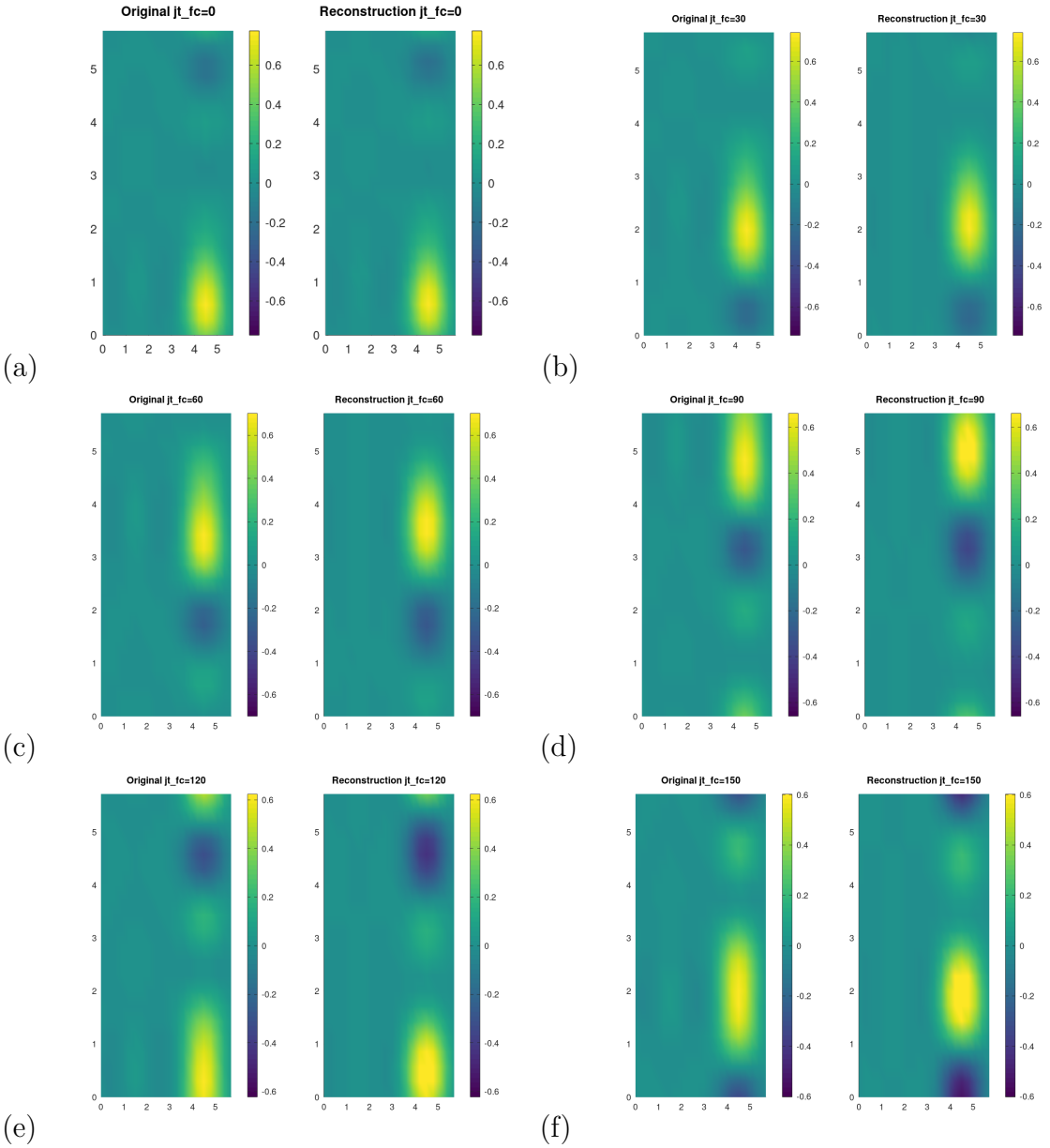


Figure 4.18: The Figures (a)-(f) show different times steps comparing the original dynamics with the field generated by the learned differential equation, where we display iterations $0, 30, ..., 150$, all for $p = [0; 1]$.

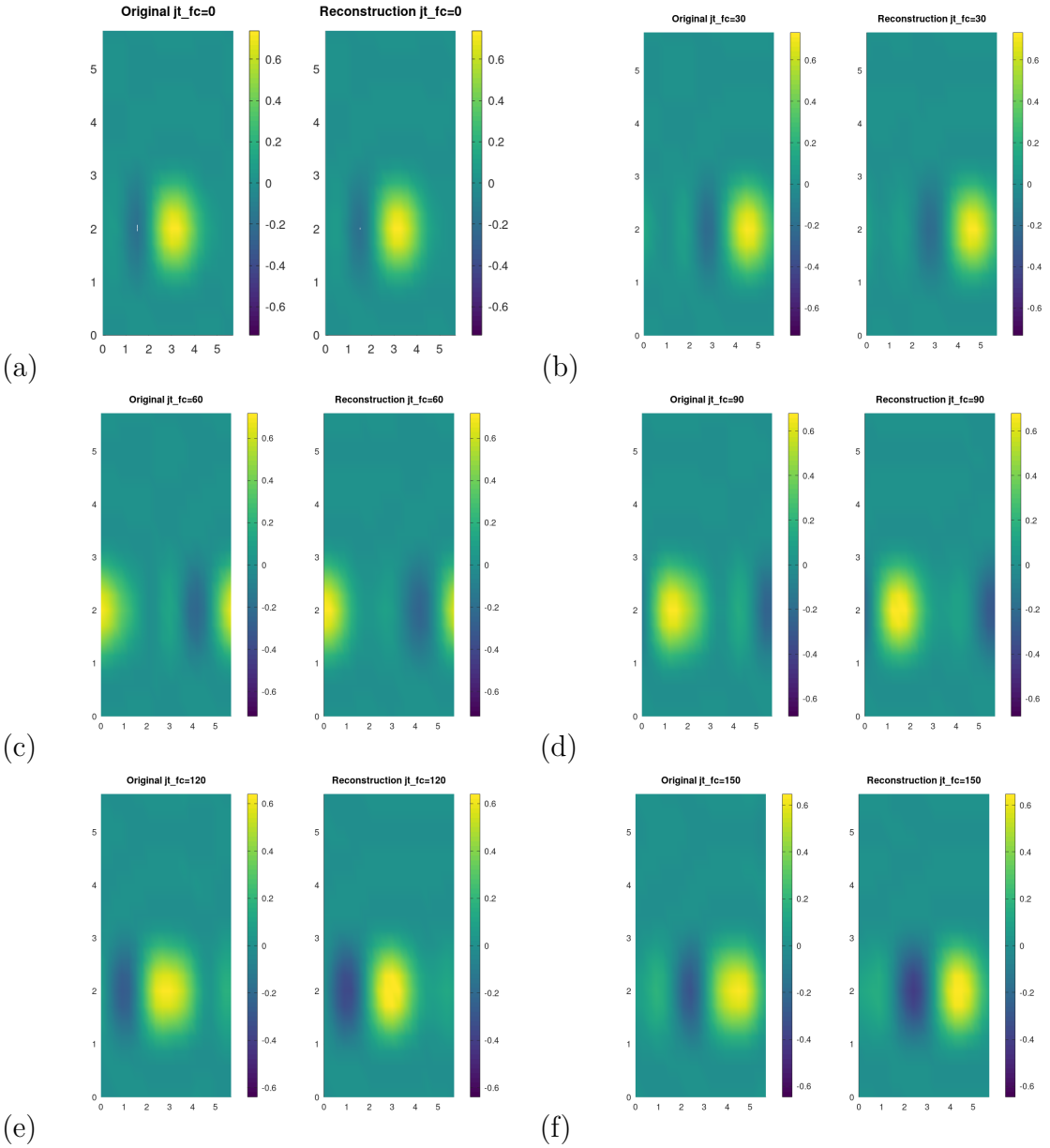


Figure 4.19: The Figures (a)-(f) show different times steps comparing the original dynamics with the field generated by the learned differential equation, where we display iterations $0, 30, ..., 150$, all for $p = [1; 0]$.

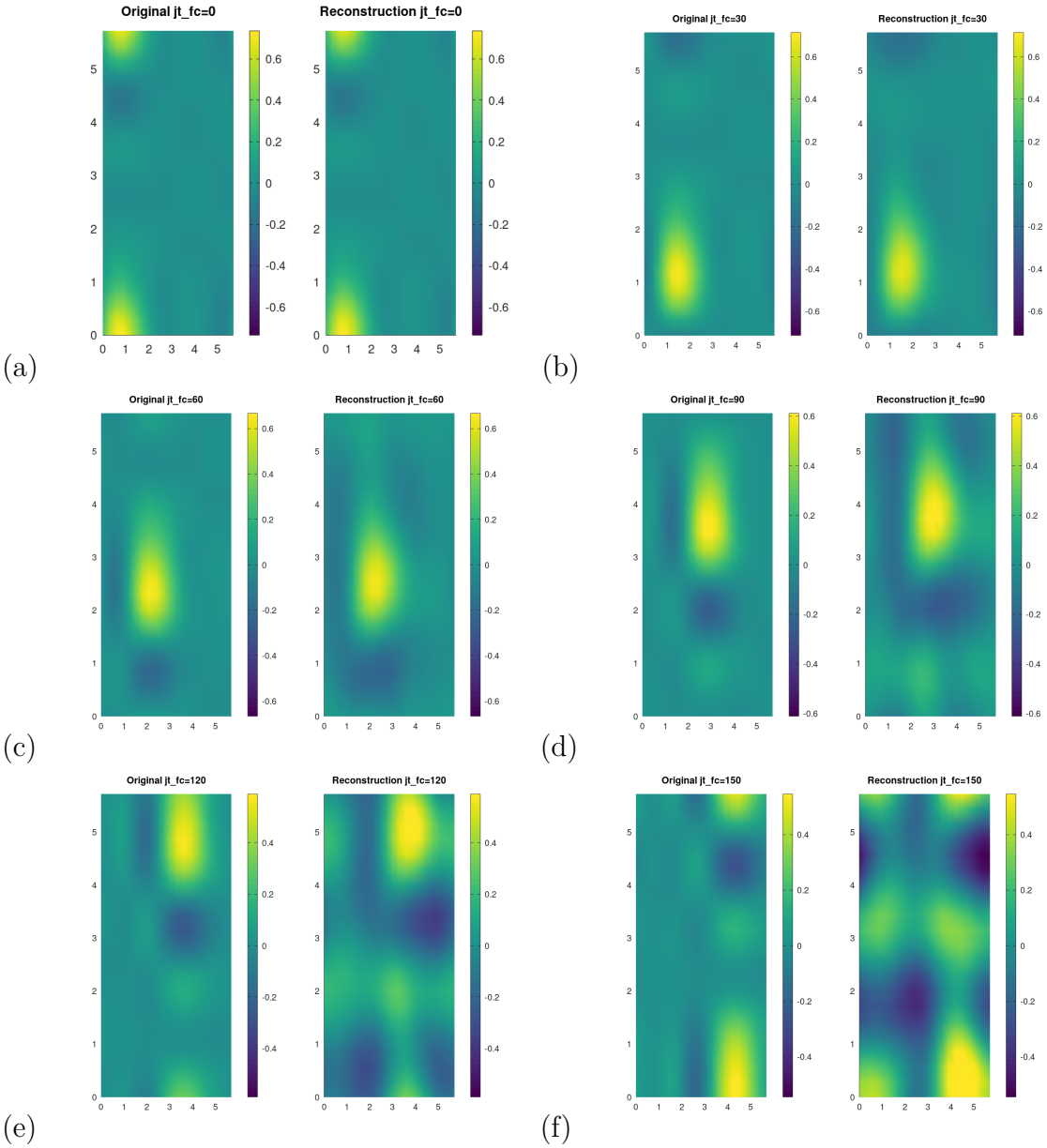


Figure 4.20: The Figures (a)-(f) show different times steps comparing the original dynamics with the field generated by the learned differential equation, where we display iterations $0, 30, ..., 150$, all for $p = [0.5; 0.86]$.

Sensitivity Experiment #2. We show a comparison of the results for two different choices of the diffusion coefficient c. For $c = 1e-4$ and $c = 1e-5$ we have carried out the same reconstruction, studying the trajectory for the true and approximated (learned) PDE starting from the same initial field u_0 in Figure 4.21.

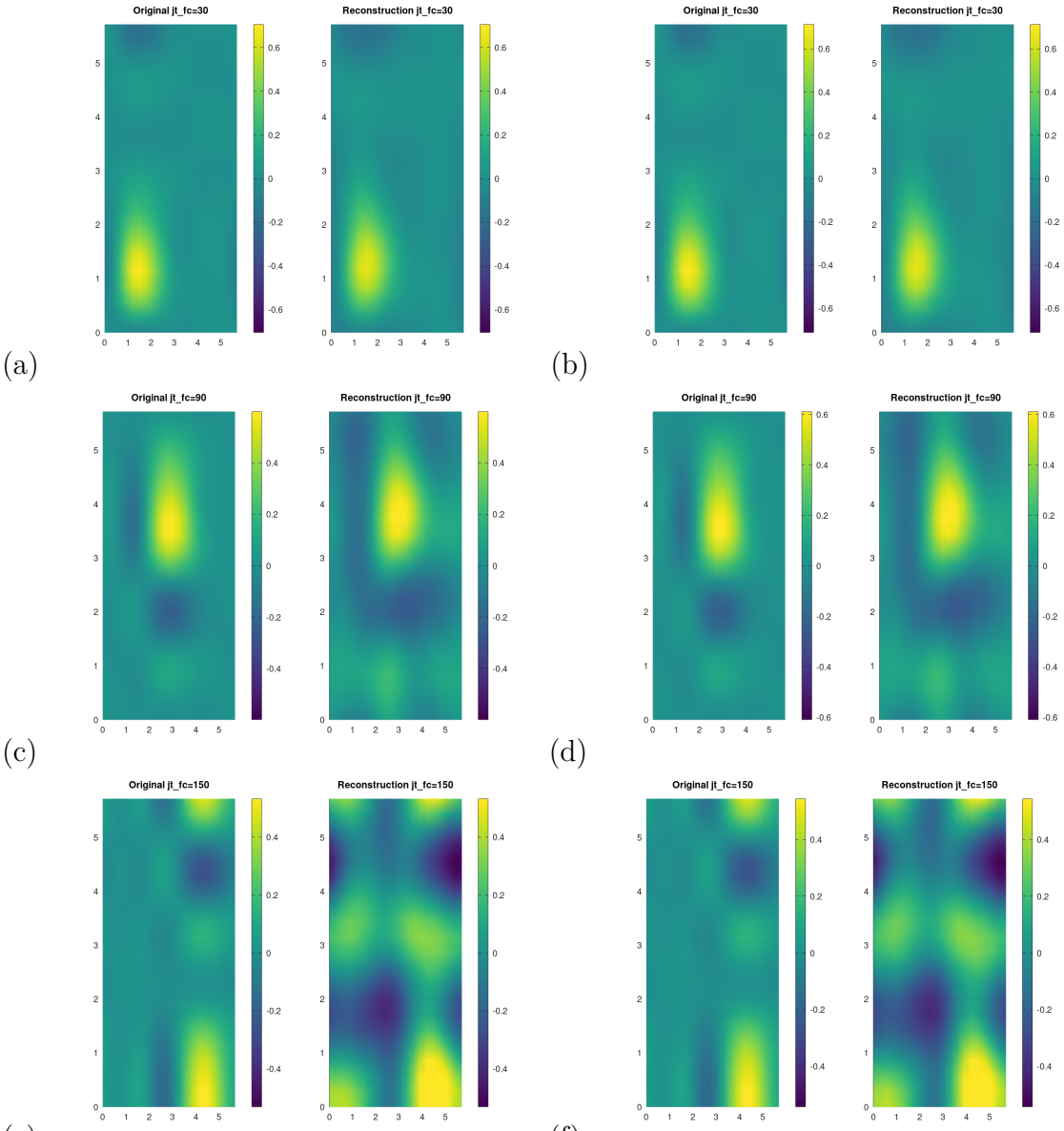


Figure 4.21: The Figures (a)-(f) show different times steps comparing the original dynamics with the field generated by the learned differential equation, where we display iterations $30, 90, 150$, starting with equal states taken from the original iteration 100. All simulations here with $p = [0.5; 0.86]$ with different diffusion $c = 1e-4$ in (a), (c) and (e) and $c = 1e-5$ in (b), (c) and (f). Fields with less diffusion are larger as expected, after 150 iterations the two fields have started to strongly diverge due to the difference between true PDE and approximated PDE.

Chapter 5

Statistical Sensitivity Analysis of Model Reconstruction

This chapter aims to investigate the sensitivity of the model reconstruction, i.e., to understand how variations in the noise level affect the accuracy of the model reconstructions or the performance of the data assimilation techniques developed in this thesis. It helps assess the robustness, identify weaknesses, and make informed decisions to enhance the model reconstruction algorithm's capabilities.

In particular, we investigate the

1. Sensitivity of the results to different levels of observational noise.
2. Sensitivity of the results to partial observations of the state.
3. Sensitivity of the results to time frequency of observations.
4. Convergence behaviour as the approximation of the model is improved.

We will present statistical evaluations for the Lorenz 63 model in Section 5.1, for the Lorenz 96 model in Section 5.2 and for the Amari Neural field model in Section 5.3. The evaluations aim to show the changes in the learning performances of the models mentioned above at varying time steps or experiences.

In the case of Lorenz 96 and Amari, we were using approximations which contain the true model. Then, it does not make sense to study the convergence of the approximation, but the convergence will be obtained when more and

more input data are used for learning. We will show this by running a sequence of experiments with more and more steps.

In addition, we also present the results of the cycled experiments carried out with some setups. We run a data assimilation cycle for each choice of the different noise levels added to these simulated experiments in this chapter. The average first guess error measures how well the assimilation works. We now do this ten times with different observation errors, leading to a curve we can display (as shown in the figures below). In general, sensitivity analysis helps quantify how these uncertainties propagate through the model and helps understand the impact on its predictive capabilities.

5.1 Sensitivity Results for Lorenz Model L63

Sensitivity analysis with respect to observational noise. In this section, we discuss the simulations for the different levels of observational noise using the Lorenz 63 model.

We carry out the 3D-VAR based model reconstruction with a growing number of nodes for 5000 time steps and study the dependency of the model reconstruction error on the observation error. For each observational noise level, we carry out an experiment. This means

- We carry out the full assimilation cycle.
- In each assimilation step we update the model coefficients c
- When the learning phase is completed, we test the reconstructed model against the true model.

Model reconstruction error is measured based on the Frobenius norm (also known as the Euclidean norm) for calculating the error of the difference between the true first guess and the reconstructed model-based first guess for all time steps of the full original trajectory.

In the chart below in Figure 5.1, we present the results of the model reconstruction error against the noise levels. It is noticeable that the noise level increase is having an impact on the model's reconstruction error, there is a clear indication that the model is sensitive to noise, even though the rate of a unit change in the axis of the noise levels is higher than that of the reconstruction error axis.

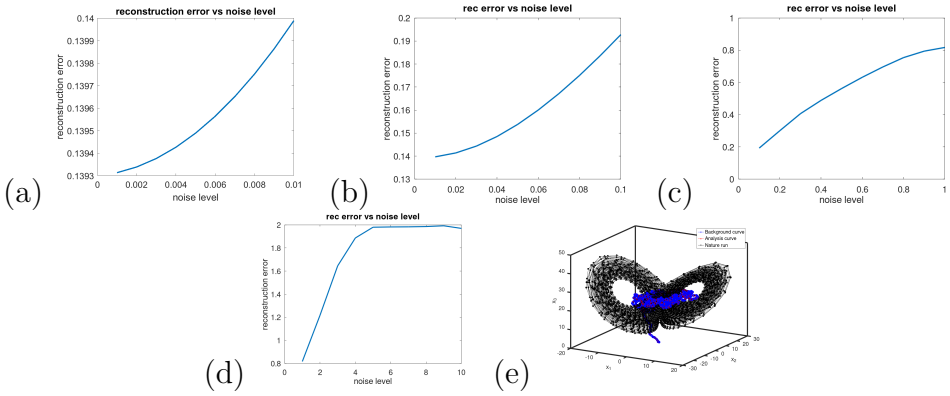


Figure 5.1: The figures (a)-(d) show the growth of the reconstruction error against the different noise levels, testing ranges from 0.001 to 10. As the observational noise level increases, the reconstruction error for the coefficients values increase also in a super-linear way. When the error increases further, the reconstruction breaks down at an error size of about $noise = 3$, we show the behaviour of the reconstructed model for $noise = 10$ in (e).

Sensitivity analysis with respect to partial observations of the state for Lorenz 63 model. We carry out reconstructions based on the Kalman Filter (KF) and a fixed number of degrees of freedom by observing one, two, or all three of the Lorenz 63 dimensions and studying different noise levels.

We have carried out experiments in the same way as above, now studying the reconstruction error when modifications of the observation operator are carried out.

In Figure 5.2 below, we present a diagram displaying the reconstruction error against the noise level at three different choices of the observation operator. We set up an observation operator of different dimensions where we first observed only the first component of the state (in blue); we then proceeded to observe both the first and second components (in red) and finally, we observed the complete state components (in yellow). It can be seen that observing more components of the state lead to better model reconstruction.

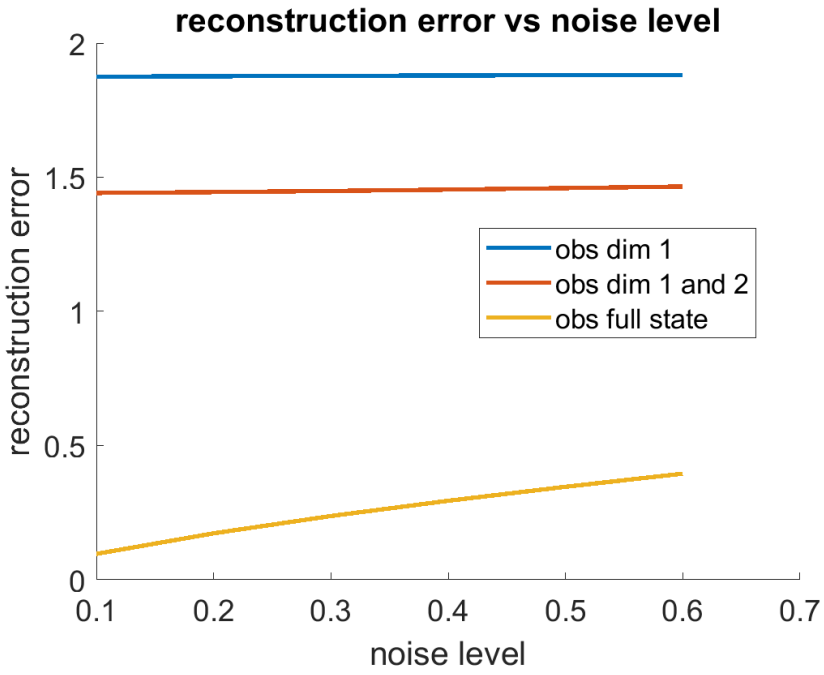


Figure 5.2: The chart shows the reconstruction error versus the noise levels when the observations are partially observed or spaced at specific intervals or dimensions, and this is compared with the observed full state.

Sensitivity analysis with respect to time frequency of observations Lorenz 63 model. Analyzing the reconstruction behaviour for the time frequency of observations is non-trivial. We could keep the total number of time steps investigated fixed or the total covered interval fixed to [0, T] and adapt the number of time steps accordingly.

In both cases, the reconstruction quality is better with a larger time frequency of observations (i.e. smaller time spacing) and corresponding assimilation and model learning steps. The error, though, increases strongly if we have less number of time steps used for learning, i.e. in the second case, the error increase is much stronger for *dtime* increasing (and then the nature run (Nnat) becoming smaller since `Nnat = ceil(T/dtime)`).

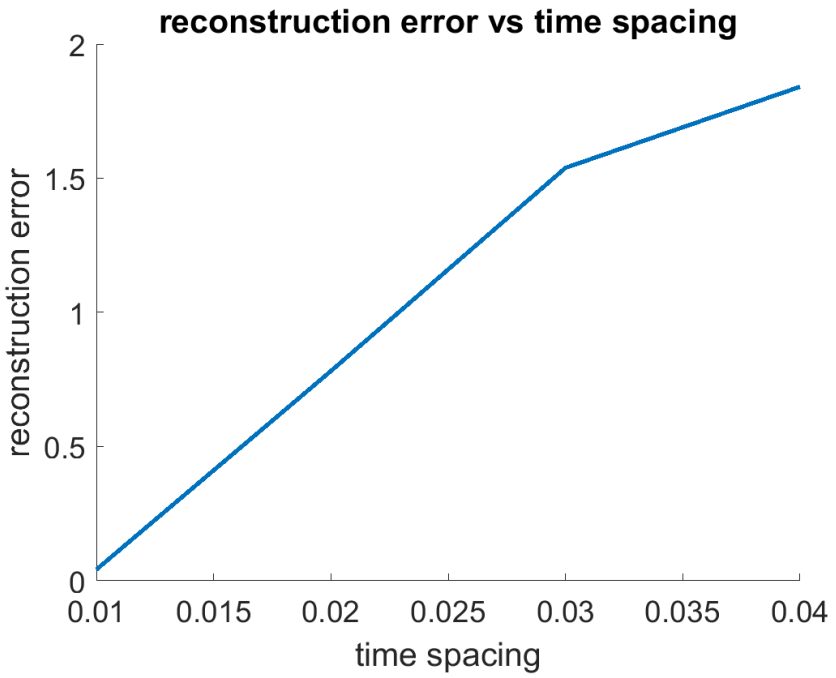


Figure 5.3: The chart shows the reconstruction error versus the time frequency with $T = 4$ of observations.

Sensitivity analysis with respect to convergence behaviour as the approximation of the model is improved Lorenz 63 model. We will study the convergence behaviour exploring two dimensions of the reconstruction procedure. The first dimension is the amount of training data, i.e. the total

number of time steps used for training. The second dimension is the model approximation quality, i.e. the dimension of the model space employed for approximating the true model by an approximate model.

First, we test the convergence of the reconstruction error when the number of simulation and learning steps for the nature run $Nnat$ is increased step by step. At the same time, the number of degrees of freedom for the learning coefficients and the testing trajectory is kept fixed.

The simulation result is shown in Figure 5.4 below, showing the reconstruction error against the number of time steps in the nature run (Nnat) used for training. The nature run represents our dynamic system's (actual) behaviour;

- a data assimilation cycle has been carried out for each fixed Nnat.
- Then, a testing cycle with the trained model has been equally calculated.
- The error has been evaluated for a fixed cycle for all tests, with a `j_norm_max`=99. This can be interpreted as a fixed evaluation period for all cases of training length.

As the nature run increases, we also see a gradual decline in the reconstruction error. We can infer that the accuracy of the model or simulation is improving as it approaches the system's actual behaviour. This behaviour suggests that the model's error predictions are getting closer (converging) to the actual outcomes represented by the nature run.

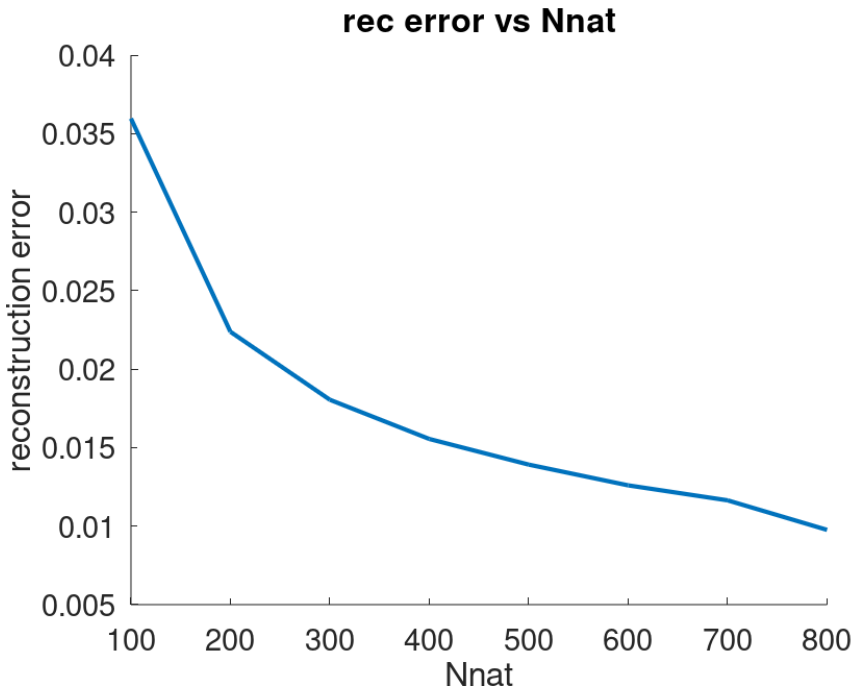


Figure 5.4: The figure shows the behaviour of the reconstruction error measured above by the first guess forecast difference to the truth is dependent on the length of the training period indicated by Nnat. We observe that the reconstruction error decreases as the number of trajectory observations and corresponding learning steps increases.

Second, we did a test for evaluating the reconstruction error and the convergence for better and better model approximations for Lorenz 63. In this instance, the situation depends on two parameters: the density (or number) of nodes of the radial basis functions and their variance sigma. The variance must be adapted to the nodes' density to achieve better approximations when the number of nodes changes.

We have carried out several experiments of reconstructions with increasing the node density but keeping sigma fixed and visualising the observation error. In particular, with sigma chosen large enough, we observe that with more nodes, i.e., with more degrees of freedom for the model approximation, the model error decays, and we can achieve better reconstructions. This behaviour also occurs where we could get minimal reconstruction errors for the model reconstruction task.

For different fixed σ, the result of these runs is reflected in Figures 5.5 and 5.6. The results of Figure 5.5 (d)-(f) and Figure 5.6 (g)-(j) show a decrease of the reconstruction error over the full trajectory when the number of degrees of freedom for the model approximation is increased.

We need to briefly discuss the results (a)-(d). For $sigma = 4, 5, 6$, when we increase the number of nodes we observe an increase in error. We note that we have strong ill-posedness of the Radial Basis Function (RBF) approximation when we have very wide ranging RBF, where increasing the number of nodes might increase the error. We need to study the ill-posedness of this particular approximation, and its consequences. However, this type of investigation is beyond the current focus of our work and needs to be part of future research.

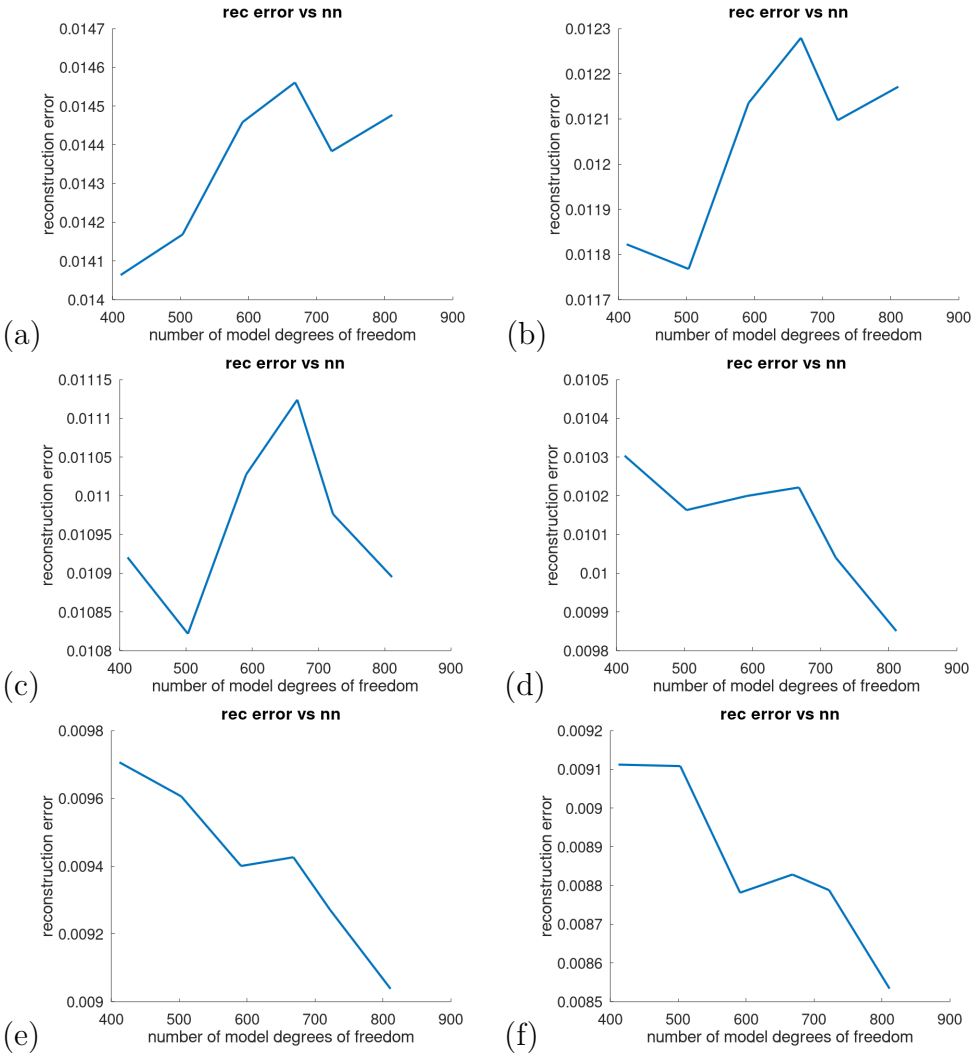


Figure 5.5: Experimenting with reconstruction error with varying `sigma`= $4, 5, 6, 7, 8, 9$ in (a) to (f). We display a visualization of the different phases of the errors as we we increase the sigma values. We observe that the reconstruction error converges slightly as we increase the values of the sigma.

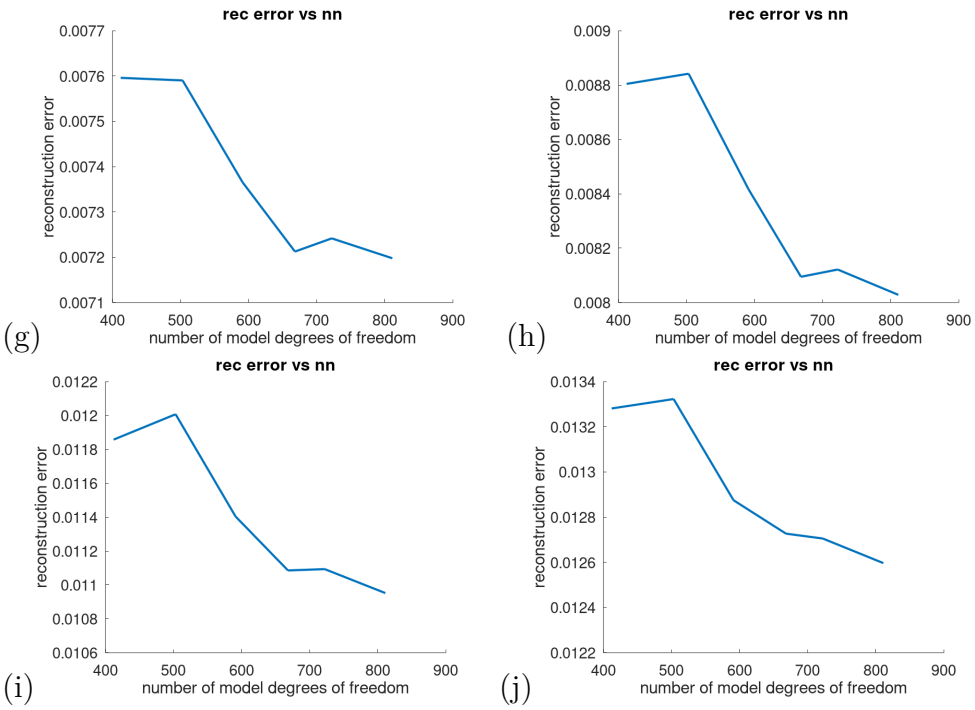


Figure 5.6: Experimenting with reconstruction error with varying `sigma`= 15, 16, 18, 20 in (g) to (j). We display a visualization of the different phases of the errors as we we increase the sigma values. We observe that the reconstruction error converges slightly as we increase the values of the sigma.

5.2 Sensitivity Results for Lorenz 96 Model

Sensitivity analysis with respect to observational noise. Different levels of observational noise for Lorenz 96. In the sensitivity analysis concerning observational noise, we chose a scenario for testing, where we observed all the nodes of the L96 system as shown below in figure 5.7. The size of the fields is between -10 and 10, such that we studied ranges of random errors ranging from 1 to 10.

We have carried out the same type of experiments as described in the framework of Lorenz 63. This means

- we carry out an assimilation cycle where we learn the model step by step during cycling.
- We then test the quality of the learned model with some appropriate metric.

Since our model approximation approach for Lorenz 96 is different, here we can actually measure the reconstruction quality by comparing the reconstructed coefficients with the true coefficients of the Lorenz 96 model, compare Figure 4.12. The reconstruction error for coefficients is L^2-norm of the differences of the reconstructed coefficient vector to the true coefficient vector.

The chart below in Figure 5.7, exhibits a similar pattern to the observational noise for the reconstruction error shown above in Figure 5.1 for Lorenz '63. A similar super-linear relationship exists in the change in unit growth rate between the noise level and the reconstruction error for the coefficients.

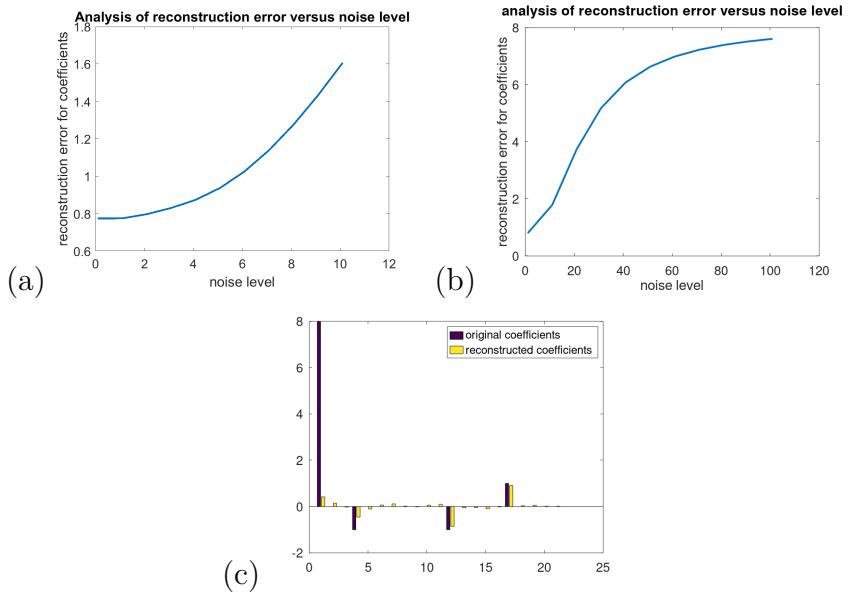


Figure 5.7: The curves (a) and (b) show the growth of the reconstruction error against the different noise levels for small noise and a break-down of the reconstruction of the main constant coefficient for large noise. As the observational noise level increases, the reconstruction error for the coefficient values increases rapidly. (c) shows the case where $noise = 10$, where the first coefficient is no longer properly reconstructed.

Sensitivity analysis with respect to partial observations of the state for Lorenz 96 model. We next study how the reconstruction error depends on the number of nodes of the L96 state observed in each assimilation step. The sensitivity analysis is shown in Figure 5.8. It shows an increase in the reconstruction error of the coefficients as the spacing of the observations increases. In more detail,

- we have chosen a case of $N = 64$ nodes for Lorenz 96 and
- observe the nodes $1 : hH : N$ with spacing $hH = [1, 2, 4, 8, 16, 32, 64]$.
- For each choice we have carried out a full reconstruction experiment and calculate the reconstruction error of the coefficients at the end.

This setup guarantees that the observations are always at the same points, but taking away observations step by step.

We observe that the reconstruction is still quite stable since it relies on the ability of the LETKF to synchronize the dynamics, which works even with limited observations as long as the observation error is within a reasonable range.

The setup covers the full range of options from observing all nodes to observing one node only.

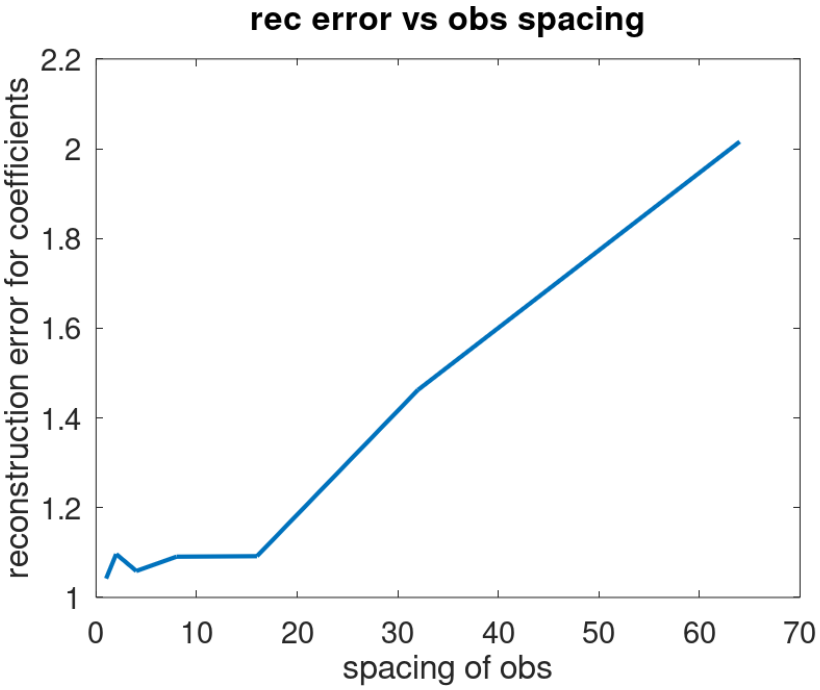


Figure 5.8: The chart shows the reconstruction error versus the observation spacing when the observations are partially observed. We observe every $n-$th state variable, where $n = 1, 2, ..., 64$.

Sensitivity analysis with respect to time frequency of observations for Lorenz 96 model. The sensitivity analysis of the time frequency for the Lorenz 96 model as shown below in 5.9 and 5.10, shows that for every unit (in terms of absolute units) increase in the time spacing for learning or assimilation, we see a corresponding increase in the reconstruction error for the coefficients.

To test the dependence on the time frequency of observations, we adapt the parameter *dtime*, which controls the times when assimilation and learning steps are carried out. To still cover the same dynamical range over an interval [0,T], we had to adapt the total number of time steps we ran the system against. We show two different images below, one with $T = 2$ and one with $T = 0.6$, but with more frequent time steps being a multiple of `dtime_0`=0.001.

We present two scenarios shown below with varying time frequencies. In both cases in Figures 5.9 and 5.10, we observe similar trends in the charts as we have seen using Lorenz '63 above.

- Again, the reconstruction quality improves with a more significant time frequency of observations (i.e. smaller time spacing) and corresponding assimilation and model learning steps.
- Similarly, the error increases strongly if we have less number of time steps used for learning, i.e. in the second case, the error increase is much stronger for *dtime* growing (and then the nature run (Nnat) becoming smaller since `Nnat = ceil(T/dtime)`).

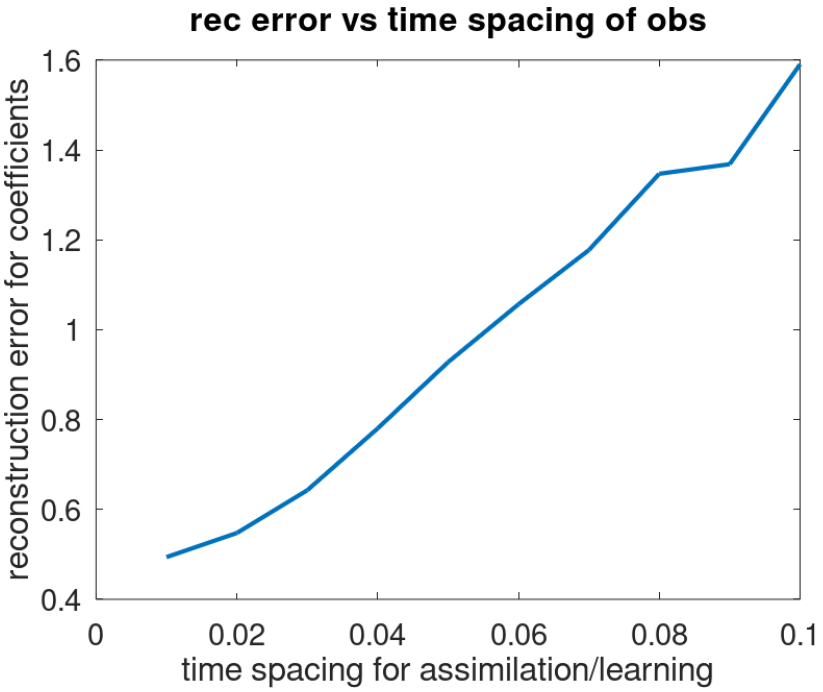


Figure 5.9: The chart shows the reconstruction error versus the time spacing/frequency with $T = 2$ of observations.

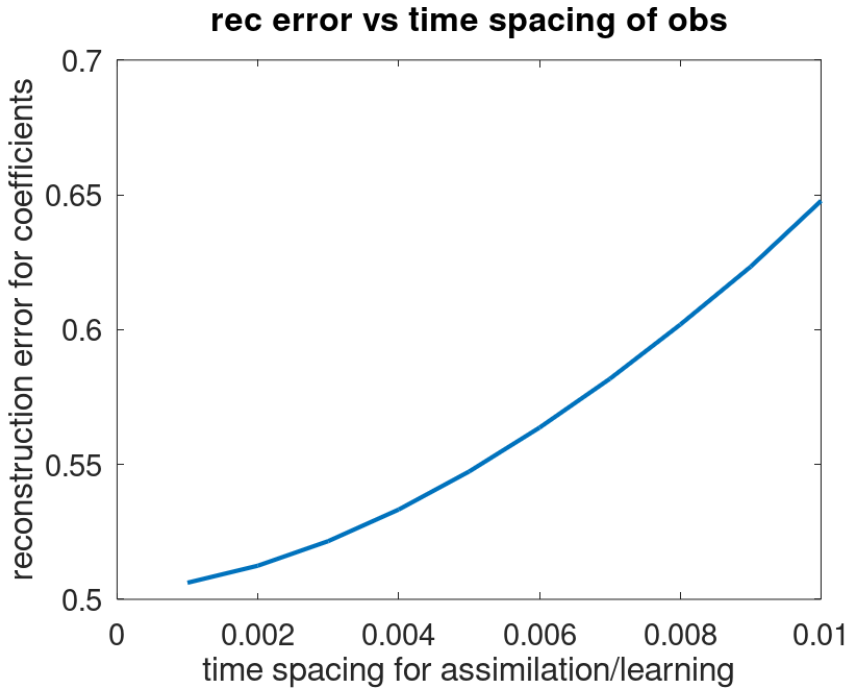


Figure 5.10: The chart shows the reconstruction error versus the time spacing/frequency with $T = 0.6$ of observations.

Sensitivity analysis with respect to convergence behaviour as the approximation of the model is improved. For the case of L96, we were using approximations which contain the true model. Then, it does not make sense to study the convergence of better and better approximations, but the convergence will be obtained when more and more input data are used for learning. We will show this by running a sequence of experiments with more and more steps.

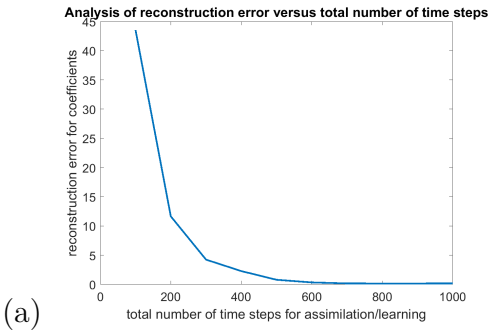
The convergence for the reconstruction of the parameters is visualized in Figures 5.11 below. We tested convergence by choosing a small parameter *dtime* and then, step by step studying the reconstruction error for longer time intervals [0, T], where $T = dtime * Nnat$ with the number of time steps $Nnat$.

- For each choice of the total number of time steps $Nnat$, we carry out a full reconstruction and then calculate the reconstruction error of the coefficients. The corresponding curve is shown in Figure 5.11(a).
- A result for 1000 time steps, $noise = 0.01$ and $dtime = 0.001$, is shown in Figure 5.11(b) comparing the original and reconstructed coefficients.
- The evolution of the error for each coefficient for 1000 time steps of learning is shown in Figure 5.11(c).

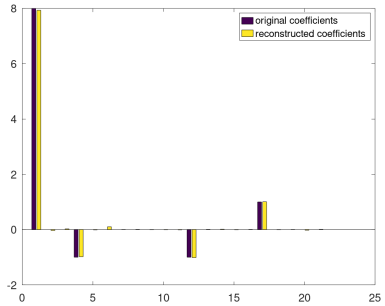
The result in Figure 5.11(a) shows the gradual decay of the reconstruction error to zero as the total number of assimilation or learning steps for the nature run increases with other parameters kept constant.

Similarly, the bar chart in Figure 5.11(b) shows the high quality of the reconstruction in comparing the original and reconstructed coefficients. We note that the reconstruction finds all the nonzero coefficients and their corresponding sizes as well as identifies all zero coefficients for which the reconstruction is also very small.

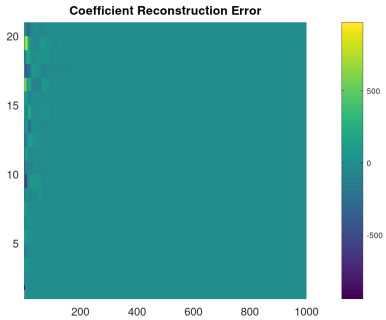
In addition, the heatmap in Figure 5.11(c) helps to visualise the coefficient reconstruction error outcome with the time steps.



(a)



(b)



(c)

Figure 5.11: Figure (a) shows the model reconstruction error depending on the length of the input trajectory used for learning in terms of the total number of time steps `Nnat` used for training. The more input data, the better the model reconstruction when all other parameters are kept fixed. The bar char for the original and reconstructed coefficients are shown in (b), while (c) shows the heatmap visualization of the evolution of the coefficient reconstruction error for the last experiment learning with 1000 time steps.

5.3 Sensitivity Results for the Amari Neural Field Equation

Sensitivity analysis with respect to observational noise. Different levels of observational noise for neuro kalman filter

We begin simulations for the Amari neural field model with predetermined neural dynamics. The neural kernel can then be computed using an inverse problems approach or a Kalman filter-based model learning method, as described in Section 3.34, and equation (3.35). We incorporate observational error into the simulated neural dynamics and then examine the reconstruction results depending on this error.

It is essential to remember that we do not begin with a true model for the neural field, but rather with a *true* or *prescribed* neural dynamics. Consequently, the model reconstruction error will be measured in terms of the error of the simulated dynamics following neural kernel learning.

Alternately, we could examine the kernel reconstruction error; however, because we use an approximative space with limited dimensions for the approximation, the reconstruction error in terms of the kernel will be constrained by the best possible approximation within this kernel space to whatever reference kernel is selected.

The experimental setup to carry out reconstruction runs for different levels of observational noise for the neuro Kalman filter is described next. The Matlab codes numbered (1-5) are shown in the Appendix 7.3.

1. First, we calculate a dynamical neural field u, which is prescribed and provided as a generic input 7.3. In this instance, it is an excitation moving in a circle as a test case.

2. For reference, we then calculate a neural kernel by *inverse methods* based on the equation
$$\frac{du}{dt} = -u + Wf(u) \tag{5.1}$$
in the third Matlab code attached to the Appendix 7.3.

3. Gaussian kernel functions G_ξ and $G_{2,\xi}$ with two different variance parameters σ and σ_2 are setup in the file #6 of Section 7.3. They consist of Gaussians around base points counted by $\xi = 1, 2, ...$, which for efficiency reasons here we have restricted to the true trajectory.

4. The full model observation operator Hc is then composed in the file #7 of Section 7.3, where for each location an excitation or inhibition in forward direction along the trajectory or in backward direction is setup, where the strength and sign is to be learned by the Kalman filter.

5. Now, for different sensitivity scenarios as described above, we carry out a Kalman filter model reconstruction in 7.3. In the code, there are various versions of reconstruction methods which can be tested, but we have focused on the *Kalman Filter* method with a particular Gaussian representation of W as given by the parameter $represent1 = 3$ for this test case, i.e. the Kalman filter is based on the form 5.1, where W is written as a sum of all the Gaussians in the form

$$W = sum_{\xi} G_{\xi} * G2'_{\xi} c_{\xi} \tag{5.2}$$

with coefficients $c_x i$ to be determined by the Kalman filter successively.

Our observations come from the simulated neural fields u (in the code named `uv`). We add random observation error to our truth in the form

```
uv_e = uv + obs_err*randn(size(uv));
```

such that observations with observation errors are given by uv_e. Clearly, this error is part of the Kalman filter equation in the form,

```
Rc = obs_err^2*eye(nn,nn);
```

with the number of points nn where the neural field is defined.

The reconstruction error is measured by the differences between the prescribed dynamics uv and the reconstructed dynamics $urec$. The error in the L2 norm over space and time is given by

```
 e_ba3(:,j) = urec(:,j+1) - uv(:,j+1);
```

calculated in the script #8 of Section 7.3 shown below.

Finally, we carry out reconstructions for different levels of observation errors in the 7.3 file, storing the input observation error and the resulting reconstruction error into vectors

```
obs_err_v(joe) = obs_err;
rec_err_v(joe) = e_ba3_norm;
```

with `e_ba3_norm` given by the *Eucledean norm* of the full space-time trajectory of the simulated field normalized by the square root of the total number of points in space and time.

The dependency of the reconstruction error on the observation error is displayed in Figure 5.12. The reconstruction error grows with observation error.

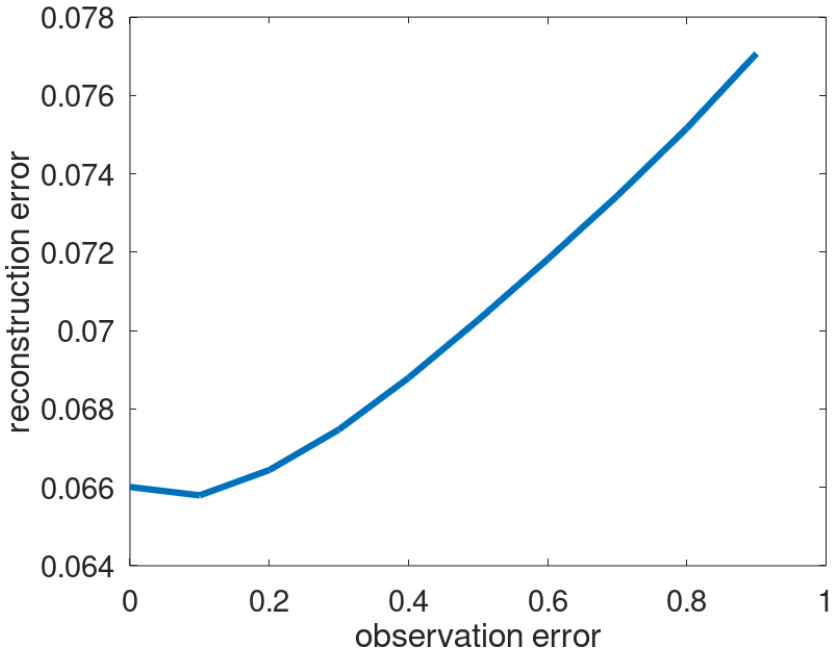


Figure 5.12: The line graph shows the reconstruction error versus the observation error for the different noise levels. There is a strong positive correlation between the observation and reconstruction errors, with growing observation error the reconstruction error increases.

Sensitivity analysis with respect to partial observations of the state. In case 1 (different levels of observational noise for neuro Kalman filter) as described above, we have chosen to observe all points of the neural domain. Here, the objective is to partially observe the neural domain by taking every n-th point of the neural excitation field when arranged as one long vector line by line.

In Figures 5.13(a)-(b) below, we show a study with partial observations in the approach for a strong constraint learning problem (which we have due

to our choice of the Gaussian Ansatz functions learning Gaussian pulses). The reconstruction is relatively stable as long as some observations cover the trajectory.

The visualization of the observation points is given by the diagram below in Figures 5.13(b) for the case when neural tissue is observed at every $5th$ successive interval. The Neuro reconstruction is capturing the original pulse well. The Kalman reconstruction captures the main features of this pulse as well. The error is displayed in the Figure (c) top left, being below 20% of the pulse.

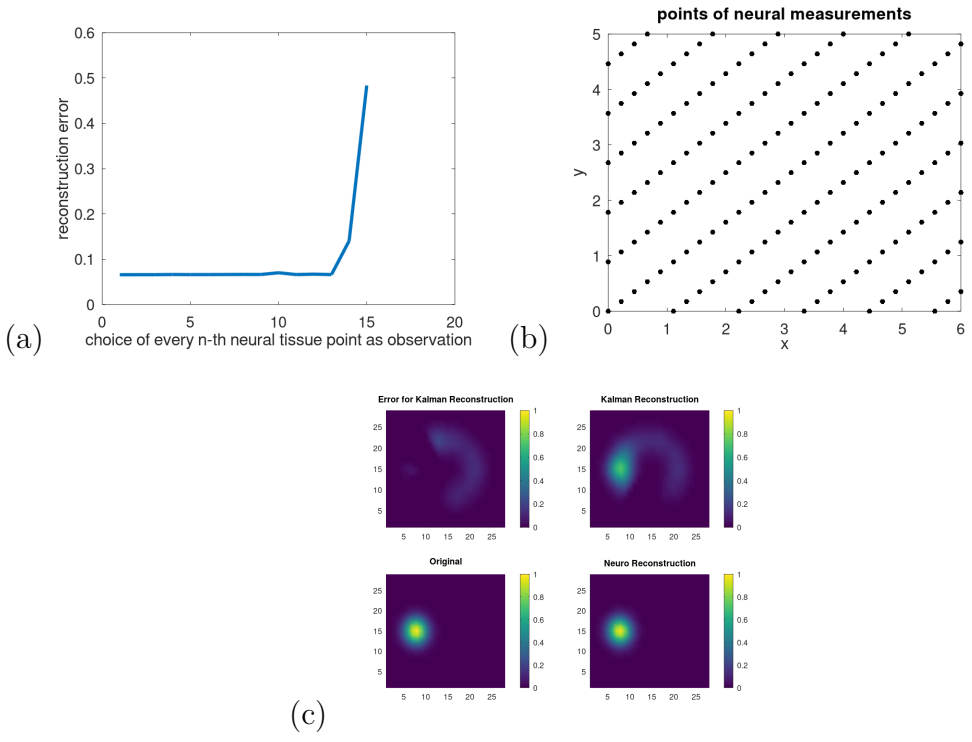


Figure 5.13: In figure 5.13 the graphics (a)-(c) show the different representations of the reconstruction study where we take an observation at every $n-$th point of the neural domain. In (a), we carried out reconstructions for the choice of the $n-$th point and show the dependence of the total reconstruction for a corresponding sequence of experiments. (b) shows the geometric setup for one of the cases with $n = 5$. (c) shows one reconstruction snapshot to provide an impression of what the errors actually mean.

Sensitivity analysis with respect to time frequency of observations.

We test reconstructions with observations taken at different time-frequency. The graphics show the reconstruction error in dependence on the number time difference where $dtime = 2$ means that the observation is taken at every 2nd-time step, $dtime = 3$ means it is taken at every 3rd-time step, etc.

The graphics in Figure 5.14 shows the dependencies of the reconstruction error on the choice of the time steps where observations take place. There is an upward trend, but there seems to be an overlay with some oscillating function, which is a typical phenomenon if oscillators are observed with increasing time frequency.

To argue that we indeed run into this phenomenon, we show the $x1 - x2$ location of the observed pulse centers of the neural pulse under consideration in the case $dtime = 6$ in Figure 5.15. Observing these now with increasing time frequency will result in the standard aliasing phenomena of signal processing.

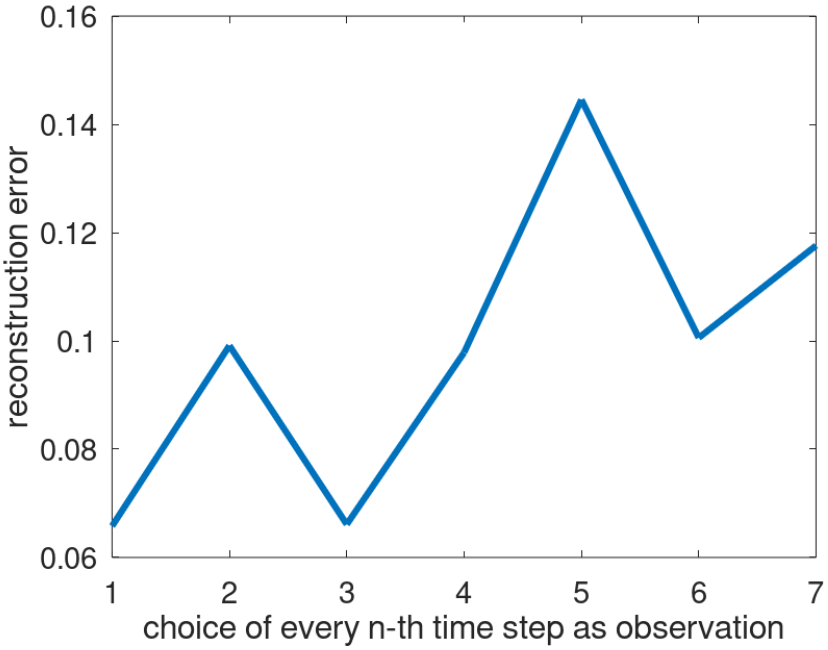


Figure 5.14: Display of the reconstruction error for carrying out the model reconstruction at every $n-$th time step of the neural dynamics. There are some aliasing effects visible, since the pulse rotates several times and when the observations cover more positions, the reconstruction is better than for the case where we observe only at smaller selection.

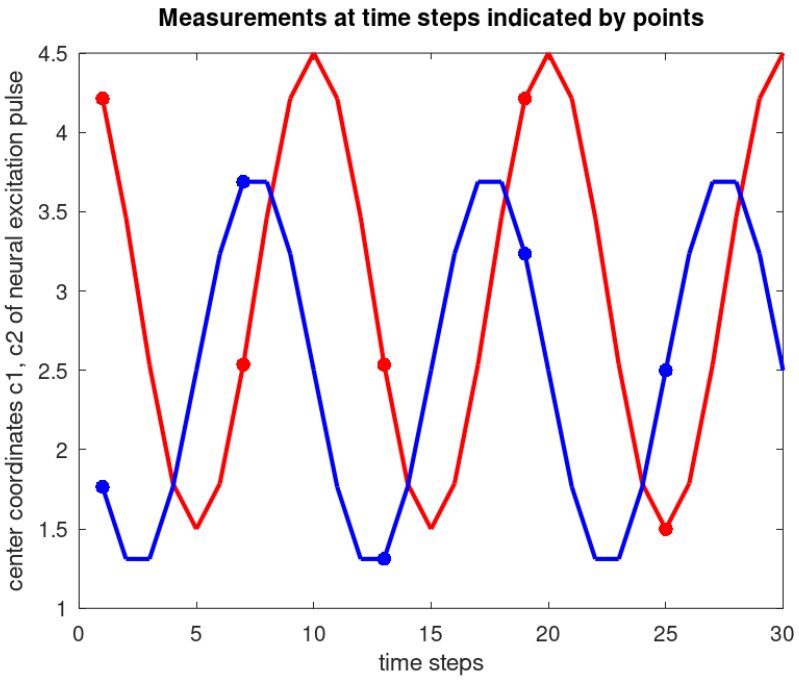


Figure 5.15: Shows the time steps measurements at selected points of the center coordinates $c1$, $c2$ of the neural excitation pulse. They also display the cycle between cosines and sines for the two coordinates, $c1$(blue) and $c2$(red) at different time-steps and points as shown on the chart.

Sensitivity analysis with respect to convergence behaviour as the approximation of the model is improved. For the case of Amari, we were using approximations which contain the true model dynamics. Then, it is not meaningful to study the convergence of better and better approximations, but the convergence will be obtained when more and more input data are used for learning. We will show this by running a sequence of experiments with more and more steps.

To demonstrate the convergence behaviour of the model, we observe the performance of the sum of the reconstruction error with increasing iteration in the number of time steps used in the learning process, as shown in Figure 5.16. It is noticeable that as expected for the setup with the true model within the approximational range, the total reconstruction error converges towards zero as we increase the number of time steps used for assimilation or learning the model.

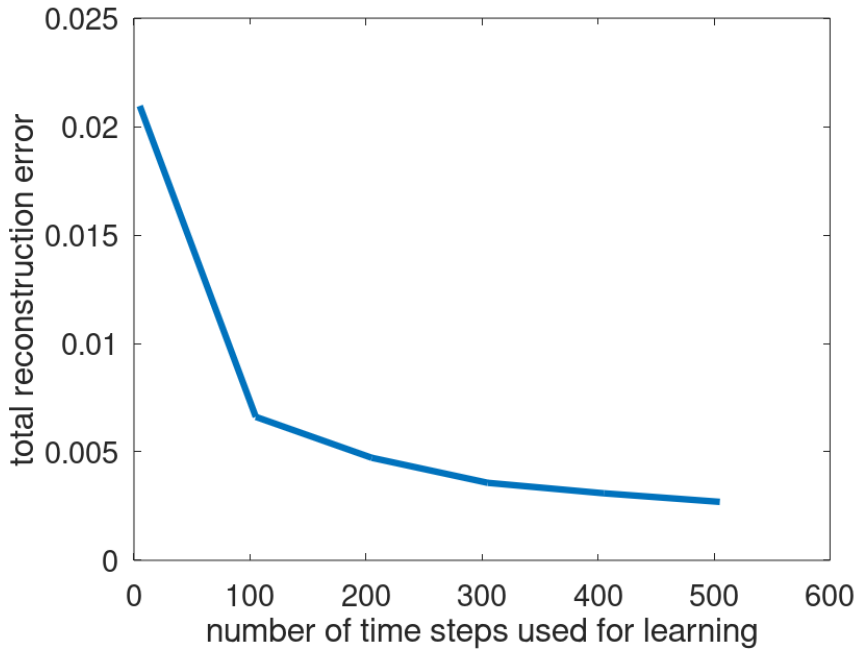


Figure 5.16: Shows a convergence study in the sense that the total reconstruction error tend towards zero as the number of time steps used for the learning increases.

Chapter 6

Conclusions and Perspectives

In the previous chapters of this thesis, we stated the challenging and often difficult situation of predicting the forcing term of a dynamic system where little or some specific knowledge is known of the underlying structure or parameters of the model. We investigate the estimation of the forcing term F of a mechanistic dynamical system model of the form $\dot{x} = F(x)$ by a high-dimensional nonlinear approach based on Polynomial or Gaussian functions. We have used the 3 model systems - Lorenz models (Lorenz '63 and '96) and the Amari Neural Field equation as representations of our dynamical systems to test the formulated algorithms. The reconstruction algorithm is then finally tested with supplementary data from a simplified version of the full reaction-diffusion system weather model as a test case.

In this chapter, we present a detailed summary of the main findings and evaluation of the results from this work. We also provide our perspectives on the future development or improvement of the techniques formulated in this thesis, in addition, we present our views on other high-impact applications where the techniques could potentially be applied for future research purposes.

6.1 Evaluation of Results

The role of data assimilation and the Kalman filter techniques are quite pivotal to the formulated model learning approaches used in this thesis. As we may recall, data assimilation is mainly concerned with the combination of observations amid random and zero-mean errors with optimal model predictions. Therefore, achieving an optimal model prediction is often very challenging since

there are systematic errors including the model suitability for making such predictions to contend with.

Broadly speaking, model reconstruction is learning models of dynamical systems and determining their accuracy and precision for predicting the future states of that system. In this thesis, we have introduced 2 different methods in Sections 3.1 and 3.2 which explain the model reconstruction approach used in the thesis. The main results are as follows:

a) **Lorenz '63** in Section 4.1, we present the results of Lorenz '63 with the model reconstruction approach as implemented in equations (3.20) and (3.21), see Section 4.1 for more details on the steps and codes respectively. We can see clearly from Figure 4.4, that the images in (a) and (b) show a close resemblance between the original and reconstructed kernels of the two-dimensional neural delay dynamics while the images in (c) and (d) follows a similar pattern displaying a column of the original and reconstructed kernel from the point indicated by the blue star to the trajectories followed by the rest of the neural patch.

   In Figure 4.5, we show the error evolution of the first guess error at different assimilation cycles, first when it was 500 and 5,000 steps. We witnessed a reduction in the errors when the time steps increased, this is obviously due to the improvement in the error approximation as more points are reached multiple times as the assimilation cycle increases.

b) **Lorenz '96** in Section 4.2, we present the results of the model learning method with Lorenz '96 using the Kalman filter approach. In Figure 4.11, we present the application of the algorithm in a higher dimensional environment which is also an alternative method to a dimension reduction approach. We displayed the results of the dynamics of the simulated variables together with the local ensemble transform kalman filter (LETKF) plots of the first guess and analysis error evolution. We also showed how the first guess and analysis mean estimates of the observation are closely matched with the nature run in addition to their deviation as well.

   In Figure 4.12, here we show the reconstruction of the model errors. First, we compared similarities in the results of the first guess field with that of the reconstructed model. We also show the same comparison in the variations for the original and reconstructed coefficients as well as the heat map for the reconstructed coefficient error which is very low.

c) the **Amari Neural Field Model** in Section 4.3, we displayed the comparative results of the prescribed model dynamics and that of the reconstructed model dynamics at different times $t = 5, 10, 15, 20, 25$. The approximate dynamics can generate the movement of the pulse, though with a slight phase error, see Figures 4.14 and 4.15 for more descriptions.

d) the **reaction-diffusion system model** in Section 4.4, we showed the competitive nature of the proposed algorithm with data of a proxy system for the full reaction-diffusion system weather model by estimating the function term of the measurement of the temperature taken.

Overall, this research has demonstrated the use of the formulated algorithms in this thesis as viable methods of reconstructing models of unknown dynamical systems with little specific structural knowledge of the system.

6.2 Limitations of the techniques used in the thesis

This section outlines the study's shortcomings, some of the potential restrictions are described below:

The nervous system is a highly complicated system with several interconnected components. Developing mathematical models that accurately represent this complexity might be time-consuming, and including all critical biological processes into a single model could be challenging.

In this thesis, we investigate a method to learn models from observations. The method is rather generic, and it works for general non-linear models. We used the classical approach to learn either the coefficients of some partial differential equation or the coefficients of the model itself represented in the form of basis functions.

Today, data science has developed further model representations based on neural networks or random forests, which are learned by minimisation techniques. Further model representation approaches are based on image segmentation and neural architectures known as auto-encoder or Unet. We do not consider such techniques in this work. However, the iterative learning techniques developed here can also be applied to the new AI architectures with neural networks.

We have treated several academic systems which are very popular in studying data assimilation approaches, including low and high dimensions. However, we did not try the techniques in real-world applications yet. It would be desirable to try the methods, e.g. on operational weather forecasting or climate science.

This work has provided important insights into how mathematical models may be utilised to idealise complex biological processes. Nonetheless, the strategies used in this study can be extended to other fields of research outside of neuroscience.

6.3 Perspectives on Techniques

One of the key perspectives on the techniques proposed in this research would be to see how competitive they will perform when comparing with or extending other well-known models or dimension reduction algorithms commonly used in the analytical and data science fields. This aspect of the techniques has not been fully explored in this research against other commonly used methods, like K-Means clustering and other classification methods, Neural Network methods, and other dimension-reduction techniques using the same datasets for effective measurement.

However, there is still much room for improvement. This research can further be extended to test with other real-time data from other physics-based systems as a means of comparison with the reaction-diffusion system weather model. Another important aspect that is well-recognized is the issue of convergence. Its realization is not the primary focus of this thesis. However, the techniques formulated in this thesis are a step towards realizing convergence, and it is an area that can be further explored.

Nevertheless, there are still some pertinent questions that could be raised about observability, i.e., do we have enough information to achieve the true forcing term? Which, in general terms, we do not have. In essence, since we have an ansatz formulated as we have done in this case, then we might still, in some sense, achieve convergence if it can be used to approximate the forcing term.

6.4 Perspectives on Applications

Our perspectives on the other high impact that this technique could be applied to are pretty broad. The technique could be adapted to other machine learning techniques where the data is unlabelled, for example, in unsupervised learning for classification and even other supervised learning techniques where data on the dynamical system is scantily available.

Another important aspect will be to apply them to other areas of engineering, fluid dynamics and on a full-scale weather forecast project where the preponderance of the pieces of literature examined on some of the relevant techniques formulated in this thesis have also been applied.

Chapter 7

Appendix

Attached below are compilations of some selected parts of the core script files used for generating and simulating data and visualizations used in this thesis including those used for testing of the algorithms formulated in the previous chapters above.

MATLAB was the main software used for the codes and visualizations presented in this thesis with occasional use of OCTAVE (a version of MATLAB freely available for download) and PYTHON for comparison and experimental testing. It is also important to note that, there may be differences in syntax between the two (MATLAB and OCTAVE) for some aspects of the codes but we expect most parts to run in both software.

7.1 Learning Low Dimensional Lorenz-63

1. Matlab Code for Lorenz '63 Model Approximation Implementation

```
B = 0.3*eye(3,3);     % background error covariance matrix
m = size(H,1);        % number of observations
R = 0.00001*eye(m,m); % data error covariance matrix
Ac = 0.04;            % coefficient controlling the decal of the covariance

xa = xv(:,1);
coeff_list = [];
xk_list    = [];
for j=1:Nnat
   xbt = sim_06_2_1_Lorenz63(xa,dtime,sigma0);
   xb = M_approx(xa,coeff_list,xk_list,Ac);  % state background
   et(j)=norm(xa-xv(:,j));                % calculate the error
   xbv(:,j) = xb;                         % save it for display
   xk_list = [xk_list, xa];               % base point list, Model analysis!
```

```
14     coeff = H'*inv(R+ H*H')*(y(:,j)-H*xb); % the analysis step
15     coeff_list = [coeff_list, coeff];      % coefficient list, Model analysis!
16     xa = M_approx(xa,coeff_list,xk_list,Ac);  % analysis state
17     xav(:,j) = xa;                         % analysis state list
18     e(j)=norm(xbv(:,j)-xv(:,j));           % calculate the error
19   end
```

Here, the model approximation approach is shown in the code below:

2. Matlab Code for Lorenz '63 Model Approximation function

```
1  function tx=M_approx(x,coeff_list,xk_list,Ac)

2  N = size(coeff_list,2);
3  A = Ac*eye(3,3);
4  tx = x;
5  if( N>0 )
6    for j=1:N
7      xk = xk_list(:,j);
8      tx = tx + exp( -(x-xk)'*A*(x-xk) )*coeff_list(:,j);
9    end
10 end
```

Lastly, next we show the code for the data assimilation steps for the Lorenz '63 model, including the initializations of the background and error correlation matrices. The following script does not yet integrate the learning, which can then be carried out by integrating its steps shown in code No. 1 into the data assimilation cycle.

3. Matlab Code for Lorenz '63 Data Assimilation step

```
1  B = 0.5*eye(3,3); % background error correlation Matrix
2  m = size(H,1);
3  R = 0.5*eye(m,m); % observation error correlation matrix

4   % Assimilation Cycle
5  xa      = x0;   % initial state for iteration
6  sigmaA  = 11;   % parameter in the Lorentz system
7  for j=1:Nnat
8     xb       = sim_06_2_1_Lorenz63(xa,dtime,sigmaA); % Calculate the background
9     xbv(:,j) = xb;                  % and save it in xbv

10    y    = yv(:,j);

11    xa   = xb + B*H'*inv(R+H*B*H')*(y-H*xb); % data assimilation step
```

```
12      xav(:,j) = xa;
13  end

14  ea = sqrt(sum((xav-xv).^2))
15  eb = sqrt(sum((xbv-xv).^2))
```

7.2 Learning Higher Dimensional Lorenz-96

7.2.1 Display Trajectories and Runge-Kutta Application

In the following script below, we present an illustration of the L96 setup to show a visualization of the nodes and activity function of the L96 model as depicted in Figure 2.2 and 2.3 above.

1 Matlab Code for Lorenz '96 - Setup

```
1  clear all;   % clear all variables
2  close all;   % close all figures
3  %-----------------------------------------------
4  % We set up the Lorenz 96 System first of all
5  % There are N nodes located on a circle.
6  % At each node we have some excitation modelled
7  % by x(j), j = 1,..., N
8  %-----------------------------------------------
9  N = 9;       % N = Number of Nodes on a Circle
10 h = 2*pi/N;  % h = angle between two nodes
11 % vector of angles for all nodes
12 angle = [0:h:2*pi-h]';
13 % vector of excitation values
14 x = 10*sin(2*angle);
15 % points in space to put the nodes
16 r1 = cos(angle);  % first coordinate
17 r2 = sin(angle);  % second coordinate
18 %-----------------------------------------------
19 % generate visualization figure
20 %-----------------------------------------------
21 fo = figure(1);
22 % first we plot a reference line
23 plot3(r1,r2,zeros(size(x)),'k.-','LineWidth',2,...
24 'Color',[.6 .6 .6], 'MarkerSize',25); hold on;
25 % second, we plot the excitation function
26 plot3(r1,r2,x,'k.-','LineWidth',2); grid on;
27 saveas(fo,'01_setup.png','png')
```

In the following code, we show the time integration evolution to show the

changes in the excitation state for x for the L96 model as shown in Figure 2.2 above.

2 Matlab Code for Lorenz '96 - We carry out the loop of the time integration

```
%-----------------------------------------------
% Here we carry out the loop of time integration
% to checking and displaying the change of the
% excitation x for our Lorenz 96 model
%-----------------------------------------------
nsteps = 1000;       % number of time steps
F      = 8;          % forcing term of Lorenz 96
t      = 0;          % time shift in Runge-Kutta
h      = 0.01;       % step size for Runge-Kutta

for j=1:(nsteps+1)   % loop over time steps
  [xtmp] = time_integration(x, t, 1, h, F);
  xp = xtmp(:,2);
  % display the current excitation state
  figure(2);
  % first plot the nodes in gray
  plot3(r1,r2,zeros(size(xp)),'k.-','LineWidth', ...
  2,'Color',[.6 .6 .6],'MarkerSize',25); hold on;
  % now plot the excitation xp as a curve
  plot3(r1,r2,xp,'k.-','LineWidth',2);
  % set axis and further grid parameters
  axis([-1 1 -1 1 -12 12])
  view(3); grid on;
  title(['Time index j=' num2str(j)]);
  hold off; % remove previous curves by hold off
  pause(.01)
  % reset initial value for the next integration step
  x = xp;
end
```

Next, we display the control code used for simulating the convergence of the chaotic and highly non-linear Lorenz-96 equation. We simulate a data assimilation cycle with an implementation of the Local Ensemble Transform Kalman Filter (LETKF) approach.

3 Matlab Code for Lorenz '96 - We display the trajectories over time and describe the integration time step size for the Runge-Kutta

```
%-----------------------------------------------
% We display the trajectories for all nodes
% individually
%-----------------------------------------------
```

```
nsteps = 100; % steps for Runge-Kutta integration
F      = 8;   % forcing term
t      = 0;   % time shift in Runge-Kutta
h      = 0.01;% integration step size Runge-Kutta

% get nsteps of the integration in one go
[x2] = time_integration(x, t, nsteps, h, F);

% show the trajectories over time
fo = figure(3);         % generate figure
Nsqrt = ceil(sqrt(N));  % to position figures
for j=1:N
  subplot(Nsqrt,Nsqrt,j)     % generate subplot
  plot(x2(j,:),'LineWidth',1)% show j-th variable
  axis tight; set(gca,'FontSize',12)
  title(['x(' num2str(j) ,')']) % display title
end
saveas(fo,'trajectories.png','png')
```

In order to have a better understanding of the vector, scalar or matrix we have in each of the steps, we look at the combination of the angles for all nodes using the angle setup vectors below.

```
 [angle 2*angle sin(2*angle) x]
ans =

    0.00000    0.00000    0.00000    0.00000
    0.69813    1.39626    0.98481    9.84808
    1.39626    2.79253    0.34202    3.42020
    2.09440    4.18879   -0.86603   -8.66025
    2.79253    5.58505   -0.64279   -6.42788
    3.49066    6.98132    0.64279    6.42788
    4.18879    8.37758    0.86603    8.66025
    4.88692    9.77384   -0.34202   -3.42020
    5.58505   11.17011   -0.98481   -9.84808
```

7.2.2 Nature Run of Lorenz-96

In the script below, we set up model parameters, initialise the true state, set up the observation operator and generate a nature run and the observations.

1 Matlab Code for Lorenz '96 - Nature Run and its Observations

```
%==========================================================================
% Setup data for data assimilation cycle
%==========================================================================
```

```
randn('seed',cc);
% use rng instead of randn
%-------------------------------------------------------------------------
% 1) Setup important parameters
%-------------------------------------------------------------------------
% Model parameters
%F = 8;          % forcing term for the true state
%N  = 50;        % model state dimension 50
%% Parameters for the numerical solution of the model
%dtime = 0.3;  % steps of integration
%% Parameters for generating observations and in
%noise   = 0.1; % noise factor on measurements  % standard deviation of observations
%-------------------------------------------------------------------------
% 2) Initialize true state
%-------------------------------------------------------------------------
%x0     = F * ones(N, 1);       % truth state at t = 0
%x0(N/2) = x0(N/2) + F/1000;  % introduce a small perturbation
x0 = 5*sin(2*pi*3*(1:N)'/N);
x = x0;                         % initial condition
%-------------------------------------------------------------------------
% 3) Setup Observation operator
%-------------------------------------------------------------------------
Hcase = 1;  % select type of H operator 1 = points, 2 = integrals
%obsloc  = 1:N; % observing every variable
obsloc = 1 : hH : N; % observing only every second variable
mobs = length(obsloc);   % number of observations
H = zeros(mobs, N);
% locations and matrix of differences
switch(Hcase)
   case 1
      % observations josen at obsloc, observing the state at these points
      H      = zeros(mobs, N);
      for i  = 1 : mobs
        H(i, obsloc(i)) = 1;
      end
 case 2
      v1org  = cos(0:2*pi/N:2*pi-1/N)';
      v2org  = sin(0:2*pi/N:2*pi-1/N)';
      v1 = v1org(obsloc');
      v2 = v2org(obsloc');
      v1mat  = repmat(v1,1,N);
      v2mat  = repmat(v2,1,N);
      v1matp = repmat(v1org',mobs,1);
      v2matp = repmat(v2org',mobs,1);
      vmat   = sqrt( (v1mat-v1matp).^2 + (v2mat-v2matp).^2 );
      % non-local H operator
      %H = real(exp(-vmat.^2*1)>0.9);
```

```
51        H = real(exp(-vmat.^2*1))>0.9;
52  %endswitch
53  end
54  %-------------------------------------------------------------------------------
55  % 4) Generate a "nature" run and the "observations"
56  %-------------------------------------------------------------------------------
57  for j=1:Nnat
58    % time integration of the initial true state
59    [x,Y,yy] = sim_06_4_1_Lorenz96(x,dtime,h,F,J,H,ko,sc,ocase);
60    xv(:,j) = x;       % save it in xv
61    y(:,j) = yy + noise*randn(size(yy,1),1);         % observation + noise

62  end
63  disp(['Nnat = ' num2str(Nnat) ' cycling steps prepared']);
```

7.2.3 Learning Coefficients of Model Representation based on the Kalman Filter

The scripts below describe the model parameters for the L96 model including that used for learning the model error.

The Lorenz '96 script for carrying out the learning algorithm based on the Kalman Filter is presented below. The code demonstrates the learning approach for the model error term or the model itself using the Kalman filter method used in Section 3.2 above. First, we show the initialized values for all the terms used in the equation in Part A and Part B provides the view of the error term for the model, coefficient observation operator Hc, the Kalman matrix and the model coefficients updates.

1 Matlab Code for Lorenz '96 - Set model parameter used for model error learning (Part A)

```
1  Nc = size(F,1);           % size of coefficient vector
2  cb = zeros(Nc,1);         % initial model bias coefficient vector c
3  %cb = F;                  %
4  cb0 = cb;
5  Bc  = 1*eye(Nc,Nc);       % covariance matrix for model coefficients
6  Bc0 = Bc;                 % initial Bc matrix
7  method_version = 'KF'     % choose '3D','4D' or 'KF'
8  Hc_matrix = [];           % initialization of complete Hc_matrix
9  r0  = 0.5;                % model error for learning
10 Rc  = r0 * eye(N,N);      % model error for learning
11 aci = 0.0000;             % factor for control of additive covariance inflation
```

2 Matlab Code for Lorenz '96 - Learning the model error or the model itself (Part B)

```
%
% Learning the model error or the model itself
%
%e_ba(:,j)  = (xbm - xa_old)/h;       % model error term
e_ba(:,j)  = (xam - xa_old)/h;       % model error term
Hc = sim_06_5_8_setup_Hc(xa_old,J);  % coefficient observation operator Hc
xbv_Hc(:,j) = xa_old + Hc*cb*h;
emb(:,j)    = xbm - xbv_Hc(:,j);     % current error
cbv(:,j)    = cb;
%
if( strcmp(method_version,'3D') )
  Kc = Bc*Hc'*inv(Rc + Hc*Bc*Hc');    % Kalman matrix
  ca = cb + Kc*(e_ba(:,j)-Hc*cb);     % ... 3DVAR
elseif( strcmp(method_version,'KF') )
  Kc = Bc*Hc'*inv(Rc + Hc*Bc*Hc');    % Kalman matrix
  ca = cb + Kc*(e_ba(:,j)-Hc*cb);     % ... KF
  Bc = (eye(Nc,Nc)-Kc*Hc)*Bc+aci*Bc0; % update Bc matrix
elseif( strcmp(method_version,'4D') )
  if( j>1 )
    e_ba_full = [e_ba_full; e_ba(:,j)];
  else
    e_ba_full = e_ba(:,j);
  end
  Hc_matrix = [Hc_matrix ; Hc];       % setup 4d observation operator
  Kc_matrix = Bc0*Hc_matrix'*inv(noise^2*eye(N*j,N*j)+Hc_matrix*Bc0*Hc_matrix');
  ca = Kc_matrix*(e_ba_full);         % ... 4DVAR
end
cb = ca;                              % update model coefficients
```

3 Matlab Code for Lorenz '96 - Hc operator

```
function Hc = sim_06_5_8_setup_Hc(x,J)

  n = length(x);             % number of variables
  jv   = (1:n)';             % vector of indices

  js(:,J+1) = jv;
  for j=1:J
    js(:,J-j+1) = jv([(end-j+1):end 1:(end-j)]);
    js(:,J+1+j) = jv([(1+j):end 1:j]);
  end
  % Hc first column
  Hc(:,1) = ones(n,1);       % add constant term
  j0 = -2*J-1;               % start of counter for -2*J:2*J when
  % Hc columns j=1,...,2*j+1 next, loop for linear terms
```

```
  for j=1:(2*J+1)
    Hc(:,1+j)= x(js(:,j));
  end
  % Hc columns for pure quadratic terms
  j1 = 1 + (2*J+1);          % start of counter
  for j=1:(2*J+1)
     Hc(:, j1+j) = x(js(:,j)).^2;
  end
  % Hc columns loop for fully mixed quadratic terms
  j2 = 1+2*(2*J+1);          % start of counter
  vs = 0;                    % counting upper diagonal terms
  for j=1:(2*J+1)
    for k=(j+1):(2*J+1)
      vs = vs + 1;
      Hc(:, j2+vs ) = x(js(:,j)) .* x(js(:,k));
    end

  end
```

4 Matlab Code for Lorenz '96 - Visualize learning model outputs

```
ca2 = max(max(abs(xv)));
ca1 = -ca2;

%-------------------------------------------------------------------------------
%
%-------------------------------------------------------------------------------
fo = figure();

subplot(3,1,1);
surf(xbv);
view(2);
caxis([ca1 ca2])
shading interp;
set(gca,'FontSize',14)
axis tight;
colorbar;
%axis('label[y]')
ylabel({'[y]'})
title('First guess field')


subplot(3,1,2);
surf(xbv_Hc);
view(2);
caxis([ca1 ca2])
shading interp;
```

```
23 set(gca,'FontSize',14)
24 axis tight;
25 colorbar;
26 %axis('label[y]')
27 ylabel({'[y]'})
28 title('First guess field from Model Reconstruction')

29 subplot(3,1,3);
30 surf(xbv_Hc-xv);
31 view(2);
32 caxis([ca1 ca2])
33 shading interp;
34 set(gca,'FontSize',14)
35 axis tight;
36 colorbar;
37 title('Model Reconstruction error')
38 savefile(fo,'model_recon_error_first_guess_field');

39 %-------------------------------------------------------------------------
40 %
41 %-------------------------------------------------------------------------
42 eba = xbv-xav;

43 fo = figure;
44 plot( mean(abs(emb)) ,'LineWidth',2);
45 hold on;
46 plot( mean(abs(eba)), 'r--','LineWidth',2);
47 plot( mean((xbv)), 'k:', 'LineWidth',2);
48 plot( mean((xbv_Hc)), 'b--', 'LineWidth',2);
49 set(gca,'FontSize',14)
50 legend('model reconstruction error','first guess error','mean fg', ...
51           'mean fg\_rec');
52 axis( [1 Nnat 0 3] )
53 savefile(fo,'model_recon_error_mean_first_guess_recon');

54 %-------------------------------------------------------------------------
55 %
56 %-------------------------------------------------------------------------
57 fo = figure;
58 bar([F cb]);
59 set(gca,'FontSize',14)
60 legend('original coefficients','reconstructed coefficients');
61 savefile(fo,'orig_recon_coefficients');

62 %-------------------------------------------------------------------------
63 %
64 %-------------------------------------------------------------------------
```

```
fo = figure;
surf( cbv-repmat(F,1,Nnat) );
view(2);
shading flat;
set(gca,'FontSize',14)
axis tight;
colorbar;
title('Coefficient Reconstruction Error')
savefile(fo,'coefficient_recon_error');
```

The figure generated by the above code example is shown in Figure 4.12.

7.2.4 Implement Local Ensemble Transform Kalman Filter (LETKF)

The script below was used to implement the LETKF approach. It shows the convergence of the model dynamics and the observation data.

This approach also helps to provide better initial conditions both for the main and ensemble forecasts. In a nutshell, it provides an analysis state obtained by performing localized analyses at each model grid point.

1 Matlab Code for Lorenz '96 - LETKF Implementation

```
%=========================================================================
% Local Ensemble Transform Kalman Filter (LETKF)
%=========================================================================
rand('seed',cc)         % use the same random numbers to achieve repeatability
randn('seed',cc)         % use the same random numbers to achieve repeatability
m = size(yy,1);         % number of observations
R = noise^2*eye(m,m);   % data error covariance matrix
method = 'LETKF';       % method for display and plotting choices
mymodel = 'Lorenz' %'Hc_model'      % choose 'Lorenz' or 'Hc_model'
% mymodel = 'Hc_model'      % choose 'Lorenz' or 'Hc_model'

% generate the initial ensemble
xa = repmat(x0,1,L) + 0.5*(rand(N,L)-0.5);  xa2 = xa;
xam = mean(xa')'; xa_old = xam;

spread_control = repmat(s_control_start_LETKF,N,1); % initial spread control
%c_aci              = 0.0; % additive covariance inflation constant
I=eye(L,L);               % standard identity matrix

% Initialization of Model Learning
sim_06_5_9_initialize_learning;
%-------------------------------------------------------------------------
```

```
% Setup the localization
%---------------------------------------------------------------------------
v1     = cos(0:2*pi/N:2*pi-1/N)';
v2     = sin(0:2*pi/N:2*pi-1/N)';
v1mat  = repmat(v1,1,N);
v2mat  = repmat(v2,1,N);
vmat   = sqrt( (v1mat-v1mat').^2 + (v2mat-v2mat').^2 );

for j=1:Nnat

  switch(mymodel)
  case 'Lorenz'
    [xb,Y,yy,yym] = sim_06_4_1_Lorenz96(xa,dtime,h,Fm,J,H,ko,sc,ocase);
  case 'Hc_model'
    for ll=1:L
      Hc = sim_06_5_8_setup_Hc(xa(:,ll),J);% coefficient observation operator Hc
      xb(:,ll) = xa(:,ll) + Hc*cb*h;
    end
    xb_ff(:,2,:)=xb;
    type =0;
    [Y,yy,yym] = sim_06_2_1_Y(xb_ff,H,type,ko,sc,ocase);
  end
  xbm = (sum(xb'))'/L;                % and its mean
  xbv(:,j) = xbm;                     % save the mean in xbv
  xbvv(:,j,:) = xb;                   % the full ensemble in xbvv

  % carry out core LETKF step
  sim_06_5_3_LETKF_96_loc_core;

  xavv(:,j,:) = xa;   % save full analysis ensemble
  xam = (sum(xa'))'/L;   % calculate analysis mean
  xav(:,j)=xam;       % save analysis mean

  %---------------------------------------------------------------------------
  % Calculate errors
  %---------------------------------------------------------------------------
  evb(j) = norm(y(:,j) - yym)/sqrt(N); % the "observation-first guess" error
  eva(j) = norm(y(:,j) - H*xav(:,j))/sqrt(N); % the "observation-first guess"
  ea(j) = norm(xav(:,j)-xv(:,j))/sqrt(N);     % error of the mean (analysis)
  eb(j) = norm(xbv(:,j)-xv(:,j))/sqrt(N);     % error of the mean (first guess)

  % model learning part
  sim_06_5_9_learning;

  xa_old = xam;

end
```

```
59  eb_LETKF_mean = mean(eb)              % mean first guess error
60  ea_LETKF_mean = mean(ea)              % mean analysis error
61  eb_LETKF_ll = sqrt(sum(eb.^2)/Nnat)   % first guess L2 error (normalized to Nnat)
62  ea_LETKF_ll = sqrt(sum(ea.^2)/Nnat)   % analysis L2 error (normalized to Nnat)
```

7.3 Neuro Reconstruction for Neural Kernel Estimation

This section shows the code used for the Gaussian Model Reconstruction and the Kalman Learning approach used for the Kernel estimation in equation (4.10) and Figure 4.16 respectively. In addition, the sensitivity analysis for the neuro kalman filter and the corresponding outputs in Figure 5.12 are also shown below.

1 Matlab Code for reconstructions of the different levels of observation errors

```
1   disp('--- control.m ---------------- ')
2   clc
3   clear all;
4   close all;

5   p1_setup;
6   p2_field_input;
7   p3_inverse;
8   p4_field_simulation;
9   %p5_model_reconstruction;
10  for joe = 1:10
11    obs_err=(0.1*joe-0.0999);
12    p7_neuro_kalman_filter
13    p8_neuro_show_dynamics
14    checking_errors;
15    obs_err_v(joe) = obs_err;
16    rec_err_v(joe) = e_ba3_norm;
17  end

18  fo10 = figure;
19  plot(obs_err_v,rec_err_v,'MarkerSize',15,'LineWidth',3);
20  xlabel('Observation Error')
21  ylabel('Reconstruction Error')
22  set(gca,'FontSize',14)
23  saveas(fo10,'error_dependencies.png','png')
```

The sensitivity and statistical analysis of the neuro kalman filter reconstruc-

tion of the results to the different levels of observational noise is carried out by the following code.

2 Matlab Code for the Sensitivity Analysis of the Neuro Kalman Filter Reconstruction

```
disp('--- p2_field_input.m ------------')

sigma  = 3;               % decay parameter for Gaussian type excitation
uv     = zeros(N,Nt);     % vector to collect excitation fields
umat   = zeros(n2,n1);    % matrix to display excitation area in 2D
mycase = 2;                % case of curve dynamics

for j=1:Nt                % time loop
  switch mycase
    case 1
      % parabula type curve of centre
      c1v(j) = 0.2+a1*0.8/Nt*j;
      c2v(j) = 0.2+3*a1/Nt*j - 3/a1*(a1/Nt*j)^2;
    case 2
      % elliptic centre curve over time
      nRo = 3;            % number of times to pass through circle
      c1v(j) = a1/2 + a1/4*cos(nRo*2*pi*j/Nt);
      c2v(j) = a2/2 - a2/4*sin(nRo*2*pi*j/Nt);
  end
  %endswitch

        umat = exp(-sigma*( (X1-c1v(j)).^2 + (X2-c2v(j)).^2 ));
        uv(:,j) = reshape(umat,N,1);
end


%-------------------------------------------------------------------
% now the graphics area
%-------------------------------------------------------------------
if( 1==0 )

for j=1:Nt
        umat = reshape(uv(:,j),n2,n1);
        fobj = figure(100);
        surf(X1,X2,umat);
        view(2);
        shading interp;
        hold off;
        pause(.1);
end

end
```

The original neural reconstruction is carried out by the following code.

3 Matlab Code for Neural Field Equation - Neuro Reconstruction

```
disp('--- p3_inverse.m ----------------')
for j=1:(Nt-1)
  % setting up psi and phi in neural field equation integration form
  %     psi = Wmat* phi
  psi(:,j) = tau*(uv(:,j+1)-uv(:,j))/ht0 + uv(:,j);
  phi(:,j) = f(uv(:,j),eta);
end

%-----------------------------------------------
% solve the inverse problem next
%-----------------------------------------------
% solving for all time steps simultaneously rewriting psi' = phi'*Wmat'
A = phi';
rhs = psi';          %
alpha = 0.01;
nn = size(A,2);
% solving A*(Wmat') = rhs based on Tikhonov regularization
Wmat = (inv(alpha*eye(nn,nn)+A'*A )\A'*rhs)';

%-----------------------------------------------
% display the reconstructed kernel Wmat
%-----------------------------------------------
if( 1==0 )
  dn1 = 7;
  dn2 = 7;
  for j1 = 1:dn1:n1
    for j2=1:dn2:n2
      jj = (n2-1)*j1+j2;
      disp(['j1, j2, jj = ' num2str(j1), ' ', num2str(j2), ' ', num2str(jj)])
      w = reshape(Wmat(:,jj),n2,n1);
      fobj200=figure(200);
      surf(X1,X2,flip(w));
      shading interp;
      hold on;
      %plot3(X2v(jj),X1v(jj),10,'k.','MarkerSize',15,'Color',[1 1 1]);
      plot3(xv1(j1),xv2(j2),10,'k.','MarkerSize',15,'Color',[1 1 1]);
      view(2);
      clim([-.1 .1])
      colorbar;
      axis tight;
      hold off;
      pause(.5);
    end
  end
```

```
43 end
```

In this script, we show what the neural kernel based dynamics will generate, which is contained in the field usimv.

4 Matlab Code for displaying the neural kernel field simulations

```
1  disp('--- p4_field_simulation.m -------')
2  ht = ht0*1;
3  usimv = zeros(nn,Nt); % generate neural kernel field dynamics
4  usimv(:,1) = uv(:,1);

5  for j=1:Nt-1
6          dsim = ht/tau*(-usimv(:,j) + Wmat*f(usimv(:,j),eta) );
7          usimv(:,j+1) = usimv(:,j)+dsim;
8  end


9  %---------------------------------------------------------------------
10 % now the graphics area
11 %---------------------------------------------------------------------
12 if( 1==0 )

13 for j=1:Nt
14         % simulated field after kernel reconstruction
15         umat = reshape(usimv(:,j),n2,n1);
16         % original field prescribed and fed into kernel reconstruction
17         umat2 = reshape(uv(:,j),n2,n1);

18         fobj = figure(100);
19   clf(100)

20         subplot(2,1,1)
21         surf(X1,X2,umat+1e-3);
22         view(2);
23         shading interp;
24   colorbar
25         hold off;
26         title('Dynamics from NFE')
27         %
28         subplot(2,1,2)
29         surf(X1,X2,umat2+1e-3);
30         view(2);
31         shading interp;
32   colorbar
33         hold off;
```

```
34          title('Prescribed Dynamics')
35          %
36          pause(.1);
37  end

38  end
```

This script allows the user to pick a point in two-dimensional space by a mouse click and then the corresponding column of the kernel W is displayed as a two-dimensional field showing the influence of a field at the chosen point to all other fields in the two-dimensional neural domain.

5 Matlab Code for two-dimensional neural domain

```
1   jj = ceil(N/2);

2   while(jj>0)
3   w = reshape(Wmat(:,jj),n2,n1);

4   fobj200=figure(200);
5   surf(X1,X2,w);
6   hold on;
7   plot3(X1v(jj),X2v(jj),10,'k.','MarkerSize',15);
8   plot3(c1v,c2v,ones(size(c1v)),'k.');
9   shading interp;
10  view(2);
11  clim([-.1 .1])
12  colorbar;
13  hold off;

14  [x,y]=ginput(1);
15  jx = ceil(x/a1*n1);
16  jy = ceil(y/a2*n2);
17  jj = n2*(jx-1)+jy;

18  end
```

This script is used to carry out the model reconstruction for the kalman filter approach.

6 Matlab Code for carrying out the neuro kalman filter model reconstruction

```
1   %
2   % Learning the model error or the model itself
```

```
%
disp('learning ...')
method_version = 'KF';
represent1 = 3;                     % case for representation of W
switch (represent1)
  case 1, 2
    Nc = nn*nn;
case 3
    Nc = 2*(Nt-1);
    p9_Gmat_setup;
    clear Hc
end
cb = zeros(Nc,1);
cbv = zeros(Nc,Nt-1);
% Our model representation is connecting each variable with each variable
Bc         = eye(Nc,Nc);            % model space covariance matrix
Bc0        = Bc;                    % for KF additive covariance inflation ACI
aci        = 0;                     % aci strength
Rc         = obs_err^2*eye(nn,nn);        % observation error covariance matrixx

randn('seed',0);
uv_e = uv + obs_err*randn(size(uv)); % add random lerror to truth
                                      % to simulate observation error

disp('... entering time loop ...')
for j=1:Nt-1
  str1 = sprintf("%d, ", j);
  fprintf(str1)

  Hc = zeros(nn,Nc);
  switch( represent1 )
    case 1
      % model observation operator
      % Ansatz du = M * u with du in R^nn, M in R^(nn x nn),
      for jj=1:nn
        Hc(jj,(nn*(jj-1)+1):(nn*jj)) = uv(:,j)';
      end
      e_ba(:,j)  = (uv(:,j+1) - uv(:,j))';  % term to learn
    case 2
      % model observation operator
      % Ansatz du = -u + W f(u)
      for jj=1:nn
        Hc(jj,(nn*(jj-1)+1):(nn*jj)) = f(uv(:,j)',eta);
      end
      e_ba(:,j) = uv(:,j+1)-(1-ht)*uv(:,j);
    case 3
      % model observation operator
```

```
47        % Ansatz du = -u + W f(u) with W = sum G_xi c_xi
48        for jj=1:Nt-1
49          Hc(:,2*jj-1) = reshape(GG(:,jj+1)*GG2(:,jj)',nn,nn)* f(uv(:,j),eta);
50          if( jj > 1 )
51            Hc(:,2*jj) = reshape(GG(:,jj-1)*GG2(:,jj)',nn,nn)* f(uv(:,j),eta);
52          else
53            Hc(:,2*jj) = reshape(zeros(nn,1)*GG2(:,jj)',nn,nn)* f(uv(:,j),eta);
54          end
55        end
56        e_ba(:,j) = uv_e(:,j+1)-(1-ht)*uv_e(:,j);
57    end
58    cbv(:,j)   = cb;                        % save current vector
59    e_bav(:,j) = e_ba(:,j)-Hc*cb;           % current error
60    %
61    if( strcmp(method_version,'3D') )
62      Kc = Bc*Hc'*inv(Rc + Hc*Bc*Hc');      % Kalman matrix
63      ca = cb + Kc*(e_ba(:,j)-Hc*cb);       % ... 3DVAR
64    elseif( strcmp(method_version,'KF') )
65      Kc = Bc*Hc'*inv(Rc + Hc*Bc*Hc');      % Kalman matrix
66      ca = cb + Kc*(e_ba(:,j)-Hc*cb);       % ... KF
67      Bc = (eye(Nc,Nc)-Kc*Hc)*Bc+aci*Bc0; % update Kalman matrix
68    elseif( strcmp(method_version,'4D') )
69      if( j>1 )
70        e_ba_full = [e_ba_full; e_ba(:,j)];
71      else
72        e_ba_full = e_ba(:,j);
73      end
74      Hc_matrix = [Hc_matrix ; Hc];         % setup 4d observation operator
75      Kc_matrix = Bc0*Hc_matrix'*inv(noise^2*eye(N*j,N*j)+Hc_matrix*Bc0*Hc_matrix');
76      ca = Kc_matrix*(e_ba_full);           % ... 4DVAR
77      end
78    cb = ca;                                % update model coefficients
79  end
80  disp('')
```

This script allows us to set up the Gaussian functions.

6 Matlab Code for for setting up the kernel functions

```
1   for j=1:Nt                  % time loop
2           GG(:,j) = reshape(exp(-sigma*( (X1-c1v(j)).^2 + (X2-c2v(j)).^2 )),nn,1);
3     sigma2 = 10;
4           GG2(:,j) = reshape(exp(-sigma2*( (X1-c1v(j)).^2 + (X2-c2v(j)).^2 )),nn,1);
5   end
```

7.4 PDE for Weather Simulation

This section shows the code used for learning the PDE by the Kalman filter approach.

```
disp(['--- p8_Kalman_learning.m ---']);

tic;
%method = '3D';            % define the method to use
method = 'KF';
nn = n1*n2;               % number of points in 2d grid
disp(['nn = ' num2str(nn) ' steps to do!']);
bv2 = zeros(ns,nn);        % initialization
svv = zeros(nt,nn);       % ~
svv2 = zeros(nt,nn);      % ~
nx = 2;                   % number of input coordinates around current location
ny = 2;                   % ~
ns = (2*nx+1)*(2*ny+1);   % total number of input coordinates
for jj=1:nn
  Bcv(jj,:,:) = eye(ns,ns);          % initialization of Bc Matrix
end
Rc = 0.1;                 % observation error, here it is a scalar only
%
svv = zeros(nt-1,nn);       % ~
svv2 = zeros(nt-1,nn);      % ~
for jt = 1: nt-1
  t = toc;                % calculate time estimate
  tt = t/jt*nt;        % ~
  str1 = sprintf("\r\r\r\r\r\r\r%10s",[num2str(t,4) '/' num2str(tt,4)]);
  fprintf([str1 ' seconds         ']);
  jj = 1;                   % counter for points in spatial grid
  for jx = 1:1:n1;
    for jy = 1:1:n2;
      js = n2*(jx-1) + jy;  % current point where variable to learn is located
      p1v(jj) = a1/n1*jx;   % point coordinate 1
      p2v(jj) = a2/n2*jy;   % point coordinate 2

      sv = uv(js,jt+1)'; % right-hand side for point js

      umat  = reshape(uv(:,jt),n2,n1);
      usmat = umat(mymod([-ny:ny]+jy,n1),mymod([-nx:nx]+jx,n1));
      Hc    = reshape(usmat,1,ns);

      % learn the coefficients
      alpha = 0.0001;        % regularization parameter for learning
      %b = inv( alpha*eye(ns,ns)+ Hc'*Hc)*Hc'*sv;
      bb = bv2(:,jj);
```

```
39        Bc = reshape(Bcv(jj,:,:),ns,ns);
40        Kc = Bc*Hc'*inv(alpha+Rc + Hc*Bc*Hc');
41        ba = bb + Kc*(sv - Hc*bb);
42        if( method == 'KF')
43          Bc2 = (eye(ns,ns)-Kc*Hc)*Bc;
44        else
45          Bc2 = Bc;
46        end
47        Bcv(jj,:,:) = Bc2;
48        %bmat = reshape(b,(2*ny+1),(2*nx+1));

49        % save for test and display
50        bv2(:,jj) = ba;
51        svv(:,jj) = sv;

52        sv2 = Hc*b;
53        svv2(:,jj) = sv2;

54        jj = jj + 1;
55      end
56    end
57  end

58  t = toc;
59  disp(['Time for complete learning of this field t=' num2str(t) ' sec']);

60  disp('Testing the learned coefficients in comparison:');
61  jj = ceil(nn/3);
62  disp([bv(1:10,jj) bv2(1:10,jj)]);
63  disp([' norm(bv - bv2)/norm(bv) = ' num2str( norm(bv-bv2)/norm(bv) ) ])
```

Bibliography

[1] A. Adadi, M. Lahmer, and S. Nasiri. Artificial intelligence and covid-19: A systematic umbrella review and roads ahead. *Journal of King Saud University - Computer and Information Sciences*, 34(8, Part B):5898–5920, 2022.

[2] M. Al-Emran, M. Al-Kabi, and G. Marques. *A Survey of Using Machine Learning Algorithms During the COVID-19 Pandemic*, pages 1–8. Research Gate, 03 2021.

[3] B. I. Alliance. The brain research through advancing innovative neurotechnologies® (brain) initiative. *The Brain Initiative*, 2013.

[4] J. Alswaihli, R. Potthast, A. Hutt, D. Saddy, and I. Bojak. Kernel reconstruction for delayed neural field equations. *Pre-print*, 2017. Preprint.

[5] S. Amari. Dynamics of patterns formation in lateral-inhibition type neural fields. *Biolobical Cybernetics*, 27:77–87, 1977.

[6] D. Amsallem, M. J. Zahr, and C. Farhat. Nonlinear model order reduction based on local reduced-order bases. *International Journal for Numerical Methods in Engineering*, 92(10):891–916, 2012.

[7] J. L. Anderson and S. L. Anderson. A monte carlo implementation of the nonlinear filtering problem to produce ensemble assimilations and forecasts. *Monthly Weather Review*, 127(12):2741 – 2758, 1999.

[8] H. M. Arnold, I. M. Moroz, and T. N. Palmer. Stochastic parametrizations and model uncertainty in the lorenz '96 system. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1991):20110479, 2013.

[9] M. M. Arzhanov, V. V. Malakhova, I. I. Mokhov, and M. R. Parfenova. Stability of relic methane hydrates under climatic changes in the holocene.

IOP Conference Series: Earth and Environmental Science, 386(1):012019, nov 2019.

[10] M. Asch, M. Bocquet, and M. Nodet. *Data Assimilation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2016.

[11] R. Aster, B. Borchers, and C. Thurber. Parameter estimation and inverse problems. *Recherche*, 67:02, 01 2012.

[12] L. Bangjun, X. Guangzhu, et al. Parameter estimation. *Ebook Central*, pages 77–113, 2017.

[13] A. G. Barto and R. S. Sutton. Chapter 19 - reinforcement learning in artificial intelligence. In J. W. Donahoe and V. Packard Dorsel, editors, *Neural-Network Models of Cognition*, volume 121 of *Advances in Psychology*, pages 358–386. North-Holland, 1997.

[14] C. Batlle and N. Roqueiro. Balanced model order reduction method for systems depending on a parameter. *IFAC-PapersOnLine*, 52(1):412–417, 2019. 12th IFAC Symposium on Dynamics and Control of Process Systems, including Biosystems DYCOPS 2019.

[15] P. Beim Graben and R. Potthast. Inverse problems in dynamic cognitive modeling. *Chaos (Woodbury, N.Y.)*, 19:015103, 04 2009.

[16] G. D. Birkhoff. *Dynamical Systems*. American Mathematical Society, 1927.

[17] BrainBox. The brain box initiative. *Brainbox Initiative*, 2016.

[18] J. Brajard, A. Carrassi, M. Bocquet, and L. Bertino. Combining data assimilation and machine learning to emulate a dynamical model from sparse and noisy observations: A case study with the lorenz 96 model. *Journal of Computational Science*, 44:101171, 2020.

[19] J. Brajard, A. Carrassi, M. Bocquet, and L. Bertino. Combining data assimilation and machine learning to infer unresolved scale parametrization. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2194):20200086, 2021.

[20] M. Breakspear. Dynamic models of large-scale brain activity. *Nat Neurosci 20*, pages 340–352, 2017.

[21] P. Bressloff. Spatiotemporal dynamics of continuum neural fields. *Journal of Physics A: Mathematical and Theoretical*, 45:033001, 12 2011.

[22] P. C. Bressloff and S. Coombes. Physics of the extended neuron. *International Journal of Modern Physics B*, 11, 1997.

[23] C. Buizza, C. Quilodrán Casas, P. Nadler, J. Mack, S. Marrone, Z. Titus, C. Le Cornec, E. Heylen, T. Dur, L. Baca Ruiz, C. Heaney, J. A. Díaz Lopez, K. S. Kumar, and R. Arcucci. Data learning: Integrating data assimilation and machine learning. *Journal of Computational Science*, 58:101525, 2022.

[24] M. Burger, H. Dirks, and J. Mueller. Inverse problems in imaging. In M. Cullen, M. A. Freitag, S. Kindermann, and R. Scheichl, editors, *Large Scale Inverse Problems*, volume 13 of *Radon Series on Computational and Applied Mathematics*. Walter de Gruyter, Berlin, 2013.

[25] M. Carlu, F. Ginelli, V. Lucarini, and A. Politi. Lyapunov analysis of multiscale dynamics: the slow bundle of the two-scale lorenz 96 model. *Nonlinear Processes in Geophysics*, 26(2):73–89, 2020.

[26] C. Q. Casas, R. Arcucci, P. Wu, C. Pain, and Y.-K. Guo. A reduced order deep data assimilation model. *Physica D: Nonlinear Phenomena*, 412:132615, 2020.

[27] S. Chen, R. Yang, R. Yang, L. Yang, X. Yang, C. Xu, B. Xu, H. Zhang, Y. Lu, and W. Liu. A parameter estimation method for nonlinear systems based on improved boundary chicken swarm optimization. *Discrete Dynamics in Nature and Society*, 2016:1–11, 01 2016.

[28] M. C. Cieslak, A. M. Castelfranco, V. Roncalli, P. H. Lenz, and D. K. Hartline. t-distributed stochastic neighbor embedding (t-sne): A tool for eco-physiological transcriptomic analysis. *Marine Genomics*, 51:100723, 2020.

[29] B. J. Cook, A. D. H. Peterson, W. Woldman, and J. R. Terry. Neural field models: A mathematical overview and unifying framework. *Mathematical Neuroscience and Applications*, Volume 2, mar 2022.

[30] S. Coombes, P. beim Graben, and R. Potthast. *Tutorial on Neural Field Theory*, pages 1–43. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[31] S. Coombes, P. beim Graben, R. Potthast, and J. Wright, editors. *Neural Fields: Theory and Applications*. Springer-Verlag Berlin Heidelberg, 2014.

[32] S. Coombes, H. Schmidt, and I. Bojak. Interface dynamics in planar neural field models. *Journal of mathematical neuroscience*, 2:9, 05 2012.

[33] A. Corigliano, M. Dossi, and S. Mariani. Model order reduction and domain decomposition strategies for the solution of the dynamic elastic–plastic structural problem. *Computer Methods in Applied Mechanics and Engineering*, 290:127–155, 2015.

[34] D. Dee. Variational bias correction of radiance data in the ecmwf system. *ECMWF Workshop on Assimilation of High Spectral Resolution Sounders in NWP*, pages 97–112, 01 2004.

[35] J. Doherty and R. J. Hunt. Two statistics for evaluating parameter identifiability and error reduction. *Journal of Hydrology*, 366(1):119–127, 2009.

[36] I. C. Education. What is machine learning. *IBM Cloud Learn Hub*, 2020.

[37] F. Engert. The big data problem: Turning maps into knowledge. *Neuron*, 83(6):1246–1248, 2014.

[38] H. W. Engl, M. Hankle, and A. Neubauer. *Regularization of Inverse Problems*. Mathematics and Its Applications. Springer Netherlands, 2000.

[39] G. Evensen, G. Burgers, and P. J. van Leeuwen. Analysis scheme in the ensemble Kalman filter. *Monthly Weather Review*, 126(6):1719–1724, 1998.

[40] G. Evensen and P. J. van Leeuwen. An ensemble kalman smoother for nonlinear dynamics. *Monthly Weather Review*, 128:1852–1867, 2000.

[41] H. Festjens, G. Chevallier, and J. Dion. Nonlinear model order reduction of jointed structures for dynamic analysis. *Journal of Sound and Vibration*, 333(7):2100–2113, 2014.

[42] G. R. Francisco Chinesta, Antonio Huerta and K. Willcox. Model order reduction. *Encyclopedia of Computational Mechanics*, pages 1–59, 2004.

[43] M. Freitag and R. Potthast. Synergy of inverse problems and data assimilation techniques. In M. Cullen, M. A. Freitag, S. Kindermann, and R. Scheichl, editors, *Large Scale Inverse Problems*, volume 13 of *Radon Series on Computational and Applied Mathematics*. Walter de Gruyter, Berlin, 2013.

[44] P. Galetsi, K. Katsaliaki, and S. Kumar. The medical and societal impact of big data analytics and artificial intelligence applications in combating pandemics: A review focused on covid-19. *Social Science and Medicine*, 301:114973, 2022.

[45] F. Galvanin, E. Cao, N. Al-Rifai, A. Gavriilidis, and V. Dua. *Model-based design of experiments for the identification of kinetic models in microreactor platforms*. Research Gate, 12 2015.

[46] M. E. Gharamti, I. Hoteit, and J. Valstar. Dual states estimation of a subsurface flow-transport coupled model using ensemble kalman filtering. *Advances in Water Resources*, 60:75–88, 2013.

[47] O. Ghattas and K. Willcox. Learning physics-based models from data: perspectives from inverse problems and model reduction. *Acta Numerica*, pages 445–554, 2021.

[48] M. Gianfelice, F. Maimone, V. Pelino, and S. Vaienti. On the recurrence and robust properties of lorenz'63 model. *Commun. Math. Phys.*, 313, 03 2011.

[49] C. W. Groetsch. *Inverse problems in the mathematical sciences*, volume 6 of *Theory and Practice of Applied Geophysics Series*. Vieweg, 1993.

[50] J. Guan, K. Simek, E. Brau, C. Morrison, E. Butler, and K. Barnard. Proceedings of the 32 nd international conference on machine learning, lille, france. In *Moderated and Drifting Linear Dynamical Systems*, volume 37, 07 2015.

[51] R. Gutenkunst, J. Waterfall, F. Casey, K. Brown, C. Myers, and J. Sethna. Universally sloppy parameter sensitivities in systems biology models. *PLOS Comput Biol*, 3:e189, 11 2007.

[52] R. Guzzi. *Data Assimilation: Mathematical Concepts and Instructive Examples*. Springer, 2016.

[53] D. Hansel and C. van Vreeswijk. The mechanism of orientation selectivity in primary visual cortex without a functional map. *Journal of Neuroscience*, 32(12):4049–4064, 2012.

[54] A. Heidari, N. Navimipour, M. Unal, and S. Toumaj. Machine learning applications for covid-19 outbreak management. *Neural Computing and Applications*, 34:15313–15348, 9 2022.

[55] G. Hooker. Forcing function diagnostics for nonlinear dynamics. *Biometrics*, 65(3):928–936, 2009.

[56] B. R. Hunt, E. J. Kostelich, and I. Szunyogh. Efficient data assimilation for spatiotemporal chaos: A local ensemble transform Kalman filter. *Physica D: Nonlinear Phenomena*, 230(1-2):112–126, 2007.

[57] C. Huntingford, E. S. Jeffers, M. B. Bonsall, H. M. Christensen, T. Lees, and H. Yang. Machine learning and artificial intelligence to aid climate change research and preparedness. *Environmental Research Letters*, 14(12):124007, nov 2019.

[58] H. S. W. Jorge A. Revelli, Miguel A. Rodriguez. Resonant phenomena in extended chaotic systems subject to external noise: The lorenz'96 model case. *Physica A*, 387:1–8, 03 2008.

[59] E. Kalnay. *Atmospheric Modeling, Data Assimilation and Predictability*. Cambridge University Press, 2002.

[60] B. Kaltenbacher, A. Neubauer, and O. Scherzer. *Iterative Regularization Methods for Nonlinear Ill-Posed Problems*. Radon Series on Computational and Applied Mathematics. De Gruyter, 2008.

[61] A. Karimi and M. R. Paul. Extensive chaos in the lorenz-96 model. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 20(4):043105, Dec 2010.

[62] M. Kawato. Reinforcement models. In *Encyclopedia of Neuroscience*, pages 89–97. Academic Press, Oxford, 2009.

[63] Y. Kim and H. Bang. *Introduction to Kalman Filter and Its Applications*, chapter Open access peer-reviewed chapter, pages 1–10. InTechOpen, 11 2018.

[64] B. Koo, H. Son, H. Kim, T. Jo, and J. Y. Yoon. Model-order reduction technique for temperature prediction and sensor placement in cylindrical steam reformer for ht–pemfc. *Applied Thermal Engineering*, 173:115153, 2020.

[65] S. Kotsuki, Y. Sato, and T. Miyoshi. Data assimilation for climate research: Model parameter estimation of large-scale condensation scheme. *Journal of Geophysical Research: Atmospheres*, 125(1):e2019JD031304, 2020. e2019JD031304 2019JD031304.

[66] R. Kotter. Neuroscience databases: tools for exploring brain structure-function relationships. *The Royal Society*, 356(1412):1111–20, 2001.

[67] R. Kress. *Linear Integral Equations*, volume 82 of *Applied Mathematical Sciences*. Springer New York, 1999.

[68] M. Lang, P. Van Leeuwen, and P. Browne. A systematic method of parameterisation estimation using data assimilation. *Tellus A: Dynamic Meteorology and Oceanograph*, 68(1):p.29012, 2016.

[69] J. W. Lichtman, H. Pfister, and N. Shavit. The big data challenges of connectomics. *Nature Neuroscience*, 17(11):1448–54, 2014.

[70] E. Lorenz. Deterministic nonperiodic flow, massachusetts institute of technology. *Journal of the Atmospheric Science*, 20, 1963.

[71] E. Lorenz. Predictability: a problem partly solved. *Seminar on Predictability, 4-8 September 1995*, 1:1–18, 1995 1995.

[72] Y. Luo, E. Weng, X. Wu, C. Gao, X. Zhou, and L. Zhang. Parameter identifiability, constraint, and equifinality in data assimilation with ecosystem models. *Ecological Applications*, 19(3):571–574, 2009.

[73] M. Maheu, S. Dahaene, and F. Meyniel. Brain signatures of a multiscale process of sequence learning in humans. *Neuroscience*, 8:e41451, 2019.

[74] K. Maik. How data assimilation helps illuminate complex biology. *SciPort RLP*, pages 1–31, 2017.

[75] H. Markram, E. Muller, Ramaswamy, et al. Reconstruction and simulation of neocortical microcircuitry. *Cell*, 163(2):456–92, 2015.

[76] S. Martina-Perez, M. J. Simpson, and R. E. Baker. Bayesian uncertainty quantification for data-driven equation learning. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 477(2254):20210426, 2021.

[77] MathWorks. Introducing machine learning, part (1-4). *Mathworks Tutorial*, 2016.

[78] L. McInnes, J. Healy, N. Saul, and L. Großberger. Umap: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861, 2018.

[79] S. Mishra Tiwari, O. Dogan, J. Akhil, S. K. Shandilya, F. Ortiz-Rodríguez, S. Bajpai, and S. Banerjee. *Applications of machine learning approaches to combat COVID-19: A survey. Lessons from COVID-19*, pages 263–287. PubMed, 01 2022.

[80] M. Moye and C. Diekman. Data assimilation methods for neuronal state and parameter estimation. *J Math Neuroscience*, Vol. 8, 2018.

[81] G. Nakamura and R. Potthast. *Inverse Modeling*. 2053-2563. IOP Publishing, 2015.

[82] A. Nogaret, C. Meliza, D. Margoliash, Abarbanel, and H. D. I. Automatic construction of predictive neuron models through large scale assimilation of electrophysiological data. *Scientific Reports*, 6:1–14, 2016.

[83] B. Odunuga, R. Potthast, and D. Saddy. On model reconstruction for dynamcial systems. *Pre-print*, 2017. Preprint.

[84] U. Parlitz, J. Schumann-Bischoff, and S. Luther. Local observability of state variables and parameters in nonlinear modeling quantified by delay reconstruction. *Chaos (Woodbury, N.Y.)*, 24:024411, 06 2014.

[85] Y. C. Peter Jan van Leeuwen and S. Reich. *Nonlinear Data Assimilation*. Frontiers in applied dynamical systems; v.2. Springer, 2015.

[86] A. M. Pinho, S. Casas, and L. Amendola. Model-independent reconstruction of the linear anisotropic stress. *Journal of Cosmology and Astroparticle Physics*, 2018(11):027–027, Nov 2018.

[87] R. Potthast. Inverse problems and data assimilation for brain equations - state and current challenges. *Conference Paper*, 2015.

[88] R. Potthast and P. Beim Graben. Existence and properties of solutions for neural field equations. *Mathematical Methods in the Applied Science*, 33(8):935–949, 2009.

[89] R. Potthast and P. Graben. Inverse problems in neural field theory. *SIAM Journal on Applied Dynamical Systems*, 8(4):1405–1433, 2009.

[90] G. J. Powers et al. The weather research and forecasting model: Overview, system efforts, and future directions. *Bulletin of the American Meteorological Society*, 98(8):1717 – 1737, 2017.

[91] D. Purves, G. J. Augustine, D. Fitzpatrick, et al. National center for biotechnology information, u.s. national library of medicine. In *Reconstruction of the Action Potential*. Neuroscience. 2nd edition. Sunderland (MA): Sinauer Associates, 2001.

[92] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng. A survey of machine learning for big data processing. *EURASIP Journal on Advances in Signal Processing*, 2016:67, 05 2016.

[93] M. Rabinovich and P. Varona. Robust transient dynamics and brain functions. *Frontiers in computational neuroscience*, 5:24, 06 2011.

[94] A. Raue, C. Kreutz, T. Maiwald, J. Bachmann, M. Schilling, U. Klingmüller, and J. Timmer. Structural and practical identifiability analysis of partially observed dynamical models by exploiting the profile likelihood. *Pubmed*, pages 1923–9, 6 2009.

[95] M. Rezapour, M. Niazi, and M. Gurcan. Machine learning-based analytics of the impact of the covid-19 pandemic on alcohol consumption habit changes among united states healthcare workers. *Scientific Reports 13*, page 6003, 4 2023.

[96] D. Rosen, R. L. Burden, J. D. Faires, and A. C. Reynolds. Numerical analysis. *The American mathematical monthly*, 87(3):231, 1980.

[97] A. Roskams and Z. Popovic. Power to the people: Addressing big data challenges in neuroscience by creating a new cadre of citizen neuroscientists. *Neuron*, 92:658–664, 11 2016.

[98] J. Samuel, P. Coulibaly, G. Dumedah, and H. Moradkhani. Assessing model state and forecasts variation in hydrologic data assimilation. *Journal of Hydrology*, 513:127–141, 2014.

[99] W. Schilders. *Introduction to Model Order Reduction*, pages 3–32. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[100] J. Schmidt, M. Marques, et al. Recent advances and applications of machine learning in solid-state materials science. *npj Computational Mater*, 5(83), 2019.

[101] R. S. Scorer. Atmospheric data analysis, roger daley, cambridge atmospheric and space science series, cambridge university press, cambridge, 1991. *International Journal of Climatology*, 12(7):763–764, 1992.

[102] Z. Shen, Y. Tang, X. Li, and Y. Gao. On the localization in strongly coupled ensemble data assimilation using a two-scale lorenz model. *Earth and Space Science*, 8(3):e2020EA001465, 2021. e2020EA001465 2020EA001465.

[103] D. Simon. *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. ProQuest Ebook Central. John Wiley & Sons, Incorporated, 2006.

[104] P. Smith, S. Dance, M. Baines, N. Nichols, and T. Scott. Variational data assimilation for parameter estimation: Application to a simple morphodynamic model. *Ocean Dynamics*, 59:697–708, 11 2009.

[105] P. Smith, G. Thornhill, S. Dance, A. Lawless, D. Mason, and N. Nichols. Data assimilation for state and parameter estimation: Application to morphodynamic modelling. *Quarterly Journal of the Royal Meteorological Society*, 139:314–327, 01 2013.

[106] S. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering.* CRC Press, 2nd edition edition, 2019.

[107] H. B. Syeda, M. Syed, K. Sexton, S. Syed, S. Begum, F. Syed, and F. Jr. The role of machine learning techniques to tackle covid-19 crisis: A systematic review. (preprint). *JMIR Medical Informatics*, 9, 08 2020.

[108] A. Tarantola. *Inverse Problem Theory and Methods for Model Parameter Estimation*. SIAM, 2005.

[109] S. Theodoridis. *Parameter Estimation*. Elsevier Science and Technology, ProQuest Ebook Central, 2015.

[110] R. Tomasello, M. Garagnani, T. Wennekers, and F. Pulvermüller. A neurobiologically constrained cortex model of semantic grounding with spiking neurons and brain-like connectivity. *Frontiers in Computational Neuroscience*, 12:88, 2018.

[111] M. Ursino, F. Cona, and E. Magosso. Mathematical models for computational neuroscience. *Modeling Methodology for Physiology and Medicine: Second Edition*, pages 311–332, 12 2013.

[112] Vaibhaw, J. Sarraf, and P. Pattnaik. Chapter 2 - brain–computer interfaces and their applications. In V. E. Balas, V. K. Solanki, and R. Kumar, editors, *An Industrial IoT Approach for Pharmaceutical Industry Growth*, pages 31–54. Academic Press, 2020.

[113] F. van der Heijden, R. Duin, D. de Ridder, and D. Tax. *Parameter Estimation*, chapter 3, pages 45–80. John Wiley & Sons, Ltd, 2004.

[114] L. van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[115] M. Viana. (what's new on lorenz strange attractors?). the mathematical intelligencer. *The Mathematical Intelligencer 22*, pages 6–19, 2000.

[116] S. Vieira, W. H. Lopez Pinaya, and A. Mechelli. Chapter 1 - introduction to machine learning. In A. Mechelli and S. Vieira, editors, *Machine Learning*, pages 1–20. Academic Press, 2020.

[117] K. Vlachas, K. Tatsis, K. Agathos, A. R. Brink, and E. Chatzi. A local basis approximation approach for nonlinear parametric model order reduction. *Journal of Sound and Vibration*, 502:116055, 2021.

[118] E. Walter and L. Pronzato. On the identifiability and distinguishability of nonlinear parametric models. *Mathematics and Computers in Simulation*, 42(2):125–134, 1996. Mathematical Modelling and Simulation in Agriculture and Bio-Industries Proceedings of the 1st IMACS-IFAC Symposium Msu2SABI.

[119] J. Wang, A. Hertzmann, and D. J. Fleet. Gaussian process dynamical models. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems*, volume 18. MIT Press, 2005.

[120] K. E. Willcox. Approximate yet accurate surrogates for large-scale simulation. *Science at Extreme Scales: Where Big Data Meets Large-Scale Computing Tutorials, Institute for Pure and Applied Mathematics, The University of Texas, Austin*, pages 1–52, 2018.

[121] J. Wouters. A brief introduction to the Lorenz-63 system. *CC Attribution 3.0 License*, 2013.

[122] K. Xu, J. Maidana, S. Castro, and P. Orio. Synchronization transition in neuronal networks composed of chaotic or non-chaotic oscillators. *PMCID: PMC5976724*, 8, 2018.

[123] T. Zhang, Z. Lu, J. Liu, and G. Liu. Parameter identification of nonlinear systems with time-delay from time-domain data. *Nonlinear Dynamics*, 104:4045–4061, 04 2021.

[124] Y. J. Zhen Li and K. Xu. Non-linear model-order reduction based on tensor decomposition and matrix product. *Institution of Engineering and Technology, Control Theory and Applications*, 12 Iss:2253–2262, 2018.

[125] Y. Zhu, J. Derber, A. Collard, D. Dee, R. Treadon, G. Gayno, J. Jung, D. Groff, Q. Liu, P. Delst, E. Liu, and D. Kleist. Variational bias correction of radiance data in the ecmwf system. In *Variational Bias Correction in the NCEP's Data Assimilation System*, 03 2014.