Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Helena Thu Phan
Vilde Taklo Kenworthy

Learning of Nonlinear Dynamics with Contraction-Based Regularization

Master's thesis in Engineering and ICT
Supervisor: Olav Egeland
June 2023

NTNU
Norwegian University of
Science and Technology

Helena Thu Phan
Vilde Taklo Kenworthy

Learning of Nonlinear Dynamics with Contraction-Based Regularization

Master's thesis in Engineering and ICT
Supervisor: Olav Egeland
June 2023

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

NTNU
Norwegian University of
Science and Technology

Preface

The presented thesis is submitted to fulfill the final requirements for obtaining the degree of Master of Science in Engineering and ICT at the Norwegian University of Science and Technology (NTNU). This thesis is written during Spring 2023 for the course "TPK4960 - Robotics and Automation, Master's Thesis" within the Department of Mechanical and Industrial Engineering.

Readers of this thesis would benefit from having a fundamental understanding of linear algebra, metric spaces, and regression analysis. However, the necessary theoretical concepts are covered in the preliminaries section. The primary objective of this thesis is to investigate and evaluate the performance of random Fourier features in approximating different kernels in a reproducing kernel Hilbert space. These features are then deployed in various regression problems. The focus is on different vector fields with distinct characteristics, such as contraction. The vector fields are both learned and simulated, and the resulting outcomes will be analyzed and discussed. This thesis contributes to validating theoretical concepts and practical applications in the field of engineering, particularly in the area of learning system dynamics.

Acknowledgments

We would like to express our sincere gratitude towards Professor Olav Egeland at NTNU for the challenging yet exciting research subject, as well as for his support, guidance, and feedback throughout the development of this thesis. Furthermore, we would like to thank Ph.D. candidate Torbjørn Smith, our discussions have been of great value in solving technical problems and complex tasks. Olav and Torbjørn's knowledge and insightful feedback have significantly contributed to improving the quality of this research. Lastly, we would like to thank our family and friends for their encouragement and support throughout the thesis.

Summary

Learning unknown dynamical systems is a complex task involving inferring the underlying system behavior from observed data without prior knowledge of the underlying equations. Understanding and accurately learning these dynamics is of great importance in various fields, such as robotics and control systems, enabling predictive modeling, control, and decision-making in complex and uncertain environments.

This thesis addresses this challenge by exploring the use of random Fourier features (RFF) to approximate different kernels in a reproducing kernel Hilbert space (RKHS) and solve various regression problems. The focus will be on learning vector fields with various characteristics. To accomplish this, three kernels were approximated: a Gaussian separable kernel, a curl-free kernel, and a symplectic kernel. The characteristics included properties such as contraction constraints and vanishing points. Regression problems were solved on both real-world and self-generated data, demonstrating the potential of RFF in learning dynamical systems. The results obtained in this thesis indicate that using RFF to approximate kernels in a RKHS for various regression problems provided accurate and satisfactory estimates while being computationally efficient.

Sammendrag

Å lære ukjente dynamiske systemer er en kompleks oppgave som involverer å utlede den underliggende systematferden fra observert data, uten forkunnskaper om de underliggende ligningene. Å forstå og nøyaktig lære dynamikken til disse systemene er av stor betydning for ulike områder, som robotikk og kontrollsystemer. Dette muliggjør prediktiv modellering, kontroll og beslutningstaking i komplekse og usikre miljøer.

Denne masteroppgaven tar opp denne utfordringen ved å utforske bruken av tilfeldige Fourier features (RFF) for å tilnærme ulike kjerner i et reproduserende kjerne Hilbert-rom (RKHS) og løse ulike regresjonsproblemer. Fokuset vil være å lære vektorfelt med ulike egenskaper. For å oppnå dette ble tre kjerner tilnærmet: en Gaussisk separerbar kjerne, en curl-fri kjerne og Disse egenskapene inkluderer kontraksjonsbegrensninger og forsvinningspunker. Regresjonsproblemer ble løst på både virkelighets og selvgenerert data, noe som demonstrerte potensialet til RFF i å lære dynamiske systemer. Resultatene utledet i denne oppgaven indikerer at bruk av RFF for å tilnærme kjerner i en RKHS for ulike regresjonsproblemer ga nøyaktige og gode estimater, samtidig som det var beregningsmessig effektivt.

Contents

List of Figures

List of Tables

Chapter 1.

Introduction

This introductory chapter provides the motivation for the master's thesis, presents relevant literature and related work, and outlines the key goals.

1.1. Motivation

The field of data-driven modeling, which is classified within the broader scope of machine learning, has become a significant research area. In cases where it is not practical or possible to derive models based on equations from first principles, data-driven approaches become useful. Furthermore, in various scientific and engineering fields, extracting physical laws from data is a central challenge in many diverse areas of science and engineering. As a result, there are many critical data-driven problems, such as understanding brain activity, climate pattern inference, and determining stability in financial markets [16]. These problems demonstrate the importance of leveraging data to gain insights, make predictions, and drive informed decision-making. This thesis is motivated by a specific method within data-driven modeling, which involves using data to learn models for robotics and control systems.

When it comes to control systems, understanding system dynamics is often crucial for designing an appropriate control system. However, there may be occasions where controlling a system with unknown dynamics is necessary or preferable. As mentioned, this could happen when the system's dynamics are unknown and can not be directly measured or derived from first principles. In such cases, a learning technique, such as regression, can be used to estimate the dynamics based on measurements of the system's inputs and states. By using regression, the system's behavior can be approximated based on a limited number of observed trajectories. This approach proves particularly useful in situations where obtaining a mathematical model of the system is difficult or impossible [26]. It is also valuable when

the system's complexity prevents accurate modeling.

Furthermore, machine learning faces a significant challenge in efficiently representing and modeling complex, high-dimensional data [22]. This challenge is especially relevant in regression tasks that aim to predict output values based on one or more input features. However, reproducing kernel Hilbert space (RKHS) and random Fourier features (RFF) presents a promising approach to overcoming this challenge. These techniques have been proven to enable efficient function approximation in a feature space, making them useful for modeling complex data. These techniques can also provide valuable insights into the trade-offs between computational efficiency and modeling performance in various regression problems.

Our specialization project [28] investigated using RFF to approximate kernels in a RKHS, yielding promising results. The project demonstrated that using RFF provided satisfactory estimates in regression problems, significantly reducing computation time compared to using the exact kernel. Hence, it is now desired to further research the potential of using RFF to estimate dynamical systems.

1.2. Related Work

Improving the precision of learning in data-driven modeling is a broad research area that is valuable for the control and robotics industry. The use of RFF to approximate kernels in a RKHS to learn these systems represents a more narrow research area within this field. The literature review revealed notable contributions to this subject, highlighting its significance.

The ability to approximate various radial basis kernels using random Fourier features was first introduced by Rahimi and Recht in 2007 [46]. They demonstrated that using RFF as input to standard linear learning algorithms could produce results comparable in accuracy, training time, and evaluation time to state-of-the-art large-scale kernel machines. The primary concept of this approach was to map input data to a low-dimensional, randomized feature space and apply existing fast linear methods. This was made possible using a RKHS, a concept proposed by Aronszajn in 1950 [6]. The literature review shows that using RFFs for approximating functions presented in an RKHS has proven highly effective for solving regression problems, as demonstrated in [46]. This method was later extended to vector-valued functions, as introduced by Brault et al. [13] and Minh [39] in 2016. Moreover, this technique enables efficient function approximation, which is advantageous in accurately modeling and predicting data in various regression tasks.

In 2018, Sindhwani et al. [51] introduced a vector field learning approach that uti-

lized random features for function approximation in learning stable vector fields. The objective was to satisfy contraction under the identity metric. The vector field was parameterized using a vector-valued reproducing kernel, and RFF was used to approximate the kernel. In addition, the paper aimed to position the equilibrium of the vector field at desired locations, also known as vanishing points, and control the local contraction at different points by utilizing convex optimization. Their approach was then demonstrated on imitation learning tasks using a real-world dataset.

Singh et al. [52] addressed the challenge of learning controlled dynamics from demonstrations. Their objective was to learn a dynamics model for a system with an unknown dynamics function. To achieve this, they used a small set of sampled data points from observed trajectories to estimate the system's behavior. By leveraging tools from RKHS and contraction theory, they introduced the concept of learning stabilizable dynamics in 2021.

Ahmadi and Khadir [1] introduced a framework for learning a dynamical system of a limited number of trajectories while subject to *side information*. In this context, side information refers to any additional knowledge about the system being learned, apart from the trajectory data. Therefore, explicitly integrating side information into the learning process was of interest to compensate for the limited number of trajectory observations. One side information defined in the paper is the incorporation of Hamiltonian systems, which will be investigated in this thesis.

In 2022, Boffi and Slotine [11] presented a nonparametric adaptive algorithm for learning unknown dynamics in a RKHS. To address the computational cost of a RKHS, the authors proposed a randomized implementation using RFF. This randomized algorithm retained the expressivity of the nonparametric input while recovering the complexity of classical parametric methods. Furthermore, the authors proposed an approximation of a symplectic kernel using RFF, enabling the estimation of a high-dimensional Hamiltonian dynamical system.

1.3. Aim of the Thesis

The aim of this thesis explores the use of random Fourier features for kernel-based regression problems for learning dynamical systems. This includes investigating how the incorporation of additional information and using different kernels can enhance the precision of the learning process. Specifically, RFF will be used to approximate three different kernels, namely the Gaussian separable, the curl-free, and the symplectic kernel. Additionally, contraction constraints and vanishing points will be incorporated into the regression problem. Furthermore, the thesis

will demonstrate the efficacy of using RFF to approximate kernels where the data is corrupted by noise.

Chapter 2.

Preliminaries

This section provides the necessary information to understand the subject matter discussed in this thesis. It covers essential definitions, concepts, and theories helpful in understanding the material. The work in [19] has served as a significant source of guidance in this regard. The references provided also contain elementary proofs excluded from this report. Finally, it is worth noting that some of the content presented in the preliminaries was included in the report for the specialization project [28].

2.1. Kernels

Kernel methods transform data from their original input space to a feature space. Through the utilization of a kernel method, it becomes possible to convert a nonlinear problem in the input space to a simple linear problem in the feature space. To accomplish this, kernel methods substitute the inner product of the data with a selected kernel function [4].

2.1.1. Positive Definite Kernel

This section is based on [40] and [49]. If a kernel, denoted as k, is symmetric and satisfies Mercer's theorem, then it is considered positive definite. A kernel is symmetric if the following holds

$$k(\boldsymbol{x}, \boldsymbol{z}) = k(\boldsymbol{z}, \boldsymbol{x}), \quad \forall\, \boldsymbol{x}, \boldsymbol{z} \in \mathcal{X} \tag{2.1}$$

where $\mathcal{X}$ is a closed subset of $\mathbb{R}^n$ and $n \in \mathbb{N}$. Given a finite set of points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathcal{X}$ and real numbers $a_1, \ldots, a_N \in \mathbb{R}$

$$\sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0 \tag{2.2}$$

If a kernel is positive definite, and $\boldsymbol{x}_1$ and $\boldsymbol{x}_2 \in \mathcal{X}$, then

$$|k(\boldsymbol{x}_1, \boldsymbol{x}_2)|^2 \leq k(\boldsymbol{x}_1, \boldsymbol{x}_1) k(\boldsymbol{x}_2, \boldsymbol{x}_2) \tag{2.3}$$

The reason for this is a characteristic of dot products known as the Cauchy-Schwarz inequality for kernels [10].

2.1.2. Reproducing Kernel

This section is based on Moore-Aronszajn's theorem [57], which states that for any set $\mathcal{X}$, a kernel k: $\mathcal{X} \times \mathcal{X}$ is positive definite if and only if it is a reproducing kernel. Let $\mathcal{H}$ be a reproducing kernel Hilbert Space (RKHS) with a reproducing kernel k. The symmetry of the inner product between two points $\boldsymbol{x}, \boldsymbol{z} \in \mathcal{X}^2$ in $\mathcal{H}$ implies that

$$k(\boldsymbol{x}, \boldsymbol{z}) = \langle k_{\boldsymbol{x}}, k_{\boldsymbol{z}} \rangle_{\mathcal{H}} = \langle k_{\boldsymbol{z}}, k_{\boldsymbol{x}} \rangle_{\mathcal{H}} = k(\boldsymbol{z}, \boldsymbol{x}) \tag{2.4}$$

Symmetry follows as a result of this. For any set of points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathcal{X}^n$ and real numbers $a_1, \ldots, a_n$, it can be shown that the kernel k is positive definite when

$$\begin{aligned}
\sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \sum_{i=1}^{n}\sum_{j=1}^{n} a_i a_j \langle k_{\boldsymbol{x}_i}, k_{\boldsymbol{x}_j} \rangle_{\mathcal{H}} \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} \langle a_i k_{\boldsymbol{x}_i}, a_j k_{\boldsymbol{x}_j} \rangle_{\mathcal{H}} \\
&= \left\langle \sum_{i=1}^{n} a_i k_{\boldsymbol{x}_i}, \sum_{j=1}^{n} a_j k_{\boldsymbol{x}_j} \right\rangle_{\mathcal{H}} \\
&= \left\| \sum_{i=1}^{n} a_i k_{\boldsymbol{x}_i} \right\|_{\mathcal{H}}^2 \\
&\geq 0
\end{aligned} \tag{2.5}$$

Now, consider a vector space $\mathcal{H}_0 \subset \mathbb{R}^{\mathcal{X}}$, where

$$\mathcal{H}_0 = \operatorname{span}\{k_{\boldsymbol{x}} : \boldsymbol{x} \in \mathcal{X}\} \tag{2.6}$$

Two arbitrary functions $f, g \in \mathcal{H}$ are given by

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} a_i k_{\boldsymbol{x}_i} = \sum_{i=1}^{n} a_i k(\boldsymbol{x}_i, \boldsymbol{x}) \tag{2.7}$$

$$g(\boldsymbol{x}) = \sum_{i=1}^{n} b_i k_{\boldsymbol{z}_i} = \sum_{i=1}^{n} b_i k(\boldsymbol{z}_i, \boldsymbol{x}) \tag{2.8}$$

The inner product is defined as

$$\langle f, g \rangle_{\mathcal{H}_0} = \sum_{i=1}^{n} \sum_{j=1}^{n} a_i b_j k(\boldsymbol{x}_i, \boldsymbol{z}_j) \tag{2.9}$$

The inner product between function f and kernel $k_{\boldsymbol{x}}$ is given by

$$\langle f, k_{\boldsymbol{x}} \rangle_{\mathcal{H}_0} = f(\boldsymbol{x}) \tag{2.10}$$

A positive definite kernel k gives

$$||f||^2_{\mathcal{H}_0} = \langle f, f \rangle_{\mathcal{H}_0} = \sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0 \tag{2.11}$$

It is also noted that

$$||k_{\boldsymbol{x}}||^2_{\mathcal{H}_0} = \langle k_{\boldsymbol{x}}, k_{\boldsymbol{x}} \rangle_{\mathcal{H}_0} = |k(\boldsymbol{x}, \boldsymbol{x})| \tag{2.12}$$

Cauchy-Schwarz inequality states that [10]

$$|l(\boldsymbol{x})| = |\langle \boldsymbol{x}, \boldsymbol{z} \rangle| \leq ||\boldsymbol{x}||\,||\boldsymbol{z}|| \tag{2.13}$$

where l is a bounded linear functional on $\mathcal{H}_0$.

Based on this, it follows that

$$|f(\boldsymbol{x})| = |\langle f, k_{\boldsymbol{x}} \rangle|_{\mathcal{H}_0} \leq ||f||_{\mathcal{H}_0} ||k_{\boldsymbol{x}}||_{\mathcal{H}_0} = ||f||_{\mathcal{H}_0} \sqrt{|k(\boldsymbol{x}, \boldsymbol{x})|} \tag{2.14}$$

which shows that if $||f||_{\mathcal{H}_0}$ tends to zero, then f will tend to zero. Based on this, it has been demonstrated that $\mathcal{H}_0$ has all the characteristics of a Hilbert space except for completeness, which classifies it as a pre-Hilbert space. A process for constructing the closure $\mathcal{H}$ of $\mathcal{H}_0$ is outlined in [6].

2.1.3. Reproducing Kernel Hilbert Spaces

A Hilbert space is considered to be a reproducing kernel Hilbert space if it has a reproducing kernel. The reproducing property and Hilbert space structure of RKHS ensure the effectiveness of many practical learning algorithms implemented in these function spaces [63].

Suppose $\mathcal{H}$ is a Hilbert space consisting of functions $f : \mathcal{X} \to \mathbb{R}$. If a kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ satisfies the following two conditions, then it is a reproducing kernel of the Hilbert space $\mathcal{H}$ with an inner product denoted as $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ [54].

1. For each $\boldsymbol{x} \in \mathcal{X}$
$$k(\boldsymbol{x}, \cdot) \in \mathcal{H} \tag{2.15}$$

2. Reproducing property, also known as the kernel trick.
$$f(\boldsymbol{x}) = \langle f, k(\boldsymbol{x}, \cdot) \rangle_{\mathcal{H}} \quad \forall\, \boldsymbol{x} \in \mathcal{X},\ \forall f \in \mathcal{H} \tag{2.16}$$

If the reproducing kernel k is positive definite, then the Hilbert space $\mathcal{H}$ is unique [6].

2.1.4. Feature Map

To map the data from the input space to the feature space F, see Figure 2.1, a nonlinear mapping function is applied [4]. This can be represented as follows

$$\psi : \mathbb{R}^d \to F \tag{2.17}$$
$$\boldsymbol{x} \to \psi(\boldsymbol{z}) \tag{2.18}$$

Let $\mathcal{X} \subset \mathbb{R}^n$ be a set, and let k: $\mathcal{X} \times \mathcal{X}$ be a kernel. The kernel function is represented as

$$k(\boldsymbol{x}, \boldsymbol{z}) = \psi(\boldsymbol{x})^T \psi(\boldsymbol{z}) \tag{2.19}$$

2.1.5. Kernel Trick

This section is based on [46]. The kernel trick generates features for algorithms relying solely on the inner product of input point pairs. For this to work, it is necessary that any positive definite kernel $k(\boldsymbol{x}, \boldsymbol{z})$, where $\boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^n$, defines an inner product and a transformation ϕ. This ensures that the inner product between the transformed data points can be calculated.

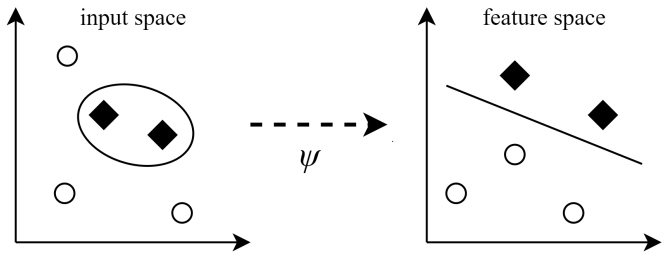


Figure 2.1.: Feature map. *Illustration* [48].

$$\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{z}) \rangle = k(\boldsymbol{x}, \boldsymbol{z}) \tag{2.20}$$

The data will be accessed by the algorithm through $k(\boldsymbol{x}, \boldsymbol{z})$ or through a kernel matrix consisting of kernel k applied to all data point pairs.

It is proposed to use a feature map that is randomized, denoted by $\psi : \mathbb{R}^n \to \mathbb{R}^d$, to map the data to an inner product space of low dimension. As a result, the inner product between the transformed data point pairs approximates their kernel evaluation.

$$k(\boldsymbol{x}, \boldsymbol{z}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{z}) \rangle \approx \psi(\boldsymbol{x})^T \psi(\boldsymbol{z}) \tag{2.21}$$

By using ψ, an explicit transformation can be achieved rather than relying on the implicit transformation provided by the kernel trick. It is worth mentioning that ψ is low-dimensional compared to the transformation ϕ. Therefore, ψ can be used to transform the input and approximate the solution of a corresponding nonlinear kernel machine. This is done by transforming the input using ψ and applying fast linear learning techniques.

2.1.6. Complexity in Kernel Methods

The power of kernel methods comes from the kernel trick, which uses a feature map $\psi(\cdot)$ to implicitly map the data into a feature space, allowing for non-linear functional learning. However, despite their applicability, kernel methods can be computationally expensive when considering algorithm complexity. A typical kernel learning algorithm requires $O(N^3)$ computation and $O(N^2)$ memory, where N is the number of data points [33]. To address this issue, Rahimi and Recht [46] proposed an approximation framework, discussed in Section 2.2, which has proven to give great results in kernel regression problems with significantly faster computation time. This framework reduces the computational complexity of ker-

nel methods by reducing them to linear learning in the feature space, allowing for computation using fast linear solvers. As a result, time and space complexity is reduced from $O(N^3)$ and $O(N^2)$ to $O(N\mathbb{R}^2)$ and $O(N\mathbb{R})$ respectively, providing substantial computational savings as long as $\mathbb{R} << N$.

2.2. Random Fourier Features

One significant disadvantage of RKHS is their computational cost [11]. However, the theory of random Fourier features (RFF) has provided a key breakthrough for overcoming this challenge. The RFF method is a widely utilized approach to improve the scalability of kernel methods, particularly in the context of kernel ridge regression (KRR). The approach was first introduced in a study by Rahimi and Recht in 2007 [46] and involves constructing random features using Bochner's Theorem and inverse Fourier transform to approximate a shift-invariant kernel [39]. This method enables the approximation of low-dimensional, randomized kernels, providing a more efficient means of processing large datasets.

In recent years, a growing interest has been in developing scalable kernel-based methods, such as KRR, using RFFs [62, 13, 61]. The material presented here is based on the work in [13] and provides an overview of Fourier transformation, Bochner's theorem, and the approximation of kernels using RFF.

2.2.1. Random Features

Rahimi and Recht [46] proposed a technique that utilizes a set of random features to project data points onto a randomly selected line, followed by applying a sinusoidal function to generate a scalar output. By drawing random lines from a distribution, the inner product of the transformed points approximates the shift-invariant kernel. The theoretical foundation for this approach is Bochner's theorem [46], which comes from classical harmonic analysis and provides valuable insights into the use of random features for this purpose.

2.2.2. Fourier Transform

This section builds upon the theories presented in [61] and explains the Fourier transform of the Gaussian radial basis function. Prior knowledge of Fourier transformation is beneficial. The significance of this finding lies in its potential to establish a connection between the Gaussian kernel and random Fourier features, which serves as a foundation for utilizing Fourier-based approximations in kernel regression.

By using a finite number of random Fourier features, it is possible to approximate the Fourier transform of the Gaussian radial basis function. This approximation is possible because the Fourier transform of the Gaussian function itself is a Gaussian function [32].

For the Gaussian radial basis function

Let the Fourier transform of a function $G(w)$ be defined by

$$g(x) = \int_{-\infty}^{\infty} G(w) e^{-iwx} dw \tag{2.22}$$

and let the inverse Fourier transform $G(w)$ be defined by

$$G(w) = \frac{1}{2\pi} \int_{-\infty}^{\infty} g(x) e^{-iwx} dx \tag{2.23}$$

Considering a function defined as

$$g(x) = e^{-\frac{x^2}{\rho^2}} \tag{2.24}$$

and applying this function to (2.23), generates the inverse Fourier transform

$$G(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-\frac{x^2}{\rho^2}} e^{-iwx} dx = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{-(\frac{x^2}{\rho^2} + iwx)} dx \tag{2.25}$$

Suppose the integral of a Gaussian function is

$$\int_{-\infty}^{\infty} e^{-(ax^2+bx)} dx = \sqrt{\frac{\pi}{a}} e^{(\frac{b^2}{4a})} \tag{2.26}$$

With $a = 1/\rho^2$ and $b = iw$, the inverse Fourier transform becomes

$$G(w) = \frac{1}{2\pi} \sqrt{\pi\rho^2} e^{-\frac{\rho^2 w^2}{4}} = \frac{1}{\sqrt{2\pi\frac{2}{\rho^2}}} e^{-\frac{1}{2}\frac{\rho^2}{2}w^2} \tag{2.27}$$

The probability density function of the normal distribution can be expressed based on the theory of normal distribution when the expected value is zero

$$p(w) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{w}{\sigma})^2} \tag{2.28}$$

From the previous analysis, it can be concluded that setting

$$\sigma^2 = \frac{2}{\rho^2} \tag{2.29}$$

$G(w)$ will represent the probability density function of a normal distribution.

For the Gaussian radial basis function with vector variable

Building upon the previous section, the case involving vector variables is now considered. In the vector case, the Fourier transform is given by

$$g(\boldsymbol{x}) = \int_{-\infty}^{\infty} G(\boldsymbol{w}) e^{-i\boldsymbol{w}^T\boldsymbol{x}} d\boldsymbol{w} \tag{2.30}$$

and the inverse Fourier transform is defined as

$$G(\boldsymbol{w}) = (\frac{1}{2\pi})^n \int_{-\infty}^{\infty} g(\boldsymbol{x}) e^{i\boldsymbol{w}^T\boldsymbol{x}} d\boldsymbol{x} \tag{2.31}$$

By replacing $g(\boldsymbol{x})$ in (2.30) with the function

$$g(\boldsymbol{x}) = e^{-\frac{\boldsymbol{x}^2}{\rho^2}} \tag{2.32}$$

the inverse Fourier transform becomes

$$G(\boldsymbol{w}) = (\frac{1}{2\pi})^n \int_{-\infty}^{\infty} e^{-\frac{\boldsymbol{x}^2}{\rho^2}} e^{-i\boldsymbol{w}^T\boldsymbol{x}} d\boldsymbol{x} = \int_{-\infty}^{\infty} e^{-(\frac{\boldsymbol{x}^2}{\rho^2} + i\boldsymbol{w}^T\boldsymbol{x})} d\boldsymbol{x} \tag{2.33}$$

Given that the integral of a Gaussian function in the case of vectors is

$$\int_{-\infty}^{\infty} e^{-(a\boldsymbol{x}^2 + \boldsymbol{b}^T\boldsymbol{x})} d\boldsymbol{x} = \sqrt{(\frac{\pi}{a})^n} e^{(\frac{||\boldsymbol{b}||^2}{4a})} \tag{2.34}$$

it is seen that $a = 1/\rho^2$ and $\boldsymbol{b} = i\boldsymbol{w}$. This results in the inverse Fourier transform

$$G(\boldsymbol{w}) = \left(\frac{1}{2\pi}\right)^n \sqrt{(\pi\rho^2)^n} e^{-\frac{\rho^2||\boldsymbol{w}||^2}{4}} = \left(\frac{1}{\sqrt{2\pi\frac{2}{\rho^2}}}\right)^n e^{-\frac{1}{2}\frac{\rho^2||\boldsymbol{w}||^2}{2}} \tag{2.35}$$

With the probability density function being

$$p(\boldsymbol{w}) = \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n e^{-\frac{1}{2}(\frac{||\boldsymbol{w}||}{\sigma})^2} \tag{2.36}$$

If σ^2 is set to be equal to

$$\sigma^2 = \frac{2}{\rho^2} \tag{2.37}$$

then $G(\boldsymbol{w})$ will be the probability density function of a normal distribution.

2.2.3. Bochner's Theorem

Bochner's theorem, explained in [46], states that a continuous kernel on $\mathbb{R}^n$, represented as

$$k(\boldsymbol{x}, \boldsymbol{z}) = g(\boldsymbol{x} - \boldsymbol{z}) \tag{2.38}$$

is positive definite if and only if the kernel k is the Fourier transform of a non-negative measure.

As mentioned, Bochner's theorem is the theoretical result leading to random Fourier features [13]. This theorem implies that any kernel k that is positive definite, continuous, and shift-invariant can be represented as the Fourier transform of a non-negative measure μ.

Let a kernel k be

$$k(\boldsymbol{x}, \boldsymbol{z}) = g(\boldsymbol{x} - \boldsymbol{z}) = \int_{\mathbb{R}^n} e^{-i\boldsymbol{w}(\boldsymbol{x}-\boldsymbol{z})} d\mu(\boldsymbol{w}) \tag{2.39}$$

If $\mu(\boldsymbol{w})$ is the cumulative distribution function for p, then

$$\mu(\boldsymbol{w}) = \int^{\boldsymbol{w}} p(\boldsymbol{w}) d\boldsymbol{w} \tag{2.40}$$

By using Leibniz integral rule it can be found that

$$d\mu = p(\boldsymbol{w})d\boldsymbol{w} \tag{2.41}$$

Note that

$$\int_{\mathbb{R}^n} d\mu = \int_{\mathbb{R}^n} p(\boldsymbol{w})d\boldsymbol{w} = 1 \tag{2.42}$$

(2.39) can therefore be rewritten

$$g(\boldsymbol{x}) = \int_{\mathbb{R}^n} e^{-i\boldsymbol{w}\boldsymbol{x}} p(\boldsymbol{w})d\boldsymbol{w} \tag{2.43}$$

(2.39) can also be written as an expectation over

$$\mu : k_0(\boldsymbol{x} - \boldsymbol{z}) = \mathbb{E}_\mu[e^{-i\boldsymbol{w}(\boldsymbol{x}-\boldsymbol{z})}] \tag{2.44}$$

Given that k is real-valued, the real part can be defined as

$$\begin{aligned} k(\boldsymbol{x}, \boldsymbol{z}) &= \mathbb{E}_\mu[\cos(\boldsymbol{w}(\boldsymbol{x} - \boldsymbol{z}))] \\ &= \mathbb{E}_\mu[\cos(\boldsymbol{w}\boldsymbol{z})\cos(\boldsymbol{w}\boldsymbol{x}) + \sin(\boldsymbol{w}\boldsymbol{z})\sin(\boldsymbol{w}\boldsymbol{x})] \end{aligned} \tag{2.45}$$

Let $\sum_{i=1}^{d} \boldsymbol{x}_i$ represent a column vector of length dm obtained by concatenating vectors $\boldsymbol{x}_i \in \mathbb{R}^m$. Given this, the feature map $\psi : \mathbb{R}^n \to \mathbb{R}^{2d}$ defined as

$$\psi(\boldsymbol{x}) = \frac{1}{\sqrt{d}} \sum_{i=1}^{d} \begin{bmatrix} \cos(\boldsymbol{w}_i\boldsymbol{x}) \\ \sin(\boldsymbol{w}_i\boldsymbol{x}) \end{bmatrix} \tag{2.46}$$

is a random Fourier feature map.

To approximate the kernel, each $\boldsymbol{w}_i$, where $i = 1, ..., d$, is sampled independently from the inverse Fourier transform μ of k_0. This produces a Monte-Carlo estimator $k(\boldsymbol{x}, \boldsymbol{z}) = \psi(\boldsymbol{x})\psi(\boldsymbol{z})$ of the kernel, as described in [13]. It is stated in [46] that the convergence of this approximation towards the desired kernel, given in equation (2.39), depends on the dimension d.

According to [13], the RFF approach requires two steps before learning can begin. The first step involves defining the randomized feature map of a given shift-invariant kernel, while the second step is to compute the randomized feature map using the spectral distribution μ. It is also shown in [46] that for the Gaussian kernel $k(\boldsymbol{x} - \boldsymbol{z}) = e^{-\gamma\|\boldsymbol{x}-\boldsymbol{z}\|^2}$, the spectral distribution $\mu(\boldsymbol{w})$ is indeed Gaussian.

2.2.4. Real-Valued Approximation of a Scalar Kernel

As observed in [46], real-valued functions can be used to represent random Fourier features. This is possible because the probability density function of the features is real, and the kernel can be expressed as an integral in the following manner

$$\boldsymbol{K}(\boldsymbol{x},\boldsymbol{z}) = \int_{\mathbb{R}^d} \cos(\boldsymbol{w}^T(\boldsymbol{x}-\boldsymbol{z}))p(\boldsymbol{w})d\boldsymbol{w} \tag{2.47}$$

When the feature map is given by

$$\boldsymbol{\psi}_w(\boldsymbol{x}) = \begin{bmatrix}\cos(\boldsymbol{w}^T\boldsymbol{x}) \\ \sin(\boldsymbol{w}^T\boldsymbol{x})\end{bmatrix} \tag{2.48}$$

it follows from trigonometric identities that $\boldsymbol{\psi}_w(\boldsymbol{x})^T\boldsymbol{\psi}_w(\boldsymbol{z}) = \cos(\boldsymbol{w}\boldsymbol{x})\cos(\boldsymbol{w}\boldsymbol{z}) + \sin(\boldsymbol{w}\boldsymbol{x})\sin(\boldsymbol{w}\boldsymbol{z}) = \cos(\boldsymbol{w}(\boldsymbol{x}-\boldsymbol{z}))$.

Bochner's theorem for the kernel is given by

$$\boldsymbol{K}(\boldsymbol{x},\boldsymbol{z}) = \int_{\mathbb{R}^d} \boldsymbol{\psi}_w(\boldsymbol{x})\boldsymbol{\psi}_w(\boldsymbol{z})p(\boldsymbol{w})d\boldsymbol{w} \tag{2.49}$$

By calculating

$$\boldsymbol{K}(\boldsymbol{x},\boldsymbol{z}) = \frac{1}{d}\sum_{i=1}^{d} \boldsymbol{\psi}_{w_i}(\boldsymbol{x})^T\boldsymbol{\psi}_{w_i}(\boldsymbol{z}) \tag{2.50}$$

an approximation of the kernel can be found. Further on, $\boldsymbol{w}_1, ..., \boldsymbol{w}_d$ can be drawn based on the probability $p(\boldsymbol{w})$.

The approximation now satisfies the following

$$\boldsymbol{K}(\boldsymbol{x},\boldsymbol{z}) = \boldsymbol{\Psi}(\boldsymbol{x})^T\boldsymbol{\Psi}(\boldsymbol{z}) \tag{2.51}$$

where

$$\boldsymbol{K}_{ij} = k(\boldsymbol{x}_i,\boldsymbol{x}_j) \approx \psi(\boldsymbol{x}_i)^T\psi(\boldsymbol{x}_j) \tag{2.52}$$

where the approximate feature map is

$$\boldsymbol{\Psi}(\boldsymbol{x}) = \frac{1}{\sqrt{d}} \begin{bmatrix} \boldsymbol{\psi}_{w_i}(\boldsymbol{x}) \\ \vdots \\ \boldsymbol{\psi}_{w_d}(\boldsymbol{x}) \end{bmatrix} = \frac{1}{\sqrt{d}} \begin{bmatrix} \cos(\boldsymbol{w}_1^T \boldsymbol{x}) \\ \sin(\boldsymbol{w}_1^T \boldsymbol{x}) \\ \vdots \\ \cos(\boldsymbol{w}_d^T \boldsymbol{x}) \\ \sin(\boldsymbol{w}_d^T \boldsymbol{x}) \end{bmatrix} \tag{2.53}$$

2.2.5. Alternative Random Fourier Feature Expressions

A real alternative formulation of the random Fourier features is mentioned in [46]. Here, an approximation of the kernel is obtained by

$$\boldsymbol{K}(\boldsymbol{x}, \boldsymbol{z}) = \frac{1}{d} \sum_{i=1}^{d} \boldsymbol{\psi}_{w_i}(\boldsymbol{x})^T \boldsymbol{\psi}_{w_i}(\boldsymbol{z}) \tag{2.54}$$

where

$$\boldsymbol{\psi}_w(\boldsymbol{x}) = \sqrt{2} \cos(\boldsymbol{w}^T \boldsymbol{x} + b) \tag{2.55}$$

It can also be given by

$$\boldsymbol{\psi}_w(\boldsymbol{x}) = \sqrt{2} \sin(\boldsymbol{w}^T \boldsymbol{x} + b) \tag{2.56}$$

As for the approximation outlined in Section 2.2.4, the value of $\boldsymbol{w}_i$ is selected with probability $p(\boldsymbol{w})$, while the value of b is chosen uniformly at random from the interval $[0, 2\pi)$.

2.2.6. Random Fourier Features for Curl-Free Kernel

The concept of curl-free kernels has been introduced for learning conservative vector fields [55], which is when the vector fields conserve energy. The curl property of vector fields is applicable when representing the rotational movement of a field. A vector field $\boldsymbol{F}$ is considered irrotational, or curl-free, if

$$\boldsymbol{\nabla} \times \boldsymbol{F} = \boldsymbol{0} \tag{2.57}$$

This indicates that the vector field has no rotational component.

It is possible to find an approximation of a curl-free kernel using RFF [51]. To find the curl-free kernel when using RFF, the Hessian of $g(\boldsymbol{x})$ defined in equation

(2.43) is calculated. The Hessian is given as

$$\nabla^2 g(\boldsymbol{x}) = \int_{\mathbb{R}^n} \left(\nabla^2 e^{-i\boldsymbol{w}^T \boldsymbol{x}} \right) p(\boldsymbol{w}) d\boldsymbol{w} \tag{2.58}$$

This gives

$$\boldsymbol{G}_{cf}(\boldsymbol{x}) = -\nabla^2 g(\boldsymbol{x}) = \int_{\mathbb{R}^n} \boldsymbol{w}\boldsymbol{w}^T e^{-i\boldsymbol{w}^T \boldsymbol{x}} p(\boldsymbol{w}) d\boldsymbol{w} \tag{2.59}$$

$\boldsymbol{G}_{cf}(\boldsymbol{x} - \boldsymbol{z})$ can then be expressed as

$$\begin{aligned} \boldsymbol{G}_{cf}(\boldsymbol{x} - \boldsymbol{z}) &= \int_{\mathbb{R}^n} \boldsymbol{w}\boldsymbol{w}^T e^{-i\boldsymbol{w}^T (\boldsymbol{x} - \boldsymbol{z})} p(\boldsymbol{w}) d\boldsymbol{w} \\ &= \int_{\mathbb{R}^n} \boldsymbol{w}\boldsymbol{w}^T e^{i\boldsymbol{w}^T \boldsymbol{x}} e^{i\boldsymbol{w}^T \boldsymbol{z}} p(\boldsymbol{w}) d\boldsymbol{w} \end{aligned} \tag{2.60}$$

Recall that

$$\boldsymbol{K}(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{G}(\boldsymbol{x} - \boldsymbol{z}) = \int_{\mathbb{R}^n} \psi_w(\boldsymbol{x}) \psi_w(\boldsymbol{z})^T p(\boldsymbol{w}) d\boldsymbol{w} \tag{2.61}$$

Comparing equation (2.60) with equation (2.61) it follows that

$$\psi_w(\boldsymbol{x}) = \boldsymbol{w} e^{-i\boldsymbol{w}^T \boldsymbol{x}} \tag{2.62}$$

The approximation for the curl-free kernel $\boldsymbol{K}_{cf}(\boldsymbol{x}, \boldsymbol{z})$ is defined as

$$\boldsymbol{K}_{cf}(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{\Psi}(\boldsymbol{x}) \boldsymbol{\Psi}(\boldsymbol{z}) \tag{2.63}$$

where it follows that

$$\boldsymbol{\Psi}(\boldsymbol{x}) = \frac{1}{\sqrt{d}} \begin{bmatrix} e^{-i\boldsymbol{w}_1^T \boldsymbol{x}} \boldsymbol{w}^T \\ \vdots \\ e^{-i\boldsymbol{w}_d^T \boldsymbol{x}} \boldsymbol{w}^T \end{bmatrix} \tag{2.64}$$

Looking at the kernel approximation expressed in equation (2.55), the curl-free approximation can be defined as

$$\boldsymbol{\Psi}_{cf}(\boldsymbol{x}) = \sqrt{\frac{2}{d}} \begin{bmatrix} \sin(\boldsymbol{w}_1^T \boldsymbol{x} + b_1) \boldsymbol{w}_1^T \\ \vdots \\ \sin(\boldsymbol{w}_d^T \boldsymbol{x} + b_d) \boldsymbol{w}_d^T \end{bmatrix} \tag{2.65}$$

where its elements are given by

$$\boldsymbol{\psi}_i = \sin(\boldsymbol{w}_i^T \boldsymbol{x} + b_i)\boldsymbol{w}_i^T \tag{2.66}$$

Alternatively, the approximation can be defined as

$$\boldsymbol{\Psi}_{cf}(\boldsymbol{x}) = \sqrt{\frac{2}{d}} \begin{bmatrix} \cos(\boldsymbol{w}_1^T \boldsymbol{x} + b_1)\boldsymbol{w}_1^T \\ \vdots \\ \cos(\boldsymbol{w}_d^T \boldsymbol{x} + b_d)\boldsymbol{w}_d^T \end{bmatrix} \tag{2.67}$$

with elements given by

$$\boldsymbol{\psi}_i = \cos(\boldsymbol{w}_i^T \boldsymbol{x} + b_i)\boldsymbol{w}_i^T \tag{2.68}$$

2.3. Kernel Regression

In this section, the theoretical aspects of kernel ridge regression will be discussed, starting by introducing the Representer Theorem, followed by the Gaussian Separable kernel. It will also present the methods of Regularized Least-Squares and how these methods with Random Fourier features can be used to approximate the exact kernel and solve an optimization problem.

2.3.1. The Representer Theorem

This section is based on [49]. Consider a nonempty set $\mathcal{X}$, with a corresponding reproducing kernel Hilbert Space $\mathcal{H}$ and let k: $\mathcal{X} \times \mathcal{X}$ be a positive definite kernel. Let $(x_1, y_1), \ldots, (x_N, y_N) \in \mathcal{X} \times \mathbb{R}$ be a training sample, g on $[0, \infty) \to \mathbb{R}$ be a strictly monotonically increasing function, and $c : (\mathcal{X} \times \mathbb{R}^2)^N \to \mathbb{R} \cup \{\infty\}$ be an arbitrary cost function. There exists a class of functions

$$\mathcal{F} = \left\{ f \in \mathbb{R}^{\mathcal{X}} \,\middle|\, f(\cdot) = \sum_{i=1}^{\infty} a_i k(\cdot, z_i),\ a_i \in \mathbb{R},\ z_i \in \mathcal{X}, ||f|| < \infty \right\} \tag{2.69}$$

$||\cdot||$ is the norm associated with the kernel k in the reproducing kernel Hilbert Space. The norm for any $z_i \in \mathcal{X}$ and $a_i \in \mathbb{R}$ is given by

$$\left\| \sum_{i=1}^{\infty} a_i k(\cdot, z_i) \right\|^2 = \sum_{i=1}^{\infty} \sum_{j=1}^{\infty} a_i a_j k(z_i, z_j) \tag{2.70}$$

Any $f \in \mathcal{F}$ that minimizes the regularized functional

$$c((x_1, y_1, f(x_1)), ..., (x_N, y_N, f(x_N))) + g(||f||) \tag{2.71}$$

can be represented in the form

$$f(\cdot) = \sum_{i=1}^{N} a_i k(\cdot, x_i) \tag{2.72}$$

This representation can be utilized for ridge regression, see Section 2.3.3.

With mean squared loss, the cost function c is defined as

$$c((x_1, y_1, f(x_1)), ..., (x_N, y_N, f(x_N))) = \frac{1}{N} \sum_{i=1}^{N} (y_i - f(x_i))^2 \tag{2.73}$$

and function g is given by

$$g(||f||) = \lambda ||f||^2, \ \lambda > 0 \tag{2.74}$$

2.3.2. Gaussian Separable Kernel

This section is based on [53], which highlights the Gaussian separable kernel as the most widely used kernel in RKHS. The scalar Gaussian kernel k: $\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is given by

$$k(\boldsymbol{x}, \boldsymbol{z}) = e^{-\frac{||\boldsymbol{x}-\boldsymbol{z}||^2}{2\sigma^2}} \tag{2.75}$$

where $\boldsymbol{x} = [x_1, ..., x_n]^T$ and $\boldsymbol{z} = [z_1, ..., z_n]^T$ are vectors. The hyperparameter σ is a scalar parameter that determines the bandwidth of the kernel and controls the degree of correlation between points.

The Gaussian kernel is shift-invariant, which means that

$$k(\boldsymbol{x}, \boldsymbol{z}) = g(\boldsymbol{x} - \boldsymbol{z}) = e^{-\frac{||\boldsymbol{x}-\boldsymbol{z}||^2}{2\sigma^2}} \tag{2.76}$$

for a positive definite function g [61]. The associated density is also Gaussian, $\mathcal{N}(0, \sigma^{-2}\boldsymbol{I}_N)$.

2.3.3. Regularized Least-Squares

This section covers regularized least-squares, a traditional statistical technique commonly referred to as ridge regression, and it is based on [54], [59] and [60]. Assume that N input data points $\boldsymbol{x}_i$ and the corresponding outputs y_i are given, where $i = 1, \ldots, N$, in other words $\boldsymbol{z} = (\boldsymbol{x}_i, y_i)_{i=1}^N$. Here $\boldsymbol{x}_i \in \mathcal{X} \subseteq \mathbb{R}^m$ and $y_i \in \mathbb{R}$.

Given an RKHS with the kernel K, the optimization problem for regularized least-squares is formulated as minimizing the cost function

$$f_{\boldsymbol{z},\lambda} = \underset{f \in \mathcal{H}_K}{\arg\min} \frac{1}{N} \sum_{i=1}^{N} ||f(\boldsymbol{x}_i) - y_i||_{\mathbb{R}}^2 + \lambda ||f||_{\mathcal{H}_K}^2 \tag{2.77}$$

The classic way to minimize a cost function is by solely using the least-square term, which is the first term on the right-hand side. However, there is a big risk of overfitting when working in the feature space. Therefore, regularization is necessary, which is the reason why the second term is included. The hyperparameter λ regulates the strength of the regularization, and the noise of the problem is associated with it.

As mentioned, the function f can be represented by the form

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} a_i k(\boldsymbol{x}, \boldsymbol{x}_i) \tag{2.78}$$

By replacing $f(\boldsymbol{x})$ in (2.77) with the right hand side of (2.78), the optimization problem can be rewritten as

$$f_{\boldsymbol{z},\lambda} = \underset{f \in \mathcal{H}_K}{\arg\min} \frac{1}{N} \sum_{i=1}^{N} \left\| \sum_{j=1}^{N} a_j k(\boldsymbol{x}, \boldsymbol{x}_j) - y_i \right\|_{\mathbb{R}}^2 + \lambda ||f||_{\mathcal{H}_K}^2 \tag{2.79}$$

The second term in (2.77) can be reformulated as

$$\begin{aligned} ||f||^2_{\mathcal{H}_K} &= \left\langle \sum_{i=1}^{N} a_i k(\cdot, \boldsymbol{x}_i), \sum_{j=1}^{N} a_j k(\cdot, \boldsymbol{x}_j) \right\rangle_{\mathcal{H}_k} \\ &= \sum_{i=1}^{N} \sum_{j=1}^{N} \langle a_i k(\cdot, \boldsymbol{x}_i), a_j k(\cdot, \boldsymbol{x}_j) \rangle_{\mathcal{H}_K} \\ &= \sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j k_{ij} \\ &= \boldsymbol{a}^T \boldsymbol{K} \boldsymbol{a} \end{aligned} \tag{2.80}$$

where $\boldsymbol{a} = [a_1, ..., a_N]^T$

This gives

$$f_{\boldsymbol{z},\lambda} = \frac{1}{N} ||\boldsymbol{K}\boldsymbol{a} - \boldsymbol{y}||^2_{\mathbb{R}} + \boldsymbol{a}^T \boldsymbol{K} \boldsymbol{a} \tag{2.81}$$

By differentiating the right-hand side of equation (2.81) with respect to $\boldsymbol{a}$ and utilizing the symmetry of $\boldsymbol{K}$, the term can be expressed in the following manner

$$f_{\boldsymbol{z},\lambda} = -\frac{1}{N} \boldsymbol{K}(\boldsymbol{y} - \boldsymbol{K}\boldsymbol{a}) + \lambda \boldsymbol{K}\boldsymbol{a} \tag{2.82}$$

Any solution should satisfy $f_{\boldsymbol{z},\lambda} = 0$, which gives

$$-\frac{1}{N} \boldsymbol{K}(\boldsymbol{y} - \boldsymbol{K}\boldsymbol{a}) + \lambda \boldsymbol{K}\boldsymbol{a} = \boldsymbol{0} \tag{2.83}$$

Multiplying all elements with $\boldsymbol{K}^{-1}$ and reorganizing gives the unique solution

$$\boldsymbol{a}(\boldsymbol{K} + \lambda N \boldsymbol{I_N}) = \boldsymbol{y} \tag{2.84}$$

If $\boldsymbol{K}$ is non-singular, $\boldsymbol{a}$ can be found by

$$\boldsymbol{a} = (\boldsymbol{K} + \lambda N \boldsymbol{I_N})^{-1} \boldsymbol{y} \tag{2.85}$$

where $\boldsymbol{a} = [a_1, ..., a_N]^T$, $\boldsymbol{y} = [y_1, ..., y_N]^T$ and $\boldsymbol{K}$ is the Gram matrix of the kernel k, $\boldsymbol{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

The exact solution for the optimization problem in the scalar case is then

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} a_i k(\boldsymbol{x}, \boldsymbol{x}_i) \tag{2.86}$$

and the value of $\boldsymbol{y}$ can be predicted.

2.3.4. Regularized Least-Squares with Random Fourier Features

When using random Fourier features in ridge regression, the inputs $\boldsymbol{x}$ are replaced with their feature map

$$\boldsymbol{x}_i \rightarrow \psi_i = \psi(\boldsymbol{x}_i) \tag{2.87}$$

The regularized least-squares problem can be resolved by utilizing Random Fourier Features to obtain an approximation of the kernel. Randomized approximated feature maps have proven to be a good mechanism to scale up kernel methods and reduce the computational expenses for large-scale tasks [27]. The following is based on [21], [34] and [46]. Let $(\boldsymbol{x}_i, y_i)$, ..., $(\boldsymbol{x}_N, y_N)$ be the set of inputs and corresponding output, where $\boldsymbol{x}_i \in \mathbb{R}^m$ and $y_i \in \mathbb{R}$. Consider the minimization problem

$$L[f] = \arg\min_{\boldsymbol{f} \in \mathcal{H}_K} \frac{1}{N} \sum_{i=1}^{N} ||f(\boldsymbol{x}_i) - y_i||_{\mathbb{R}}^2 + \lambda ||f||_{\mathcal{H}_K}^2 \tag{2.88}$$

$f(\boldsymbol{x})$ is a linear combination of N functions $k(\boldsymbol{x}, \boldsymbol{x}_i)$, and it is given by

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} a_i k(\boldsymbol{x}, \boldsymbol{x}_i) \tag{2.89}$$

It is possible to define a feature map $\boldsymbol{\psi} : \mathcal{X} \rightarrow \mathbb{R}^d$, which has the following property

$$k(\boldsymbol{x}, \boldsymbol{z}) \approx \boldsymbol{\psi}(\boldsymbol{x})^T \boldsymbol{\psi}(\boldsymbol{z}) \tag{2.90}$$

where $\boldsymbol{\psi}(\boldsymbol{x}) = [\psi(\boldsymbol{x}_i), ..., \psi(\boldsymbol{x}_d)]^T \in \mathbb{R}^d$.

The inner product is given by

$$\boldsymbol{\psi}(\boldsymbol{x})^T \boldsymbol{\psi}(\boldsymbol{z}) = \frac{1}{d} \sum_{i=1}^{d} \psi_i(\boldsymbol{x}) \psi_i(\boldsymbol{z}) \tag{2.91}$$

and function f is given by

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} a_i k(\boldsymbol{x}, \boldsymbol{x}_i) \approx \sum_{i=1}^{d} a_i \boldsymbol{\psi}(\boldsymbol{x})^T \boldsymbol{\psi}(\boldsymbol{x}_i) = \sum_{i=1}^{d} a_i \psi_i(\boldsymbol{x}) = \boldsymbol{\psi}(\boldsymbol{x})^T \boldsymbol{\alpha} \tag{2.92}$$

where $\boldsymbol{\alpha}$ is

$$\boldsymbol{\alpha} = \sum_{i=1}^{d} a_i \boldsymbol{\psi}(\boldsymbol{x}_i) \in \mathbb{R}^d \tag{2.93}$$

Rather than being expressed as a linear combination of N functions $k(\boldsymbol{x}, \boldsymbol{x}_i)$, $f(\boldsymbol{x})$ is now represented as a linear combination of d functions $\psi_i(\boldsymbol{x})$.

Based on [52] it is shown that the norm f is given by

$$\begin{aligned} ||f||^2_{\mathcal{H}_k} &= \sum_{i=1}^{d} \sum_{j=1}^{d} \langle a_i, k(\boldsymbol{x}_i, \boldsymbol{x}_j) a_j \rangle \\ &\approx \sum_{i=1}^{d} \sum_{j=1}^{d} \langle a_i \boldsymbol{\psi}(\boldsymbol{x}_i), a_j \boldsymbol{\psi}(\boldsymbol{x}_j) \rangle \\ &= \left\langle \sum_{i=1}^{d} a_i \boldsymbol{\psi}(\boldsymbol{x}_i), \sum_{j=1}^{d} a_j \boldsymbol{\psi}(\boldsymbol{x}_j) \right\rangle \\ &= \langle \boldsymbol{\alpha}, \boldsymbol{\alpha} \rangle \\ &= ||\boldsymbol{\alpha}||^2 \end{aligned} \tag{2.94}$$

It is now possible to reformulate the optimization problem over $\boldsymbol{\alpha} \in \mathbb{R}^d$ instead. The following is based on [37].

Vector $\boldsymbol{f} = [f(\boldsymbol{x}_1), ..., f(\boldsymbol{x}_N)]^T$ is given by

$$\boldsymbol{f} = \boldsymbol{\Psi}^T \boldsymbol{\alpha} \tag{2.95}$$

where $\boldsymbol{\Psi}$ is matrix containing the transformed inputs, $\boldsymbol{\Psi}_{ij} = \psi_i(\boldsymbol{x}_j)$

$$\boldsymbol{\Psi} = [\boldsymbol{\psi}(\boldsymbol{x}_1), ..., \boldsymbol{\psi}(\boldsymbol{x}_N)] \in \mathbb{R}^{d \times N}, \ \boldsymbol{\Psi}^T = \begin{bmatrix} \boldsymbol{\psi}(\boldsymbol{x}_1) \\ \vdots \\ \boldsymbol{\psi}(\boldsymbol{x}_N) \end{bmatrix} \in \mathbb{R}^{N \times d} \tag{2.96}$$

The kernel matrix is given by

$$\boldsymbol{K} = \boldsymbol{\Psi}^T\boldsymbol{\Psi} \in \mathbb{R}^{N\times N} \tag{2.97}$$

with elements

$$\boldsymbol{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j) \approx \boldsymbol{\psi}(\boldsymbol{x}_i)^T\boldsymbol{\psi}(\boldsymbol{x}_j) \tag{2.98}$$

$\boldsymbol{\Psi}\boldsymbol{\Psi} \in \mathbb{R}^{d\times d}$, and has the following elements

$$\{\boldsymbol{\Psi}\boldsymbol{\Psi}^T\}_{ij} = \sum_{k=1}^{d} \boldsymbol{\psi}_k(\boldsymbol{x}_i)\boldsymbol{\psi}_k(\boldsymbol{x}_j) \tag{2.99}$$

By defining vector $\boldsymbol{y} = [y_1, ..., y_N]^T$, the optimization problem can be rewritten to

$$\begin{aligned}\min_{\boldsymbol{\alpha}\in\mathbb{R}^d} L[\boldsymbol{\alpha}] &= \frac{1}{N}||\boldsymbol{\Psi}^T\boldsymbol{\alpha} - \boldsymbol{y}||^2 + \lambda||\boldsymbol{\alpha}||^2 \\ &= \frac{1}{N}(\boldsymbol{\Psi}^T\boldsymbol{\alpha} - \boldsymbol{y})^T(\boldsymbol{\Psi}^T\boldsymbol{\alpha} - \boldsymbol{y}) + \lambda\boldsymbol{\alpha}^T\boldsymbol{\alpha} \\ &= \frac{1}{N}(\boldsymbol{\alpha}^T\boldsymbol{\Psi}\boldsymbol{\Psi}^T\boldsymbol{\alpha} - 2\boldsymbol{\alpha}\boldsymbol{\Psi}^T\boldsymbol{y} + \boldsymbol{y}^T\boldsymbol{y}) + \lambda\boldsymbol{\alpha}^T\boldsymbol{\alpha}\end{aligned} \tag{2.100}$$

By differentiating L with $\boldsymbol{\alpha}$, and setting it equal to zero, the minimum will be achieved. This gives the solution

$$\frac{1}{N}(-2\boldsymbol{\Psi}\boldsymbol{y} + 2\boldsymbol{\Psi}\boldsymbol{\Psi}^T\boldsymbol{\alpha}) + 2\lambda\boldsymbol{\alpha} = \boldsymbol{0} \tag{2.101}$$

$$(\boldsymbol{\Psi}\boldsymbol{\Psi}^T + \lambda N\boldsymbol{I})\boldsymbol{\alpha} = \boldsymbol{\Psi}\boldsymbol{y} \tag{2.102}$$

$$\boldsymbol{\alpha} = (\boldsymbol{\Psi}\boldsymbol{\Psi}^T + \lambda N\boldsymbol{I})^{-1}\boldsymbol{\Psi}\boldsymbol{y} \tag{2.103}$$

This solution is of dimension d.

2.4. Vector-Valued RKHS

This section is focused on the regularized least-squares problem for the case where the inputs and outputs can be vectors. This can be considered as an extension of the properties of the scalar case.

2.4.1. Moore-Aronszajn Theorem for the Vector Case

The Moore-Aronszajn theorem discussed in Section 2.1.2 can be generalized so that it applies to the case of vector-valued functions. This section is based on [39] and [38].

Consider a vector valued function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$, and a function $\boldsymbol{K} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^{m\times m}$. The function $\boldsymbol{K}$ is a positive definite kernel if

$$\boldsymbol{K}(\boldsymbol{x}, \boldsymbol{z})^T = \boldsymbol{K}(\boldsymbol{z}, \boldsymbol{x}), \quad \forall\, \boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^n \tag{2.104}$$

and

$$\sum_{i=1}^{N}\sum_{j=1}^{N}\langle \boldsymbol{y}_i, \boldsymbol{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)\boldsymbol{y}_j\rangle \geq \boldsymbol{0} \tag{2.105}$$

for every set of points $\{\boldsymbol{x}_i\}_{i=1}^N \in \mathbb{R}^n$ and $\{\boldsymbol{y}_i\}_{i=1}^N \in \mathbb{R}^m$. If both equation (2.104) and (2.105) are satisfied, then there exists a RKHS, $\mathcal{H}_K$, with a kernel $\boldsymbol{K}$.

Let $\boldsymbol{K}_{\boldsymbol{x}}\boldsymbol{y} : \mathbb{R}^n \to \mathbb{R}^m$ be a function given by

$$(\boldsymbol{K}_{\boldsymbol{x}}\boldsymbol{y})(\boldsymbol{z}) = \boldsymbol{K}(\boldsymbol{z}, \boldsymbol{x})\boldsymbol{y} \in \mathbb{R}^m, \quad \forall \boldsymbol{z} \in \mathbb{R}^n \tag{2.106}$$

Function $\boldsymbol{K}_{\boldsymbol{x}}\boldsymbol{y}$ defines the following set

$$\mathcal{H}_0 = \text{span}\{\boldsymbol{K}_{\boldsymbol{x}}\boldsymbol{y} \mid \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{y} \in \mathbb{R}^m\} \tag{2.107}$$

where the elements are functions from $\mathbb{R}^n$ to $\mathbb{R}^m$. Let function $\boldsymbol{f}$ and $\boldsymbol{g}$ be defined as followed

$$\boldsymbol{f} = \sum_{i=1}^{N} \boldsymbol{K}_{x_i}\boldsymbol{y}_i \in \mathcal{H}_0, \quad \boldsymbol{g} = \sum_{j=1}^{N} \boldsymbol{K}_{z_j}\boldsymbol{w}_j \in \mathcal{H}_0 \tag{2.108}$$

with inner product

$$\langle \boldsymbol{f}, \boldsymbol{g} \rangle_{\mathcal{H}_0} = \left\langle \sum_{i=1}^{N} \boldsymbol{K}_{\boldsymbol{x}_i} \boldsymbol{y}_i, \sum_{j=1}^{N} \boldsymbol{K}_{\boldsymbol{z}_j} \boldsymbol{w}_j \right\rangle_{\mathcal{H}_0} = \sum_{i,j=1}^{N} \langle \boldsymbol{y}_i, \boldsymbol{K}(\boldsymbol{x}_i, \boldsymbol{z}_j) \boldsymbol{w}_j \rangle \tag{2.109}$$

If $\mathcal{H}_K$ is the completion of $\mathcal{H}_0$, it is the RKHS of vector-valued functions from $\mathbb{R}^n$ to $\mathbb{R}^m$. The inner product can then be defined as

$$\langle \boldsymbol{K}_{\boldsymbol{x}} \boldsymbol{y}, \boldsymbol{K}_{\boldsymbol{z}} \boldsymbol{w} \rangle_{\mathcal{H}_K} = \langle \boldsymbol{y}, \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{z}) \boldsymbol{w} \rangle \tag{2.110}$$

The reproducing property is

$$\langle \boldsymbol{f}(\boldsymbol{x}), \boldsymbol{y} \rangle = \langle \boldsymbol{f}, \boldsymbol{K}_{\boldsymbol{x}} \boldsymbol{y} \rangle_{\mathcal{H}_K}, \quad \boldsymbol{f} \in \mathcal{H}_K \tag{2.111}$$

2.4.2. Vector-valued Curl-Free Kernel

Let $\boldsymbol{K}_{cf}(\boldsymbol{x}, \boldsymbol{z})$ be a matrix-valued curl-free kernel, where

$$\boldsymbol{K}_{cf}(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{G}_{cf}(\boldsymbol{x} - \boldsymbol{z}) \tag{2.112}$$

The kernel is found using the Hessian of the scalar Gaussian kernel [51]. In [36], the Hessian is defined as

$$\boldsymbol{G}_{cf}(\boldsymbol{x}) = -\nabla^2 g(\boldsymbol{x}) \tag{2.113}$$

where

$$\nabla^2 g(\boldsymbol{x}) = \left\{ \frac{\partial^2 g(\boldsymbol{x})}{\partial x_i \partial x_j} \right\} \tag{2.114}$$

and the scalar Gaussian kernel is defined in equation (2.76).

The expression for the matrix-valued curl-free kernel is therefore

$$\boldsymbol{K}_{cf}(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{G}_{cf}(\boldsymbol{x} - \boldsymbol{z}) = \frac{1}{\sigma^2} e^{-\frac{||\boldsymbol{x}-\boldsymbol{z}||^2}{2\sigma^2}} \left[\boldsymbol{I} - \left(\frac{\boldsymbol{x} - \boldsymbol{z}}{\sigma} \right) \left(\frac{\boldsymbol{x} - \boldsymbol{z}}{\sigma} \right)^T \right] \tag{2.115}$$

The selection of this kernel results in vector fields in the corresponding RKHS being curl-free. They can, according to [51], be interpreted as gradient flows with respect to a potential field $\boldsymbol{V}$

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}) = -\nabla \boldsymbol{V}(\boldsymbol{x}) \tag{2.116}$$

2.4.3. Vector-Valued Regularized Least-Squares

This section is based on [2] and [41]. Consider N inputs $\{\boldsymbol{x}_i\}_{i=1}^N$ and their corresponding outputs $\{\boldsymbol{y}_i\}_{i=1}^N$, where $\boldsymbol{x}_i \subseteq \mathbb{R}^n$ and $\boldsymbol{y}_i \subseteq \mathbb{R}^m$. Here, n and m represents the dimension of the inputs and outputs, respectively. This leads to $\boldsymbol{z} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$. In order to extend the regularized least-squares problem described in (2.77) to the vector-valued case, it can be rewritten as follows

$$\boldsymbol{f}_{\boldsymbol{z},\lambda} = \underset{\boldsymbol{f}\in\mathcal{H}_K}{\arg\min} \frac{1}{N}\sum_{i=1}^{N} ||\boldsymbol{f}(\boldsymbol{x}_i) - \boldsymbol{y}_i||^2_{\mathbb{R}^{mN}} + \lambda ||\boldsymbol{f}||^2_{\mathcal{H}_K} \tag{2.117}$$

In the scalar case, the solution to the problem is

$$\boldsymbol{f}(\boldsymbol{x}) = \sum_{i=1}^{N} \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{x}_i)\boldsymbol{a}_i \quad \in \mathbb{R}^m \tag{2.118}$$

given by the representer theorem, and the kernel being $\boldsymbol{K}(\boldsymbol{x}, \boldsymbol{x_i}) : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^{m\times m}$. When assuming that the kernel is Gaussian separable, it is given by

$$\boldsymbol{K}(\boldsymbol{x}, \boldsymbol{z}) = k(\boldsymbol{x}, \boldsymbol{z})\boldsymbol{I}_m = \begin{bmatrix} k(\boldsymbol{x}, \boldsymbol{z}) & 0 & \dots & 0 \\ 0 & k(\boldsymbol{x}, \boldsymbol{z}) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & k(\boldsymbol{x}, \boldsymbol{z}) \end{bmatrix} \tag{2.119}$$

where k: $\mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ is the scalar shift-invariant Gaussian kernel described in (2.75).

(2.118) can be written as

$$\boldsymbol{f}(\boldsymbol{x}) = [\boldsymbol{K}(\boldsymbol{x}, \boldsymbol{x}_1), ..., \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{x}_N)] \begin{bmatrix} \boldsymbol{a}_1 \\ \vdots \\ \boldsymbol{a}_N \end{bmatrix} \tag{2.120}$$

The coefficients $\boldsymbol{a}$ can be found from

$$(\boldsymbol{K} + \lambda N \boldsymbol{I}_{Nm})\boldsymbol{a} = \boldsymbol{y} \tag{2.121}$$

Which gives

$$\boldsymbol{a} = (\boldsymbol{K} + \lambda N \boldsymbol{I}_{Nm})^{-1} \boldsymbol{y} \tag{2.122}$$

This is the same as

$$\begin{bmatrix} \boldsymbol{a}_1 \\ \vdots \\ \boldsymbol{a}_N \end{bmatrix} = \left(\begin{bmatrix} \boldsymbol{K}(\boldsymbol{x}_1, \boldsymbol{x}_1) & \cdots & \boldsymbol{K}(\boldsymbol{x}_1, \boldsymbol{x}_N) \\ \vdots & \ddots & \vdots \\ \boldsymbol{K}(\boldsymbol{x}_N, \boldsymbol{x}_1) & \cdots & \boldsymbol{K}(\boldsymbol{x}_N, \boldsymbol{x}_N) \end{bmatrix} + N\lambda \boldsymbol{I}_{Nm} \right)^{-1} \begin{bmatrix} \boldsymbol{y}_1 \\ \vdots \\ \boldsymbol{y}_N \end{bmatrix} \tag{2.123}$$

One coefficient vector $\boldsymbol{a}_i \in \mathbb{R}^m$ is given by

$$\sum_{j=1}^{N} \boldsymbol{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) \boldsymbol{a}_j + N\lambda \boldsymbol{I}_N \boldsymbol{a}_i = \boldsymbol{y}_i, \; i = 1, ..., N, \; \boldsymbol{a}_i \in \mathbb{R}^m \tag{2.124}$$

Regarding the dimensions, $\boldsymbol{a} = [\boldsymbol{a}_1, ..., \boldsymbol{a}_N] \in \mathbb{R}^{Nm}$ where $\boldsymbol{a}_i \in \mathbb{R}^m$ and $\boldsymbol{y} = [\boldsymbol{y}_1, ..., \boldsymbol{y}_N] \in \mathbb{R}^{Nm}$. The kernel is a matrix $\boldsymbol{K} \in \mathbb{R}^{Nm \times Nm}$, where $\boldsymbol{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathbb{R}^{m \times m}$.

2.4.4. Random Fourier Features for Vector-Valued Functions

Random feature approximations to kernel functions can be extended to matrix-valued kernels [13, 39]. Given data points $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ for $i = 1, \ldots, N$, with $\boldsymbol{x}_i \in \mathbb{R}^n$ and $\boldsymbol{y}_i \in \mathbb{R}^m$, the optimization problem described in (2.88) can be rewritten so that it applies to vector-valued functions

$$\min_{\boldsymbol{f} \in \mathcal{H}_K} L[\boldsymbol{f}] = \frac{1}{N} \sum_{i=1}^{N} ||\boldsymbol{f}(\boldsymbol{x}_i) - \boldsymbol{y}_i||^2_{\mathbb{R}^{mN}} + \lambda ||\boldsymbol{f}||^2_{\mathcal{H}_K} \tag{2.125}$$

Given $\boldsymbol{f} = [f_1, ..., f_m]^T$, where $f_i : \mathbb{R}^m \to \mathbb{R}$, each function is real-valued and defined by the scalar kernel $k(\boldsymbol{x}, \boldsymbol{z})$ in the RKHS.

Recall that for a scalar kernel, an approximation can be defined as (2.90) and (2.91) by using RFF. Combining these two equations lets the scalar kernel expression be obtained as

$$K(\boldsymbol{x}, \boldsymbol{z}) \approx \boldsymbol{\psi}(\boldsymbol{x})^T \boldsymbol{\psi}(\boldsymbol{z}) = \sum_{i=1}^{d} \psi_i(\boldsymbol{x}) \psi_i(\boldsymbol{z}) \tag{2.126}$$

An approximation of the matrix kernel $\boldsymbol{K}(\boldsymbol{x}, \boldsymbol{z})$ can then be given by

$$\begin{aligned}\boldsymbol{K}(\boldsymbol{x}, \boldsymbol{z}) &= \boldsymbol{\psi}(\boldsymbol{x})^T \boldsymbol{\psi}(\boldsymbol{z}) \boldsymbol{I}_m \\ &= \begin{bmatrix} \boldsymbol{\psi}(\boldsymbol{x})^T \boldsymbol{\psi}(\boldsymbol{z}) & 0 & \dots & 0 \\ 0 & \boldsymbol{\psi}(\boldsymbol{x})^T \boldsymbol{\psi}(\boldsymbol{z}) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \boldsymbol{\psi}(\boldsymbol{x})^T \boldsymbol{\psi}(\boldsymbol{z}) \end{bmatrix}\end{aligned} \tag{2.127}$$

$\boldsymbol{\Psi}$ can be defined as

$$\boldsymbol{\Psi}(\boldsymbol{x}) = \boldsymbol{\psi}(\boldsymbol{x}) \otimes \boldsymbol{I}_m = \begin{bmatrix} \psi_1(\boldsymbol{x}) \boldsymbol{I}_m \\ \vdots \\ \psi_d(\boldsymbol{x}) \boldsymbol{I}_m \end{bmatrix} \in \mathbb{R}^{dm \times m} \tag{2.128}$$

and

$$\boldsymbol{\Psi}(\boldsymbol{x})^T = \boldsymbol{\psi}(\boldsymbol{x})^T \otimes \boldsymbol{I}_m = [\psi_1(\boldsymbol{x}) \boldsymbol{I}_m \ \dots \ \psi_d(\boldsymbol{x}) \boldsymbol{I}_m] \in \mathbb{R}^{m \times md} \tag{2.129}$$

where $\otimes$ is the Kronecker product and m is the dimension.

According to [52], the kernel matrix $\boldsymbol{K}$ can now be expressed as

$$\boldsymbol{K}(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{\Psi}(\boldsymbol{x})^T \boldsymbol{\Psi}(\boldsymbol{z}) \tag{2.130}$$

For the scalar case, the function $\boldsymbol{f}$ can be approximated as

$$\boldsymbol{f}(\boldsymbol{x}) = \sum_{i=1}^{N} \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{x}_i) \boldsymbol{a}_i \approx \sum_{i=1}^{N} \boldsymbol{\Psi}(\boldsymbol{x})^T \boldsymbol{\Psi}(\boldsymbol{x}_i) \boldsymbol{a}_i = \sum_{i=1}^{N} \boldsymbol{\Psi}(\boldsymbol{x}) \boldsymbol{a}_i = \boldsymbol{\Psi}(\boldsymbol{x})^T \boldsymbol{\alpha} \tag{2.131}$$

where $\boldsymbol{a}_i \in \mathbb{R}^m$ and

$$\boldsymbol{\alpha} = \sum_{i=1}^{N} \boldsymbol{\Psi}(\boldsymbol{x}_i) \boldsymbol{a}_i \in \mathbb{R}^{dm} \tag{2.132}$$

Since $\boldsymbol{\Psi}(\boldsymbol{x})$ is given by (2.128), function $\boldsymbol{f}(\boldsymbol{x})$ can be viewed as a linear combination of d functions $\psi_i(\boldsymbol{x})$. Therefore the approximation can be expressed as

$$\boldsymbol{f}(\boldsymbol{x}) = [\psi_1(\boldsymbol{x}) \boldsymbol{I}_m \dots \psi_d(\boldsymbol{x}) \boldsymbol{I}_m] \boldsymbol{\alpha} = \sum_{i=1}^{d} \psi(\boldsymbol{x}) \boldsymbol{I}_m \boldsymbol{\alpha} \tag{2.133}$$

The optimization problem can be reformulated to solve for $\boldsymbol{\alpha}$ instead of $\boldsymbol{f}$, which

reduces the problem's dimension to dm.

By using the same derivation as in equation (2.94), but replacing $\boldsymbol{f}$ with (2.131), the norm of $\boldsymbol{f}$ can be reformulated to

$$||\boldsymbol{f}||^2_{\mathcal{H}_K} = \langle \boldsymbol{\alpha}, \boldsymbol{\alpha} \rangle = ||\boldsymbol{\alpha}||^2 \tag{2.134}$$

$\boldsymbol{\Phi}$ can be defined as

$$\boldsymbol{\Phi} = \begin{bmatrix} \boldsymbol{\Psi}(\boldsymbol{x}_1)^T \\ \vdots \\ \boldsymbol{\Psi}(\boldsymbol{x}_N)^T \end{bmatrix} \in \mathbb{R}^{Nm \times dm} \tag{2.135}$$

Vector $\boldsymbol{F}$ can be written as

$$\boldsymbol{F} = \begin{bmatrix} \boldsymbol{f}(\boldsymbol{x}_1) \\ \vdots \\ \boldsymbol{f}(\boldsymbol{x}_N) \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Psi}(\boldsymbol{x}_1)^T \\ \vdots \\ \boldsymbol{\Psi}(\boldsymbol{x}_N)^T \end{bmatrix} \boldsymbol{\alpha} \quad \in \mathbb{R}^{Nm} \tag{2.136}$$

Which gives

$$\boldsymbol{F} = \boldsymbol{\Phi}\boldsymbol{\alpha} \tag{2.137}$$

The vector $\boldsymbol{Y}$ is given by

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{y}_1 \\ \vdots \\ \boldsymbol{y}_N \end{bmatrix} \in \mathbb{R}^{Nm} \tag{2.138}$$

By rewriting (2.125) to be solved for $\boldsymbol{\alpha}$, the following holds

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^{dm}} L[\boldsymbol{\alpha}] = \frac{1}{N}(\boldsymbol{\Phi}\boldsymbol{\alpha} - \boldsymbol{Y})^T(\boldsymbol{\Phi}\boldsymbol{\alpha} - \boldsymbol{Y}) + \lambda \boldsymbol{\alpha}^T \boldsymbol{\alpha} \tag{2.139}$$

To obtain the minimum for (2.139), the equation is differentiated with respect to $\boldsymbol{\alpha}$ and set equal to zero. The solution to the problem is then given by

$$(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda N \boldsymbol{I}_{dm})\boldsymbol{\alpha} = \boldsymbol{\Phi}^T\boldsymbol{Y} \tag{2.140}$$

$$\boldsymbol{\alpha} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda N \boldsymbol{I}_{dm})^{-1}\boldsymbol{\Phi}^T\boldsymbol{Y} \tag{2.141}$$

which, as mentioned, has the dimension dm.

Dimensions for curl-free kernel

In the case of a curl-free kernel approximation, some dimensions will be different than in the case of a Gaussian kernel.

Note that while $\boldsymbol{\Psi}(\boldsymbol{x})$ for the Gaussian kernel is given by

$$\boldsymbol{\Psi}(\boldsymbol{x}) = \boldsymbol{\psi}(\boldsymbol{x}) \otimes \boldsymbol{I}_m \ \in \mathbb{R}^{dm \times m} \tag{2.142}$$

$\boldsymbol{\Psi}_{cf}(\boldsymbol{x})$ for the curl-free kernel will be

$$\boldsymbol{\Psi}_{cf}(\boldsymbol{x}) = \boldsymbol{\psi}_{cf}(\boldsymbol{x}) \ \in \mathbb{R}^{d \times m} \tag{2.143}$$

This shows that the feature maps have different dimensions, where $\psi(\boldsymbol{x}) \in \mathbb{R}$ and $\psi_{cf}(\boldsymbol{x}) \in \mathbb{R}^m$. As for $\boldsymbol{\alpha}$, it has dimension d, and $\boldsymbol{\Phi} \in \mathbb{R}^{Nm \times d}$. On the other hand, the vectors $\boldsymbol{F}$ and $\boldsymbol{Y}$ will have the same dimensions as for the Gaussian kernel, which is Nm.

2.5. RKHS for Vector Fields

This section introduces an extension to regularized least-squares so that a constraint, based on contraction, can be applied to a subset of data points. This constraint is applied to a vector field, meaning the dimensions of $\boldsymbol{x}$ and $\boldsymbol{y}$ will be the same, thus implying $n = m$.

2.5.1. Contraction Analysis

This section is based on [35] and [51]. Consider the system

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) \tag{2.144}$$

where $\dot{\boldsymbol{x}} \in \mathbb{R}^m$ is the velocity vector at position $\boldsymbol{x} \in \mathbb{R}^m$. If $\boldsymbol{f}$ is continuously differentiable, then equation (2.144) will yield the differential relation

$$\delta\dot{\boldsymbol{x}} = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} \delta\boldsymbol{x} \tag{2.145}$$

where $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}$ is the Jacobian of $\boldsymbol{f}$.

Here, $\delta\boldsymbol{x}$ denotes a virtual displacement, an infinitesimal spatial displacement at a fixed time between neighboring trajectories. By defining the squared distance

between two neighboring trajectories as $\delta \boldsymbol{x}^T \delta \boldsymbol{x}$, the rate of change is given by

$$\frac{d}{dt}(\delta \boldsymbol{x}^T \delta \boldsymbol{x}) = 2\delta \boldsymbol{x}^T \delta \dot{\boldsymbol{x}} = 2\delta \boldsymbol{x}^T \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} \delta \boldsymbol{x} \tag{2.146}$$

Let $\lambda_{max}(\boldsymbol{x})$ denote the largest eigenvalue of the symmetric part of the Jacobian of $\boldsymbol{f}$. The symmetric part is given by

$$\frac{1}{2}\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}^T\right) \tag{2.147}$$

Assume that λ_{max} is strictly negative, which means that

$$\lambda_{max}(\boldsymbol{x}) \leq -\mu, \ \exists \mu(\boldsymbol{x}) > 0, \ \forall \boldsymbol{x} \geq 0 \tag{2.148}$$

If the Jacobian is negative definite, given by

$$\frac{1}{2}\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}^T\right) \leq -\mu(\boldsymbol{x})\boldsymbol{I} < 0 \tag{2.149}$$

then

$$\frac{d}{dt}(\delta \boldsymbol{x}^T \delta \boldsymbol{x}) \leq -2\mu(\boldsymbol{x})\delta \boldsymbol{x}^T \delta \boldsymbol{x} \tag{2.150}$$

and the distance between neighboring trajectories will shrink.

Integrating both sides of equation (2.150) results in

$$||\delta \boldsymbol{x}|| \leq ||\delta \boldsymbol{x}_0|| e^{-\int_0^t \mu(\boldsymbol{x})dt} \tag{2.151}$$

This shows that any infinitesimal length $||\delta \boldsymbol{x}||$ converges exponentially to zero.

Contraction region

This theory on contraction region can be found in [35]. Given a specific trajectory, consider a ball of constant radius centered around it. The ball is designed to remain within a contraction region at all times. This property arises due to the negative definiteness of the Jacobian matrix within the contraction region. As a result, any trajectory that initiates inside the ball will remain inside it and exponentially converge toward the given trajectory. Any length segment within the ball will decrease exponentially over time. Figure 2.2 illustrates this. Note that

if the whole state space is a contraction region, global exponential convergence will be guaranteed.

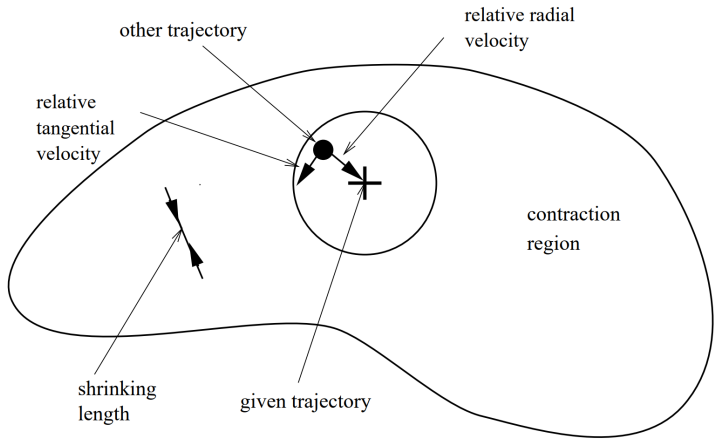


Figure 2.2.: Convergence of two trajectories. *Illustration* [35].

Hence, an open, connected set, also known as a region, of the state space for the system in equation (2.144) is a contraction region if the Jacobian of $\boldsymbol{f}$ is negative definite in that region. This ensures that any two trajectories that start within the region will converge exponentially toward each other.

2.5.2. Contraction as a Constraint

According to [35], a condition that must be satisfied for contraction to occur is

$$\boldsymbol{\nabla}^T \boldsymbol{f}(\boldsymbol{x}) + (\boldsymbol{\nabla}^T(\boldsymbol{f}(\boldsymbol{x}))^T \preceq -\mu \boldsymbol{I} \tag{2.152}$$

where $\boldsymbol{\nabla}^T$ denotes transposing the gradient. $\boldsymbol{\nabla}^T \boldsymbol{f}(\boldsymbol{x})$ is the Jacobian of $\boldsymbol{f}$ given by

$$\boldsymbol{\nabla}^T \boldsymbol{f}(\boldsymbol{x}) = \left\{ \frac{\partial f_i}{\partial x_j} \right\} \tag{2.153}$$

and

$$\boldsymbol{\nabla} \boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f_1(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\boldsymbol{x})}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial f_m(\boldsymbol{x})}{\partial x_m} \end{bmatrix} \tag{2.154}$$

Based on equation (2.129) and (2.131), the approximation of $\boldsymbol{f}$ can be written as

$$\begin{aligned}\boldsymbol{f}(\boldsymbol{x}) &= [\psi_1(\boldsymbol{x})\boldsymbol{I}_m \cdots \psi_d(\boldsymbol{x})\boldsymbol{I}_m]\begin{bmatrix}\boldsymbol{\alpha}_1\\ \vdots \\ \boldsymbol{\alpha}_d\end{bmatrix} \\ &= \psi_1(\boldsymbol{x})\boldsymbol{\alpha}_1 + \cdots + \psi_d(\boldsymbol{x})\boldsymbol{\alpha}_d\end{aligned} \tag{2.155}$$

where each element of $\boldsymbol{f}(\boldsymbol{x})$ is defined as

$$f_i(\boldsymbol{x}) = \psi_1(\boldsymbol{x})\alpha_{1i} + \cdots + \psi_d(\boldsymbol{x})\alpha_{di} \tag{2.156}$$

This gives

$$\frac{f_i(\boldsymbol{x})}{\partial x_j} = \frac{\psi_1(\boldsymbol{x})}{\partial x_j}\alpha_{1i} + \ldots + \frac{\psi_d(\boldsymbol{x})}{\partial x_j}\alpha_{di} \tag{2.157}$$

and

$$\boldsymbol{\nabla}\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} \frac{\psi_1(\boldsymbol{x})}{\partial x_1}\alpha_{11} + \cdots + \frac{\psi_d(\boldsymbol{x})}{\partial x_1}\alpha_{d1} & \cdots & \frac{\psi_1(\boldsymbol{x})}{\partial x_m}\alpha_{11} + \cdots + \frac{\psi_d(\boldsymbol{x})}{\partial x_m}\alpha_{d1} \\ \vdots & \ddots & \vdots \\ \frac{\psi_1(\boldsymbol{x})}{\partial x_1}\alpha_{1m} + \cdots + \frac{\psi_d(\boldsymbol{x})}{\partial x_1}\alpha_{dm} & \cdots & \frac{\psi_1(\boldsymbol{x})}{\partial x_m}\alpha_{1m} + \cdots + \frac{\psi_d(\boldsymbol{x})}{\partial x_m}\alpha_{dm} \end{bmatrix} \tag{2.158}$$

Based on equation (2.153) and (2.157), the Jacobian for the approximation of $\boldsymbol{f}$ can be expressed as

$$\boldsymbol{\nabla}^T\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{\nabla}^T(\boldsymbol{\Psi}(\boldsymbol{x})^T)\boldsymbol{\alpha} = \sum_{k=1}^{d}\boldsymbol{\alpha}_k\boldsymbol{\nabla}^T\psi_k(\boldsymbol{x}) \tag{2.159}$$

Upon examining equation (2.159), it can be seen that the inequality constraint expressed in equation (2.152) can be reformulated so that it is applicable for RFF

$$\sum_{k=1}^{d}(\boldsymbol{\alpha}_k\boldsymbol{\nabla}^T\psi_k(\boldsymbol{x}) + \boldsymbol{\nabla}\psi_k(\boldsymbol{x})\boldsymbol{\alpha}_k^T) \preceq -\mu\boldsymbol{I}_m \tag{2.160}$$

This inequality constraint is affine in $\boldsymbol{\alpha}_k$.

Gradient of the feature map

As mentioned in Section 2.2.5, the feature map $\psi_k(\boldsymbol{x})$ can be defined as

$$\psi_k(\boldsymbol{x}) = \sqrt{2}\cos(\boldsymbol{w}^T\boldsymbol{x} + b) \tag{2.161}$$

Then the derivative of ψ_k is

$$\nabla\psi_k(\boldsymbol{x}) = -\sqrt{2}\sin(\boldsymbol{w}^T\boldsymbol{x} + b)\boldsymbol{w} \tag{2.162}$$

2.5.3. Curl-Free Kernel Approximation with RFF

With the curl-free kernel approximation defined as in equation (2.65), the learned function $\boldsymbol{f}(\boldsymbol{x}) = \sum_{i=1}^{N} \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{x}_i)\boldsymbol{c}_i$ can be approximated with an RFF kernel approximation as

$$\begin{aligned}
\boldsymbol{f}(\boldsymbol{x}) &= \boldsymbol{\Psi}(\boldsymbol{x})^T\boldsymbol{\alpha} \\
&= \sqrt{\frac{2}{d}}[\sin(\boldsymbol{w}_1^T\boldsymbol{x} + b_1)\boldsymbol{w}_1^T \ldots \sin(\boldsymbol{w}_d^T\boldsymbol{x} + b_d)\boldsymbol{w}_d^T]\begin{bmatrix}\alpha_1 \\ \vdots \\ \alpha_d\end{bmatrix} \\
&= \sqrt{\frac{2}{d}}[\sin(\boldsymbol{w}_1^T\boldsymbol{x} + b_1)\boldsymbol{w}_1^T\alpha_1 + \ldots + \sin(\boldsymbol{w}_d^T\boldsymbol{x} + b_d)\boldsymbol{w}_d^T\alpha_d] \in \mathbb{R}^m
\end{aligned} \tag{2.163}$$

where $\{\alpha_k\}_{k=1}^d \in \mathbb{R}$ are the scalar components of $\boldsymbol{\alpha} \in \mathbb{R}^m$. Each of the m components of $\boldsymbol{f}$ can be written as a sum

$$\begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_m(\boldsymbol{x}) \end{bmatrix} = \sqrt{\frac{2}{d}}\begin{bmatrix} \sin(\boldsymbol{w}_1^T\boldsymbol{x} + b_1)w_{11}\alpha_1 + \ldots + \sin(\boldsymbol{w}_d^T\boldsymbol{x} + b_1)w_{d1}\alpha_d \\ \vdots \\ \sin(\boldsymbol{w}_1^T\boldsymbol{x} + b_1)w_{1m}\alpha_1 + \ldots + \sin(\boldsymbol{w}_d^T\boldsymbol{x} + b_1)w_{dm}\alpha_d \end{bmatrix} \tag{2.164}$$

where $\{w_{ij}\}_{j=1}^m \in \mathbb{R}$ denotes the scalar components of the vectors $\{\boldsymbol{w}_i\}_{i=1}^d \in \mathbb{R}^m$.

Using the RFF approximation, the expression for the Jacobian of $\boldsymbol{f}$ can now be obtained

$$\boldsymbol{\nabla f}(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f_1(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\boldsymbol{x})}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m(\boldsymbol{x})}{\partial x_1} & \cdots & \frac{\partial f_m(\boldsymbol{x})}{\partial x_n} \end{bmatrix} \tag{2.165}$$

Observe that each element in the Jacobian can be expressed as

$$\frac{\partial f_i(\boldsymbol{x})}{\partial x_j} = \sqrt{\frac{2}{d}} \left(w_{1j} \cos(\boldsymbol{w}_1^T \boldsymbol{x} + b_1) w_{1i} \alpha_1 + \ldots + w_{dj} \cos(\boldsymbol{w}_d^T \boldsymbol{x} + b_d) w_{di} \alpha_d \right) \tag{2.166}$$

The Jacobian can now be written as the sum of outer products as seen below

$$\begin{aligned} D\boldsymbol{f}(\boldsymbol{x}) &= \sum_{k=1}^{d} \boldsymbol{w}_k \boldsymbol{\nabla}^T \left(\sqrt{\frac{2}{d}} \sin(\boldsymbol{w}_k^T \boldsymbol{x} + b_k) \right) \alpha_k \\ &= \sum_{k=1}^{d} \boldsymbol{w}_k \sqrt{\frac{2}{d}} \begin{bmatrix} \cos(\boldsymbol{w}_k^T \boldsymbol{x} + b_k) w_{k1} \\ \vdots \\ \cos(\boldsymbol{w}_k^T \boldsymbol{x} + b_k) w_{km} \end{bmatrix}^T \alpha_k \end{aligned} \tag{2.167}$$

2.5.4. Regularized Least-Squares with Contraction Constraints

In [51] the optimization problem is given by

$$\min_{\boldsymbol{f} \in \mathcal{H}_K} L[\boldsymbol{f}] = \frac{1}{N} \sum_{i=1}^{N} ||\boldsymbol{f}(\boldsymbol{x}_i) - \boldsymbol{y}_i||^2 + \lambda ||\boldsymbol{f}||_{\mathcal{H}_K}^2 \tag{2.168}$$

$$\text{subject to } \frac{1}{2}(\boldsymbol{J}(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})^T) \preceq -\mu \boldsymbol{I}_m \tag{2.169}$$

The notation $\preceq$ in equation (2.169) indicates that the left-hand side of the equation is negative definite with eigenvalues that are less than or equal to $-\mu$, where μ is a positive scalar.

$\boldsymbol{J}(\boldsymbol{x})$ is the Jacobian of $\boldsymbol{f}(\boldsymbol{x})$, defined as

$$\boldsymbol{J}(\boldsymbol{x}) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \boldsymbol{\nabla}^T \boldsymbol{f}(\boldsymbol{x}) \ \in \mathbb{R}^{m \times m} \tag{2.170}$$

It should be noted that the contraction constraints in equation (2.169) can be enforced using only a subset of points M.

2.5.5. Random Fourier Features with Contraction Constraints

It is possible to reformulate regularized least-squares with a contraction condition so that it can be applied to the usage of RFF. The optimization problem needs to be reformulated so that it is solved for $\boldsymbol{\alpha}$. Based on equation (2.137) and (2.138), the problem can be written as

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^d} L[\boldsymbol{\alpha}] = (\boldsymbol{\Phi}\boldsymbol{\alpha} - \boldsymbol{Y})^T(\boldsymbol{\Phi}\boldsymbol{\alpha} - \boldsymbol{Y}) + \lambda\boldsymbol{\alpha}^T\boldsymbol{\alpha} \tag{2.171}$$

When RFF is used, the Jacobian is given by

$$\boldsymbol{J}(\boldsymbol{x}) = \boldsymbol{\nabla}^T(\boldsymbol{\Psi}(\boldsymbol{x})^T)\boldsymbol{\alpha} \tag{2.172}$$

as shown in Section 2.5.2.

The optimization problem with a contraction condition can therefore be expressed as

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^{dm}} L[\boldsymbol{\alpha}] = (\boldsymbol{\Phi}\boldsymbol{\alpha} - \boldsymbol{Y})^T(\boldsymbol{\Phi}\boldsymbol{\alpha} - \boldsymbol{Y}) + \lambda\boldsymbol{\alpha}^T\boldsymbol{\alpha} \tag{2.173}$$

$$\text{subject to } \frac{1}{2}(\boldsymbol{\nabla}^T(\boldsymbol{\Psi}(\boldsymbol{x})^T)\boldsymbol{\alpha} + \boldsymbol{\alpha}^T\boldsymbol{\nabla}\boldsymbol{\Psi}(\boldsymbol{x})) \preceq -\mu\boldsymbol{I}_m \tag{2.174}$$

2.6. Vanishing Point

Further work in [51] leads to developing a set of vector fields that vanish at specific points of interest. These targeted points are essentially the equilibrium points one aims to learn in the associated dynamical system. An equilibrium state, also known as a steady state, is a state where the system is stable and does not change over time [23].

2.6.1. RKHS Vector Fields Vanishing on a Point Set

Given a RKHS denoted as $\mathcal{H}_K$, consider the subset of functions that vanish on a set of points $Z = \{\boldsymbol{x}_1^*, ..., \boldsymbol{x}_p^*\}$. The points in Z represent the desired equilibrium points.

$$\mathcal{H}_K^Z = \{\boldsymbol{f} \in \mathcal{H}_K : \boldsymbol{f}(\boldsymbol{x}_i^*) = 0 \in \mathbb{R}^m, \boldsymbol{x}_i^* \in Z\} \tag{2.175}$$

It should be emphasized that $\mathcal{H}_K^Z$ forms a closed subspace of $\mathcal{H}_K$ and, as such, constitutes a distinct RKHS that is associated with a modified kernel function K^Z.

Let $\mathcal{H}_K^Z \subseteq \mathcal{H}_K$, which is an RKHS defined by the matrix-valued kernel expressed as

$$\boldsymbol{K}^Z(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{z}) - \boldsymbol{K}(\boldsymbol{x}, Z)\boldsymbol{K}(Z, Z)^{-1}\boldsymbol{K}(Z, \boldsymbol{z}) \tag{2.176}$$

The equation above is a generalization of Theorem 116 in [9]. Here, $S = \{\boldsymbol{x}_i \in \mathbb{R}^m\}_{i=1}^N$ and $S' = \{\boldsymbol{z}_i \in \mathbb{R}^m\}_{i=1}^{N'}$ is any two sets of points. $\boldsymbol{K}(S', S)$ is the Gram matrix on any matrix-valued kernel $\boldsymbol{K}$ on S, S', which is an $N'm \times Nm$ matrix defined by the $n \times n$ blocks given by

$$\boldsymbol{G}_{ij} = \boldsymbol{K}(\boldsymbol{z}_i, \boldsymbol{x}_j) \in \mathbb{R}^{m \times m} \tag{2.177}$$

Therefore, starting with any base matrix-valued kernel $\boldsymbol{K}$, $\boldsymbol{K}^Z$ can be defined as in (2.176). By doing so, one can generate its corresponding RKHS as a subset of vector fields that are ensured to vanish on Z, the targeted set of equilibrium points.

Regularized least-squares with vanishing point

Let $(\boldsymbol{x}_i, \boldsymbol{z}_i)$ for $i = 1, ..., N$ and $\boldsymbol{x}_i, \boldsymbol{z}_i \in \mathbb{R}^m$ be a set of data points. If the objective is to make vector fields vanish at certain points, the resulting optimization problem will be

$$\boldsymbol{f}_{z,\lambda} = \underset{\boldsymbol{f} \in \mathcal{H}_K^Z}{\arg\min} \frac{1}{N} \sum_{i=1}^{N} ||\boldsymbol{f}(\boldsymbol{x}_i) - \boldsymbol{z}_i||_{\mathbb{R}^{\mathbb{N}}}^2 + \lambda ||\boldsymbol{f}||_{\mathcal{H}_K^Z}^2 \tag{2.178}$$

$$\text{subject to } \frac{1}{2}(\boldsymbol{J}(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})^T) \preceq -\mu \boldsymbol{I}_m \tag{2.179}$$

Where $\boldsymbol{J}(\boldsymbol{x})$ is the Jacobian of function $\boldsymbol{f}$, described in equation (2.170).

The problem is now over the subset $\mathcal{H}_K^Z$, instead of $\mathcal{H}_K$.

2.6.2. Random Fourier Features Vanishing on a Point Set

This section will discuss the steps to transform a matrix-valued feature map ψ to ψ^Z, so that $\psi^Z(\boldsymbol{x})$ vanishes on Z, according to the procedure in [51].

Given a set of points $X = (\boldsymbol{x}_1, ..., \boldsymbol{x}_N)$, a matrix-valued feature map is defined

$$\boldsymbol{\Psi}(X) = [\boldsymbol{\Psi}(\boldsymbol{x}_1), ..., \boldsymbol{\Psi}(\boldsymbol{x}_Z)] \in \mathbb{R}^{dm \times Nn} \tag{2.180}$$

For $\boldsymbol{K}(\boldsymbol{x}, \boldsymbol{z}) \approx \boldsymbol{\Psi}(\boldsymbol{x})^T \boldsymbol{\Psi}(\boldsymbol{z})$ it can be derived that

$$\begin{aligned}
\boldsymbol{K}^Z(\boldsymbol{x}, \boldsymbol{z}) &= \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{z}) - \boldsymbol{K}(\boldsymbol{x}, Z)\boldsymbol{K}(Z, Z)^{-1}\boldsymbol{K}(Z, \boldsymbol{z}) \\
&= \boldsymbol{\Psi}(\boldsymbol{x})^T \boldsymbol{\Psi}(\boldsymbol{z}) - \boldsymbol{\Psi}(\boldsymbol{x})^T \boldsymbol{\Psi}(Z)(\boldsymbol{\Psi}(Z)^T \boldsymbol{\Psi}(Z))^{-1} \boldsymbol{\Psi}(Z)^T \boldsymbol{\Psi}(\boldsymbol{z}) \\
&= \boldsymbol{\Psi}(\boldsymbol{x})^T [\boldsymbol{I} - \boldsymbol{\Psi}(Z)(\boldsymbol{\Psi}(Z)^T \boldsymbol{\Psi}(Z))^{-1} \boldsymbol{\Psi}(Z)^T] \boldsymbol{\Psi}(\boldsymbol{z}) \\
&= \boldsymbol{\Psi}(\boldsymbol{x})^T [\boldsymbol{I} - \boldsymbol{P}_{\boldsymbol{\Psi}(Z)}] \boldsymbol{\Psi}(\boldsymbol{z}) \\
&= \boldsymbol{\Psi}(\boldsymbol{x})^T \boldsymbol{P}^{\perp}_{\boldsymbol{\Psi}(Z)} \boldsymbol{\Psi}(\boldsymbol{z})
\end{aligned} \tag{2.181}$$

where $\boldsymbol{P}_{\boldsymbol{\Psi}(Z)}$ is the orthogonal projector onto the range of $\boldsymbol{\Psi}(Z)$.

It follows that the matrix $\boldsymbol{P}^{\perp}_{\boldsymbol{\Psi}(Z)}$ can be factorized as the product of a lower-triangular matrix $\boldsymbol{L}$ and its transpose $\boldsymbol{L}^T$ using Cholesky factorization [12], given by

$$\boldsymbol{P}^{\perp}_{\boldsymbol{\Psi}(Z)} = \boldsymbol{L}\boldsymbol{L}^T \tag{2.182}$$

Therefore, in order to find $\boldsymbol{L}$ in equation (2.185), it is necessary to first compute $\boldsymbol{P}^{\perp}_{\boldsymbol{\Psi}(Z)}$. It can be observed from equation (2.181) that

$$\boldsymbol{P}_{\boldsymbol{\Psi}(Z)} = \boldsymbol{\Psi}(Z)(\boldsymbol{\Psi}(Z)^T \boldsymbol{\Psi}(Z))^{-1} \boldsymbol{\Psi}(Z)^T \tag{2.183}$$

and

$$\boldsymbol{P}^{\perp}_{\boldsymbol{\Psi}} = \boldsymbol{I} - \boldsymbol{P}_{\boldsymbol{\Psi}(Z)} \tag{2.184}$$

Note that this is for some $\boldsymbol{L} \in \mathbb{R}^{dm \times dm}$ and $\boldsymbol{P}^{\perp}_{\boldsymbol{\Psi}(Z)} \in \mathbb{R}^{dm \times dm}$ in the case of a Gaussian separable kernel. For a curl-free kernel the dimensions will be $\boldsymbol{L} \in \mathbb{R}^{d \times d}$ and $\boldsymbol{P}^{\perp}_{\boldsymbol{\Psi}(Z)} \in \mathbb{R}^{d \times d}$. The dimensions of the identity matrix $\boldsymbol{I}$ vary depending on which kernel is used and is of the same size as $\boldsymbol{P}_{\boldsymbol{\Psi}(Z)}$. It is important to highlight that, in the presence of numerical noise, ensuring that matrix $\boldsymbol{P}$ is Hermitian and positive-definite is important.

This leads to a new feature map, which can be defined as

$$\boldsymbol{\Psi}^Z(\boldsymbol{x}) = \boldsymbol{L}^T \boldsymbol{\Psi}(\boldsymbol{x}) \tag{2.185}$$

The new feature map satisfies

$$\boldsymbol{K}^Z(\boldsymbol{x},\boldsymbol{z}) = \boldsymbol{\Psi}^Z(\boldsymbol{x})^T\boldsymbol{\Psi}^Z(\boldsymbol{y}) \tag{2.186}$$

Notice that even though the kernel $\boldsymbol{K}^Z(\boldsymbol{x},\boldsymbol{z})$ is not shift-invariant, this particular structure still retains the ability to be a low-rank feature map while ensuring that $\boldsymbol{\Psi}^Z(\boldsymbol{x})$ vanishes on Z.

Regularized least-squares with RFF vanishing on a point set

Given a set of points $(\boldsymbol{x}_i,\ \boldsymbol{y}_i)$ for $i = 1,\dots,N$, where $\boldsymbol{x}_i, \boldsymbol{y}_i \in \mathbb{R}^m$, and a chosen subset of the data points, given by $(\boldsymbol{x}_j,\ \boldsymbol{y}_j)$ for $j = 1,\dots,M$. If it is desired that vector fields vanish at specific points, the optimization problem becomes

$$\min_{\boldsymbol{\alpha}\in\mathbb{R}^{dm}} L[\boldsymbol{\alpha}] = (\boldsymbol{\Phi}^{\boldsymbol{Z}}\boldsymbol{\alpha} - \boldsymbol{Y})^T(\boldsymbol{\Phi}^{\boldsymbol{Z}}\boldsymbol{\alpha} - \boldsymbol{Y}) + \lambda\boldsymbol{\alpha}^T\boldsymbol{\alpha} \tag{2.187}$$

$$\text{subject to } \frac{1}{2}\sum_{i=1}^{M}(\boldsymbol{J}_{\boldsymbol{\Psi}_i^Z}(\boldsymbol{x}_i)\boldsymbol{\alpha}_i + \boldsymbol{J}^T_{\boldsymbol{\Psi}_i^Z}(\boldsymbol{x}_i)\boldsymbol{\alpha}_i) \preceq -\mu\boldsymbol{I}_m \tag{2.188}$$

Compared to equation (2.179) it can be seen that

$$\boldsymbol{J}(\boldsymbol{x}) = \boldsymbol{J}_{\boldsymbol{\Psi}^Z}(\boldsymbol{x})\boldsymbol{\alpha} \tag{2.189}$$

$\boldsymbol{J}_{\boldsymbol{\Psi}_i^Z}$ denotes the $m \times m$ Jacobian of $\boldsymbol{\Psi}_i^Z$ [51]. This leads to

$$\boldsymbol{J}_{\boldsymbol{\Psi}^Z}(\boldsymbol{x}) = \boldsymbol{\nabla}^T(\boldsymbol{\Psi}^Z(\boldsymbol{x})^T) = \boldsymbol{\nabla}^T(\boldsymbol{\Psi}(\boldsymbol{x})^T L) \tag{2.190}$$

$\boldsymbol{\Phi}^Z$ is given by

$$\boldsymbol{\Phi}^Z = \begin{bmatrix} \boldsymbol{\Psi}^Z(\boldsymbol{x}_1)^T \\ \vdots \\ \boldsymbol{\Psi}^Z(\boldsymbol{x}_N)^T \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Psi}(\boldsymbol{x}_1)^T L \\ \vdots \\ \boldsymbol{\Psi}(\boldsymbol{x}_N)^T L \end{bmatrix} \tag{2.191}$$

The dimension for $\boldsymbol{\Phi}^Z$ and $\boldsymbol{\alpha}$ will be the same as in Section 2.4.4.

Note that

$$\boldsymbol{J}_{\boldsymbol{\Psi}^Z}(\boldsymbol{x})\boldsymbol{\alpha} = \begin{bmatrix} \frac{\psi_1^Z(\boldsymbol{x})}{\partial x_1}\alpha_{11} + \cdots + \frac{\psi_d^Z(\boldsymbol{x})}{\partial x_1}\alpha_{d1} & \cdots & \frac{\psi_1^Z(\boldsymbol{x})}{\partial x_m}\alpha_{11} + \cdots + \frac{\psi_d^Z(\boldsymbol{x})}{\partial x_m}\alpha_{d1} \\ \vdots & \ddots & \vdots \\ \frac{\psi_1^Z(\boldsymbol{x})}{\partial x_1}\alpha_{1m} + \cdots + \frac{\psi_d^Z(\boldsymbol{x})}{\partial x_1}\alpha_{dm} & \cdots & \frac{\psi_1^Z(\boldsymbol{x})}{\partial x_m}\alpha_{1m} + \cdots + \frac{\psi_d^Z(\boldsymbol{x})}{\partial x_m}\alpha_{dm} \end{bmatrix} \tag{2.192}$$

2.7. Hamiltonian Dynamics

A Hamiltonian system is a well-established framework for describing the time evolution of systems that possess conserved quantities, known as *Hamiltonians* [55]. One of the most significant of these conserved quantities is the system's total energy, which remains constant over time. This section explores the fundamentals of Hamiltonian dynamics, including the Hamiltonian function, the Hamiltonian equations of motion, the symplectic kernel, and the symplectic integrator. The applications of Hamiltonian dynamics for a pendulum will also be presented. The theoretical framework for Hamiltonian dynamics presented in this section is primarily based on [20] and [42].

2.7.1. Hamilton's Equations of Motion

Hamilton's equations of motion are closely related to Lagrange's equation of motion and rely on generalized coordinates and energy expressions [20].

Consider the Lagrangian

$$\mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) = T(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) - U(\boldsymbol{q}) \tag{2.193}$$

with generalized coordinates $\boldsymbol{q}$, kinetic energy $T(\cdot)$ and potential energy $U(\cdot)$.

The Hamiltonian H can be described from the Lagrangian $\mathcal{L}$, where a change of variables is needed. While the Lagrangian variables are $(\boldsymbol{q}, \dot{\boldsymbol{q}})$, the Hamiltonian's are $(\boldsymbol{q}, \boldsymbol{p}) \in \mathbb{R}^d$. Usually, the variables $\boldsymbol{q} = (q_1, ..., q_N)$ and $\boldsymbol{p} = (p_1, ..., p_N)$ represent the position and the momentum respectively. The rationale behind this change can be explored in greater depth in [42].

Defining the vector of momentum variables as

$$\boldsymbol{p} = \frac{\partial L(\boldsymbol{q}, \boldsymbol{\phi}(\boldsymbol{q}, \boldsymbol{p}, t))}{\partial \boldsymbol{\phi}} \tag{2.194}$$

and using Legendre transformation leads to the Hamiltonian being defined as

$$H(\boldsymbol{q}, \boldsymbol{p}, t) = \boldsymbol{p}^T \boldsymbol{\phi}(\boldsymbol{q}, \boldsymbol{p}, t) - L(\boldsymbol{q}, \boldsymbol{\phi}(\boldsymbol{q}, \boldsymbol{p}, t), t) \tag{2.195}$$

Through a system of ordinary differential equations called Hamilton's equations of motion

$$\dot{\boldsymbol{q}} = \frac{\partial H(\boldsymbol{q}, \boldsymbol{p}, t)^T}{\partial \boldsymbol{p}}, \qquad \dot{\boldsymbol{p}} = -\frac{\partial H(\boldsymbol{q}, \boldsymbol{p}, t)^T}{\partial \boldsymbol{q}} + \boldsymbol{\tau} \tag{2.196}$$

the evolution of the Hamiltonian function $H(\boldsymbol{q}, \boldsymbol{p})$ can be described. The dot notation represents derivatives with respect to the time variable t [17].

Note that for many mechanical systems, for instance, harmonic oscillators, the Hamiltonian will be the total energy of the system. In other words

$$H(\boldsymbol{q}, \boldsymbol{p}) = T(\boldsymbol{p}) + U(\boldsymbol{q}) \tag{2.197}$$

The time derivative of the Hamiltonian

The time derivative of the Hamiltonian is an important property regarding the energy conservation of the system. From [20], the time derivative of the Hamiltonian is derived from the chain rule

$$\frac{dH}{dt} = \frac{\partial H}{\partial \boldsymbol{p}}\dot{\boldsymbol{p}} + \frac{\partial H}{\partial \boldsymbol{q}}\dot{\boldsymbol{q}} + \frac{\partial H}{\partial t} \tag{2.198}$$

Inserting $\dot{\boldsymbol{q}}$ and $\dot{\boldsymbol{p}}$ from equation (2.196) gives

$$\frac{dH}{dt} = \dot{\boldsymbol{q}}^T \left(-\frac{\partial H^T}{\partial \boldsymbol{q}} + \boldsymbol{\tau} \right) + \frac{\partial H}{\partial \boldsymbol{q}}\dot{\boldsymbol{q}} + \frac{\partial H}{\partial t} = \dot{\boldsymbol{q}}^T \boldsymbol{\tau} + \frac{\partial H}{\partial t} \tag{2.199}$$

This leads to the time derivative of the Hamiltonian

$$\frac{dH}{dt} = \dot{\boldsymbol{q}}^T \boldsymbol{\tau} + \frac{\partial H}{\partial t} \tag{2.200}$$

Now suppose the Hamiltonian is time independent, that is $H = H(\boldsymbol{q}, \boldsymbol{p})$, and the system is unactuated, meaning $\boldsymbol{\tau} = \boldsymbol{0}$, then

$$\frac{dH(\boldsymbol{q}, \boldsymbol{p})}{dt} = 0 \tag{2.201}$$

2.7.2. Hamiltonian Dynamics

Now that Hamilton's equation of motion is introduced, the Hamiltonian system with symplectic dynamics [11] can be described by

$$\dot{\boldsymbol{q}} = \frac{\partial H(\boldsymbol{q}, \boldsymbol{p})}{\partial \boldsymbol{p}}, \qquad \dot{\boldsymbol{p}} = -\frac{\partial H(\boldsymbol{q}, \boldsymbol{p})}{\partial \boldsymbol{q}} \tag{2.202}$$

where $\boldsymbol{q}, \boldsymbol{p} \in \mathbb{R}^n$, and the state vector becomes $\boldsymbol{x} = [\boldsymbol{q}^T, \boldsymbol{p}^T] \in \mathbb{R}^{2n}$.

This gives the gradient of H

$$\boldsymbol{\nabla} H = \begin{bmatrix} \boldsymbol{\nabla}_q H \\ \boldsymbol{\nabla}_p H \end{bmatrix} \tag{2.203}$$

and an alternative notation of the dynamics is

$$\begin{bmatrix} \dot{\boldsymbol{q}} \\ \dot{\boldsymbol{p}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} \\ -\boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\nabla}_q H \\ \boldsymbol{\nabla}_p H \end{bmatrix} \tag{2.204}$$

From [11], the symplectic $2n \times 2n$ matrix is defined as

$$\boldsymbol{J} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{I} \\ -\boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \tag{2.205}$$

where $\boldsymbol{I}$ is the identity matrix of the dimension of $\boldsymbol{p}$ or $\boldsymbol{q}$ [24]. The dynamical system is then given by

$$\dot{\boldsymbol{x}} = \boldsymbol{J} \boldsymbol{\nabla} H(\boldsymbol{x}) \tag{2.206}$$

Note that a system is Hamiltonian if and only if it is symplectic [20].

2.7.3. Symplectic Kernel

To ensure that the kernel used in Hamiltonian dynamic modeling satisfies the symplectic property, the curl-free kernel is extended by adding a term proportional to the gradient of a scalar potential function.

Assume a curl-free kernel $\boldsymbol{K}_{cf}(\boldsymbol{x}, \boldsymbol{z}) = \boldsymbol{G}_{cf}(\boldsymbol{x} - \boldsymbol{z})$ as described in equation (2.115). Then according to [11], the symplectic kernel is defined as

$$\boldsymbol{K}(\boldsymbol{x}, \boldsymbol{z}) = -\boldsymbol{J} \boldsymbol{\nabla}^2 \boldsymbol{g}(\boldsymbol{x} - \boldsymbol{z}) \boldsymbol{J}^T \tag{2.207}$$

where $\mathcal{H}$ is a RKHS defined by the kernel $\boldsymbol{K}$. Hence, it becomes possible to represent any function $\boldsymbol{f}$ belonging to $\mathcal{H}$ in the following form

$$\boldsymbol{f}(\boldsymbol{x}) = \sum_{i=1}^{N} \boldsymbol{K}(\boldsymbol{x}, \boldsymbol{x}_i) \boldsymbol{a}_i = \sum_{i=1}^{N} \boldsymbol{G}(\boldsymbol{x} - \boldsymbol{x}_i) \boldsymbol{a}_i \tag{2.208}$$

Inserting (2.207) into the equation above gives

$$\begin{aligned}\boldsymbol{f}(\boldsymbol{x}) &= -\boldsymbol{J}\boldsymbol{\nabla}\sum_{i=1}^{N}\boldsymbol{\nabla}^T g(\boldsymbol{x}-\boldsymbol{x}_i)\boldsymbol{J}^T\boldsymbol{a}_i \\ &= -\boldsymbol{J}\boldsymbol{\nabla}\sum_{i=1}^{N}\boldsymbol{\nabla}^T g(\boldsymbol{x}-\boldsymbol{x}_i)\boldsymbol{\alpha}_i\end{aligned} \tag{2.209}$$

where $\boldsymbol{\alpha}_i = \boldsymbol{J}^T\boldsymbol{a}_i$. Defining the Hamiltonian as

$$H(\boldsymbol{x}) = -\sum_{i=1}^{N}\boldsymbol{\nabla}^T g(\boldsymbol{x}-\boldsymbol{x}_i)\boldsymbol{\alpha}_i \tag{2.210}$$

leads to

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{J}\boldsymbol{\nabla}H(\boldsymbol{x}) = \dot{\boldsymbol{x}} \tag{2.211}$$

this describes the Hamiltonian dynamics, and note that this corresponds to the equation described in (2.206).

2.7.4. Symplectic Kernel from a Gaussian Kernel

A symplectic kernel can be derived from the Gaussian separable kernel defined in equation (2.76). Similarly, as for the derivation of a curl-free kernel from a Gaussian kernel, the symplectic kernel can be defined as

$$\boldsymbol{G}(\boldsymbol{x}) = \frac{1}{2\sigma^2}e^{-\frac{\boldsymbol{x}^T\boldsymbol{x}}{2\sigma^2}}\boldsymbol{J}\left(\boldsymbol{I}-\frac{\boldsymbol{x}\boldsymbol{x}^T}{\sigma^2}\right)\boldsymbol{J}^T \tag{2.212}$$

In the case of real-valued RFF, the approximation of the symplectic kernel is

$$\boldsymbol{K}(\boldsymbol{x},\boldsymbol{z}) = \boldsymbol{G}(\boldsymbol{x}-\boldsymbol{z}) = \boldsymbol{J}\boldsymbol{\Psi}(\boldsymbol{x})^T\boldsymbol{\Psi}(\boldsymbol{z})\boldsymbol{J}^T \tag{2.213}$$

which is derived from

$$\begin{aligned}\boldsymbol{G}(\boldsymbol{x}-\boldsymbol{z}) &= \boldsymbol{J}\int_{\mathbb{R}^n}\boldsymbol{w}\boldsymbol{w}^T e^{-i\boldsymbol{w}^T(\boldsymbol{x}-\boldsymbol{z})}p(\boldsymbol{x})d\boldsymbol{w}\boldsymbol{J}^T \\ &= \boldsymbol{J}\int_{\mathbb{R}^n}\boldsymbol{w}\boldsymbol{w}^T\cos(\boldsymbol{w}^T(\boldsymbol{x}-\boldsymbol{z}))p(\boldsymbol{w})d\boldsymbol{w}\boldsymbol{J}^T\end{aligned} \tag{2.214}$$

Here, $\boldsymbol{\Psi(x)}$ is given by

$$\boldsymbol{\Psi}(\boldsymbol{x}) = \frac{1}{\sqrt{d}} \begin{bmatrix} \cos(\boldsymbol{w}_1^T \boldsymbol{x}) \boldsymbol{w}_1^T \\ \sin(\boldsymbol{w}_1^T \boldsymbol{x}) \boldsymbol{w}_1^T \\ \vdots \\ \cos(\boldsymbol{w}_d^T \boldsymbol{x}) \boldsymbol{w}_d^T \\ \sin(\boldsymbol{w}_d^T \boldsymbol{x}) \boldsymbol{w}_d^T \end{bmatrix} \tag{2.215}$$

Note that equation (2.65) and (2.67) offers an alternative definition for $\boldsymbol{\Psi(x)}$.

By using the feature map described in equation (2.67), the following vector field is given

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{J} \sum_{i=1}^{d} \alpha_i \cos(\boldsymbol{w}_d^T \boldsymbol{x} + b_d) \boldsymbol{w}_d \tag{2.216}$$

2.7.5. Symplectic Characteristics

This section, based on [25] and [5], outlines important characteristics that make a Hamiltonian system symplectic. One fundamental property of Hamiltonian systems is that the Hamiltonian $H(\boldsymbol{p}, \boldsymbol{q})$ is a *first integral* of the system described in equation (2.196). Furthermore, the property of *symplecticity* exhibited by the flow is of importance [25].

Volume preservation - Liouville's theorem

It is stated in [25] that every symplectic transformation and symplectic integrator applied to a Hamiltonian system preserves volume in phase space. The phase space is $2n$-dimensional where $\boldsymbol{p} = [p_1, \ldots, p_n]^T$ and $\boldsymbol{q} = [q_1, \ldots, q_n]^T$.

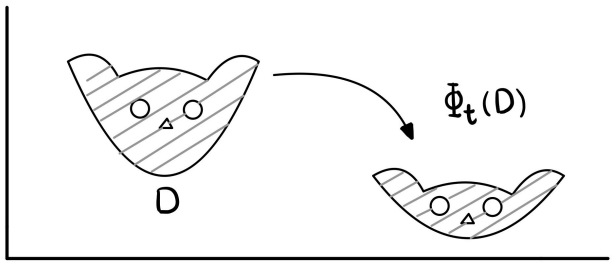


Figure 2.3.: Conservation of volume. *Illustration* [25].

Given a system of ordinary differential equations in the phase space

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) \tag{2.217}$$

where $\boldsymbol{x} = [\boldsymbol{p}^T, \boldsymbol{q}^T]^T$ is the state vector and

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} -\frac{\partial H}{\partial \boldsymbol{q}} \\ \frac{\partial H}{\partial \boldsymbol{p}} \end{bmatrix} \tag{2.218}$$

Then the phase flow is defined as

$$\boldsymbol{\Phi}_t(\boldsymbol{x}(0)) = \boldsymbol{x}(t) \tag{2.219}$$

Here, $\boldsymbol{x}$ denotes the solution of the Hamiltonian dynamics at time t with initial value $\boldsymbol{x}(0)$.

Liouville's theorem states that the phase flow $\Phi_t(D)$ of a region D in the phase space, which represents a material volume of particles following Hamiltonian dynamics, preserves the volume $\mathrm{vol}(D)$. In other words, for any region D in the phase space

$$\mathrm{vol}(\boldsymbol{\Phi}_t(D)) = \mathrm{vol}(D) \tag{2.220}$$

which is illustrated in Figure 2.3.

Equation (2.220) is derived from the fact that $\boldsymbol{f}(\boldsymbol{x})$ has zero divergence, which follows from

$$\mathrm{div}\boldsymbol{f}(\boldsymbol{x}) = \frac{\partial}{\partial \boldsymbol{p}}\left(-\frac{\partial H}{\partial \boldsymbol{q}}\right) + \frac{\partial}{\partial \boldsymbol{q}}\left(\frac{\partial H}{\partial \boldsymbol{p}}\right) = 0 \tag{2.221}$$

For a detailed explanation of how the preservation of volume in the phase space is related to the vector field $\boldsymbol{f}(\boldsymbol{x})$, refer to pages 68-70 in [5].

Symplectic transformations

When studying the flow property, it is common to study two-dimensional parallelograms in $\mathbb{R}^{2n}$. Given the parallelogram P spanned by two vectors in the $(\boldsymbol{p}, \boldsymbol{q})$ space

$$\boldsymbol{\xi} = \begin{pmatrix} \boldsymbol{p}^{\xi} \\ \boldsymbol{q}^{\xi} \end{pmatrix}, \qquad \boldsymbol{\eta} = \begin{pmatrix} \boldsymbol{p}^{\eta} \\ \boldsymbol{q}^{\eta} \end{pmatrix} \tag{2.222}$$

where $\boldsymbol{p}^{\xi}, \boldsymbol{q}^{\xi}, \boldsymbol{p}^{\eta}, \boldsymbol{q}^{\eta} \in \mathbb{R}^n$.

The vectors are written using the basis vectors in $\mathbb{R}^{2n}$ as

$$\boldsymbol{\xi} = q_1^{\xi}\boldsymbol{e}_1 + \ldots q_n^{\xi}\boldsymbol{e}_n + p_1^{\xi}\boldsymbol{e}_{n+1} + \ldots + p_n^{\xi}\boldsymbol{e}_{2n} \tag{2.223}$$

$$\boldsymbol{\eta} = q_1^{\eta}\boldsymbol{e}_1 + \ldots q_n^{\eta}\boldsymbol{e}_n + p_1^{\eta}\boldsymbol{e}_{n+1} + \ldots + p_n^{\eta}\boldsymbol{e}_{2n} \tag{2.224}$$

The vectors can be projected onto the plane by $\boldsymbol{e}_i$ and $\boldsymbol{e}_{n+1}$ for $i = 1, \ldots, n$ as

$$\boldsymbol{\xi}_{i,n+1} = q_i^{\xi}\boldsymbol{e}_i + p_i^{\xi}\boldsymbol{e}_{n+i} \tag{2.225}$$

$$\boldsymbol{\eta}_{i,n+1} = q_i^{\eta}\boldsymbol{e}_i + p_i^{\eta}\boldsymbol{e}_{n+i} \tag{2.226}$$

The sum of the projected areas in the quadratic form can then be defined by

$$\omega^2(\boldsymbol{\xi}, \boldsymbol{\eta}) = \sum_{i=1}^{n} (q_i^{\xi} p_i^{\eta} - p_i^{\xi} q_i^{\eta}) \tag{2.227}$$

This equation can also be written by the 2-form matrix expression as

$$\omega^2(\boldsymbol{\xi}, \boldsymbol{\eta}) = \boldsymbol{\xi}^T \boldsymbol{J} \boldsymbol{\eta} \tag{2.228}$$

where the symplectic matrix $\boldsymbol{J}$ is the same as in (2.205).

To achieve a symplectic transformation, consider a linear transformation applied to both $\boldsymbol{\xi}$ and $\boldsymbol{\eta}$. Suppose that both vectors undergo a linear transformation, yielding $\boldsymbol{A}\boldsymbol{\xi}$ and $\boldsymbol{A}\boldsymbol{\eta}$ respectively, where $\boldsymbol{A} \in \mathbb{R}^{2n \times 2n}$, then

$$\omega^2(\boldsymbol{A}\boldsymbol{\xi}, \boldsymbol{A}\boldsymbol{\eta}) = \boldsymbol{\xi}^T \boldsymbol{A}^T \boldsymbol{J} \boldsymbol{A} \boldsymbol{\eta} \tag{2.229}$$

If the transformation defined by $\boldsymbol{A}$ satisfies

$$\omega^2(\boldsymbol{A}\boldsymbol{\xi}, \boldsymbol{A}\boldsymbol{\eta}) = \omega^2(\boldsymbol{\xi}, \boldsymbol{\eta}) \quad \forall \boldsymbol{\xi}, \boldsymbol{\eta} \tag{2.230}$$

it is said to be symplectic. Note that the following now holds

$$\boldsymbol{\xi}^T \boldsymbol{A}^T \boldsymbol{J} \boldsymbol{A} \boldsymbol{\eta} = \boldsymbol{\xi}^T \boldsymbol{J} \boldsymbol{\eta} \tag{2.231}$$

which means that for the linear transformation $\boldsymbol{A}$, the sum of the areas of the projections of the parallelogram onto all the coordinate planes defined by q_i and p_i remains unchanged, as seen in Figure 2.4. From equation (2.231) it can be observed that

$$\boldsymbol{A}^T \boldsymbol{J} \boldsymbol{A} = \boldsymbol{J} \tag{2.232}$$

and according to [25], the transformation defined by $\boldsymbol{A}$ obtains the symplectic property if and only if this holds.

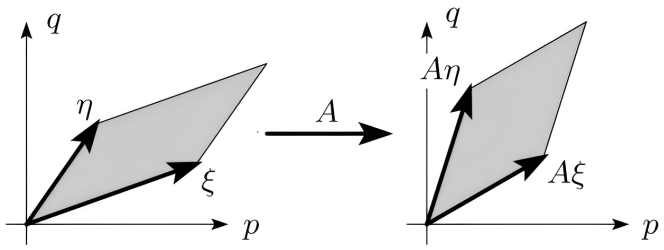


Figure 2.4.: Symplecticity of a linear mapping. *Illustration* [25].

In the case of a nonlinear transformation, given by $\boldsymbol{g}(\xi)$ and $\boldsymbol{g}(\eta)$, the transformation $\boldsymbol{g}$ is considered to be symplectic if its Jacobian matrix $\boldsymbol{g}'(x)$ is symplectic, meaning that

$$\boldsymbol{g}'(\boldsymbol{x})^T \boldsymbol{J} \boldsymbol{g}'(\boldsymbol{x}) = \boldsymbol{J} \tag{2.233}$$

2.7.6. Approximating Hamiltonian Dynamics

This section is based on [43] and [24]. It is stated in [43] that when working with Hamiltonian dynamics in practice, it is necessary to approximate Hamilton's equations. This is done by discretizing time, using a small step size denoted as h. Two of the methods that can be used are described in this section, namely explicit Euler's method and the leapfrog method.

Explicit Euler's method

When using explicit Euler's method to simulate dynamical systems, such as Hamiltonian dynamics, the following steps are performed

$$\boldsymbol{p}_{n+1} = \boldsymbol{p}_n + h\frac{d\boldsymbol{p}_n}{dt} = \boldsymbol{p}_n - h\nabla_{\boldsymbol{q}} H(\boldsymbol{q}_n, \boldsymbol{p}_n) \tag{2.234}$$

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n + h\frac{d\boldsymbol{q}_n}{dt} = \boldsymbol{q}_n + h\nabla_{\boldsymbol{p}} H(\boldsymbol{q}_n, \boldsymbol{p}_n) \tag{2.235}$$

Note that by assuming that $H(\boldsymbol{q}, \boldsymbol{p}) = T(\boldsymbol{p}) + U(\boldsymbol{q})$, equation (2.234) and (2.235) can be reformulated to

$$\boldsymbol{p}_{n+1} = \boldsymbol{p}_n - h\frac{\partial U}{\partial \boldsymbol{q}}(\boldsymbol{q}_n) \tag{2.236}$$

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n + h\frac{\partial T}{\partial \boldsymbol{p}}(\boldsymbol{p}_n) \tag{2.237}$$

The explicit Euler method is first-order accurate [44], meaning that the error discretization is $O(\Delta t)$.

The leapfrog method

The leapfrog method, also known as the Störmer-Verlet method, is a symplectic integrator method, and it can be applied to Hamiltonian systems. According to [43], even better results can be obtained with this method. The following steps are performed [24]

$$\boldsymbol{p}_{n+\frac{1}{2}} = \boldsymbol{p}_n - \frac{h}{2}\nabla_{\boldsymbol{q}} H(\boldsymbol{p}_{n+\frac{1}{2}}, \boldsymbol{q}_n) \tag{2.238}$$

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n + \frac{h}{2}\left(\nabla_{\boldsymbol{q}} H(\boldsymbol{p}_{n+\frac{1}{2}}, \boldsymbol{q}_n) + \nabla_{\boldsymbol{p}} H(\boldsymbol{p}_{n+\frac{1}{2}}, \boldsymbol{q}_{n+1})\right) \tag{2.239}$$

$$\boldsymbol{p}_{n+1} = \boldsymbol{p}_{n+\frac{1}{2}} - \frac{h}{2}\nabla_{\boldsymbol{p}} H(\boldsymbol{p}_{n+\frac{1}{2}}, \boldsymbol{q}_{n+1}) \tag{2.240}$$

Assuming that $H(\boldsymbol{q}, \boldsymbol{p}) = T(\boldsymbol{p}) + U(\boldsymbol{q})$, it can be reformulated to

$$\boldsymbol{p}_{n+\frac{1}{2}} = \boldsymbol{p}_n - \frac{\partial U}{\partial \boldsymbol{q}}(\boldsymbol{q}_n) \tag{2.241}$$

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n + h\frac{\partial T}{\partial \boldsymbol{p}}(\boldsymbol{p}_{n+\frac{1}{2}}) \tag{2.242}$$

$$\boldsymbol{p}_{n+1} = \boldsymbol{p}_{n+\frac{1}{2}} - \frac{h}{2}\frac{\partial U}{\partial \boldsymbol{q}}(\boldsymbol{q}_{n+1}) \tag{2.243}$$

The leapfrog method achieves second-order accuracy by initially taking a half step for the momentum variables, followed by computing a full step for the position variables using the updated momentum values, and finally calculating another half step for the momentum variables using the new position values. The error discretization is $O\left(\Delta t^2\right)$.

2.7.7. Hamiltonian Dynamics of a Pendulum

The equations of motion for a pendulum with a point mass m can be defined by the differential equation

$$\ddot{\boldsymbol{\theta}} + \frac{g}{L}\sin\boldsymbol{\theta} = 0 \tag{2.244}$$

where L is the length of the pendulum, g is the acceleration due to gravity, and $\boldsymbol{\theta}$ is the angular displacement [8]. This can be rewritten to

$$\begin{aligned} \dot{\boldsymbol{\theta}} &= \boldsymbol{\omega} \\ \dot{\boldsymbol{\omega}} &= -\frac{g}{L}\sin\boldsymbol{\theta} \end{aligned} \tag{2.245}$$

For a pendulum, the kinetic energy is given by $T = \frac{1}{2}mL^2\dot{\boldsymbol{\theta}}^2$ and the potential energy is $U = mgL(1-\cos\boldsymbol{\theta})$. Based on this the Lagrangian can be expressed as

$$\mathcal{L} = \frac{1}{2}mL^2\dot{\boldsymbol{\theta}}^2 - mgL(1-\cos\boldsymbol{\theta}) \tag{2.246}$$

The generalized coordinate for the Hamiltonian dynamics is set to $\boldsymbol{q} = \boldsymbol{\theta}$, and the momentum is

$$\boldsymbol{p} = \frac{\partial L}{\partial \dot{\boldsymbol{q}}} = mL^2\dot{\boldsymbol{\theta}} \tag{2.247}$$

which gives $\dot{\boldsymbol{\theta}} = \frac{\boldsymbol{p}}{mL^2}$.

Given equation (2.197), the Hamiltonian for a pendulum system can be represented as follows

$$H(\boldsymbol{q}, \boldsymbol{p}) = \frac{1}{2}\frac{\boldsymbol{p}^2}{mL^2} + mgL(1 - \cos \boldsymbol{q}) \tag{2.248}$$

and Hamiltonian dynamics is

$$\begin{aligned} \dot{\boldsymbol{q}} &= \frac{\partial H}{\partial \boldsymbol{p}} = \frac{\boldsymbol{p}}{mL^2} \\ \dot{\boldsymbol{p}} &= -\frac{\partial H}{\partial \boldsymbol{q}}\dot{\boldsymbol{q}} = -mgL \sin \boldsymbol{q} \end{aligned} \tag{2.249}$$

which leads to $\ddot{\boldsymbol{q}} = -\frac{g}{L} \sin \boldsymbol{q}$.

In addition, it can be seen that the gradient potential energy is given by

$$\frac{\partial U}{\partial \boldsymbol{q}} = mgL \sin(\boldsymbol{q}) \tag{2.250}$$

Chapter 3.

Method

This chapter presents the methodology used to model dynamical systems using regression. The optimization tools used, the method to generate data, the algorithms used to solve various regression problems, and the corresponding parameter settings will be explained. The theory presented in the Preliminaries chapter was in order to learn the models. Python is used to solve the regression problems, and the code can be found in [29]. The code is developed from scratch.

3.1. Python Tools for Optimization and Numerical Solution Solving

This section provides an overview of the Python tools used in this thesis. The optimization tools PICOS and MOSEK were used to solve the regression problems where contraction constraints are incorporated. PICOS was used to define the regression problems, while MOSEK was used to solve the problem.

3.1.1. MOSEK

MOSEK is a powerful and reliable optimization software package designed to solve complex optimization problems efficiently [3]. This tool is iterative and provides a range of solvers to address linear, quadratic, conic, and nonlinear optimization problems, as well as the ability to handle complex constraints. It is particularly well-suited for large-scale optimization problems and has a reputation for being one of the most robust and efficient optimization solvers available.

3.1.2. PICOS

Python Interface for Conic Optimization Solvers (PICOS) is a Python-based modeling language that provides an interface for different optimization solvers, including MOSEK [47]. It is built on top of NumPy and enables users to define optimization problems, decision variables, and integrate external data to define objective functions and constraints of the problem. An advantage of using PICOS is the ability to define the optimization problem as a high-level model, and thereby reducing the focus on technical details.

3.1.3. Numdifftools

When dealing with problems involving both vanishing point and contraction constraints, the mathematical expression necessary to compute the gradient of the feature map ψ became complex, leading to a significant increase in computation time. Here, the referenced ψ is used in equation (2.192). This made it difficult to obtain an analytical solution compared to the scenario without vanishing points.

Because of these limitations, it was necessary to use a Python package to compute the gradient. To address this, the Python library *numdifftools* was used, which solves automatic numerical differentiation problems [14]. This library uses an adaptive finite difference method, along with a Richardson extrapolation methodology, to provide a highly accurate approximated result.

3.1.4. Solving Ordinary Differential Equations

To evaluate the results obtained from the regression problems in this thesis, it was necessary to solve ordinary differential equations (ODEs). To do this, the Python-based SciPy library was used. SciPy has built-in solvers that solve initial value problems for ODE systems [58]. The solver used in the simulations was *solve_ivp*.

This solver requires an initial value, a function, and a time span as input. It then performs numerical integration of the system of ODEs [50]. The system of ODEs is represented by the equations

$$\frac{dy}{dt} = f(t, y) \tag{3.1}$$

$$y(t_0) = y_0 \tag{3.2}$$

3.2. Data Generation

In this thesis, most of the regression problems use a real-world dataset. The dataset is obtained from [18], and it is a LASA handwriting experiment. Additional details can be found in Section 3.3.1. However, when learning with Hamiltonian systems, it was necessary to self-generate trajectories, and the process is explained in Section 3.4.1.

3.2.1. Generating RFF Parameters for Learning

As described in Section 2.2.3, in order to use RFF in learning, a feature map ψ of a shift-invariant kernel needs to be defined. This thesis uses three different feature maps, each based on a specific kernel. Equations (2.55), (2.65), and (2.67) define the feature maps associated with the Gaussian separable kernel, the curl-free kernel, and the symplectic kernel, respectively. When feature maps are mentioned further in this thesis, these are the ones referred to.

To use the feature maps in the learning process, it is necessary to generate their parameters, $\boldsymbol{w}$ and b. As mentioned in Section 2.2.5, $\boldsymbol{w}_i \in \mathbb{R}^N$ is selected with probability $p(\boldsymbol{w})$ from a normal distribution, while $b_i \in \mathbb{R}$ is randomly selected from the uniform distribution $[0, 2\pi]$. The shape of $\boldsymbol{w}$ depends on the dimension m, resulting in the shape $(m,\, d)$, and b will always have the shape $(1,\, d)$, where d is the number of samples. A trial-and-error approach was used to generate the parameters until satisfactory values were obtained.

3.3. Learning with LASA Benchmark

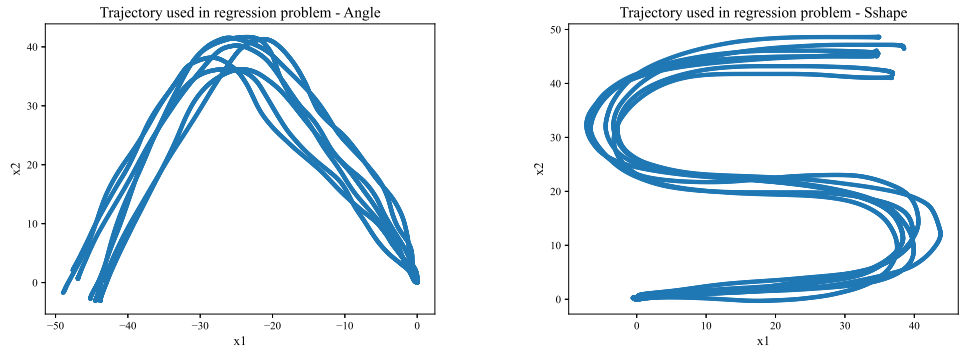
This section presents the methodology for solving regression problems with the LASA library dataset obtained from [18]. The regression problems presented in this section are based on the research findings reported in [51]. The code developed for this problem can be found under the directory *lasa* in the GitHub repository [29].

3.3.1. The Dataset

To replicate two-dimensional human handwriting movements and learn their corresponding vector fields, our Python program was developed implementing the LASA library dataset [30] sourced from [18]. This dataset consists of .mat files, which made it necessary to use SciPy's loadmap function to work with MATLAB files in a Python program.

The dataset includes 30 handwriting movements and is commonly used as a benchmark for evaluating the effectiveness of random Fourier features and regression techniques. Out of the 30 motions, 26 feature a single pattern, three have two patterns, and one has three patterns. The demonstrations in the dataset were recorded from pen input using a Tablet-PC.

The dataset consists of two-dimensional motions, denoted as $\boldsymbol{\zeta} = [x; y] \in \mathbb{R}^2$. The dataset includes seven demonstrations for each pattern, with varying starting points ending at the same final point. However, the starting points for each demonstration are near one another. The demonstrations contain the same number of data points and may intersect. Each demonstration consists of 1000 position x and velocity $\dot{x}$ measurements. It was decided to learn the dynamics of two patterns in the dataset, namely the Angle- and S-shape, and Figure 3.1 shows the seven demonstrations for these. The experimental parameters used in this study can be seen in Table 3.3.



(a) (b)

Figure 3.1.: The seven human handwriting demonstrations (a) Angle-shape (b) S-shape

It is important to emphasize that the learnt model in this study is invariant to time, implying that its behavior and response to input remain consistent over time. The model relies solely on spatial information to derive motion, eliminating the need for time-warping trajectory alignment before training. However, this time-invariance feature also implies that the model's generalization capability relies on the spatial environment, as discussed in [31]. As a result, if the motion initiates from an inconvenient location, the model's accuracy may be limited to only half of the pattern. A case in point is if the motion starts close to the middle of the demonstration, such as at point $[-25, 35]$ for the Angle-shape model in Figure 3.1(a), the reproduction may be inadequate. Depending on the intended

application, this generalization feature may be advantageous or disadvantageous. To ensure precise reproduction of the desired pattern, it is important to position the origin of the frame of reference at the starting point, as done in this thesis.

3.3.2. Dataset Splitting

To ensure reliable validation of the results obtained during the modeling, the dataset was divided into training and test sets. This division will allow for an unbiased evaluation of a model's performance on unseen data during real-world predictions [56]. As presented in Section 3.3.1, the LASA library dataset includes seven demonstrations for each pattern. Among these demonstrations, four were allocated for training the model, while the remaining three were reserved for testing purposes, as indicated in Table 3.3. This resulted in an approximately 60/40 split.

For most of the regression problems addressed in this thesis, a training dataset of 1000 points was created by calculating the mean of position and velocity measurements from the four training demonstrations. This resulted in a single trajectory, and this approach aligns with the one used in [51]. However, when RFF was used to estimate a vector field without contraction and vanishing points, refer to Algorithm 1, two models were trained.

The first model was trained by using the mean trajectory, while the second model was trained on all four training demonstrations individually without computing their mean. This resulted in the model being trained on four distinct trajectories, leading to a training set consisting of 4000 points. The objective was to compare the two resulting simulations to determine the potential benefits of using the single trajectory as a training approach for regression problems.

Training the model

During the training process, the value of $\boldsymbol{\alpha}$ was determined by using the training set. The different methods for calculating $\boldsymbol{\alpha}$ are explained in Section 3.3.4. Once $\boldsymbol{\alpha}$ was obtained, a trajectory was simulated using the numerical solver *solve_ivp*. As discussed in Section 3.1.4, *solve_ivp* requires a function as one of its inputs. This function is used to learn the dynamics of the system and the specific function defined depended on whether vanishing points were present or not.

For the cases without vanishing points, the function used was

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{\Psi}(\boldsymbol{x})\boldsymbol{\alpha} \tag{3.3}$$

while for the cases with vanishing points, the function used was

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{\Psi}^{Z}(\boldsymbol{x})\boldsymbol{\alpha} = \boldsymbol{L}^{T}\boldsymbol{\Psi}(\boldsymbol{x})\boldsymbol{\alpha} \tag{3.4}$$

Once the value of $\boldsymbol{\alpha}$ was estimated using the training set, it was evaluated using different test sets to assess the accuracy of the approximation. Finally, the *solve_ivp* function was used again to approximate the trajectory $\boldsymbol{f}$ for each test set, and the resulting approximations were plotted against the actual trajectory.

Table 3.1 outlines the initial conditions and timespan used for training and testing the model with the Angle-shape. Similarly, when using the S-shape, Table 3.2 presents the corresponding values for training and testing.

Dataset	Initial condition	Timespan
Training datasets 1-4	[-44.2241 -2.1552]	[0. 2.9655]
Test dataset 5	[-47.5862 2.0679]	[0. 2.8368]
Test dataset 6	[-46.8966 0.6897]	[0. 3.0366]
Test dataset 7	[-48.9655 -1.7241]	[0. 3.1252]

Table 3.1.: Train and Test values for Angle-shape

Dataset	Initial condition	Timespan
Training datasets 1-4	[35.7252 44.0041]	[0. 4.6176]
Test dataset 5	[34.5554 45.3539]	[0. 4.4489]
Test dataset 6	[34.5553 46.0738]	[0. 4.7592]
Test dataset 7	[38.5148 46.4337]	[0. 4.6556]

Table 3.2.: Train and Test values for S-shape

3.3.3. Parameters Used for Learning

When it comes to learning dynamical systems, selecting the correct parameters for testing can significantly influence the accuracy of replicating trajectories. Therefore, choosing parameters was crucial for achieving the best possible results. To determine which hyperparameters should be used in the regression problems, a set of candidate values was tested based on the values used in [51].

Table 3.3 and Table 3.4 display the parameters and their corresponding values used for learning. Table 3.3 is the same for both shapes, whereas Table 3.4 presents the values used for the regression problems with RFF, which vary based on the chosen shape. Notice the difference in the number of samples between the Angle- and S-shape. This is because a larger number of samples were required

to achieve satisfactory results when training the S-shape. The reason the Angle-shape requires less number of samples to provide a satisfactory shape is because of its simplicity of representation. The Angle-shape has a straightforward nature with one change in direction and can therefore be accurately captured with fewer data points. Conversely, the S-shape has two direction changes and therefore requires denser sampling to accurately represent smoother and more complex curves.

Parameter	Value
Number of points with mean, N	1000
Number of points without mean, N	4000
Number of contraction points, M	250
Number of training datasets	4
Number of test datasets	3
Vanish point	(0, 0)

Table 3.3.: LASA Handwriting - Parameters

Parameter	Angle	Sshape
Number of samples, d	100	200
σ	10	10
λ	0.1	0.1
μ	0	0

Table 3.4.: LASA Handwriting - RFF Parameters

3.3.4. Algorithms used in Implementation

In this section, a variety of algorithms are introduced. These algorithms were used to solve regression problems to learn models involving the estimation of vector fields using RFF. All of the algorithms were developed based on the theory presented in the preliminaries, and the feature maps used are described in equation (2.55) for the Gaussian separable kernel and equation (2.65) in the case of a curl-free kernel. The Gaussian separable feature map was selected based on the results obtained in the specialization project [28], which showed that it provided a slightly better result than the feature map given in equation (2.48). While the majority of algorithms described in this section focus on outlining the approach for estimating $\boldsymbol{\alpha}$, one algorithm describes the process of finding $\boldsymbol{L}$ used in regression problems with vanishing points. After estimating $\boldsymbol{\alpha}$ in the different regression problems, it was used to estimate the trajectory using *solve_ivp*, as outlined in Section 3.3.2. The last algorithm presented, describes the process for estimating the function $\boldsymbol{f}$, resulting in the estimated trajectory.

As previously stated, the use of PICOS was necessary to define the regression problems and MOSEK to solve them in the cases where contraction constraints were incorporated. An example with explanation of how this was implemented in Python can be seen in Appendix A.3, where the code is developed based on Algorithm 2.

Algorithm for vector-valued regression

Algorithm 1 was used to solve a vector-valued regression problem, where the kernel is approximated using RFF. The kernel used is the Gaussian separable kernel. The code based on this algorithm can be found in the file *lasa_gaussian_separable* and *lasa_gaussian_separable_without_mean* in the GitHub repository [29].

Algorithm 1 Estimating $\boldsymbol{\alpha}$ with Gaussian separable feature map

Input: N data points $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ where $\boldsymbol{x}_i \in \mathbb{R}^n, \boldsymbol{y}_i \in \mathbb{R}^m$
Scalar feature map $\sqrt{2}\cos(\boldsymbol{w}_i^T \boldsymbol{x}_i + b_i)$
Dimension m
m samples $\boldsymbol{w}_1, ..., \boldsymbol{w}_m \in \mathbb{R}^m$ with d samples $\boldsymbol{w}_i = w_1, ..., w_d \in \mathbb{R}^N$
d samples b_1, ..., $b_d \in \mathbb{R}$
Tuning parameter λ
Output: Estimated $\boldsymbol{\alpha}$
Create the $Nm \times dm$ matrix $\boldsymbol{\Phi}$
for $i \in 1 : N$ **do**
  for $j \in 1 : d$ **do**
    $\boldsymbol{\psi}[\mathrm{j}] = \sqrt{2}\cos(\boldsymbol{w}_j^T \boldsymbol{x}_i + b_j)$
  end for
  $\boldsymbol{\Psi}(\boldsymbol{x}_i) = \boldsymbol{\psi}(\boldsymbol{x}_i) \otimes \boldsymbol{I}_m$
  $\boldsymbol{\Phi}[i] = \boldsymbol{\Psi}(\boldsymbol{x}_i)^T$
end for
Solve equation $(\boldsymbol{\Phi}^T \boldsymbol{\Phi} + \lambda d \mathbf{I}_{dm})\boldsymbol{\alpha} = \boldsymbol{\Phi}^T \boldsymbol{Y}$ for $\boldsymbol{\alpha}$

Algorithm for contraction constraints

Algorithm 2 was used to finding $\boldsymbol{\alpha}$ in the regression problem where contraction constraints were considered. The algorithm is developed based on the theory presented in Section 2.5.5. The Python code for this can be found in *lasa_gaussian_separable* in the GitHub repository [29].

Algorithm 2 Estimating $\boldsymbol{\alpha}$ in regression problem with contraction

Input: N data points $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ where $\boldsymbol{x}_i, \boldsymbol{y}_i \in \mathbb{R}^m$
Scalar feature map $\sqrt{2}\cos(\boldsymbol{w}_i^T \boldsymbol{x}_i + b_i)$
Gradient of scalar feature map $-\sqrt{2}\sin(\boldsymbol{w}_i^T \boldsymbol{x}_i + b_i)\boldsymbol{w}_i$
Dimension m
m samples $\boldsymbol{w}_1, ..., \boldsymbol{w}_m \in \mathbb{R}^m$ with d samples $\boldsymbol{w}_i = w_1, ..., w_d \in \mathbb{R}^N$
d samples b_1, ..., $b_d \in \mathbb{R}$
Tuning parameter λ
Parameter μ
M constraint points

Output: Estimated $\boldsymbol{\alpha}$

```
Create the Nm × dm matrix Φ
for i ∈ 1 : N do
    for j ∈ 1 : d do
        Ψ[j] = √2 cos(w_j^T x_i + b_j)
    end for
    Φ[i] = Ψ(x_i)^T
end for
Apply contraction constraints to the regression problem
for i ∈ 1 : M do
    Calculate J(x_i)
    for j ∈ 1 : d do
        ∇ψ_j = −√2 sin(w_j^T x_i + b_j) w_j
        J(x_i) += 1/2 ( α_j ∇ψ_j^T(x_i) + (α_j ∇ψ_j^T(x_i))^T )
    end for
    Add constraints s.t. J(x_i) ⪯ −μ I_m to the problem
end for
Set min_{α ∈ R^{dm}} [(Φα − Y)^T (Φα − Y) + λ α^T α] as the objective of the problem
Solve for α
```

Algorithm for finding L

As seen in Section 2.6.2, the inclusion of vanishing points in regression problems using RFF necessitates the computation of a lower-triangular matrix $\boldsymbol{L}$. The approach to find $\boldsymbol{L}$ is shown in Algorithm 3. The algorithm shows both how it is calculated when there is a curl-free feature map and when there is a Gaussian separable feature map. The dimension for $\boldsymbol{\Phi}$ for the curl-free kernel will be $d \times Zm$, and for the other kernel, it will be $dm \times Zm$. The code for finding $\boldsymbol{L}$ in the curl-free case can be found in the file *lasa_curl_free* and for the Gaussian separable case, it can be found in *lasa_vanish* in the GitHub repository [29].

Algorithm 3 Finding the lower-triangular matrix $\boldsymbol{L}$

Input: Vanishing points $\{\boldsymbol{x}_i^*\}_{i=1}^{Z} \in \mathbb{R}^m$
Number of vanish points Z
Dimension m
m samples $\boldsymbol{w}_1, ..., \boldsymbol{w}_m \in \mathbb{R}^m$ with d samples $\boldsymbol{w}_i = w_1, ..., w_d \in \mathbb{R}^N$
d samples b_1, ..., $b_d \in \mathbb{R}$
Output: Lower-triangular matrix $\boldsymbol{L}$
Calculate $\boldsymbol{\Phi}(Z)$
for $i \in 1 : Z$ **do**
  for $j \in 1 : d$ **do**
    if curl-free feature map **then**
      $\boldsymbol{\psi}[i] = \sqrt{2}\sin(\boldsymbol{w}_j^T \boldsymbol{x}_i^* + b_j)\boldsymbol{w}_j^T$
    else
      $\boldsymbol{\psi}[j] = \sqrt{2}\cos(\boldsymbol{w}_j^T \boldsymbol{x}_i^* + b_j)$
    end if
  end for
  if curl-free feature map **then**
    $\boldsymbol{\Psi}[i] = \boldsymbol{\psi}(\boldsymbol{x}_i^*)$
  else
    $\boldsymbol{\Psi}[i] = \boldsymbol{\psi}(\boldsymbol{x}_i^*) \otimes \boldsymbol{I}_m$
  end if
  $\boldsymbol{\Phi}[i] = \boldsymbol{\Psi}(\boldsymbol{x}_i^*)^T$
end for
Calculate equation $\boldsymbol{P}^{\perp}_{\boldsymbol{\Phi}(Z)} = \boldsymbol{I} - (\boldsymbol{\Phi}(Z)(\boldsymbol{\Phi}(Z)^T\boldsymbol{\Phi}(Z))^{-1}\boldsymbol{\Phi}(Z)^T)$
Ensure P is a Hermitian positive-definite matrix given numerical noise
$\boldsymbol{P}^{\perp}_{\boldsymbol{\Phi}(Z)} = \frac{\boldsymbol{P}^{\perp}_{\boldsymbol{\Phi}(Z)} + \boldsymbol{P}^{\perp}_{\boldsymbol{\Phi}(Z)^T}}{2} + \boldsymbol{I} * 1e{-}12$
Take the Cholesky factorization of $\boldsymbol{P}_{\boldsymbol{\Phi}}$ to find $\boldsymbol{L}$

Algorithms for vanishing point

For the models that required a vector field to vanish at specific points, both contraction and non-contraction cases were considered. Algorithm 4 can be used to address the problem when there is no contraction. Algorithm 5, on the contrary, can be used to solve regression problems with contraction constraints. As mentioned in Section 3.1, finite difference was used to calculate the gradient of the feature map ψ, because of the computational cost and complexity of solving the mathematical expression in equation (2.192). This is used in Algorithm 5. The code can be found in the file *lasa_vanish* in [28].

Algorithm 4 Estimating $\boldsymbol{\alpha}$ in problem with vanishing point

Input: N data points $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ where $\boldsymbol{x}_i, \boldsymbol{y}_i \in \mathbb{R}^m$
Scalar feature map $\sqrt{2}\cos(\boldsymbol{w}_i^T \boldsymbol{x}_i + b_i)$
Dimension m
m samples $\boldsymbol{w}_1, ..., \boldsymbol{w}_m \in \mathbb{R}^m$ with d samples $\boldsymbol{w}_i = w_1, ..., w_d \in \mathbb{R}^N$
d samples b_1, ..., $b_d \in \mathbb{R}$
Tuning parameter λ
Lower triangular matrix $\boldsymbol{L}$

Output: Estimated $\boldsymbol{\alpha}$

```
Create the Nm × dm matrix Φ^Z
for i ∈ 1 : N do
    for j ∈ 1 : d do
        ψ[j] = √2 cos(w_j^T x_i + b_j)
    end for
    Ψ(x_i) = ψ(x_i) ⊗ I_m
    Ψ^Z(x_i) = L^T Ψ(x_i)
    Φ^Z[i] = Ψ^Z(x_i)^T
end for
Solve equation (Φ^T Φ + λdI_dm)α = Φ^T Y for α
```

Algorithm 5 Estimating $\boldsymbol{\alpha}$ in problem with vanishing point and contraction

Input: N data points $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ where $\boldsymbol{x}_i, \boldsymbol{y}_i \in \mathbb{R}^m$
Scalar feature map $\sqrt{2}\cos(\boldsymbol{w}_i^T \boldsymbol{x}_i + b_i)$
Dimension m
m samples $\boldsymbol{w}_1, ..., \boldsymbol{w}_m \in \mathbb{R}^m$ with d samples $\boldsymbol{w}_i = w_1, ..., w_d \in \mathbb{R}^N$
d samples b_1, ..., $b_d \in \mathbb{R}$
Tuning parameter λ
Parameter μ
M constraint points
Lower triangular matrix $\boldsymbol{L}$

Output: Estimated $\boldsymbol{\alpha}$

Create the $Nm \times dm$ matrix $\boldsymbol{\Phi}^Z$
for $i \in 1 : N$ **do**
  for $j \in 1 : d$ **do**
    $\boldsymbol{\Psi}[j] = \sqrt{2}\cos(\boldsymbol{w}_j^T \boldsymbol{x}_i + b_j)$
  end for
  $\boldsymbol{\Psi}^Z(\boldsymbol{x}_i) = \boldsymbol{L}^T \boldsymbol{\Psi}(\boldsymbol{x}_i)$
  $\boldsymbol{\Phi}^Z[i] = \boldsymbol{\Psi}^Z(\boldsymbol{x}_i)^T$
end for
Apply contraction constraints to the regression problem
for $i \in 1 : M$ **do**
  Calculate $\boldsymbol{\nabla}\boldsymbol{\Psi}^Z(\boldsymbol{x}_i)$ using finite difference
  for $j \in 1 : d$ **do**
    $\nabla\psi_j = \boldsymbol{\nabla}\boldsymbol{\Psi}^Z(\boldsymbol{x}_i)[j]$
    $\boldsymbol{J}(\boldsymbol{x}_i) \mathrel{+}= \frac{1}{2}\left(\boldsymbol{\alpha}_j \nabla\psi_j^T(\boldsymbol{x}_i) + (\boldsymbol{\alpha}_j \nabla\psi_j^T(\boldsymbol{x}_i))^T\right)$
  end for
  Add constraints s.t. $\boldsymbol{J}(\boldsymbol{x}_i) \preceq -\mu \boldsymbol{I}_m$ to the problem
end for
Set $\min_{\boldsymbol{\alpha} \in \mathbb{R}^{dm}} [(\boldsymbol{\Phi}^Z\boldsymbol{\alpha} - \boldsymbol{Y})^T(\boldsymbol{\Phi}^Z\boldsymbol{\alpha} - \boldsymbol{Y}) + \lambda\boldsymbol{\alpha}^T\boldsymbol{\alpha}]$ as the objective of the problem
Solve for $\boldsymbol{\alpha}$

Algorithm for curl-free feature map with vanishing point and contraction

A regression problem with both vanishing points and contraction constraints was considered for the curl-free case. This can be seen in the Algorithm 6. It should be noted that Algorithm 5 shares similarities with the current algorithm, except for the difference in dimensions. Specifically, $\boldsymbol{\alpha} \in \mathbb{R}^d$ for the current algorithm compared to $\boldsymbol{\alpha} \in \mathbb{R}^{dm}$. The code for this can be seen in the *lasa_curl_free* file, in the GitHub repository [29].

Algorithm 6 Estimating $\boldsymbol{\alpha}$ in curl-free problem with vanishing point and contraction

Input: N data points $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ where $\boldsymbol{x}_i, \boldsymbol{y}_i \in \mathbb{R}^m$
Feature map $\sqrt{2}\sin(\boldsymbol{w}_i^T \boldsymbol{x}_i + b_i)\boldsymbol{w}_i^T$
Dimension m
m samples $\boldsymbol{w}_1, ..., \boldsymbol{w}_m \in \mathbb{R}^m$ with d samples $\boldsymbol{w}_i = w_1, ..., w_d \in \mathbb{R}^N$
d samples b_1, ..., $b_d \in \mathbb{R}$
Tuning parameter λ
Parameter μ
M constraint points
Lower triangular matrix $\boldsymbol{L}$

Output: Estimated $\boldsymbol{\alpha}$

Create the $Nm \times d$ matrix $\boldsymbol{\Phi}^Z$
for $i \in 1 : N$ **do**
  for $j \in 1 : d$ **do**
    $\boldsymbol{\Psi}[j] = \sqrt{2}\sin(\boldsymbol{w}_j^T \boldsymbol{x}_i + b_j)\boldsymbol{w}_j^T$
  end for
  $\boldsymbol{\Psi}^Z(\boldsymbol{x}_i) = \boldsymbol{L}^T \boldsymbol{\Psi}(\boldsymbol{x}_i)$
  $\boldsymbol{\Phi}^Z[i] = \boldsymbol{\Psi}^Z(\boldsymbol{x}_i)^T$
end for
Apply contraction constraints to the regression problem
for $i \in 1 : M$ **do**
  Calculate $\boldsymbol{\nabla}\boldsymbol{\Psi}^Z(\boldsymbol{x}_i)$ using finite difference
  for $j \in 1 : d$ **do**
    $\nabla\psi_j = \boldsymbol{\nabla}\boldsymbol{\Psi}^Z(\boldsymbol{x}_i)[j]$
    $\boldsymbol{J}(\boldsymbol{x}_i)$ += $\frac{1}{2}\left(\boldsymbol{\alpha}_j \nabla\psi_j^T(\boldsymbol{x}_i) + (\boldsymbol{\alpha}_j \nabla\psi_j^T(\boldsymbol{x}_i))^T\right)$
  end for
  Add constraints s.t. $\boldsymbol{J}(\boldsymbol{x}_i) \preceq -\mu \boldsymbol{I}_m$
end for
Set $\min_{\boldsymbol{\alpha} \in \mathbb{R}^d} [(\boldsymbol{\Phi}^Z \boldsymbol{\alpha} - \boldsymbol{Y})^T (\boldsymbol{\Phi}^Z \boldsymbol{\alpha} - \boldsymbol{Y}) + \lambda \boldsymbol{\alpha}^T \boldsymbol{\alpha}]$ as the objective of the problem
Solve for $\boldsymbol{\alpha}$

Algorithm for finding the function approximation

After finding the estimated $\boldsymbol{\alpha}$ for the different regression problems, the approximation of function $\boldsymbol{f}(\boldsymbol{x})$ was calculated for the learnt model. Algorithm 7 shows how this can be done.

Algorithm 7 Approximation of function $\boldsymbol{f}$

Input: N data points $\{\boldsymbol{x}_i\}_{i=1}^{N} \in \mathbb{R}^m$
Estimated $\boldsymbol{\alpha}$
Feature map $\boldsymbol{\Psi}$
Output: Estimated function $\boldsymbol{f}$
Find the estimated function $\boldsymbol{f}$
for $i \in 1 : N$ **do**
$\boldsymbol{f}(\boldsymbol{x}_i) = \sum_{j=1}^{N} \boldsymbol{\Psi}(\boldsymbol{x}_i)\boldsymbol{\alpha}_j$
end for

3.4. Learning Hamiltonian Systems

This section explains the approach for solving regression problems associated with Hamiltonian dynamics, specifically for a pendulum. In the GitHub repository [29] under the folder *hamilton*, the code developed for these regression problems can be accessed.

3.4.1. Generating Trajectories

When working with Hamilton systems, generating data points for different trajectories was desired. Two numerical integration methods were used for this purpose: the leapfrog integrator and the explicit Euler method, both presented in Section 2.7.6. Using these methods allowed for a simulation of the system's evolution over time, which would help understand the behavior of the Hamiltonian dynamics. This included analyzing the properties of stability and conservation of energy.

Both methods were used to generate a trajectory over a defined time interval with a specific step size. Based on the old position $\boldsymbol{q}$, the old momentum $\boldsymbol{p}$ and the step size, the methods calculated the new $\boldsymbol{q}$ and $\boldsymbol{p}$. The methods started with an initial condition and were iteratively applied to generate subsequent points along the trajectory.

In this thesis, the decision was made to use identical parameter values for both methods when generating the trajectories. This choice would ensure consistency and comparability in the resulting trajectories, establishing a fair analysis of the methods' behavior. These values can be found in Table 3.5.

Parameter	Value
Initial condition, (q_0, p_0)	(2, 0)
Time interval	[0, 10]
Step size	0.01

Table 3.5.: Numerical integration - Parameters

The methods were used to generate trajectories for a pendulum. The parameters specific to this system used in the trajectory generation are listed in Table 3.6.

Parameter	Value
Mass, m	1
Length of pendulum, L	1
Acceleration of gravity, g	9.81

Table 3.6.: Pendulum - Parameters

Following the trajectory generation, the values found for $\boldsymbol{q}$ and $\boldsymbol{p}$ were used to calculate the Hamiltonian dynamics of the pendulum system, specifically $\dot{\boldsymbol{p}}$ and $\dot{\boldsymbol{q}}$. The Hamiltonian dynamics of a pendulum can be seen in Section 2.7.7.

3.4.2. Learning Hamiltonian Dynamics with RFF

After generating the two trajectories, the leapfrog-generated trajectory was chosen for learning purposes. RFF was used to approximate a symplectic kernel and a Gaussian separable kernel, resulting in a sympletic feature map described in equation (2.67) and a Gaussian separable feature map described in equation (2.55). These feature maps were then used in regression problems to learn and understand the trajectory dynamics. The main objective of this approach was to compare the performance of the two feature maps in regression problems involving the learning of Hamiltonian dynamics. The comparison involved evaluating each feature map's accuracy, stability, and capability to estimate vector fields. Additionally, studying their robustness to generalizing unseen data is essential to understand how reliable each feature map was for modeling these dynamic systems.

To ensure a fair comparison, identical parameters were used for both regression problems. The specific values of these parameters can be found in Table 3.7.

The trajectory generated using the leapfrog method was trained twice for each kernel: once with noise intentionally added to the training set and once without noise. When noise is incorporated into a dataset, it hinders the model's ability to memorize the training samples due to their continuous variation. Consequently, this leads to developing a more robust model that exhibits lower generalization

Parameter	Value
Noise, σ_{est}	0.01
Number of points, N	1000
Number of samples, d	30
Dimension, m	2
σ	5
λ	0.001

Table 3.7.: Hamiltonian System - RFF Parameters

error [15]. To add noise, standard deviation with zero mean was used, and the value can be seen in Table 3.7.

To solve for $\boldsymbol{\alpha}$ and estimate the trajectory generated using the Gaussian separable feature map, Algorithm 1 was used. Meanwhile, Algorithm 8 was deployed to estimate the trajectory where the symplectic feature map was used.

Algorithm 8 Estimating $\boldsymbol{\alpha}$ with a symplectic feature map

Input: N data points $(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathbb{R}^N$ where $\boldsymbol{x}_i \in \mathbb{R}^n, \boldsymbol{y}_i \in \mathbb{R}^m$
Scalar feature map $\sqrt{2}\cos(\boldsymbol{w}_i^T \boldsymbol{x}_i + b_i)\boldsymbol{w}_i^T$
Dimension m
m samples $\boldsymbol{w}_1, ..., \boldsymbol{w}_m \in \mathbb{R}^m$ with d samples $\boldsymbol{w}_i = w_1, ..., w_d \in \mathbb{R}^N$
d samples b_1, ..., $b_d \in \mathbb{R}$
Tuning parameter λ
Symplectic matrix $\boldsymbol{J}$
Output: Estimated $\boldsymbol{\alpha}$
Create the $Nm \times d$ matrix $\boldsymbol{\Phi}$
for $i \in 1 : N$ **do**
  for $j \in 1 : d$ **do**
    $\boldsymbol{\psi}$[j] $= \sqrt{2}\cos(\boldsymbol{w}_j^T \boldsymbol{x}_i + b_j)\boldsymbol{w}_j^T$
  end for
  $\boldsymbol{\Psi}(\boldsymbol{x}_i) = \boldsymbol{J}\boldsymbol{\psi}(\boldsymbol{x}_i)$
  $\boldsymbol{\Phi}[i] = \boldsymbol{\Psi}(\boldsymbol{x}_i)^T$
end for
Solve equation $(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda d\mathbf{I}_d)\boldsymbol{\alpha} = \boldsymbol{\Phi}^T\boldsymbol{Y}$ for $\boldsymbol{\alpha}$

3.5. Comparison Metrics

Comparison metrics were used to evaluate and compare the performance of the different models and algorithms. In addition, these metrics allow for determining which model and regression technique performs best.

3.5.1. Reproduction Accuracy

In comparing the accuracy of learned dynamical systems for both LASA-shape trajectories and Hamiltonian dynamics, reproduction accuracy is a key criterion. This criterion assesses the system's ability to replicate the positions observed in the training and test demonstrations. For the LASA benchmark dataset, this evaluation assumes that the system is initialized with the same initial conditions as the human movement and is integrated for the same duration as the human movement, denoted as T [51]. For Hamiltonian dynamics, T represents the maximum time duration in which the simulation will run.

The measurement of the reproduction error, considering l demonstration trajectories, is given by

$$\text{TrajectoryError} = \frac{1}{l}\sum_{i=1}^{l}\frac{1}{T_i}\sum_{t=0}^{T_i}||\boldsymbol{x}_t^i - \hat{\boldsymbol{x}}_t^i||_2 \tag{3.5}$$

This reproduction error is also used in [51]. It gives the mean trajectory error for a given shape model, where T denotes the actual time of the human demonstration movement, $\boldsymbol{x}$ is the actual trajectory, and $\hat{\boldsymbol{x}}$ is the estimated trajectory. For Hamiltonian dynamics, the reproduction error is determined as the mean error between the leapfrog-generated trajectory $\boldsymbol{x}$ and the trajectory $\hat{\boldsymbol{x}}$ estimated from the learned dynamics.

In the case of Hamiltonian systems, the residual error was computed at each point, denoted by i, to assess the difference between the predicted values and the actual values for both the position $\boldsymbol{q}$ and momentum $\boldsymbol{p}$. This was achieved by utilizing the equation

$$\text{ResidualError} = \hat{\boldsymbol{x}}_i - \boldsymbol{x}_i \tag{3.6}$$

Additionally, the total error of $\boldsymbol{q}$ and $\boldsymbol{q}$ at each point was determined by calculating the Euclidean norm. This was accomplished using the equation

$$\text{TotalError} = \sqrt{q_i^2 + p_i^2} \tag{3.7}$$

The errors obtained from equations (3.6) and (3.7) were plotted to provide a visual representation of the accuracy of the estimated trajectories.

3.5.2. Computation Time for Training

To gain insights into how different characteristics in vector fields will impact the computation time when solving the regression problems using the LASA dataset, the model's training time was analyzed. In order to do this, a Python decorator utility was developed to measure the time it takes to approximate $\boldsymbol{\alpha}$ in the different regression problems. The decorator is based on [45]. This utility records the start and end times of a method's execution and calculates the difference between them.

Chapter 4.

Results

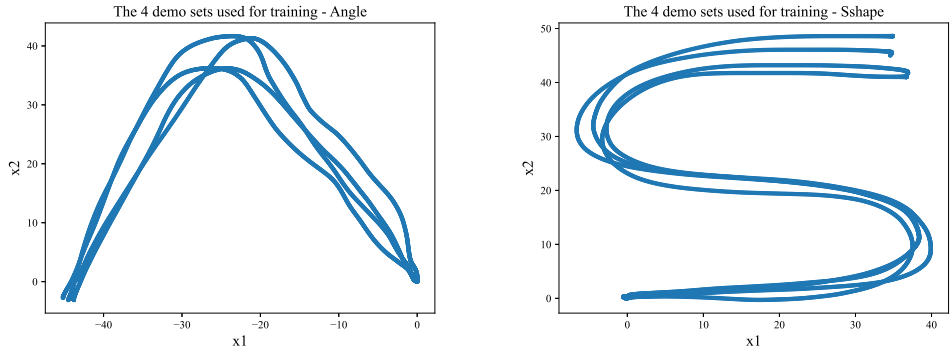
This section presents the results obtained in the thesis by implementing the methodologies outlined in the Method chapter. These methodologies included learning and simulating the models described in Section 3.3 and 3.4. The results include plots, errors, and computation time for different functions. The outcomes offer an understanding of how models using regression and RFF perform on simulated and real-world data.

4.1. LASA Handwriting with RFF

The results achieved by solving different regression problems using the LASA data benchmark are presented in this section. The algorithms applied in the analysis are detailed in Section 3.3.4, while the resulting plots and associated error measurements are displayed here. As mentioned in Section 3.3.1, it was decided to learn the dynamics for the Angle- and S-shape.

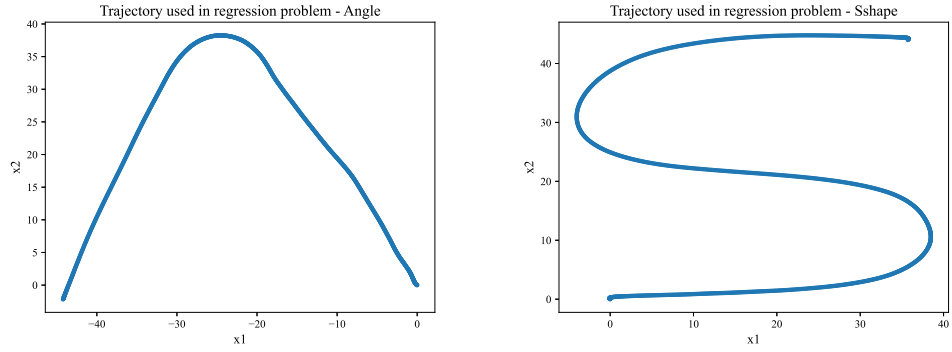
Figure 4.1 shows the four demonstrations used for training. As noted in Section 3.3.1, the demonstrations begin at different initial positions but end at the same point, which is evident from the plots. The mean trajectory was calculated for the four demonstrations, which were then used for learning in most regression problems, as stated in Section 3.3.3. After computing the mean, the resulting trajectories can be seen in Figure 4.2.

For the following results, each subplot shows the result for (a) the Angle-shape dataset and (b) the S-shape dataset, except for the plots where the learnt model trajectories are compared to the actual trajectories.



(a) (b)

Figure 4.1.: The four trajectories used for training (a) Angle-shape (b) S-shape



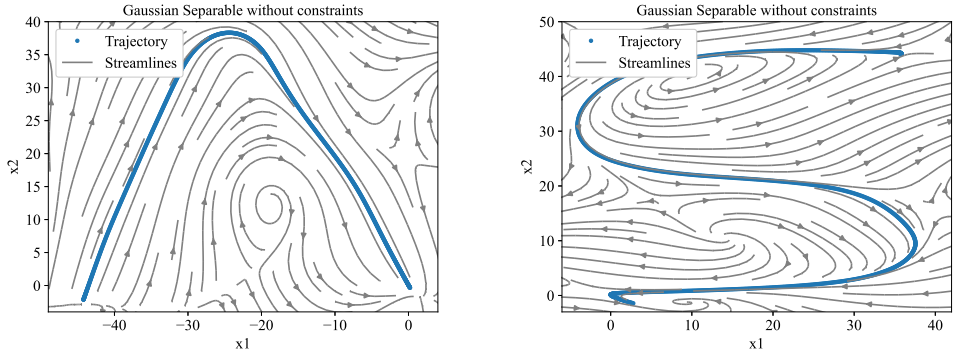
(a) (b)

Figure 4.2.: The mean trajectory used for training (a) Angle-shape (b) S-shape

4.1.1. Learnt Models with Gaussian Separable Feature Map

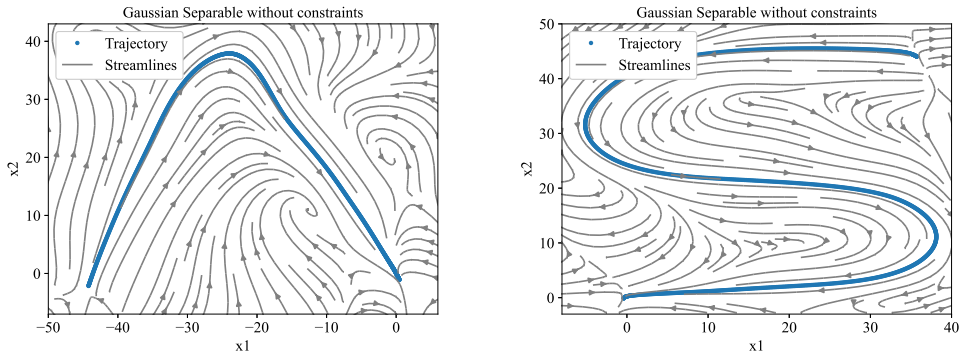
The results of learning a model that uses RFF to approximate a Gaussian separable kernel are displayed in this section. Two different vector fields were estimated, one where contraction constraints were enforced during the learning process and another where no contraction constraints were applied. In the absence of contraction, two regression problems were solved, one where the mean of the training demonstrations was calculated and one where it was not, as mentioned in Section 3.3.2. This resulted in one model being trained on a single trajectory and one model on four trajectories. The learnt model trained on a single trajectory is shown in Figure 4.3, while the learnt model trained on four trajectories can be

seen in Figure 4.4.



(a) (b)

Figure 4.3.: Learnt model with Gaussian separable feature map trained on a single trajectory (a) Angle-shape (b) S-shape



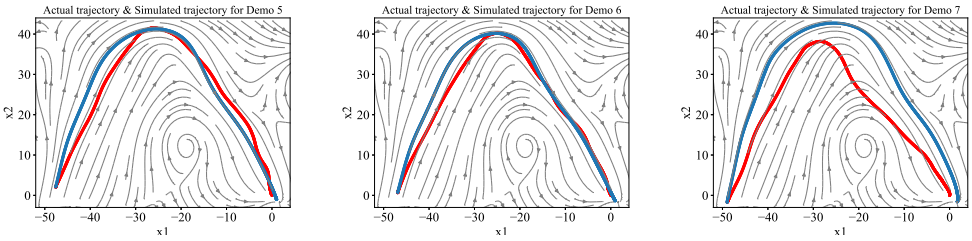
(a) (b)

Figure 4.4.: Learnt model with Gaussian separable feature map trained on four trajectories (a) Angle-shape (b) S-shape

In both cases where the mean was taken and not, the estimated $\boldsymbol{\alpha}$ was tested on the last three demonstration sets. The test sets were estimated separately. The results can be seen in Figure 4.5 and Figure 4.6 when the mean was taken, while Figure 4.7 and Figure 4.8 show the case where no mean was taken. The plots compare the estimated and actual trajectories, where the blue line represents the estimated trajectory while the red line represents the actual trajectory.

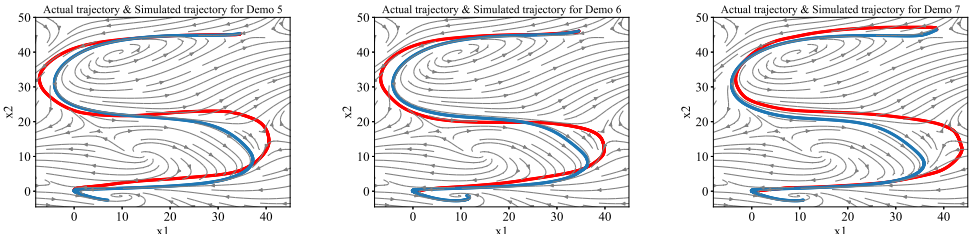
The error measurements between the actual and estimated trajectories were also

found for both cases, and the result can be seen in Table 4.1 and Table 4.2.



(a) (b) (c)

Figure 4.5.: Angle-shape - Comparison between actual and estimated trajectory for model trained on a single trajectory with a Gaussian separable feature map (a) Demo 5 (b) Demo 6 (c) Demo 7

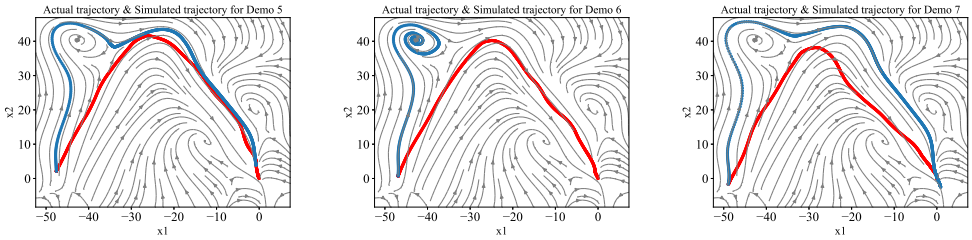


(a) (b) (c)

Figure 4.6.: S-shape - Comparison between actual and estimated trajectory for model trained on a single trajectory with a Gaussian separable feature map (a) Demo 5 (b) Demo 6 (c) Demo 7

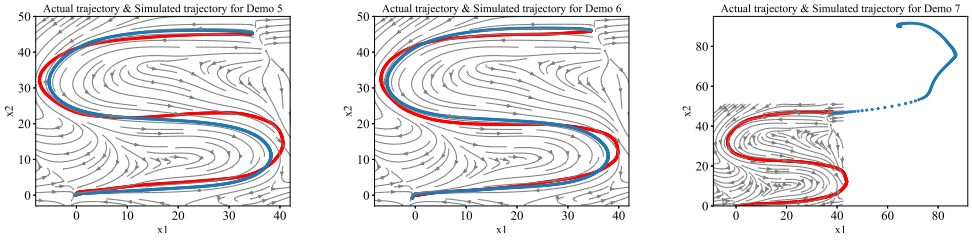
Error for Gaussian separable feature map with mean		
Test set	Angle-shape	S-shape
Demoset 5	1.7508	1.9645
Demoset 6	1.6415	3.0726
Demoset 7	1.7658	1.9650
Mean trajectory error	1.7194	2.3341

Table 4.1.: Error measurements between actual and estimated trajectory for model trained on a single trajectory with a Gaussian separable feature map



(a) (b) (c)

Figure 4.7.: Angle-shape - Comparison between actual and estimated trajectory for model trained on four trajectories with a Gaussian separable feature map (a) Demo 5 (b) Demo 6 (c) Demo 7

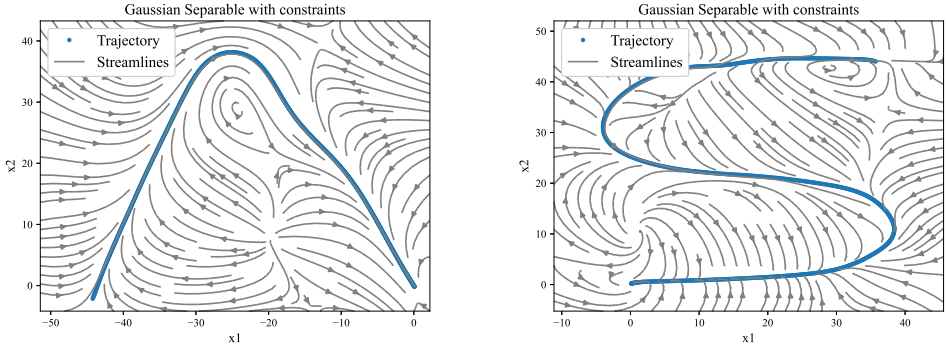


(a) (b) (c)

Figure 4.8.: S-shape - Comparison between actual and estimated trajectory for model trained on four trajectories with a Gaussian separable feature map (a) Demo 5 (b) Demo 6 (c) Demo 7

Error for Gaussian separable feature map without mean		
Test/demo set	Angle-shape	S-shape
Demoset 5	2.8838	0.7612
Demoset 6	10.1051	1.5059
Demoset 7	4.4088	17.6026
Mean trajectory error	5.7993	6.6232

Table 4.2.: Error measurements between actual and estimated trajectory for the model trained on four trajectories with a Gaussian separable feature map



(a) (b)

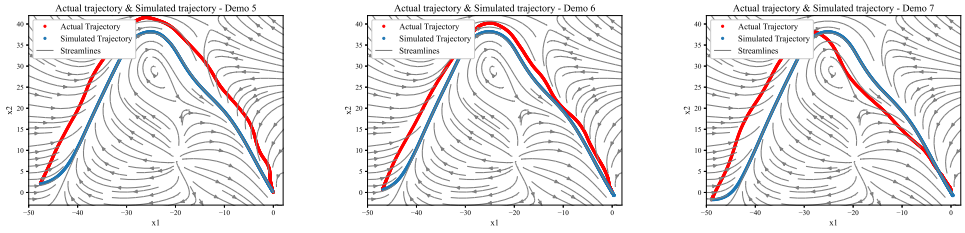
Figure 4.9.: Learnt model with Gaussian separable feature map and contraction constraints (a) Angle-shape (b) S-shape

Table 4.3 displays the computation time for the functions used to approximate $\boldsymbol{\alpha}$ for the different regression problems presented in this section.

Computation time for solving $\boldsymbol{\alpha}$		
Function	Angle-shape	S-shape
alpha_approx without mean	3.4586s	7.5489s
alpha_approx with mean	0.8560s	2.1191s
alpha_approx_with_constraint	27.3204s	56.6461s

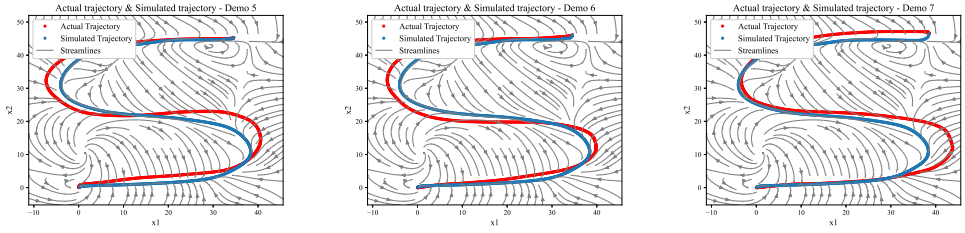
Table 4.3.: Computation time for solving $\boldsymbol{\alpha}$ in regression problem with Gaussian separable feature map

The estimated $\boldsymbol{\alpha}$ was tested on the remaining demonstrations, and the result can be seen in Figure 4.10 and Figure 4.11. The error between the estimated and actual trajectory was calculated, and it can be found in Table 4.4



(a) (b) (c)

Figure 4.10.: Angle-shape - Comparison between actual and estimated trajectory for model with Gaussian separable feature map and contraction (a) Demo 5 (b) Demo 6 (c) Demo 7



(a) (b) (c)

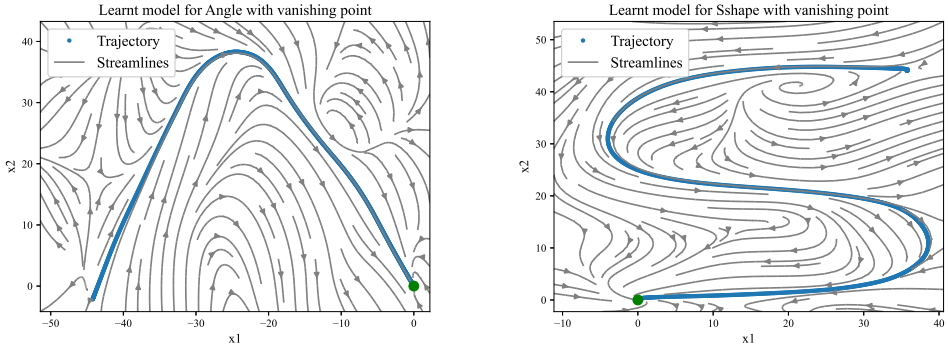
Figure 4.11.: S-shape - Comparison between actual and estimated trajectory for model with Gaussian separable feature map and contraction (a) Demo 5 (b) Demo 6 (c) Demo 7

Error for Gaussian separable feature map with contraction		
Test/demo set	Angle-shape	S-shape
Demoset 5	1.0729	0.8582
Demoset 6	0.7960	1.3056
Demoset 7	0.8125	0.7270
Mean trajectory error	0.8938	0.9636

Table 4.4.: Error measurements between actual and estimated trajectory model with Gaussian separable feature map and contraction

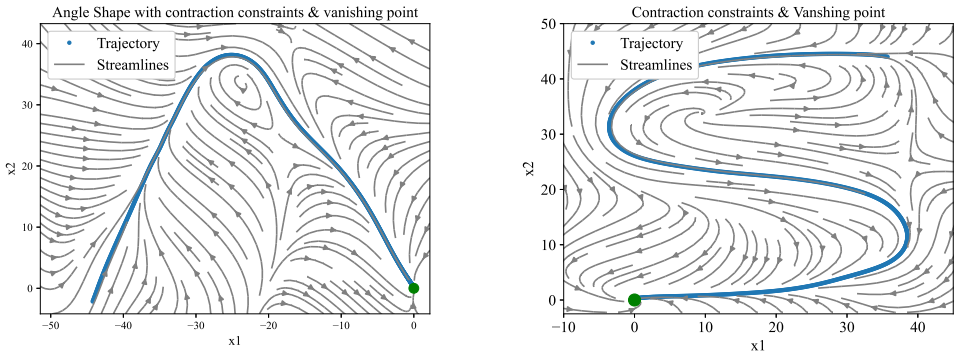
4.1.2. Learnt Models with Vector Field Vanishing on a Point Set

This section presents the results obtained when RFF is used to approximate a Gaussian separable kernel, and it is desired that the vector field vanishes at a specified point. One model was learned considering that no contraction was present, and the result obtained can be seen in Figure 4.12. A second model was learned when contraction constraints were incorporated into the regression problem, as shown in Figure 4.13. Notice that the plots contain a green dot representing the location of the vanishing point.



(a) (b)

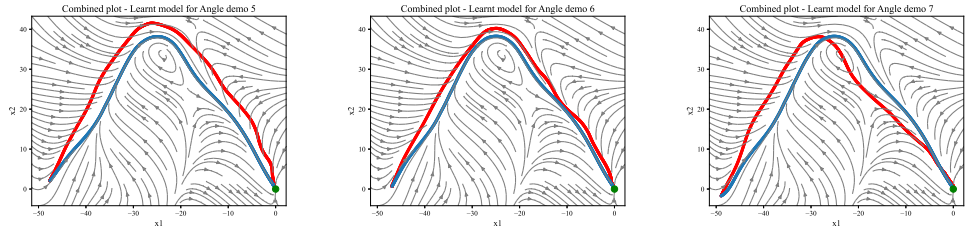
Figure 4.12.: Learnt model with vanishing point and Gaussian separable feature map (a) Angle-shape (b) S-shape



(a) (b)

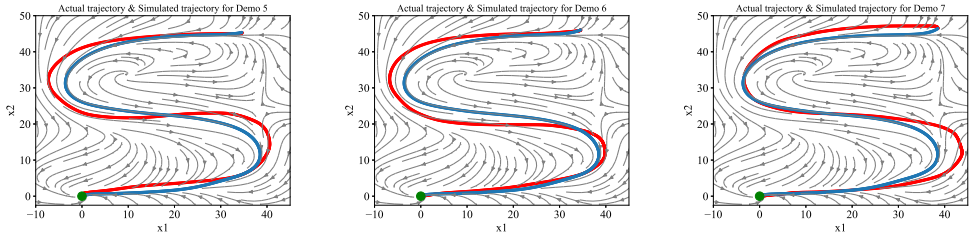
Figure 4.13.: Learnt model with vanishing point, contraction constraints and Gaussian separable feature map (a) Angle-shape (b) S-shape

The estimation of $\boldsymbol{\alpha}$ in the case of contraction was then used to find an estimate of the test demonstrations. The estimated trajectories were plotted with the actual trajectory, which can be seen in Figure 4.14 and 4.15.



(a) (b) (c)

Figure 4.14.: Angle-shape - Comparison between actual and estimated trajectory for model with contraction constraints and vanishing point (a) Demo 5 (b) Demo 6 (c) Demo 7



(a) (b) (c)

Figure 4.15.: S-shape - Comparison between actual and estimated trajectory for model with contraction constraints and vanishing point (a) Demo 5 (b) Demo 6 (c) Demo 7

The computation time to find the two estimations of $\boldsymbol{\alpha}$ was also calculated and can be seen in Table 4.5.

The mean error between the actual and estimated trajectory for the different test demonstrations was calculated, and the results can be seen in Table 4.6.

Computation time for estimating α		
Function	Angle-shape	S-shape
alpha_approx_z	1.2785s	3.6893s
alpha_approx_with_constraint_z	34.3604s	66.1032s

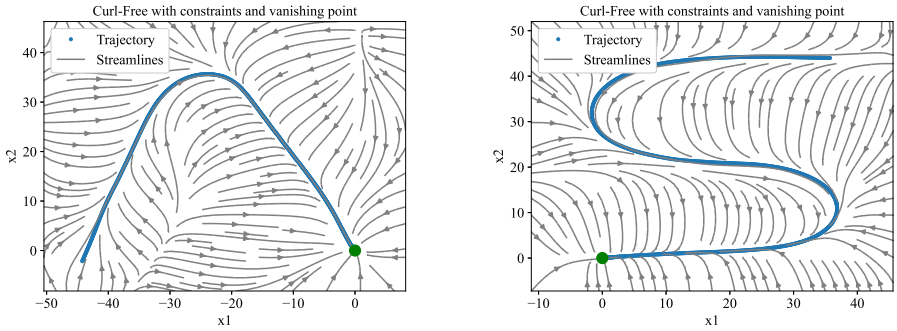
Table 4.5.: Computation time for estimating α in regression problems with vanishing point and Gaussian separable feature map

Error for learnt model with vanishing point and contraction		
Test/demo set	Angle-shape	S-shape
Demoset 5	1.1356	0.8422
Demoset 6	0.7620	1.2770
Demoset 7	0.9534	0.7221
Mean trajectory error	0.9503	0.9471

Table 4.6.: Error measurements between actual and estimated trajectory for model with vanishing point, contraction and Gaussian separable feature map

4.1.3. Learnt Model with Curl-Free Feature Map

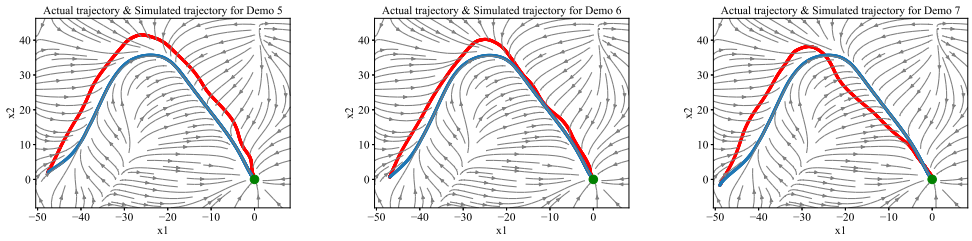
In this section, the results of training a model that uses RFF to approximate a curl-free kernel. Figure 4.16 displays the results of the simulation for the learnt model involving contracting vector fields with a curl-free feature map while incorporating a vanishing point.



(a) (b)

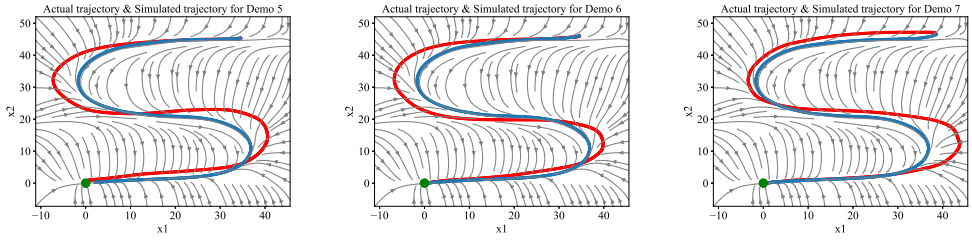
Figure 4.16.: Learnt curl-free model with vanishing point and contraction (a) Angle-shape (b) S-shape

The estimation of $\boldsymbol{\alpha}$ found from the learnt models presented in Figure 4.16 was used to find an estimate of the three test demonstrations for both Angle- and S-shape. These estimations were then plotted with the actual trajectory, as seen in Figure 4.17 and Figure 4.18.



(a) (b) (c)

Figure 4.17.: Angle-shape - Comparison between actual and estimated trajectory for model with vanishing points, contraction and curl-free feature map (a) Demo 5 (b) Demo 6 (c) Demo 7



(a) (b) (c)

Figure 4.18.: S-shape - Comparison between actual and estimated trajectory for model with vanishing points, contraction and curl-free feature map (a) Demo 5 (b) Demo 6 (c) Demo 7

The computation time for estimating $\boldsymbol{\alpha}$ was calculated, and it can be seen in Table 4.7.

The error measurements between the estimated trajectories and actual trajectories for the test demos were calculated for both Angle and S-shape, and the result is shown in Table 4.8.

Computation time for estimating α		
Function	Angle-shape	S-shape
alpha_approx_with_constraint_cf	24.5403s	58.2451s

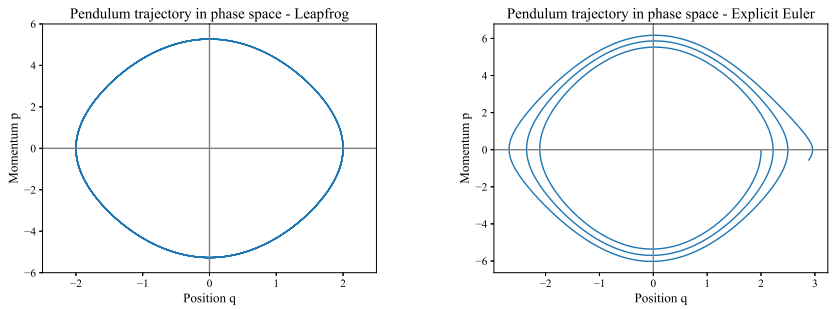
Table 4.7.: Computation time for estimating $\boldsymbol{\alpha}$ in regression problem with curl-free feature map

Error for curl-free feature map		
Test/demo set	Angle-shape	S-shape
Demoset 5	1.6316	0.9493
Demoset 6	1.4436	1.4383
Demoset 7	1.0377	0.8934
Mean trajectory error	1.3709	1.0936

Table 4.8.: Error measurements between actual and estimated trajectory for model with vanishing point, contraction and curl-free feature map

4.2. Hamiltonian Systems with RFF

This section presents the results obtained from estimating the Hamiltonian dynamics of a pendulum. The results from using the two feature maps to estimate a trajectory generated by leapfrog are presented, followed by the outcomes from adding noise to the generated trajectory. In the study, the symplectic kernel and the Gaussian separable kernel were approximated using RFF. This approximation resulted in a symplectic and a Gaussian separable feature map.



(a) (b)

Figure 4.19.: Generated trajectories for a pendulum system with step size = 0.01 (a) Leapfrog method (b) Explicit Euler method

As mentioned in Section 3.4.1, it was necessary to generate trajectories for the pendulum system in order to use them for learning. Figure 4.19 shows the two generated trajectories in phase space, where the same initial point and step size were used. The trajectory in Figure 4.19(a) was obtained using the leapfrog method, while the trajectory in Figure 4.19(b) was obtained using explicit Euler.

4.2.1. Hamiltonian Dynamics Model with RFF

As mentioned in Section 3.4.2, the trajectory used for learning was the one generated by using the leapfrog method and the resulting vector field used in the regression problems is presented in Figure 4.20.

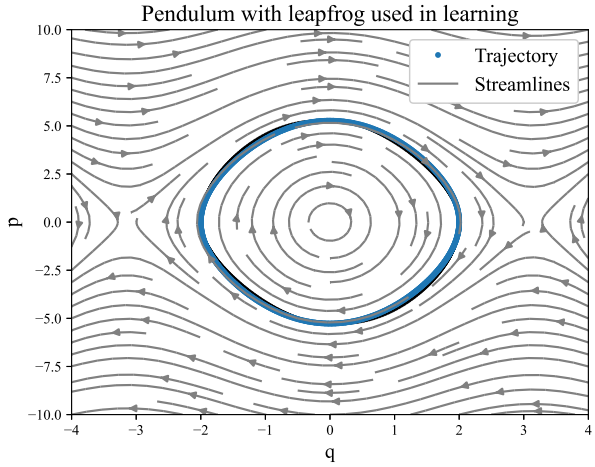


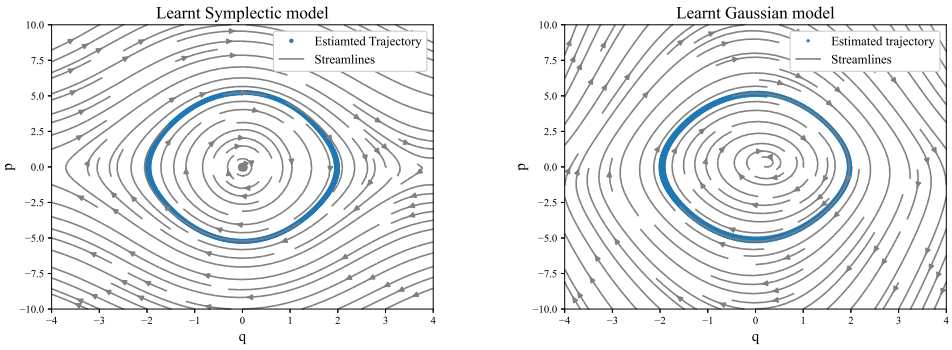
Figure 4.20.: Pendulum dynamics used in regression problem

The model was learned using two different feature maps and then simulated to generate the estimated trajectories. The approximated vector fields are shown in Figure 4.21.

Furthermore, to compare the similarities and differences between the actual and estimated trajectories, they were plotted in the same figure, as illustrated in Figure 4.22.

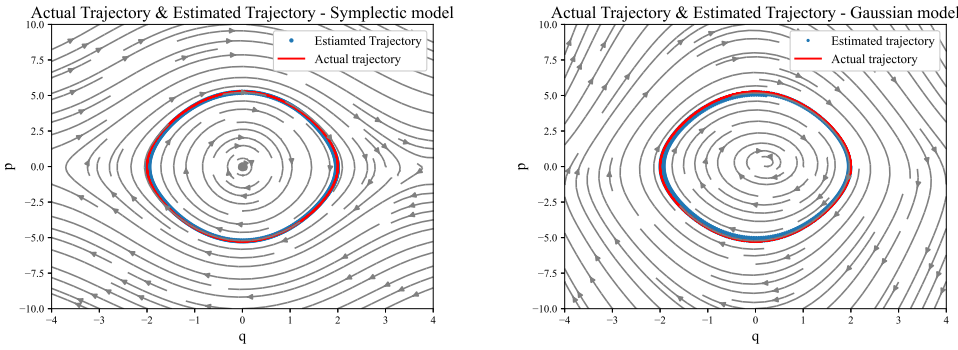
To better understand the error progression for the position and momentum, the errors at each point were plotted on a time-axis, as illustrated in Figure 4.23. The dark blue plot represents the error for the position, while the light blue plot represents the error for the momentum.

In Figure 4.24, the total error between the actual and estimated trajectory is plotted at each point, with the time-axis displaying the error scale.



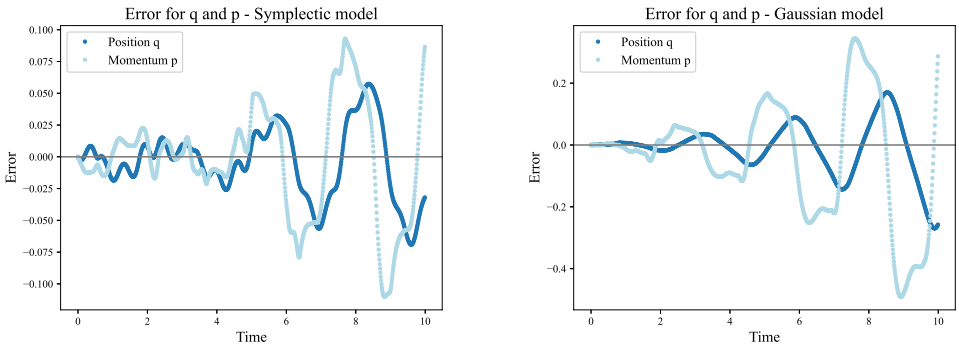
(a) (b)

Figure 4.21.: Learnt pendulum dynamics (a) Symplectic feature map (b) Gaussian separable feature map



(a) (b)

Figure 4.22.: Estimated trajectory compared to actual trajectory (a) Symplectic feature map (b) Gaussian separable feature map



(a) (b)

Figure 4.23.: Error for position $\boldsymbol{q}$ and momentum $\boldsymbol{p}$ (a) Symplectic feature map (b) Gaussian separable feature map

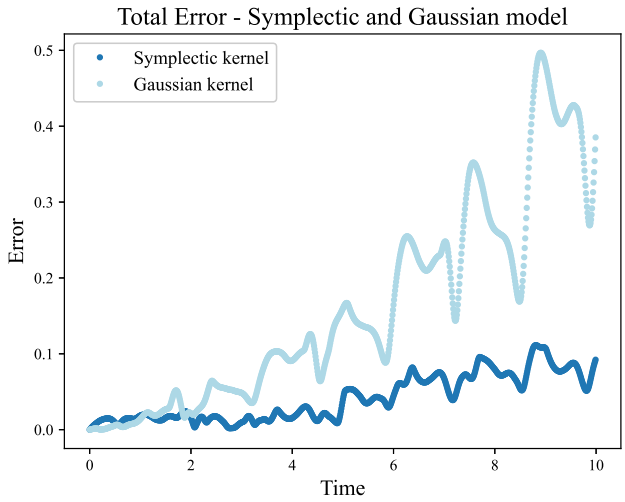


Figure 4.24.: Total error for the estimated trajectories

4.2.2. Hamiltonian Dynamics Model with Added Noise

The following results were obtained when noise was added to the training data. Figure 4.25 displays the vector field used in the regression problem under this circumstance, and leapfrog was again the method used to generate the trajectory.

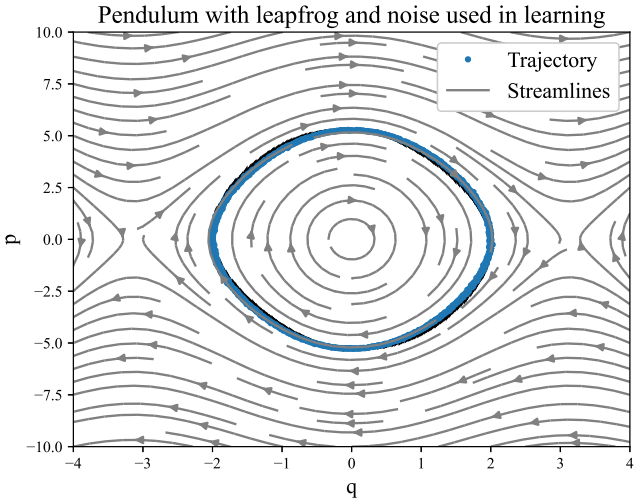


Figure 4.25.: Pendulum dynamics used in regression problem with noise

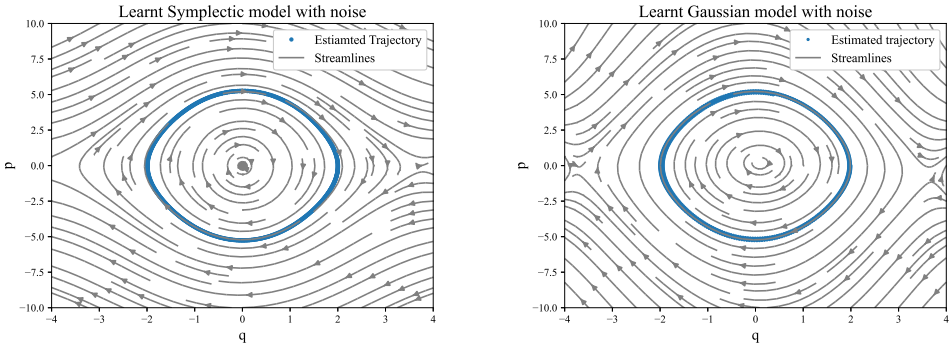
The learnt models using RFF are presented in Figure 4.26. In Figure 4.26(a), the symplectic feature map was used in the regression problem, while in Figure 4.26(b) the Gaussian separable feature map was used in the regression problem.

Figure 4.27 shows the estimated trajectories and streamlines plotted against the actual trajectory.

The errors for the estimated position and momentum are shown for both the symplectic feature map and the Gaussian separable feature map in Figure 4.28.

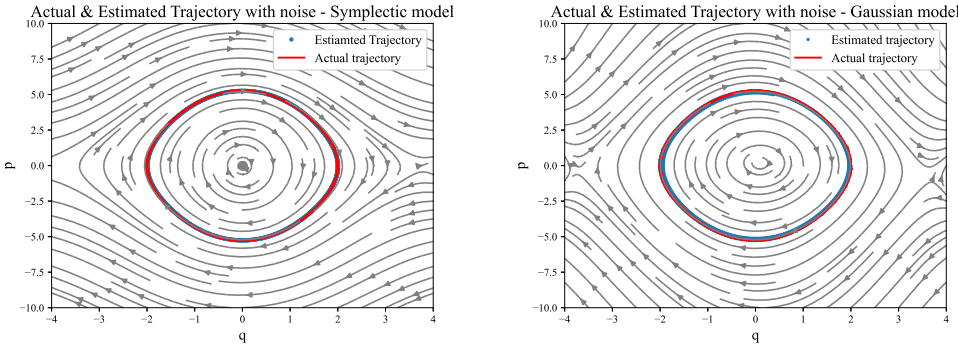
The plot in Figure 4.29 displays the total error as a function over time between the actual and estimated trajectory in the case where noise was added.

The mean trajectory error between the actual and estimated trajectory is presented in Table 4.9. It presents the error for the case where noise was added and the case when it was not.



(a) (b)

Figure 4.26.: Learnt model with noise (a) Symplectic feature map (b) Gaussian separable feature map

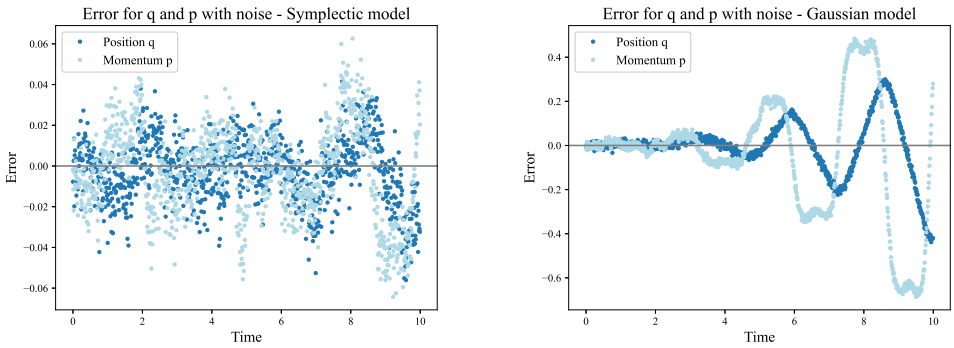


(a) (b)

Figure 4.27.: Estimated trajectory compared to actual trajectory with noise (a) Symplectic feature map (b) Gaussian separable feature map

Error for learnt model - Pendulum		
Feature map	MTE	MTE with noise = 0.01
Symplectic	0.00414	0.00238
Gaussian separable	0.01593	0.02186

Table 4.9.: Mean trajectory error measurements for learnt Pendulum dynamics



(a) (b)

Figure 4.28.: Error for position $\boldsymbol{q}$ and momentum $\boldsymbol{p}$ with noise (a) Symplectic feature map (b) Gaussian separable feature map

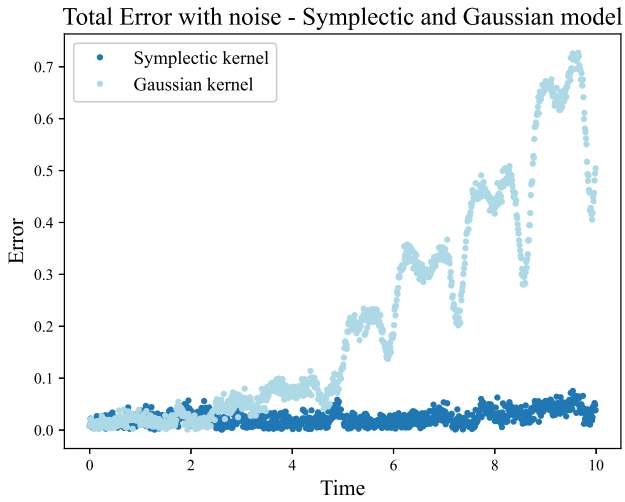


Figure 4.29.: Total error for the estimated trajectories with noise

Chapter 5.

Discussion

This section will outline the anticipated results for the learnt models, followed by an analysis and comparison of the findings presented in the Results chapter. Furthermore, future work will be discussed.

5.1. Expectations

As demonstrated in the specialization project [28], using RFF for kernel approximation resulted in reasonable estimates with reduced computational cost. Consequently, given that the hyperparameters are carefully selected, the use of RFF is expected to give reasonable estimates for the problems outlined in this thesis with low computational cost. Furthermore, since RFF is used to approximate a vector field, the expected behavior of the estimated streamlines will be similar to the actual streamlines.

5.1.1. Expectations for LASA Models

Computation time for the different shapes

In terms of computation time, it is anticipated that the Angle-shape will exhibit faster performance compared to the S-shape. This expectation is based on the difference in the number of samples, with the Angle-shape having 100 samples and the S-shape having 200.

Estimating with and without taking the mean of training sets

As mentioned in 3.3.2, most results were generated by taking the mean of the four training demonstrations, leading to a single trajectory. When taking the mean of the training sets, the impact of noisy data points is expected to be reduced,

leading to a more robust model that is less sensitive to such noise. This can lead to a more stable and reliable model that generalizes better to new, unseen data. On the other hand, not taking the mean of the training sets can be useful in situations where the data has a clear structure or pattern. By not taking the mean, the model can capture the underlying structure or pattern more accurately and potentially achieve better performance on the training data. For the LASA dataset, taking the mean gives a training set of 1000 points, while not taking the mean lets the model be trained on all 4000 points. Creating a single trajectory for model training using four trajectories is expected to be more beneficial in our case, as it will help reduce the impact of minor variations in the individual patterns and make the training process more stable. It is also expected that the computation time will be reduced in the case where the mean trajectory is used.

Estimating with contraction constraints

Based on the contraction theory presented in Section 2.5.2, it is expected that adding contraction constraints to the regression problem would lead to performance improvement in robustness and accuracy. It is anticipated that all trajectories close to the estimated trajectory would converge toward it. This will enforce smoothness and consistency in the learned vector field, preventing overfitting and improving its generalization performance. In regions where the contraction constraints are satisfied, the streamlines are expected to converge towards each other and the estimated trajectory, with the distance between them decreasing exponentially. The resulting convergence would increase the accuracy of the estimated flow by constraining the magnitude of the Jacobian of the learned vector field. Ultimately, this reduces unrealistic or unstable flow behavior.

Applying contraction constraints to the regression problem during the learning process is also anticipated to increase the computation time. This is because each point in the applied constraints needs to be differentiated, as seen in Algorithm 2, resulting in additional steps required to estimate $\boldsymbol{\alpha}$. Furthermore, the incorporation of constraints made using a numerical optimization solver like MOSEK necessary. This approach is expected to result in increased computation time as it uses an iterative method to obtain a solution, as compared to using a closed-form solution.

Estimating with vanishing point

For the problems where vanishing points are defined, improvements in the coherence of the resulting vector fields are expected. This means that the streamlines should not contain abrupt changes in direction but have smooth and consistent flow patterns. Furthermore, vanishing points are expected to act as attractors and

guide streamlines toward specific regions. Therefore, the streamlines around the vanishing points should converge toward them. The vanishing point in this thesis is (0, 0), as seen in Table 3.3. Hence, it is expected that the streamlines converge toward this point. It is also expected that the equilibrium of the estimated trajectory is placed precisely at the specified point.

The presence of vanishing points is expected to increase the computation time compared to their absence. This is because estimating $\boldsymbol{\alpha}$ requires more steps when vanishing points are present. This can be seen in the algorithms and in Section 2.6.2, where the process of defining the new feature map $\boldsymbol{\Psi}^Z = \boldsymbol{L}^T\boldsymbol{\Psi}$ with the property of vanishing on Z requires additional steps, one of which involves finding the values of $\boldsymbol{L}$.

Estimating curl-free vector field

When estimating a vector field using a curl-free feature map instead of a Gaussian separable feature map, it is expected that several differences could arise. As Section 2.4.2 explains, a vector field is considered curl-free if it has zero curls. Hence, the vector fields estimated with a curl-free feature map are expected to maintain the direction and magnitude of their streamlines. In this case, the streamlines are anticipated to exhibit gradient flows and not form closed loops or rotations.

Furthermore, the computation time for estimating $\boldsymbol{\alpha}$ in the curl-free case is expected to be faster compared to non-curl-free. This is because of the dimensions, where $\boldsymbol{\alpha}$ is in $\mathbb{R}^d$ in the curl-free case compared to $\mathbb{R}^{dm}$.

5.1.2. Expectations for Learning Hamiltonian Dynamics

Benefits of using a symplectic integrator for Hamiltonian systems

Given the theoretical framework presented in Section 2.7.6, the leapfrog method is expected to yield better results than the explicit Euler method. This is primarily because explicit Euler is only first-order accurate, while leapfrog is second-order accurate, as previously explained. Consequently, it can be inferred that the leapfrog method can handle larger step sizes without numerical instability. In contrast, the explicit Euler method is expected to generate poor results when the step size is too large. As a result, when using the same step size h, the leapfrog method is generally expected to yield more accurate results than the explicit Euler method.

The leapfrog method is also anticipated to outperform the explicit Euler method in conserving energy. Being a symplectic integrator, the leapfrog method preserves the Hamiltonian structure of the system. This property contributes to better en-

ergy conservation, resulting in more accurate and stable simulations over time. In contrast, the explicit Euler method is a non-symplectic integrator and therefore does not preserve the symplectic structure of the system. Consequently, numerical errors can accumulate over time, causing the estimate to deviate from the system's actual behavior. This deviation from the expected trajectory violates the conservation of energy. This violation is expected to be observed by coordinate values exhibiting unbounded growth and expanding the region of interest over time.

Estimating Hamiltonian dynamics

Approximating a symplectic kernel using RFF is expected to enhance the accuracy of results for Hamiltonian systems. This is because the generated symplectic feature map satisfies the symplectic conditions, as outlined in Section 2.7.3. Therefore the symplectic structure of the original Hamiltonian system will be preserved even after transformation through the feature space. By comparing it to the Gaussian separable feature map, it is anticipated that the symplectic feature map will demonstrate better energy conservation over an extended period and enhanced accuracy in learning Hamiltonian dynamics.

The preliminaries state that the leapfrog method is a symplectic integrator known for preserving the symplectic structure. Since the dataset was generated using this method, the dataset itself inherently exhibits a symplectic structure. Consequently, the symplectic feature map is expected to outperform the Gaussian separable feature map on this dataset. As mentioned in Section 2.7.3, the symplectic kernel is designed to capture and model the symplectic dynamics of the system, making it well-suited for accurately representing the underlying behavior. However, it is important to note that the Gaussian separable feature map, despite not explicitly preserving the symplectic structure, can still capture the overall patterns and structure of the data. Thus, while the symplectic feature map is anticipated to perform better, the Gaussian separable feature map can still provide reasonable results by capturing the general characteristics of the dataset.

Introducing noise to the dataset

As discussed in Section 3.4.2, including noise in a dataset enhances the model's robustness, mitigating the risk of overfitting. Consequently, this improvement is anticipated to be reflected in the obtained results. Additionally, adding noise provides a more realistic representation of the complexity present in real-world datasets. Therefore, it is expected to understand better how the model will perform when faced with diverse and challenging scenarios encountered in real-world applications.

Based on the standard deviation of 0.01 observed in Table 3.7, it can be inferred that the added noise to the data is relatively insignificant. This implies that the noise is unlikely to impact the underlying patterns of the data significantly but rather introduce subtle variability. This subtle noise is expected to preserve the essential characteristics of the data while contributing to smooth transitions and improved robustness in both regression problems as long as the regression parameters are carefully adjusted. By incorporating such small-magnitude noise, the Gaussian separable feature map is anticipated to yield smooth and continuous estimations, effectively capturing the data's patterns and providing reliable predictions. When dealing with the symplectic feature map, it is important to note that any added noise should not interfere with the inherent symplectic properties of the data. This guarantees that the symplectic feature map can accurately capture the data's patterns and dynamics, even in the presence of noise. Ultimately, this leads to reliable predictions that align with the symplectic nature of the system.

Although both feature maps are expected to perform well with added noise, the symplectic feature map is anticipated to outperform the Gaussian separable feature map. This is because of its symplectic characteristics, which can be useful in capturing the underlying Hamiltonian dynamics.

5.2. Comparing Results - LASA

In this section, an analysis and comparison will be conducted on the results presented in Section 4.1, showing the outcomes from solving regression problems using the LASA dataset.

5.2.1. Estimation with and without Mean

Based on Figure 4.3-4.8, as well as Table 4.1 and 4.2, the findings indicate that the most accurate results were obtained when training the model on a single trajectory. Specifically, this was achieved by taking the mean of four demonstrations to create a single training set consisting of 1000 points. Both models provided a good estimate for the model when training it, where the model trained on four different trajectories even performed slightly better for the S-shape. This can be observed in Figure 5.1, where the model trained on a single trajectory shows a slight deviation in the learned S-shape towards the end.

Comparing Table 4.1 with Table 4.2 and Figures 4.5-4.8, it is evident that the trained model provides better results for the test demonstrations when it is trained on a single trajectory. The mean trajectory error (MTE) for the Angle-shape was 1.7194, and 2.3341 for the S-shape when trained on a single trajectory. In contrast,

(a) (b)

Figure 5.1.: End part of the estimated trajectory for S-shape model (a) Trained on a single trajectory (b) Trained on four trajectories

the MTE was 5.7993 for the Angle-shape and 6.6232 for the S-shape when trained on four trajectories. The difference is significant, indicating that the best estimate is obtained when taking the mean, which was the expected outcome.

On the contrary, it is worth mentioning that the model, which was trained on four trajectories, performed well on demos 5 and 6 for the S-shape. The first two test demos had errors of 0.7612 and 1.5059, respectively, which were better than the errors obtained when taking the mean, where the errors were 1.9645 and 3.0726. However, the last test demo resulted in a significantly poor outcome, with an error of 17.6026 compared to 1.9650 when taking the mean. This indicates that training the model on a single trajectory is more effective in finding an estimate that works for all the demos. Additionally, when it comes to the Angle-shape, training the model on a single trajectory outperforms training on four trajectories for all test demos, with errors around 1.7 compared to the best result of 2.8838 for the four-trajectory model.

Table 4.3 provides the computation time for both shapes, revealing noteworthy differences. Specifically, learning the S-shape has a significantly higher computational cost compared to the Angle-shape. This was expected, due to the requirement of 200 samples for accurate outcomes in the case of the S-shape, in contrast to the 100 samples needed for the Angle-shape.

In line with the expectations, the single trajectory model demonstrated faster computation times for both shapes. In the case of a single trajectory, the computation time was 0.8560s and 2.1191s for the Angle-shape and S-shape, respectively. For the model trained on four trajectories, it was 3.4586s and 7.5489s. The results highlight the computational efficiency of the single trajectory model. That being said, the computation times for both models are relatively fast, demonstrating the computational efficiency of using RFF.

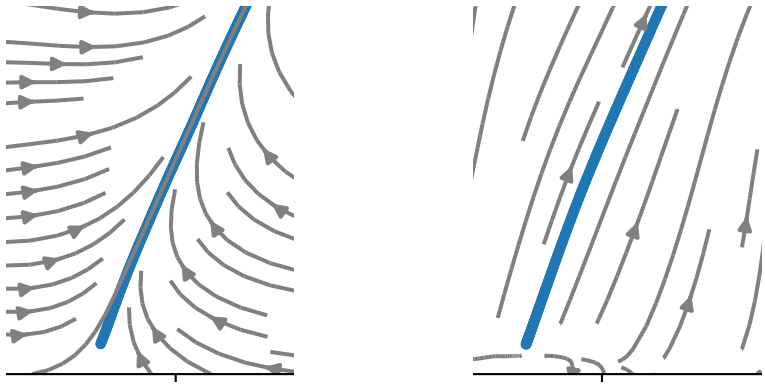
Consequently, based on the test results, it was decided that the preferred approach for the remaining regression problems would be to train the model on a single

trajectory.

5.2.2. Estimation with and without Contraction

The incorporation of contraction constraints in the regression problem, as observed in Figure 4.9, has a noticeable impact on the robustness of the streamlines. In contrast to Figure 4.3 without contraction, including these constraints significantly improves the overall stability of the streamlines. As expected, the contraction constraints resulted in the streamlines converging towards the estimated trajectories and converging toward each other. This can be observed in Figure 4.9, where the streamlines around the estimated trajectory point toward it.

There are clear differences between the streamlines in Figure 4.3 and Figure 4.9. For instance, this can be observed in the initial part of the two trajectories, seen in Figure 5.2. The streamlines in Figure 5.2(b) follow the estimated trajectory, while the streamlines in Figure 5.2(a) converge toward it. This converging behavior aligns with the theory and expected results, indicating that using RFF to learn vector fields with contraction constraints can provide accurate and reliable results.



(a) (b)

Figure 5.2.: Initial part of estimated trajectory for Angle-shape model (a) With contraction (b) Without contraction

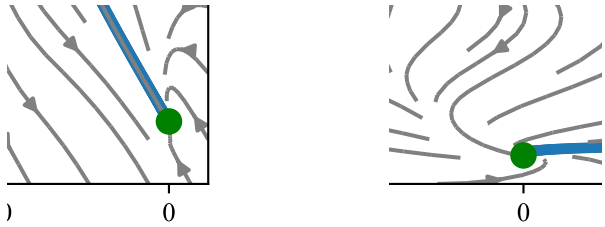
Empirical support for our findings can be observed when examining the estimated trajectory and its corresponding error measurements for the test sets. These estimates can be seen in Figure 4.5-4.6 and Figure 4.10-4.11. In particular, when contraction was incorporated, the MTE for the Angle-shape was found to be 0.8938, as shown in Table 4.4, which is essentially lower than the MTE of 1.7194 obtained without contraction, as shown in Table 4.1. Similarly, the MTE for the S-shape was found to be 0.9636 with contraction, compared to 2.3341 without.

These results demonstrate that the contraction constraints lead to significant improvements in accuracy and performance during the learning process. They also indicate that the model's ability to estimate trajectories is better in the presence of contraction.

That being said, incorporating contraction constraints significantly increased the computational cost required to estimate $\boldsymbol{\alpha}$. As shown in Table 4.3, the computational time for the Angle- and S-shape without contraction was 0.8560s and 2.1191s, respectively, while with contraction, it increased to 27.3204s and 56.6461s. As mentioned in Section 5.1.1, this increase in computational cost was expected, considering the need to incorporate constraints into the model and use MOSEK and PICOS to solve the regression problem. Thus, this additional computational cost is necessary to increase the accuracy and reliability of the simulated learned vector fields.

5.2.3. Estimation with and without Vanishing Point

By examining the vector field in Figure 4.12, it becomes evident that all streamlines around the vanishing point (0, 0) converge towards it for both the Angle- and S-shape. This is easier to observe in Figure 5.3, which provides a close-up of the region around the vanishing point. It can also be observed that the estimated trajectory ends at the desired point, represented by the green dot. These results are consistent with the expectations outlined in Section 5.1.1 and the theory presented in the preliminaries.



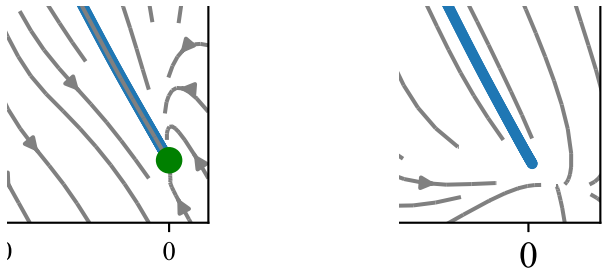
(a) (b)

Figure 5.3.: Estimated vector field around the vanishing point (0, 0) (a) Angle-shape (b) S-shape

To ensure that the results were satisfactory, it was decided to compare the case where a vanishing point was defined to the case where there was none. A closer look around the vanishing point for the Angle- and S-shape is shown in Figure 5.4 and Figure 5.5, respectively. The difference between the streamlines with

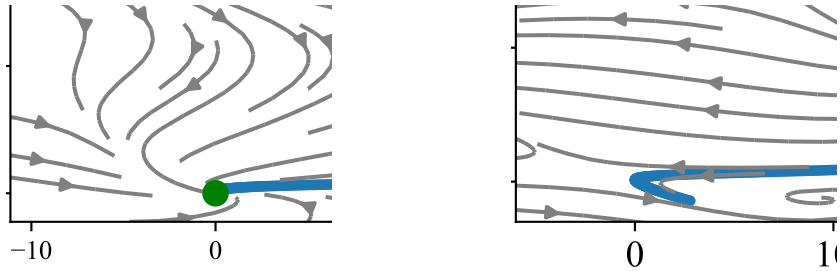
and without a defined vanishing point is clear in both cases. In Figure 5.4(a) and Figure 5.5(a) the streamlines near the vanishing point converge towards it. Conversely, in Figure 5.4(b) and Figure 5.5(b), most of the streamlines move along the estimated trajectory, and the trajectories do not end at the point (0, 0).

The difference between the models with and without a vanishing point is especially evident for the S-shape. None of the streamlines in Figure 5.5(b) converge to (0, 0), whereas all streamlines in Figure 5.5(a) do. Additionally, the estimated trajectory without a vanishing point does not end in (0, 0), while it does in Figure 5.5(a). This difference is consistent with the theory presented in the preliminaries. Therefore, it can be concluded that the RFF method is reliable in estimating vector fields with vanishing points and is likely to produce accurate results.



(a) (b)

Figure 5.4.: Estimated vector field around point (0, 0) for Angle-shape (a) With vanishing point (b) Without vanishing point



(a) (b)

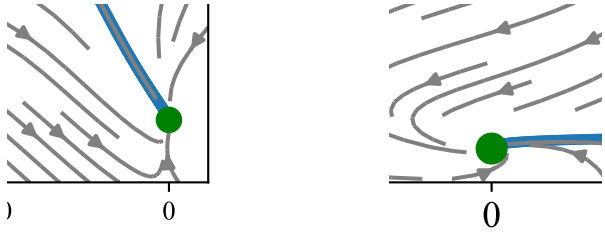
Figure 5.5.: Estimated trajectory around point (0, 0) for S-shape (a) With vanishing point (b) Without vanishing point

Based on the comparison between Table 4.3 and Table 4.5, it can be inferred that the presence of vanishing points leads to increased computational time than in

their absence. For instance, for the Angle-shape, the computation time for trajectory estimation increased from 0.8560s to 1.2785s when a vanishing point was incorporated. This observation is consistent with the expectations since defining a vanishing point requires additional steps when solving the regression problem.

5.2.4. Estimation with Vanishing Point and Contraction

The results were satisfactory when both contraction and a vanishing point were present. As expected, the streamlines surrounding the estimated trajectory converged toward it. Additionally, the estimated trajectories end in (0, 0), and there are clear convergences toward the vanishing point, which is better visualized in Figure 5.6.

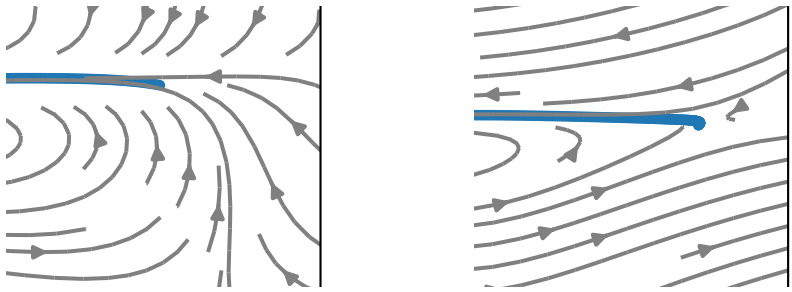


(a) (b)

Figure 5.6.: Region around (0, 0) for estimated vector field with contraction and vanishing point (a) Angle-shape (b) S-shape

Comparing the vanishing point models

For the models involving vanishing points, the expected behavior was for convergence to occur solely around the vanishing point in Figure 4.12, while convergence throughout the estimated trajectory was anticipated in Figure 4.13. This becomes apparent in the area around the initial point of the estimated trajectory, as demonstrated for the S-shape in Figure 5.7. In Figure 5.7(a), all streamlines converge towards the estimated trajectory, whereas in Figure 5.7(b), some streamlines deviate away due to the absence of contraction. However, around the vanishing point, observed by comparing Figure 5.6 with Figure 5.4(a) and Figure 5.5(a), it becomes evident that they closely resemble each other. For both the Angle- and S-shape, the streamlines close to the vanishing point converge towards it, and the trajectories end in (0, 0).

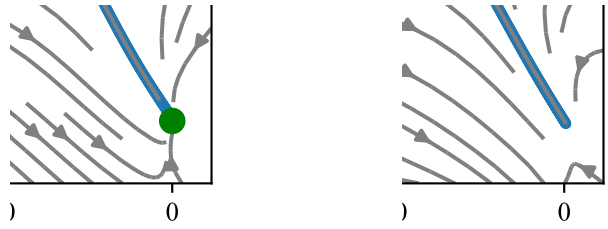


(a) (b)

Figure 5.7.: Initial part of estimated trajectory with vanishing point for S-shape model (a) With contraction (b) Without contraction

Comparing the contraction models

The simulated vector fields obtained from the regression problem with only contraction and the regression problem with both vanishing point and contraction exhibit both differences and similarities. Figure 4.13 and Figure 4.9 show closely resembling streamlines. However, there are distinctions around the vanishing point, particularly for the simulated Angle-shape as shown in Figure 5.8. Furthermore, it can be observed that in the absence of vanishing points, streamlines do not necessarily converge towards (0, 0), as seen in Figure 5.8(b).



(a) (b)

Figure 5.8.: Region around (0, 0) for estimated vector field with contraction for Angle-shape model (a) With vanishing point (b) Without vanishing point

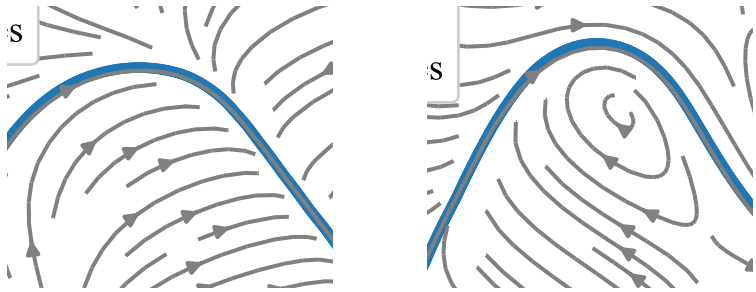
Based on Table 4.6 and the corresponding figures, Figure 4.14 and 4.15, it is evident that training the model with both vanish point and contraction constraints generates good results. The MTE for the Angle-shape was 0.9503, and for the S-shape, it was 0.9471, indicating that the estimated $\boldsymbol{\alpha}$ obtained after training the model was close to accurate. That being said, the trained model with only

contraction generated slightly better results. The computation time was also faster without vanishing points, taking 27.3204s for the Angle-shape and 56.6461s for the S-shape. In contrast, it took 34.3604s and 66.1032s when incorporating a vanishing point, as seen in Table 4.5. This outcome was expected since learning a model with a vanishing point requires more steps, as mentioned.

Although the computation time increased in the presence of vanishing points and the error was slightly larger, the results were satisfactory for both vanishing points and contraction constraints. These findings strongly support that using RFF to approximate kernels in regression problems is a reliable method for estimating vector fields with different characteristics.

5.2.5. Estimation with Curl-Free Feature Map

The simulations in Figure 4.16 have the expected behavior of a curl-free vector field, with streamlines lacking curls and closed loops. The presence of both contraction and a vanishing point is also apparent in the simulated vector fields in the figure.

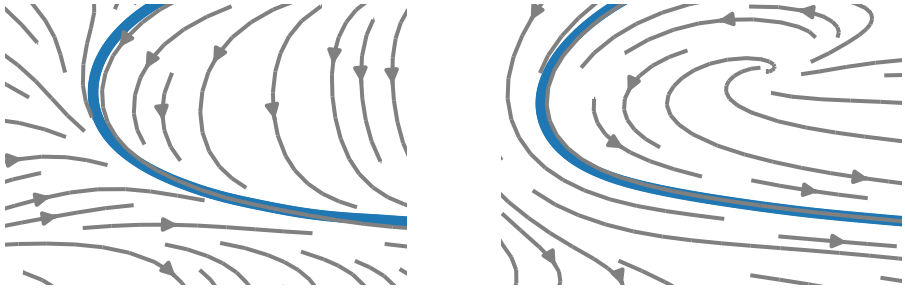


(a) (b)

Figure 5.9.: Middle region of the estimated trajectory for the Angle-shape with vanishing point and contraction (a) Curl-free feature map (b) Gaussian separable feature map

Significant differences in the streamlines and their rotational patterns become apparent by comparing the results using a curl-free feature map to those using a Gaussian separable feature map. This can be observed in Figure 5.9 for the Angle-shape and Figure 5.10 for the S-shape.

While using both feature maps caused the streamlines to converge toward the estimated trajectory, the curl-free streamlines do not exhibit rotations or loops in the flow, in contrast to the Gaussian separable feature map. These properties can be seen in Figure 5.9(b), which aligns with the theory that a curl-free vector field



(a) (b)

Figure 5.10.: Middle region of the estimated trajectory for the S-shape with vanishing point and contraction (a) Curl-free feature map (b) Gaussian separable feature map

should not have any curl, and its streamlines should be gradient flows.

The curl-free model had a slightly higher error than the Gaussian separable model. As seen in Table 4.8, the MTE values for the curl-free model were 1.3709 and 1.0936 for the Angle-shape and S-shape, respectively, compared to 0.9503 and 0.9471 for the Gaussian separable model, seen in Table 4.6. However, the obtained results remain reasonable, as evidenced by the close resemblance between the estimated and actual trajectories in Figure 4.17 and Figure 4.18.

Regarding computational time, the results show that the estimation of $\boldsymbol{\alpha}$ is faster for the curl-free feature map than for the Gaussian separable feature map. This can be observed in Table 4.5 and Table 4.7. One possible explanation for this finding is the difference in complexity between the two feature maps. As previously stated, the regression problem involving a curl-free feature map has a lower dimension compared to the Gaussian separable one. In particular, when estimating $\boldsymbol{\alpha}$ in the curl-free case, $\boldsymbol{\alpha}$ is in $\mathbb{R}^d$, where $d = 100$ for the Angle-shape and $d = 200$ for the S-shape, while in the Gaussian separable case, $\boldsymbol{\alpha}$ is in $\mathbb{R}^{dm}$, where $m = 2$. Hence, the difference in the number of samples for $\boldsymbol{\alpha}$, 100 versus 200 for the Angle-shape and 200 versus 400 for the S-shape, impact the computational time required. Therefore, the number of parameters increases for the Gaussian separable model, leading to more computations necessary, increasing the computation time, as expected.

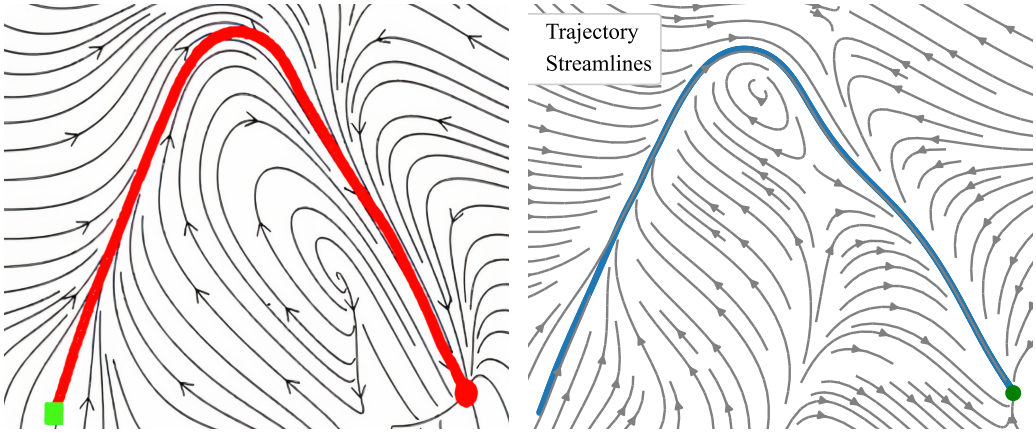
5.2.6. Comparison with External Results

To evaluate the accuracy of the streamlines in the vector fields learned in this thesis, the vector fields generated in [51] were used for comparison. These vector

fields share the same characteristics as the ones simulated in Figure 4.13 and Figure 4.16. Specifically, both sets of vector fields are approximated using RFF with either a Gaussian separable kernel or a curl-free kernel. Additionally, both approaches incorporate contraction constraints and a vanishing point.

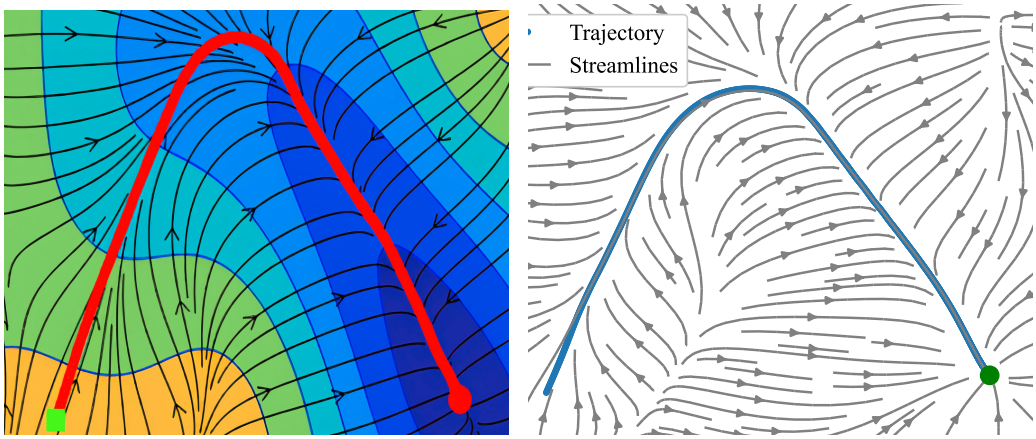
A comparison of the contracting vector fields generated in this thesis with those presented in [51] reveals strong similarities. In Figure 5.11, a comparison of the vector fields obtained when using a Gaussian separable feature map is illustrated. It can be observed that the estimated vector fields exhibit a close resemblance to one another, as the streamlines converge in similar regions.

Similarly, in Figure 5.12, where the feature map is curl-free, the streamlines have no rotations and contract in similar regions. Based on these observations, it is reasonable to assume that the results achieved in this thesis regarding learning stable vector fields from the LASA-benchmark dataset are satisfactory.



(a) (b)

Figure 5.11.: Vector Fields learned for Gaussian separable feature map (a) Result from Sindhwani et al. *Illustration* [51] (b) Result from thesis



(a) (b)

Figure 5.12.: Vector Fields learned for curl-free feature map (a) Result from Sindhwani et. al *Illustration* [51]. (b) Results from thesis

5.3. Comparing Results - Hamiltonian Dynamics

This section analyses and compares the results outlined in Section 4.2. The focus will be on comparing the symplectic feature map and Gaussian separable feature map.

5.3.1. Generating Trajectories with Numerical Methods

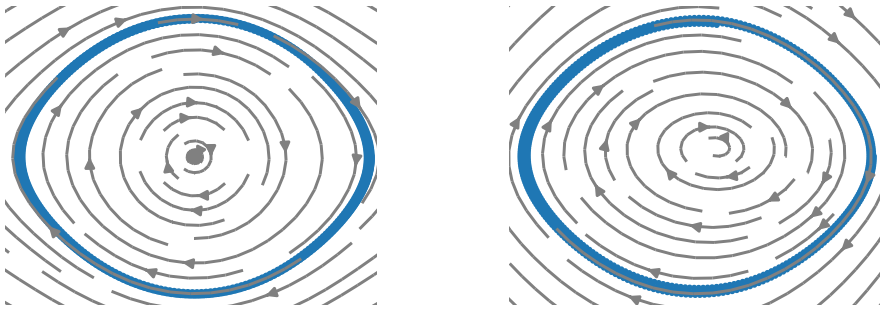
Looking at Figure 4.19, it becomes evident that the leapfrog and explicit Euler methods, generate different trajectories. While both trajectories have the same initial position $(2, 0)$, they exhibit substantial differences in their final positions. As explained in Section 2.7.6, using the symplectic approach to update the pendulum's position and momentum at half-time steps results in a trajectory that preserves the system's energy. This is visually demonstrated in Figure 4.19(a), where the pendulum sustains its oscillations along the same path, unlike the explicit Euler method shown in Figure 4.19(b), where the trajectory spirals outward. This indicates that energy is added to the trajectory, and it diverges to infinity. Consequently, energy is not conserved, and the trajectory is unstable. Based on this, it was determined that deploying the trajectory generated by the explicit Euler method for learning would not be beneficial. Instead, the leapfrog method demonstrated favorable outcomes, making it the preferred choice for generating trajectories for the purpose of learning Hamiltonian dynamics of a pendulum.

5.3.2. Estimating Hamiltonian Dynamics of a Pendulum

The results obtained for estimating the Hamiltonian dynamics of a pendulum, seen in Figure 4.21, clearly indicate that the symplectic feature map provides a more accurate depiction of the actual vector field than the Gaussian separable feature map. This outcome aligns with the theory presented in the preliminaries and the expected results.

Upon examining the learnt model using the symplectic feature map, it becomes apparent that the streamlines exhibit clear resemblances to the actual streamlines. This observation suggests that the feature map successfully maintains the system's geometric structure, specifically the symplectic structure of the Hamiltonian, leading to a more precise prediction.

On the contrary, the streamlines for the Gaussian separable model exhibit slight deviations in the center, suggesting less effective preservation of the symplectic structure. As a non-symplectic function, the Gaussian separable feature map was not expected to guarantee energy conservation, which resulted in inaccuracies in the simulation. Figure 5.13 shows the inner circle for the two learnt models, making it easier to observe the differences.

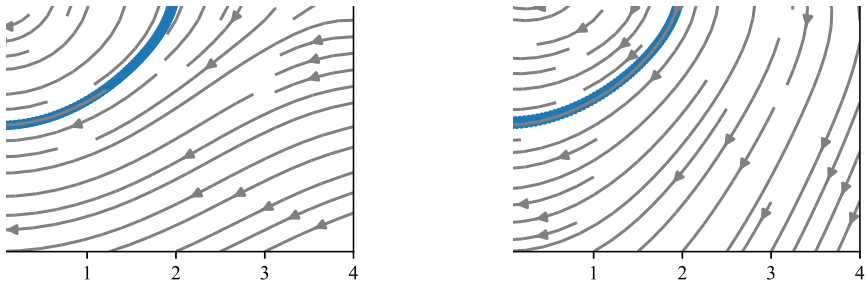


(a) (b)

Figure 5.13.: Estimated vector field zoomed in on the trajectory (a) Symplectic feature map (b) Gaussian separable feature map

Further examination of Figure 5.13 shows that the symplectic feature map outperforms the Gaussian separable feature map in accurately representing the streamlines surrounding the outer region of the estimated trajectory. Figure 5.14(b) displays more vertically oriented and downward-pointing streamlines, indicating a less precise capture of the underlying dynamics of the pendulum system. In contrast, Figure 5.14(a) exhibits horizontally-flowing streamlines that closely follow the trajectory's shape, and it closely resembles the actual streamlines, seen in Figure 4.20. As a result, these observations highlight the benefit of using a symplectic model for learning Hamiltonian systems.

This discrepancy aligns with expectations, as the symplectic kernel is designed to preserve symplectic properties. Conversely, the Gaussian separable feature map may face challenges in accurately capturing these characteristics. It is important to note that the Gaussian separable feature map's sub-optimal performance could be attributed to the choice of tuning parameters, such as the regularization parameter, for instance, λ. Selecting different parameter values could have potentially resulted in improved results.



(a) (b)

Figure 5.14.: Estimated vector field in the lower-right-hand side (a) Symplectic feature map (b) Gaussian separable feature map

Looking at Figure 4.22, it becomes evident that relying solely on visual analysis to distinguish between the estimated and actual trajectories is challenging. A close resemblance between the actual trajectory depicted in red and the estimated trajectory in blue makes it difficult to perceive their differences. This suggests that both the symplectic and Gaussian separable feature maps yield satisfactory results in trajectory estimation. However, to obtain a precise measure of the differences and evaluate the accuracy of the estimations, it was necessary to calculate numerical error estimates.

Error analysis

The error estimates presented in Figure 4.23 indicate that using the Gaussian separable feature map yields higher error values than the symplectic feature map. For instance, the highest error value for the momentum $\boldsymbol{p}$ for the former is approximately 0.5, while it is closer to 0.1 for the latter.

Additionally, the error comparison presented in Figure 4.24 further support the superior performance of the symplectic feature map compared to the Gaussian separable feature map. Both feature maps exhibit a similar initial trend of low error values. However, as time evolves, the Gaussian separable model consistently

demonstrates higher error rates. Particularly, beginning from 2 seconds, the error associated with the Gaussian separable model exhibits a consistent increase, resulting in subsequent errors and ultimately leading to an exponential growth of the overall error.

Moreover, considering the error analysis presented in Table 4.9, it can be concluded that the symplectic feature map outperforms the Gaussian separable feature map in accurately estimating the Hamiltonian dynamics of a pendulum. The MTE for the symplectic model is significantly better at 0.00414, compared to 0.01593 for the Gaussian separable model. These results clearly demonstrate that using a symplectic feature map provides more accurate estimations. However, it is important to mention that the Gaussian separable feature map still produces reasonable trajectory estimations, as indicated by its MTE of 0.01593, which is satisfactory.

5.3.3. Adding Noise to the Training Data

To evaluate the performance of the models, it was desirable to introduce noise into the training data. By doing this, improvements in results were observed when using the symplectic feature map.

Several observations can be made upon examining the impact of noise on the vector field estimation. First, it can be observed that the vector field in Figure 4.25 closely resembles the noise-free vector field depicted in Figure 4.20. This similarity is consistent with the expectations, as the small standard deviation in the incorporated noise resulted in minimal noticeable variation from the noise-free case.

When looking at the estimated streamlines and trajectories in the presence of noise, seen in Figure 4.26, both feature maps demonstrate the ability to estimate the actual trajectory with reasonable accuracy, similar to the noise-free scenario. However, there are noticeable differences in their performance regarding the estimated streamlines. The symplectic model demonstrates a close resemblance to the actual streamlines, accurately capturing the system's symplectic characteristics. In contrast, the Gaussian separable feature map exhibits deviations in the estimated streamlines and faces more challenges in accurately capturing the symplectic nature of the system.

To summarize the visual analysis of the results, it can be observed that the introduced noise had minimal impact on the vector field estimation. However, there were notable differences in the performance of the two feature maps. The symplectic model exhibited better alignment with the actual streamlines and successfully captured the symplectic characteristics of the pendulum system. It is important

to note that these results may have varied if different tuning parameters were chosen. Therefore, it is crucial to carefully tune the parameters to optimize the performance of the estimation methods.

Error analysis

Similar to the noise-free trajectory scenario, Figure 4.27 exhibited minimal visual difference between the actual and estimated trajectories. Consequently, the error between the actual and estimated trajectories was plotted to provide a more detailed analysis of the estimation accuracy. The error plots displayed in Figure 4.28 indicate that the inclusion of noise results in better outcomes for trajectory estimated with the symplectic feature map. Upon analyzing the error axis, notice that the inclusion of noise decreased the error interval from approximately $[-0.1, 0.1]$ to $[-0.06, 0.06]$ for the symplectic model. Conversely, the estimation using the Gaussian separable feature map results in poorer estimation with the interval increasing from approximately $[-0.5, 0.4]$ to $[-0.7, 0.5]$. These results suggest that the symplectic feature map is more prone to noisy data, as expected.

The numerical errors further support the assumption that the symplectic feature map outperforms the Gaussian separable feature map in the regression problem. Studying Figure 4.29, it can be seen that for the Gaussian separable model, the error of the model increase in an exponential trend, which can be observed from the light blue line. The learnt model experiences a cumulative error, which grows more extensive as it continues to learn. On the contrary, the total error for the symplectic model remains relatively constant over time, forming a horizontal line. These results indicate a stable error estimation that does not progressively increase with time. This observation is consistent with the theory that a symplectic kernel can preserve energy in a manner distinct from a standard kernel.

The models with added noise, Figure 4.29, compared to the noise-free models, Figure 4.24, reveals significant differences in the observed trends of the error plots. In the absence of noise, the two results exhibit a similar pattern, with the symplectic feature map consistently displaying lower errors than the Gaussian separable feature map. However, when noise is introduced, the total error associated with the Gaussian separable feature map maintains the same pattern over time as for the noise-free scenario, while the symplectic feature map demonstrates notable improvements. The total error range for the Gaussian separable feature map increases from $[0.0, 0.5]$ to $[0.0, 0.7]$, while for the symplectic feature map, it decreases from $[0.0, 0.1]$ to $[0.0, 0.75]$, as discussed earlier.

Furthermore, as seen in Table 4.9, the MTE for the symplectic model demonstrates significant enhancement, decreasing from 0.00414 to 0.00238, which is a clear improvement. The MTE for the Gaussian separable model on the other

hand, increased increased from 0.01593 to 0.02186, indicating a decrease in model performance.

Based on the mentioned observations, there is a clear indication that using RFF with a symplectic kernel is superior in estimating the Hamiltonian dynamics of a pendulum, regardless of the presence of noise. This outcome strongly aligns with the theory presented in the preliminaries. Notably, the symplectic feature map exhibits impressive capabilities in preserving the symplectic structure of a system, even when subjected to noise. In contrast, the RFF approximation using the Gaussian separable kernel struggles to capture the symplectic structure but demonstrates proficiency in accurately estimating the actual trajectory. These findings highlight the effectiveness of using RFF for kernel approximation and regression problems, as it yields satisfactory results.

5.4. Additional Considerations

Even though the results obtained were satisfactory, there are some other factors worth mentioning that may have influenced the outcomes.

5.4.1. The use of Finite Difference Method

As explained in the Method section, *numdifftools* was used to determine the gradient of feature map ψ in the regression problems solved to obtain the simulations in Figure 4.13 and Figure 4.16. This Python library uses finite difference method (FDM) to approximate the gradient of a given function, as explained in Section 3.1. While FDM can give satisfactory outcomes, it does not calculate the exact gradient, which may cause minor errors in the estimated gradient and ultimately result in errors in the estimated vector fields. Consequently, the error computed for the simulations could have been smaller if the exact gradient had been calculated. However, based on the results obtained, it is evident that the estimated gradient of ψ is quite close to the correct value. Hence, it is reasonable to believe that *numdifftools* is a valuable tool for estimating complex gradients as it significantly reduces computational time.

5.4.2. Limitations due to Generated RFF Parameters

Since both $\boldsymbol{w}$ and b in the feature maps were randomly generated, the estimation may fail to adequately represent the trajectories, potentially leading to low approximation accuracy. Additionally, if the RFF parameters are not optimized properly, the resulting feature map may fail to capture the key characteristics of the target function. Furthermore, the approximation accuracy may decrease if the

number of used random features is insufficient, whereas using too many features may result in overfitting. Because of this, generating reasonable RFF parameters and selecting an appropriate number of samples for the two shapes was important. As mentioned, this was done through a trial-and-error process until the results were satisfactory. However, even better results could probably be obtained with different RFF parameters. Therefore, it would be interesting to discover a method to generate optimal parameters.

5.4.3. Computational Efficiency with RFF

Based on the computation times presented in this thesis, it can be concluded that the use of RFF in regression problems is computationally efficient. The results showed that the computation times ranged from $0.8560s$ to $34.3604s$ for the Angle-shape, and $2.1191s$ to 66.1032 for the S-shape. These findings demonstrate that RFF remains efficient in modeling vector fields, regardless of whether additional characteristics are incorporated into the learning process. As mentioned in Section 2.1.6, the computational efficiency of RFF is due to its ability of dimensionality reduction and scalability benefits when handling large-scale tasks.

5.5. Future Work

Due to time and resource limitations, this thesis could not test the presented methods on more complex systems. However, as a promising area for future research, it would be interesting to apply the methods used in this thesis to real-world dynamic systems like cranes and quadcopters.

While this thesis focused on contraction constraints, it would be interesting to investigate the effectiveness of other types of constraints in regression problems using RFF. As discussed in Section 1.1, Ahmadi and Khadir [1] introduced a framework that incorporates side information to assist the learning process. Their study demonstrated that incorporating side information led to a closer alignment between the learned vector field and the true behavior. Although this thesis incorporated one of their side information, namely Hamiltonian systems, it would be interesting to explore other side information presented in their article and incorporate them into regression problems with RFF. Additionally, Singh et al. [52] proposed a stabilizability constraint that offers promising avenues for further exploration.

To further enhance the accuracy of estimation through kernel approximation using RFF, exploring a more efficient approach for determining optimal RFF parameters would be beneficial instead of relying on a trial-and-error method. For

example, one possible approach could be to investigate different nonlinear optimization techniques. Moreover, the article by Avron and Indyk [7] demonstrates a promising way to determine the optimal number of samples, thus improving the overall performance of RFF-based methods.

The findings from both the thesis and the specialization project [28] demonstrates that using RFF to approximate kernels in a RKHS yields accurate outcomes while reducing the complexity of the problem. Moving forward, it would be interesting to compare the performance of RFF-based methods with other state-of-the-art approaches, such as neural networks, in terms of accuracy and computation time.

Chapter 6.

Conclusion

This thesis investigated using random Fourier features (RFF) to approximate different kernels in a reproducing kernel Hilbert space (RKHS) to solve various regression problems. The primary focus has been on estimating vector fields with different characteristics, to see how these can improve the precision of the learning process. Simulations were conducted on both a real-world dataset and a self-generated one, and the results indicate that using RFF provided accurate estimates and was computationally efficient.

One of the regression problems solved in this thesis involved a vector field with no specified constraints. This was modeled twice, where one was trained on a single trajectory, by taking the mean of the four demonstrations, and the second was trained on four. It was concluded that training on a single trajectory provided the most accurate results. Another regression problem involved a vector field with contraction constraints, leading to a more complex problem and increased computation time. However, the learnt model was adequate and provided accurate results. It was also observed that the streamlines in the contracting vector field were more robust than the ones without contraction, and the precision in the learning process was increased.

The thesis also explored regression problems where it was required to have the vector field vanish at specific points. In order to do this, two models were trained with a Gaussian separable feature map. The first model was trained with the incorporation of contraction constraints in the regression problem, while the second model was without. The results obtained for both models aligned with the theoretical expectations. Additionally, a model was trained on a curl-free feature map enforcing both vanishing point and contraction during the learning process, yielding satisfactory results.

Furthermore, two numerical methods were used to generate trajectories for the Hamiltonian dynamics of a pendulum, namely the explicit Euler and the leapfrog

method. The leapfrog method gave the best results, and its generated trajectory was then used to solve different regression problems. Following this, two different kernels, specifically the Gaussian separable and the symplectic kernel, were approximated with RFF to estimate the dynamics of a pendulum, resulting in their corresponding feature maps. The trajectory was generated twice for each feature map, one with the inclusion of noise and one without. In both scenarios, the symplectic feature map outperformed the Gaussian separable feature map, showcasing its capability to capture the symplectic dynamics of the system. Conversely, the streamlines estimated using the Gaussian separable feature map exhibited less satisfactory performance. Nevertheless, it is essential to note that despite its limitations in estimating streamlines for symplectic systems, the Gaussian separable feature map still provided satisfactory result when estimating the trajectory.

Overall, it can be concluded that using RFF to estimate different kernels in a RKHS provides reliable results and can estimate various vector fields with different characteristics. However, the selection of a suitable kernel for the problem and generating satisfactory RFF parameters is crucial for achieving optimal results. In the future, it would be interesting to compare the performance of RFF-based methods to other state-of-the-art methods for vector field approximation, such as neural networks, in terms of accuracy, computation time, and errors.

References

[1] A. A. Ahmadi and B. El Khadir. "Learning dynamical systems with side information". In: *Learning for Dynamics and Control*. PMLR. 2022, pp. 718–727.

[2] M. A. Alvarez, L. Rosasco, N. D. Lawrence, et al. "Kernels for vector-valued functions: A review". In: *Foundations and Trends® in Machine Learning* 4.3 (2012), pp. 195–266.

[3] MOSEK ApS. "Mosek optimizer API for Python". In: *Version* 9.17 (2022), pp. 6–4.

[4] A. Apsemidis, S. Psarakis, and J. M. Moguerza. "A review of machine learning kernel methods in statistical process monitoring". In: *Computers & Industrial Engineering* 142 (2020), p. 106376.

[5] V. I. Arnold. *Mathematical methods of Classical Mechanics*. Springer-Verlag, 1989.

[6] N. Aronszajn. "Theory of reproducing kernels". In: *Transactions of the American mathematical society* 68.3 (1950), pp. 337–404.

[7] H. Avron, M. Kapralov, C. Musco, A. Musco C. Velingker, and A. Zandieh. "Fourier features for kernel ridge regression: Approximation bounds and statistical guarantees." In: *International conference on machine learning* (2017), pp. 253–262.

[8] A. Beléndez, C. Pascual, D. I. Méndez, T. Beléndez, and C. Neipp. "Exact solution for the nonlinear pendulum". In: *Revista brasileira de ensino de fısica* 29 (2007), pp. 645–648.

[9] A. Berlinet and C. Thomas-Afnan. *Reproducing Kernel Hilbert Spaces in Probability and Statistics*. Springer Link, 2014.

[10] R. Bhatia and C. Davis. "A Cauchy-Schwarz inequality for operators with applications". In: *Linear algebra and its applications* 223 (1995), pp. 119–129.

[11] N. M. Boffi, S. Tu, and J. Slotine. "Nonparametric adaptive control and prediction: theory and randomized algorithms". In: *The Journal of Machine Learning Research* 23 (2022).

[12] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004, pp. 669–672.

[13] R. Brault, M. Heinonen, and F. d'Alché-Buc. "Random Fourier Features For Operator-Valued Kernels". In: *Asian Conference on Machine Learning* 63 (2016), pp. 110–125.

[14] P. A. Brodtkorb and J. D'Errico. "Numdifftools documentation". In: (2018).

[15] J. Brownlee. *Better deep learning: train faster, reduce overfitting, and make better predictions*. Machine Learning Mastery, 2018.

[16] S. L. Brunton, J. L. Proctor, and J. N. Kutz. "International Conference on Machine Learning". In: *Proceedings of the National Academy of Sciences* 113.15 (2016).

[17] Z. Chen, J. Zhang, M. Arjovsky, and L. Bottou. "Symplectic Recurrent Neural Networks". In: *CoRR* abs/1909.13334 (2019).

[18] *Downloads - Stable Estimator of Dynamical Systems (SEDS)*. https://cs.stanford.edu/people/khansari/download.html. Accessed: 2023-04-24.

[19] O. Egeland. "Optimization". NTNU, unpublished. 2023.

[20] O. Egeland and J. T. Gravdahl. *Modeling and Simulation for Automatic Control*. Marine Cybernetics AS, 2002.

[21] T. Evgeniou, M. Pontil, and T. Poggio. "Regularization networks and support vector machines". In: *Advances in computational mathematics* 13.1 (2000), pp. 1–50.

[22] J. Fan and R. Li. "Statistical challenges with high dimensionality: Feature selection in knowledge discovery". In: *arXiv preprint math/0602133* (2006).

[23] R. Haag, D. Kastler, and E. B. Trych-Pohlmeyer. "Stability and equilibrium states". In: *Communications in Mathematical Physics* 38 (1974), pp. 173–193.

[24] E. Hairer, C. Lubich, and G. Wanner. "Geometric numerical integration illustrated by the Störmer–Verlet method". In: *Acta numerica* 12 (2003), pp. 399–450.

[25] E. Hairer, G. Wanner, and C. Lubich. *Geometric numerical integration: Structure preserving algorithms for ordinary differential equations*. Springer, 2006.

[26] F. E. Harrell et al. *Regression modeling strategies: with applications to linear models, logistic regression, and survival analysis*. Vol. 608. Springer, 2001.

[27] P. Huang, H. Avron, Sainath T.N., V. Sindhwani, and B Ramabhadran. "Kernel Methods Match Deep Neural Networks in Timit". In: *IEEE International Conference on Acoustics, Speech, and Signal Processing* (2014), pp. 205–209.

[28] V. Kenworthy and H. Phan. “Learning-Based Adaptive Control Based on Reinforcement Learning”. NTNU, unpublished. 2022.

[29] V. T. Kenworthy and H. T. Phan. *GitHub Repository*. https://github.com/helenatp/tpk4960. 2023.

[30] S. M. Khansari-Zadeh and A. Billard. “Learning stable nonlinear dynamical systems with gaussian mixture models”. In: *IEEE Transactions on Robotics* 27.5 (2011), pp. 943–957.

[31] S. M. Khansari-Zadeh and O. Khatib. “Learning Potential Functions from Human Demonstrations with Encapsulated Dynamic and Compliant Behaviors”. In: *Autonomous Robots* 41.1 (2017), pp. 45–69.

[32] J.E. Kirkwood. *Mathematical Physics with Partial Differential Equations*. Academic Press, 2013, pp. 327–350.

[33] Z. Li. “Sharp analysis of random fourier features in classification”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* (2021).

[34] F. Liu, X. Huang, Y. Chen, and J. AK. Suykens. “Random features for kernel approximation: A survey on algorithms, theory, and beyond”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.10 (2021), pp. 7128–7148.

[35] W. Lohmiller and J-J. E. Slotine. “On contraction analysis for non-linear systems”. In: *Automatica* 34.6 (1998), pp. 683–696.

[36] I. Macêdo and R. Castro. *Learning divergence-free and curl-free vector fields with matrix-valued kernels*. IMPA, 2010.

[37] R. Meyer, M. Weichselbaum, and A. W. Hauser. “Machine learning approaches toward orbital-free density functional theory: Simultaneous training on the kinetic energy density functional and its functional derivative”. In: *Journal of chemical theory and computation* 16.9 (2020), pp. 5685–5694.

[38] C. A. Micchelli and M. Pontil. “On learning vector-valued functions”. In: *Neural computation* 17.1 (2005), pp. 177–204.

[39] H. Q Minh. “Operator-Valued Bochner Theorem, Fourier Feature Maps for Operator-Valued Kernels, and Vector-Valued Learning”. In: *Journal of Machine Learning Research* abs/1608.05639 (2016).

[40] H. Q. Minh, P. Niyogi, and Y. Yao. “Mercer’s theorem, feature maps, and smoothing”. In: *International Conference on Computational Learning Theory*. Springer. 2006, pp. 154–168.

[41] H.Q. Minh and V. Sindhwani. “Vector-valued Manifold Regularization.” In: *ICML*. 2011.

[42] D. Morin. “Chapter 15 The Hamiltonian method”. In: (2008). URL: https://scholar.harvard.edu/files/david-morin/files/cmchap15.pdf.

[43] R. M. Neal et al. “MCMC using Hamiltonian dynamics”. In: *Handbook of markov chain monte carlo* 2.11 (2011), p. 2.

[44] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Vol. 153. Springer Link, 2003.

[45] *Python decorator to measure execution time*. https://dev.to/kcdchennai/python-decorator-to-measure-execution-time-54hk. Accessed: 2022-11-10.

[46] A. Rahimi and B. Recht. “Random Features for Large-Scale Kernel Machines”. In: *Advances in Neural Information Processing Systems*. Ed. by J. Platt, D. Koller, Y. Singer, and S. Roweis. Vol. 20. Curran Associates, Inc., 2007.

[47] G. Sagnol and M. Stahlberg. “PICOS: A Python interface to conic optimization solvers”. In: *Journal of Open Source Software* 7.70 (Feb. 2022), p. 3915. ISSN: 2475-9066. DOI: 10.21105/joss.03915.

[48] B. Scholkopf and A.J. Smola. *Learning with Kernels*. The MIT Press, 2002.

[49] B. Schölkopf, R. Herbrich, and A. J. Smola. “A generalized representer theorem”. In: *International conference on computational learning theory*. Springer. 2001, pp. 416–426.

[50] *scipy.integrate.solve_ivp*. https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html. Accessed: 2023-03-16.

[51] V. Sindhwani, S. Tu, and M. Khansari. “Learning Contracting Vector Fields For Stable Imitation Learning”. In: (2018).

[52] S. Singh, S. Richards, V. Sindhwani, J-J. Slotine, and M. Pavone. “Learning stabilizable nonlinear dynamics with contraction-based regularization”. In: *The International Journal of Robotics Research* 40.10-11 (2021).

[53] I. Steinwart and C. Scovel. “Fast rates for support vector machines using Gaussian kernels”. In: *The Annals of Statistics* 35.2 (2007), pp. 575–607.

[54] J. Suzuki. *Kernel Methods for Machine Learning with Math and Python: 100 Exercises for Building Logic*. Springer Nature, 2022.

[55] Y. Tanaka, T. Iwata, and N. Ueda. “Symplectic Spectrum Gaussian Processes: Learning Hamiltonians from Noisy and Sparse Data”. In: *Conference on Neural Information Processing Systems* (2022).

[56] A. Vabalas, E. Gowen, E. Poliakoff, and A. J. Casson. “Machine learning algorithm validation with a limited sample size”. In: *PloS one* 14.11 (2019), e0224365.

[57] J-P Vert. *Aronszajn’s theorem*. https://members.cbio.mines-paristech.fr/~jvert/svn/kernelcourse/notes/aronszajn.pdf.

[58] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python". In: *Nature methods* 17.3 (2020), pp. 261–272.

[59] K. Vu, J. C. Snyder, L. Li, M. Rupp, B. F. Chen, T. Khelif, K-R Müller, and K. Burke. "Understanding kernel ridge regression: Common behaviors from simple functions to density functionals". In: *International Journal of Quantum Chemistry* 115.16 (2015), pp. 1115–1128.

[60] M. Welling. "Kernel Ridge Regression". In: *Max Welling's Classnotes in Machine Learning* (2013).

[61] J. Yang, V. Sindhwani, H. Avron, and M. Mahoney. "Quasi-Monte Carlo feature maps for shift-invariant kernels". In: *International Conference on Machine Learning*. PMLR. 2016, pp. 485–493.

[62] H. Zhen, L. Ming, and C. Zhang. "Dependent Online Kernel Learning With Constant Number of Random Fourier Features". In: *IEEE Transactions on Neural Networks and Learning Systems* 26.10 (2015).

[63] D-X. Zhou. "Derivative reproducing properties for kernel methods in learning theory". In: *Journal of computational and Applied Mathematics* 220.1-2 (2008), pp. 456–463.

Appendix A.

Code

The source code for the results conducted in this thesis can be found at https://github.com/helenatp/tpk4960.

A.1. LASA

All the code related to the LASA handwriting dataset simulations can be accessed in the directory named *src/lasa*. Each file within this directory has a markdown section labeled *Read data* at the beginning. This is where the desired shape can be selected. If you want to run the *Angle* dataset, comment out the corresponding code line, and do the same for the *Sshape* dataset if desired.

A.2. Hamiltonian Dynamics

The file named *hamilton_pendulum* has a markdown section at the beginning labeled *Noise*. Under this markdown, you will find a variable named *noise*. This variable is of Boolean type, and its value determines whether noise is incorporated into the trajectory. To execute the file with noise included, set the variable to *True*. Conversely, set the value to *False* if you want to run it without noise.

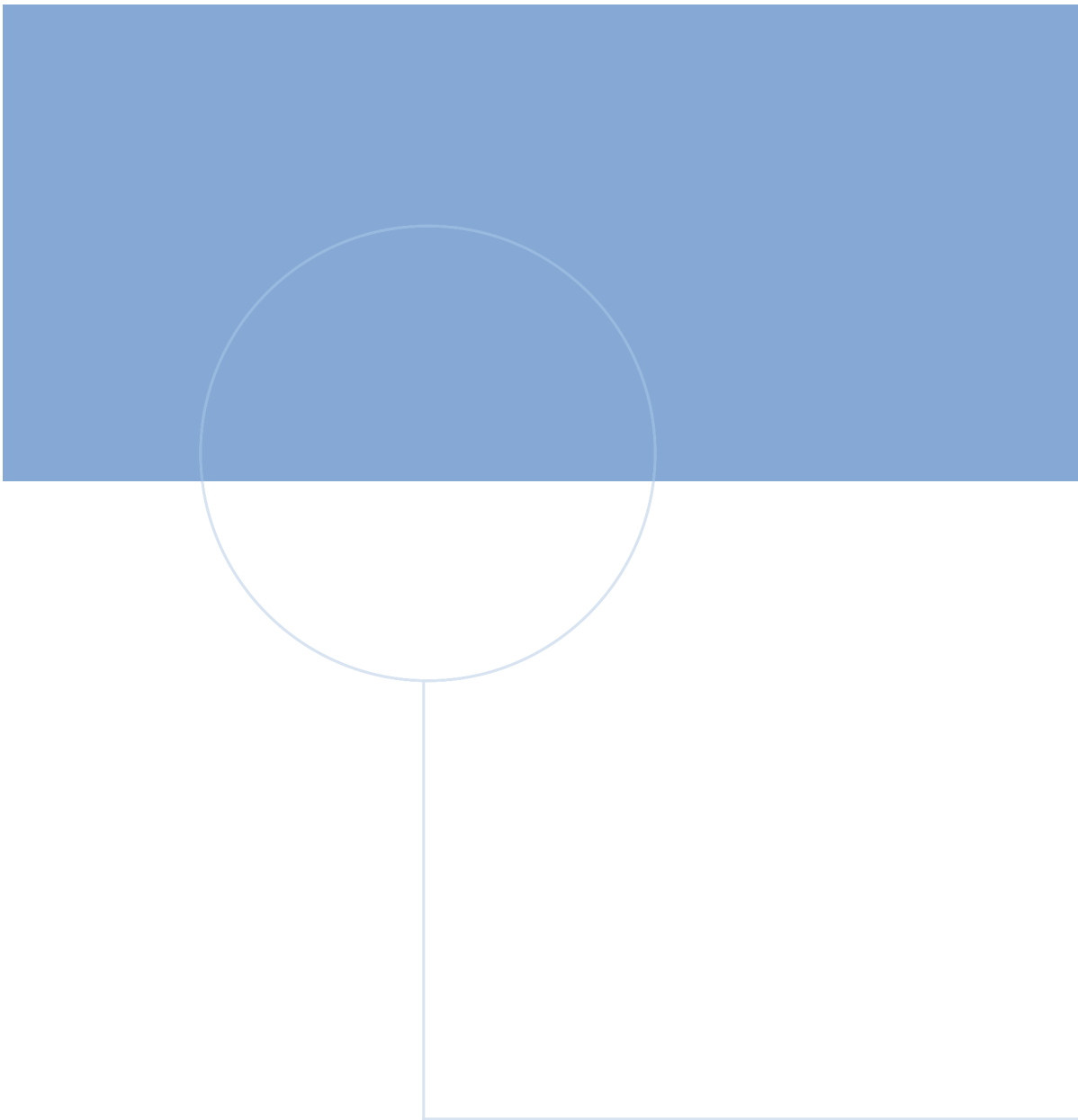
A.3. Solving Regression Problems using PICOS and MOSEK

This section demonstrates how PICOS and MOSEK can be implemented to solve a regression problem. The code below shows how Algorithm 2 is implemented using these optimization tools. As mentioned in Section 3.1 PICOS is used to

create the optimization problem, while MOSEK is used to solve it. In the code, *pc* refers to PICOS.

To solve the optimization problem using PICOS, a systematic approach was followed. An empty problem object was created to serve as the container for the optimization problem, seen in line 4. Constant parameters, representing fixed values in the problem, were defined as seen in lines 5-12. Following this, the optimization variable for the problem was defined, which in this case is $\boldsymbol{\alpha}$, seen in line 13. The contraction constraints were then defined and added to the problem, as seen in lines 15-24. Then, the objective function, representing the quantity to be minimized, was defined and set in the problem object, as seen in lines 25 and 26. Finally, at line 27, the problem was solved with MOSEK defined as the desired solver.

```
1     def alpha_approx_with_constraint(x, y, w, b, dim, d, lam, N, mu,
      constraint_points):
2         phi_ = phi(x, w, b, N, d, dim)
3         mu = mu * np.eye(dim)
4         problem = pc.Problem()
5         phi_param = pc.Constant('phi_', phi_)
6         lam_param = pc.Constant('lam', lam)
7         dim_param = pc.Constant('dim',dim)
8         d_param = pc.Constant('d', d)
9         mu_param = pc.Constant('mu', mu)
10        constraint_points_param = pc.Constant('constraint_points',
      constraint_points)
11        y_reshaped = np.array(np.ravel([y[0], y[1]], 'F'))
12        y_reshaped_param = pc.Constant('y_reshaped', y_reshaped)
13        alpha_var = pc.RealVariable('alpha_var', (d_param*dim_param, 1))
14        # Creating constraints
15        for i in range(constraint_points_param):
16            constraint_index = i*np.int64(np.floor(len(x[0])/
      constraint_points))
17            x_i = x[:, constraint_index]
18            gradient = np.zeros(dim)
19            for j in range(d):
20                index = 2*j
21                gradient_of_psi_param = pc.Constant('gradient_of_psi',
      gradient_of_psi(x_i, w[:, j], b[:, j]))
22                jacobi = alpha_var[index:index+2] * gradient_of_psi_param.T
23                gradient = gradient + 0.5 * (jacobi + jacobi.T)
24            problem.add_constraint(gradient << mu_param)
25        obj = ((phi_param * alpha_var) - y_reshaped_param).T * ((phi_param
      * alpha_var) - y_reshaped_param) + lam_param*(alpha_var.T * alpha_var)
26        problem.set_objective('min', obj)
27        problem.solve(solver='mosek')
28        alpha_var = alpha_var.np.reshape((-1,))
29        return alpha_var
```



NTNU

Norwegian University of
Science and Technology