Imperial College London

MSc STATISTICS

Imperial College London

Kernel-Based Inference Methods for Ordinary Differential Equations

Author:
Yanni Papandreou
(CID: 00955392)

Supervisor:
Dr. Andrew Duncan

A thesis submitted for the degree of

MSc Statistics

October 13, 2019

Declaration

The work contained in this thesis is my own work unless otherwise stated.

Signature:

Acknowledgements

I would like to thank Dr. Andrew Duncan for his guidance and constructive feedback during the completion of this work.

Abstract

The aim of this project is to consider the use of Maximum Mean Discrepancy (**MMD**) as a tool for inferring the true parameter value which generates data from a particular class of generative models based on Ordinary Differential Equations. We will have two main goals: the first is to demonstrate how **MMD** can be used as an appropriate cost function which will then be minimized via various gradient descent approaches in order to infer the true parameter value. The second goal will be to consider the Adjoint Method as a means of speeding up the estimation procedure in cases where the parameter space is very high dimensional. Numerical experiments will be used for the illustration of both objectives. Jupyter notebooks containing the code for these experiments can be found at the following GitHub repository: `https://github.com/nepoComplex/Kernel-based-inference-ODEs.git`.

List of Figures

Contents

Chapter 1

Introduction

1.1 Motivation

In this project we will be concerned with investigating the use of Maximum Mean Discrepancy (**MMD**) as a means of performing inference of generative statistical models based on Ordinary Differential Equations (**ODEs**). In many cases, such models are intractable, in the sense that the likelihood will not have a closed, computable form. A generative model is essentially a parametric family of probability measures from which we can obtain samples for any choice of parameter. To be more specific, given a Borel probability space $(\mathcal{U}, \mathcal{F}, \mathbb{U})$, [1, p. 2] calls a generative model any probability measure $\mathbb{P}_{\boldsymbol{\theta}}$ which is the pushforward $G_{\boldsymbol{\theta}}^{\#}\mathbb{U}$ of the probability measure $\mathbb{U}$ with respect to a measurable parametric map $G_{\boldsymbol{\theta}} : \mathcal{U} \to \mathcal{X}$ which is called the *generator*[1]. In order to generate n independent realizations from the model we first sample n i.i.d. realizations $\{u_i\}_{i=1}^n \overset{\text{i.i.d.}}{\sim} \mathbb{U}$ and then map these realizations using the generator to obtain n independent realizations from $G_{\boldsymbol{\theta}}^{\#}\mathbb{U}$, $\{y_i = G_{\boldsymbol{\theta}}(u_i)\}_{i=1}^n$. It can be seen that generating samples from these models can be relatively straightforward, while computing the likelihood is not necessarily possible, as an associated positive density may not exist or may not be computable. In these circumstances alternatives to maximum likelihood estimation are required for inference.

In this work we will consider complex generative models which are based on **ODEs**. To be more specific we consider the following initial value problems (**IVPs**):

$$\frac{d\mathbf{x}^{(i)}}{dt} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}(t); \boldsymbol{\xi}^{(i)}), \quad \mathbf{x}^{(i)}(0) = \boldsymbol{\epsilon}^{(i)}, \quad i = 1, 2, \ldots, n \tag{1.1}$$

where the function on the RHS of the **ODEs** depends on some vector of parameters $\boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^p$ whose value we would like to infer and also might depend on some noise $\{\boldsymbol{\xi}^{(i)}\}_{i=}^m \overset{i.i.d.}{\sim} \mathcal{D}_1(\boldsymbol{\theta})$ where $\mathcal{D}_1(\boldsymbol{\theta})$ is some distribution which might also depend on the unknown parameters $\boldsymbol{\theta}$. The $\boldsymbol{\epsilon}^{(i)}$'s are the initial conditions for the **ODEs** which can be random, in general we have $\{\boldsymbol{\epsilon}^{(i)}\}_{i=1}^m \overset{i.i.d.}{\sim} \mathcal{D}_2(\theta)$ where again $\mathcal{D}_2(\boldsymbol{\theta})$ is some distribution which might depend on the unknown parameters. We shall be assuming in this project that the noise terms $\boldsymbol{\xi}^{(i)}$ and $\boldsymbol{\epsilon}^{(i)}$ are finite dimensional; in particular $\boldsymbol{\xi}^{(i)} \in \mathbb{R}^s$ and $\boldsymbol{\epsilon}^{(i)} \in \mathbb{R}^d$ (*note*: this is the same dimension as the dimension of $\mathbf{x}^{(i)}$).

Remark. If instead we have that the noise term $\boldsymbol{\xi}^{(i)}$ is infinite dimensional (e.g. if it is a path valued random variable, like Brownian motion) then (1.1) includes Stochastic Differential Equations (**SDEs**).

[1] This should not be confused with the concept of an infinitesimal generator for Markov processes.

We assume that the function $f_{\boldsymbol{\theta}}$ is sufficiently differentiable w.r.t. all its arguments and that the distributions $\mathcal{D}_1(\boldsymbol{\theta})$, $\mathcal{D}_2(\boldsymbol{\theta})$ can be sampled from given any particular choice of parameter vector $\boldsymbol{\theta}$. Our generative model is now as follows: we take as our u_i the pair $(\boldsymbol{\epsilon}^{(i)}, \boldsymbol{\xi}^{(i)})$. The generator in this framework, $G_{\boldsymbol{\theta}}$, is formally the map which takes the pair $u_i = (\boldsymbol{\epsilon}^{(i)}, \boldsymbol{\xi}^{(i)})$ and 'solves' the **IVPs** (1.1) to yield potentially noisy observations of solutions of (1.1): $y_i = \{\mathbf{h}(\mathbf{x}^{(i)}(t)) + \mathbf{e}^{(i)}\}_{t\in T}$, where $\mathbf{h}$ is some function determining the structure of the observation process, $\mathbf{e}^{(i)}$ is some white-noise process, and T is some set of times on which we desire the trajectory.[2]

Remark. In the case of noisy measurements of a deterministic **ODE** mentioned in the preceding remark, where the $\{\mathbf{e}_r^{(i)}\}$ are i.i.d. $\mathcal{N}_d(\mathbf{0}, \sigma^2 I)$, the likelihood function for the unknown parameter $\boldsymbol{\theta}$ can be explicitly written down. This can be done as follows: first we have, using properties of the multivariate normal distribution, that the observations are distributed as follows:

$$\mathbf{y}_r^{(i)} = \mathbf{x}^{(i)}(t_r; \boldsymbol{\theta}) + \mathbf{e}_r^{(i)} \sim \mathcal{N}_d(\mathbf{x}^{(i)}(t_r; \boldsymbol{\theta}), \sigma^2 I)$$

Since the observations for different samples (i.e. different i) and at different times (i.e. different r) are all independent of each other the likelihood for the unknown parameters can be written as:

$$l(\boldsymbol{\theta}) = \prod_i \prod_r \left[\frac{\exp\left(-\frac{1}{2\sigma^2} (\mathbf{y}_r^{(i)} - \mathbf{x}^{(i)}(t_r; \boldsymbol{\theta}))^T (\mathbf{y}_r^{(i)} - \mathbf{x}^{(i)}(t_r; \boldsymbol{\theta})) \right)}{(2\pi)^{d/2} \sigma^d} \right]$$

Maximum likelihood estimation can then be used to estimate the true parameters. For more general **ODE** models this is no longer possible and this leads us to consider the methods presented in this project.

Our aim in this project is to consider the use of Maximum Mean Discrepancy as a tool for inferring the true parameter value which generates data from the above generative model. We will have two main goals: the first is to demonstrate how **MMD** can be used as an appropriate cost function which will then be minimized using gradient descent approaches in order to infer the true parameter value. Our second goal will be to consider the use of the so called Adjoint Method to speed up our estimation algorithm in cases where our parameter space Θ is very high dimensional (i.e. for large p). Numerical experiments will be used for illustration in both cases.

We will first provide a brief overview of various methods commonly used for parameter inference in **ODEs**/dynamical systems. The estimation of parameters in dynamical systems is important in many fields of science and engineering because it is often the case that physical, chemical or biological processes can be accurately represented by systems of **ODEs** with unknown parameters [2, p. 698].

1.2 Brief Overview of Common Methods for Inference in Dynamical Systems

Often in the literature on parameter inference in dynamical systems the scenario of noisy observations of a deterministic **ODE** at discrete times is considered. In this framework

[2] If the set T is countable then y_i is considered a vector of observations of the trajectories of the **IVPs** (1.1) on some time-grid; however T need not be countable, but this case is less computationally relevant.

the problem essentially becomes a question of regression; the goal is to use these noisy observations to infer the true parameter values. The most commonly used method is based on finding the choice of parameter vector which minimizes the least-squares cost function between the observations and the predictions of the model (see for instance [3] or [4]). This minimization often is performed using a gradient descent style approach similar to the methods we will be considering in this project.

A popular alternative approach, proposed by Varah (see [5]), instead first performs a non-parametric estimation of the trajectory (often via a spline approach). This estimated functional form of the trajectory is then used to compute estimated derivative values. Minimization of the least-squares cost between these estimated derivative values together with the derivative values computed using the observed data and RHS of the **ODEs** is then undertaken in an attempt to infer the true parameters. Building on this work Poyton et al. [2, p. 706] considered an iterative algorithm which essentially iterates between carrying out the above two steps and then using a model-based roughness penalty to improve the fitted splines. In [6] a similar approach involving a pseudo-least squares estimator is considered.

Other methods have been investigated in the literature. We briefly cite these now. In [7] and [8] Markov Chain Maximum Likelihood methods and Expectation Maximization approaches are discussed for the parameter estimation problem (*note*: [8] deals with **SDE** models). In [9] a cross-entropy approach is investigated. A multi-shooting method is discussed in [10]. Bayesian approaches to the parameter estimation of dynamical systems have also been considered. For instance [11] considers the use of Gaussian Process (**GP**) regression to perform Bayesian inference of parameters in nonlinear ordinary and delay differential equations. Gaussian Processes have also been used in [12] where the authors propose to learn a non-parametric **ODE** model using **GP** vector fields.

In this project we will instead be considering systems where we only have noisy initial conditions and our observations are noiseless (i.e. we consider dropping $\boldsymbol{\xi}^{(i)}, \mathbf{e}^{(i)}$ and keeping the $\boldsymbol{\epsilon}^{(i)}$ in (1.1)). We are thus in a situation where essentially a probability distribution is fed into a differential equation, is transformed, and a potentially different distribution is obtained as the output. Instead of using the least-squares discrepancy between the observed data and the predicted data from the model we will instead be using Maximum Mean Discrepancy, a metric on the space of probability measures, as a discrepancy between the observed empirical output distribution and the generated empirical output distribution. Before explaining our approach to the parameter estimation problem we will first go over relevant background theory.

1.3 Background Theory

In this section we provide a brief overview of the relevant theory which we will be using. We will proceed by first going over the definition of a reproducing kernel Hilbert space (**RKHS**) and then listing the relevant major results. We will then provide a brief overview of the kernel mean embedding of probability distributions into an **RKHS** before finally moving on to a discussion of Maximum Mean Discrepancy.

1.3.1 Reproducing Kernel Hilbert Spaces

We now describe one possible definition of a reproducing kernel Hilbert space. We will follow the exposition given in [13]. The reader is invited to consult this reference for further details on **RKHS**'s. A **RKHS** is a Hilbert space, $\mathcal{H}$, of functions from some non-empty set $\mathcal{X}$ to the reals[3], i.e. $\mathcal{H} \subseteq \mathbb{R}^{\mathcal{X}}$ which has some additional properties which means that it is relatively well-behaved; in particular if two functions $f, g \in \mathcal{H}$ are close with respect to the norm of $\mathcal{H}$ then $f(x)$ and $g(x)$ are close in $\mathbb{R}$, for all $x \in \mathcal{X}$, with respect to the standard norm in $\mathbb{R}$. We will denote the inner product in $\mathcal{H}$ by $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ and the corresponding norm as $\| \cdot \|_{\mathcal{H}}$. In order to properly define a **RKHS** we must first define what an evaluation functional is. Following the definition in [13, p.7] we can define an evaluation functional as follows:

Definition 1.3.1. (Evaluation functional) Let $\mathcal{H}$ be a Hilbert space of real-valued functions, defined on a non-empty set $\mathcal{X}$. For a fixed $x \in \mathcal{X}$, the map $\delta_x : \mathcal{H} \to \mathbb{R}$ which maps $f \mapsto f(x)$ is called the (Dirac) evaluation functional at x.

Having defined what an evaluation functional is we are now in the position to define a **RKHS**. We follow the definition in [13, p.7]:

Definition 1.3.2. (Reproducing kernel Hilbert space) A Hilbert space $\mathcal{H}$ of real-valued function, defined on a non-empty set $\mathcal{X}$ is said to be a Reproducing Kernel Hilbert Space (**RKHS**) if δ_x is continuous for all $x \in \mathcal{X}$.

This definition has many useful consequences. Before moving on to discuss these, we first present an equivalent characterisation of continuity for linear operators between normed linear spaces given in [13, p.6] which will be useful to us:

Theorem 1.3.1. Let $(\mathcal{F}, \| \cdot \|_{\mathcal{F}})$ and $(\mathcal{G}, \| \cdot \|_{\mathcal{G}})$ be normed linear spaces. If $L : \mathcal{F} \to \mathcal{G}$ is a linear operator, then the following three conditions are equivalent:

1. L is a bounded operator [4].
2. L is continuous on $\mathcal{F}$.
3. L is continuous at one point of $\mathcal{F}$.

Since δ_x is clearly a linear operator from $\mathcal{H} \to \mathbb{R}$ we can easily see that definition 1.3.2 is equivalent to the requirement that δ_x is a bounded operator for all $x \in \mathcal{X}$.

From the definition 1.3.2 of a **RKHS** together with Theorem 1.3.1 we get the following very simple corollary ([13, see p.7]):

Corollary 1.3.2.1. (*Norm convergence in $\mathcal{H}$ implies pointwise convergence*). If a sequence of functions $(f_n)_{n \in \mathbb{N}}$ in $\mathcal{H}$ converges in **RKHS** norm to another function $f \in \mathcal{H}$ then the sequence converges pointwise to f, i.e. if $\lim_{n \to \infty} \|f_n - f\|_{\mathcal{H}} = 0$ then $\lim_{n \to \infty} f_n(x) = f(x)$ for all $x \in \mathcal{X}$.

[3] The theory can be extended to complex valued functions but we will be working only with real-valued functions here.

[4] A linear operator is said to be bounded if it has a finite operator norm, i.e. $\|L\| := \sup_{f \in \mathcal{F}} \frac{\|Lf\|_{\mathcal{G}}}{\|f\|_{\mathcal{F}}} < \infty$.

Proof. We bound $|f_n(x) - f(x)|$ as follows:

$$\begin{aligned} |f_n(x) - f(x)| &= |\delta_x(f_n) - \delta_x(f)| \\ &= |\delta_x(f_n - f)| \\ &\leq \|\delta_x\| \|f_n - f\|_{\mathcal{H}} \end{aligned}$$

Now $\|\delta_x\| < \infty$ as $\mathcal{H}$ is an **RKHS** and so δ_x is continuous and hence a bounded operator by Theorem 1.3.1. The result then follows using the assumption that $\|f_n - f\|_{\mathcal{H}} \to 0$ as $n \to \infty$. ■

So far we have managed to define what a **RKHS** is without mentioning anything about kernels. We will now aim to explain the role of kernels for a **RKHS**. We begin by giving the definition of a reproducing kernel which we take from [13, p.8]:

Definition 1.3.3. (Reproducing kernel) Let $\mathcal{H}$ be a Hilbert space of real-valued functions defined on a non-empty set $\mathcal{X}$. A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a *reproducing kernel* of $\mathcal{H}$ if it satisfies:

- $\forall x \in \mathcal{X}, \quad k(\cdot, x) \in \mathcal{H}$
- $\forall x \in \mathcal{X}, \forall f \in \mathcal{H}, \quad \langle f, k(\cdot, x) \rangle_{\mathcal{H}} = f(x)$ (the reproducing property).

In particular[5], we have, for any $x, y \in \mathcal{X}$,

$$k(x, y) = \langle k(\cdot, x), k(\cdot, y) \rangle_{\mathcal{H}} \tag{1.2}$$

We will now show that any **RKHS** $\mathcal{H}$ has a unique reproducing kernel k ([13, see p.8-9]). We first state the Riesz Representation Theorem which we will need for our proof ([13, see p.6]):

Theorem 1.3.2. (Riesz representation) In a Hilbert space $\mathcal{F}$, all continuous linear functionals are of the form $\langle \cdot, g \rangle_{\mathcal{F}}$, for some $g \in \mathcal{F}$.

We can now prove the following (we follow the proof from [13, p.9]):

Proposition 1.3.1. (Existence and Uniqueness of reproducing kernel) Any **RKHS** $\mathcal{H}$ has a unique reproducing kernel k. I.e. a Hilbert space of functions $\mathcal{H} \subseteq \mathbb{R}^{\mathcal{X}}$ ($\mathcal{X} \neq \emptyset$) is a reproducing kernel Hilbert space if and only if it has a reproducing kernel. Further if a reproducing kernel exists it is unique.

Proof. (Existence) Suppose $\mathcal{H}$ is a reproducing kernel Hilbert space. We show it has a reproducing kernel. For any $x \in \mathcal{X}$ we have from definition 1.3.2 that the Dirac evaluational functional at x, δ_x, is continuous and so by the Riesz representation theorem (Theorem 1.3.2) we have that there exists $f_{\delta_x} \in \mathcal{H}$ such that

$$\delta_x(f) = \langle f, f_{\delta_x} \rangle_{\mathcal{H}} \quad \forall f \in \mathcal{H}.$$

We can now define $k(y, x) := f_{\delta_x}(y)$ for all $x, y \in \mathcal{X}$. Clearly k is a reproducing kernel as we have $k(\cdot, x) \in \mathcal{H}$ and $\langle f, k(\cdot, x) \rangle_{\mathcal{H}} = \langle f, f_{\delta_x} \rangle_{\mathcal{H}} = \delta_x(f) = f(x)$ for all $x \in \mathcal{X}$.

[5]Note that the symmetry of the inner product, together with (1.2), ensures that k is symmetric.

Now for the converse, suppose $\mathcal{H} \subseteq \mathbb{R}^{\mathcal{X}}$ is a Hilbert space with a reproducing kernel k. We show that $\mathcal{H}$ is a **RKHS**. To do so we show that δ_x is a bounded operator for all $x \in \mathcal{X}$. Take any $x \in \mathcal{X}$ and any $f \in \mathcal{H}$ and bound $\delta_x(f)$ as follows:

$$\begin{aligned} |\delta_x(f)| &= |f(x)| \\ &= |\langle f, k(\cdot, x)\rangle_{\mathcal{H}}| \\ &\leq \|f\|_{\mathcal{H}} \|k(\cdot, x)\|_{\mathcal{H}} \\ &= \left(\langle k(\cdot, x), k(\cdot, x)\rangle_{\mathcal{H}}\right)^{1/2} \|f\|_{\mathcal{H}} \\ &= k(x, x)^{1/2} \|f\|_{\mathcal{H}} \end{aligned}$$

Where we used the Cauchy-Schwarz inequality and the reproducing property (1.2) to simplify the upper bound. Now using the fact that $k(x, x) < \infty$ we conclude that $\|\delta_x\| < \infty$ and so δ_x is a bounded operator and hence is continuous as required.

(Uniqueness) Now suppose $\mathcal{H}$ has two reproducing kernels k_1, k_2. We have for any $f \in \mathcal{H}$ and for any $x \in \mathcal{X}$ the following:

$$\langle f, k_1(\cdot, x) - k_2(\cdot, x)\rangle_{\mathcal{H}} = \langle f, k_1(\cdot, x)\rangle_{\mathcal{H}} - \langle f, k_2(\cdot, x)\rangle_{\mathcal{H}} = f(x) - f(x) = 0$$

In particular, if we choose to take $f = k_1(\cdot, x) - k_2(\cdot, x)$ we obtain $\|k_1(\cdot, x) - k_2(\cdot, x)\|_{\mathcal{H}}^2 = 0$ and so $k_1(\cdot, x) \equiv k_2(\cdot, x)$. From this it follows that $k_1 = k_2$ as required. ■

From the proof of Proposition 1.3.1 we can see that the kernel k plays the role of the *representor of evaluation* in $\mathcal{H}$.

We will now conclude this subsection by briefly discussing the Moore-Aronszajn Theorem, which essentially states that for every positive definite function $k(x, y)$ there exists a unique **RKHS** $\mathcal{H}$ which has reproducing kernel k. In order to do so we must first define what a positive definite function is. We follow the definition from [13, p.9]:

Definition 1.3.4. (Positive definite functions) A symmetric function $h : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is positive definite if $\forall n \geq 1, \forall (a_1, \ldots, a_n) \in \mathbb{R}^n, \forall (x_1, \ldots, x_n) \in \mathcal{X}^n$ we have,

$$\sum_{i=1}^{n} \sum_{j=1}^{n} a_i a_j h(x_i, x_j) \geq 0 \tag{1.3}$$

The function $h(\cdot, \cdot)$ is said to be *strictly* positive definite if for mutually distinct $\{x_i\}$, equality in (1.3) holds only when all the a_i are 0.[6]

Any reproducing kernel is a positive definite function which is a simple corollary of the following lemma taken from [13, see p.10]:

Lemma 1.3.3. Let $\mathcal{F}$ be any Hilbert space, $\mathcal{X}$ a non-empty set and $\phi : \mathcal{X} \to \mathcal{F}$. then the function $h(x, y) := \langle \phi(x), \phi(y)\rangle_{\mathcal{F}}$ is positive definite.

For a proof of the above lemma see [13, p. 10]. From this lemma it follows that any reproducing kernel is positive definite:

[6] We follow the terminology frequently used in the machine learning literature as opposed to the more common terminology in linear algebra of "positive semi-definite" vs. "positive definite".

Corollary 1.3.3.1. Reproducing kernels are positive definite.

Proof. For a reproducing kernel k in an **RKHS** $\mathcal{H}$ the reproducing property (1.2) gives us that,

$$k(x, y) = \langle k(\cdot, x), k(\cdot, y) \rangle_{\mathcal{H}}$$

and so taking $\phi : \mathcal{X} \to \mathcal{H}$ to be the map $\phi(x) = k(\cdot, x)$ we see that Lemma 1.3.3 gives us that k is positive definite. ■

Before finally stating and discussing the Moore-Aronszajn Theorem we make a brief remark:

Remark. There is a distinction made between a kernel and a reproducing kernel in the literature which can be useful. A kernel is defined as follows [13, see p.11]:

Definition 1.3.5. (Kernel) Let $\mathcal{X}$ be a non-empty set. The function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is said to be a *kernel* if there exists a real Hilbert space $\mathcal{H}$ and a map $\phi : \mathcal{X} \to \mathcal{H}$ such that $\forall x, y \in \mathcal{H}$ we have,

$$k(x, y) = \langle \phi(x), \phi(y) \rangle_{\mathcal{H}}$$

This map ϕ is often referred to as the feature map and the space $\mathcal{H}$ as the feature space.[7] This will become a very important concept when we discuss **MMD** later on.

We thus see that a kernel is a function which can be written as an inner product in feature space and so is positive definite via Lemma 1.3.3. We can also easily see that any reproducing kernel k in an RKHS $\mathcal{H}$ is also a kernel as we can write $k(x, y) = \langle k(\cdot, x), k(\cdot, y) \rangle_{\mathcal{H}} = \langle \phi(x), \phi(y) \rangle_{\mathcal{H}}$ where the feature map is defined by $\phi(x) := k(\cdot, x)$.

We now finally present the Moore-Aronszajn Theorem [13, see p.16]:

Theorem 1.3.4. (Moore-Aronszajn) Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a positive definite function. There exists a unique **RKHS** $\mathcal{H} \subseteq \mathbb{R}^{\mathcal{X}}$ with reproducing kernel k. Moreover, if the space $\mathcal{H}_0 := \text{span}[\{k(\cdot, x)\}_{x \in \mathcal{X}}]$ is endowed with the inner product

$$\langle f, g \rangle_{\mathcal{H}_0} = \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \beta_j k(x_i, y_j), \tag{1.4}$$

where $f = \sum_{i=1}^{n} \alpha_i k(\cdot, x_i)$ and $g = \sum_{j=1}^{m} \beta_j k(\cdot, y_j)$, then $\mathcal{H}_0$ is a valid pre-**RKHS**. [8]

We do not go over the proof of Theorem 1.3.4. A comprehensive proof is given in [13, see p.12-18]. We now note the significance of this theorem. We saw earlier that any **RKHS** $\mathcal{H}$ necessarily has a unique reproducing kernel k which is also a positive definite function and is also a kernel. We also have seen that any kernel is a positive definite function. The Moore-Aronszajn theorem tells us that for any positive definite function k there is a unique **RKHS** $\mathcal{H}$ with reproducing kernel k. This means that any positive definite function is necessarily a reproducing kernel. Thus, the three concepts of kernels, reproducing kernels and positive definite functions are equivalent. This result is very powerful and will be useful when we discuss **MMD** in the next section.

[7] A given kernel can have more than one feature map.

[8] The notion of a pre-**RKHS** is discussed in greater depth in [13]. It is essentially a space which is used to give us the unique **RKHS** mentioned in Theorem 1.3.4.

1.3.2 Maximum Mean Discrepancy

We will now motivate the Maximum Mean Discrepancy (**MMD**) which will be the vital ingredient to all our subsequent algorithms. **MMD** is a particular instance of an Integral Probability Metric (**IPM**). An Integral Probability Metric $\gamma_{\mathcal{F}}$ is a pseudometric[9] on probability distributions. To be more specific, if we let $\mathscr{P}$ denote the set of all Borel probability measures on a topological space $(\mathcal{X}, \mathcal{A})$, then the **IPM** between $\mathbb{P}, \mathbb{Q} \in \mathscr{P}$ is defined to be [14, see p. 1518]:

$$\gamma_{\mathcal{F}}(\mathbb{P}, \mathbb{Q}) := \sup_{f \in \mathcal{F}} \left| \int_{\mathcal{X}} f \mathrm{d}\mathbb{P} - \int_{\mathcal{X}} f \mathrm{d}\mathbb{Q} \right| \tag{1.5}$$

where $\mathcal{F}$ is a class of real-valued, bounded and measurable functions on $\mathcal{X}$.

The choice of $\mathcal{F}$ is important and leads to different **IPM**s. Intuitively, the "larger" and richer $\mathcal{F}$ is the more discerning $\gamma_{\mathcal{F}}$ is; if one chooses $\mathcal{F}$ to be sufficiently "large" then $\gamma_{\mathcal{F}}$ will in fact be a metric on $\mathscr{P}$. However, the richer (and "larger") $\mathcal{F}$ is, the harder it will be to calculate/estimate $\gamma_{\mathcal{F}}$. We list below some well-known examples of choices of $\mathcal{F}$ which lead to the **IPM** being an actual metric:

- Choosing $\mathcal{F} = \{f : \|f\|_\infty \leq 1\} =: \mathcal{F}_{TV}$, where $\|f\|_\infty = \sup_{x \in M} |f(x)|$, yields the *total variation distance*, $\gamma_{\mathcal{F}_{TV}} =: TV$
- Choosing $\mathcal{F} = \{\mathbb{1}_{(-\infty,t]} : t \in \mathbb{R}^d\} =: \mathcal{F}_{KS}$ yields the familiar *Kolmogorov distance*, $\gamma_{\mathcal{F}_{KS}}$
- Choosing $\mathcal{F} = \{e^{i\langle \omega, \cdot \rangle} : \omega \in \mathbb{R}^d\} =: \mathcal{F}_c$, where i is the imaginary unit, yields $\gamma_{\mathcal{F}_c}(\mathbb{P}, \mathbb{Q})$ to be the maximal absolute difference between the characteristic functions of $\mathbb{P}$ and $\mathbb{Q}$. The uniqueness theorem for characteristic functions yields that $\gamma_{\mathcal{F}_c}$ is a metric on $\mathscr{P}$.

For further examples one can consult [14, p. 1519].

Having defined what an **IPM** is we are now in a position to define Maximum Mean Discrepancy. **MMD** is obtained by choosing $\mathcal{F}$ in the definition of an **IPM** to be the unit ball in an **RKHS** $\mathcal{H}_k$ with reproducing kernel k. In particular we will denote the Maximum Mean Discrepancy associated with the **RKHS** $\mathcal{H}_k$ by γ_k, which is defined as:

$$\gamma_k(\mathbb{P}, \mathbb{Q}) := \sup_{f \in \mathcal{F}_k} \left| \int_{\mathcal{X}} f \mathrm{d}\mathbb{P} - \int_{\mathcal{X}} f \mathrm{d}\mathbb{Q} \right| \tag{1.6}$$

where $\mathcal{F}_k$ is the unit ball in the **RKHS** $\mathcal{H}_k$, i.e. $\mathcal{F}_k := \{f \in \mathcal{H}_k : \|f\|_{\mathcal{H}_k} \leq 1\}$. The choice of kernel k is also important and affects whether or not γ_k will be a metric. This will be discussed at greater depth later. We now list some advantages that **MMD** has over other choices of $\mathcal{F}$ in an **IPM** (these advantages come from [14, p. 1519-1520]):

- In most applications the probability measures $\mathbb{P}$ and $\mathbb{Q}$ are unknown and all we have access to are random, i.i.d. (independent and identically distributed) samples

[9] A pseudometric on a set X is a function $d : X \times X \to \mathbb{R}$ which satisfies: $d(x, x) = 0 \ \forall x \in X$, $d(x, y) = d(y, x) \ \forall x, y \in X$ and $d(x, z) \leq d(x, y) + d(y, z) \ \forall x, y, z \in X$. Thus, a pseudometric is *almost* a metric on X. If we have in addition that $d(x, y) = 0 \implies x = y$ then d is a metric on X.

from each measure, i.e. $\{x_i\}_{i=1}^n \sim \mathbb{P}$, $\{y_j\}_{j=}^m \sim \mathbb{Q}$. These samples can be used to estimate $\gamma_{\mathcal{F}}$; one such approach is to compute $\gamma_{\mathcal{F}}(\mathbb{P}, \mathbb{Q})$ by approximating $\mathbb{P}, \mathbb{Q}$ with the corresponding empirical measures $\mathbb{P}_n = \frac{1}{n}\sum_{i=1}^n \delta_{x_i}, \mathbb{Q}_m = \frac{1}{n}\sum_{j=1}^m \delta_{y_j}$[10]. By choosing $\mathcal{F}$ to be $\mathcal{F}_k$ it can be shown that this estimator, $\gamma_k(\mathbb{P}_n, \mathbb{Q}_m)$, is a $\sqrt{\frac{nm}{n+m}}$ -consistent[11] estimator of $\gamma_k(\mathbb{P}, \mathbb{Q})$, for all $\mathbb{P}, \mathbb{Q}$ when k is measurable and bounded. It is also known that if k is translation invariant on $\mathcal{X} = \mathbb{R}^d$ then the rate is independent of the dimension d, which is very important when one works with high dimensions. Other choices of $\mathcal{F}$ might not yield consistent estimators when using this procedure (e.g. $\mathcal{F}_{TV}$ or $\mathcal{F}_c$) or if they do the rate of convergence might be dependent on the dimension d (for a more detailed discussion see [14, p. 1519]).

- γ_k is in principle relatively easier to calculate than other **IPM**s; as we shall see later, under certain conditions, $\gamma_k^2(\mathbb{P}, \mathbb{Q})$ is simply a sum of expectations of the kernel k.
- Since γ_k can, under certain conditions, be calculated entirely in terms of the kernel k it can be used in applications where the domain is arbitrary such as sets of graphs, or strings, whereas other choices of $\mathcal{F}$ can only deal with $\mathcal{X} = \mathbb{R}^d$.

We will now try to motivate **MMD** in a more natural way, in order to get a more intuitive grasp on it. In order to do this we must first discuss Hilbert Space embeddings of probability distributions. This is a generalization of the kernel feature map $\phi : \mathcal{X} \to \mathcal{H}$ seen in section 1.3.1 which maps a point in $\mathcal{X}$ into the **RKHS** $\mathcal{H}$ to kernel feature maps which map probability measures on $\mathcal{X}$ into the **RKHS**. We follow the presentation in [15, p. 26-31].

To motivate this concept we will first consider the example of a Dirac measure δ_x where x is some fixed point in $\mathcal{X}$, i.e.

$$\delta_x(A) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases} \tag{1.7}$$

For any measurable f on $\mathcal{X}$ we have,

$$\int_{\mathcal{X}} f(t)\mathrm{d}\delta_x(t) = f(x) \tag{1.8}$$

This can be re-written using the reproducing property as,

$$\begin{aligned} \int_{\mathcal{X}} f(t)\mathrm{d}\delta_x(t) &= \int_{\mathcal{X}} \langle f, k(\cdot, t)\rangle_{\mathcal{H}} \mathrm{d}\delta_x(t) \\ &= \left\langle f, \int_{\mathcal{X}} k(\cdot, t)\mathrm{d}\delta_x(t) \right\rangle_{\mathcal{H}} \\ &= \langle f, k(\cdot, x)\rangle_{\mathcal{H}} \end{aligned}$$

We can see from this that the integral $\int_{\mathcal{X}} k(\cdot, t)\mathrm{d}\delta_x(t)$ acts as a representer of the measure δ_x in the **RKHS** $\mathcal{H}$ just as $k(\cdot, x)$ is the representer of x in the **RKHS**. We can actually

[10] δ_x is here taken to be a (Dirac) point mass at x. We note that we use the same notation as for the Dirac Evaluation Functional; the particular meaning should hopefully be clear from the context.

[11] An estimator is said to be N-consistent if the sampling error is $\mathcal{O}_p(1/N)$.

view this integral as a representer of evaluation of the following functional which takes the expectation w.r.t (with respect to) δ_x:

$$f \longmapsto \int_{\mathcal{X}} f(t)\mathrm{d}\delta_x(t) \tag{1.9}$$

This example is trivial as (1.9) is equivalent to taking the inner product $\langle f, k(\cdot, x)\rangle_{\mathcal{H}}$ as both simply give $f(x)$. (1.9) however provides us with a measure-theoretic viewpoint of representing a point in a **RKHS**. We can now make sense of the following feature map of a Dirac measure,

$$\delta_x \longmapsto \int_{\mathcal{X}} k(\cdot, t)\mathrm{d}\delta_x(t) \tag{1.10}$$

This concept can be extended to include measures which are a linear combination of Dirac measures via the feature map,

$$\sum_{i=1}^{n} a_i\delta_{x_i} \longmapsto \int_{\mathcal{X}} k(\cdot, t)\mathrm{d}\left(\sum_{i=1}^{n} a_i\delta_{x_i}(t)\right) = \sum_{i=1}^{n} a_i \int_{\mathcal{X}} k(\cdot, t)\mathrm{d}\delta_{x_i}(t) \tag{1.11}$$

where the a_i are constants and the x_i are fixed points in $\mathcal{X}$ (see [15] p.28-29 for a discussion of this).

We can now take all of this once step further and consider an arbitrary measure $\mathbb{P}$ on $\mathcal{X}$ and define a feature map,

$$\mathbb{P} \longmapsto \int_{\mathcal{X}} k(\cdot, t)\mathrm{d}\mathbb{P}(t) =: \Pi[\mathbb{P}] \tag{1.12}$$

where the integral in (1.12) is interpreted as a Bochner integral[12]. The feature map (1.12) is what we will refer to as the Hilbert Space Embedding of the probability measure $\mathbb{P}$ into the **RKHS** $\mathcal{H}$ with reproducing kernel k. This embedding is the analogue of a kernel feature map of points to probability measures.

We now present a lemma found in [15, see p.30] which gives a sufficient condition for the embedding defined in (1.12) to actually exist and belong to $\mathcal{H}$:

Lemma 1.3.5. If $\mathbb{E}_{X\sim\mathbb{P}}\left[\sqrt{k(X,X)}\right] < \infty$ then $\Pi[\mathbb{P}] \in \mathcal{H}$ and $\mathbb{E}_{X\sim\mathbb{P}}[f(X)] = \langle f, \Pi[\mathbb{P}]\rangle_{\mathcal{H}}$.

Proof. Let $T_{\mathbb{P}} : \mathcal{H} \to \mathbb{R}$ be the linear functional defined by $T_{\mathbb{P}}[f] := \mathbb{E}_{X\sim\mathbb{P}}[f(X)]$. We show that $T_{\mathbb{P}}$ is a bounded operator. For any $f \in \mathcal{H}$ we have:

$$\begin{aligned}
|T_{\mathbb{P}}[f]| &= |\mathbb{E}_{X\sim\mathbb{P}}[f(X)]| \\
&\leq \mathbb{E}_{X\sim\mathbb{P}}\,|f(X)| \\
&= \mathbb{E}_{X\sim\mathbb{P}}\,|\langle f, k(\cdot, X)\rangle_{\mathcal{H}}| \\
&\leq \mathbb{E}_{X\sim\mathbb{P}}\,[\|k(\cdot, X)\|_{\mathcal{H}}\|f\|_{\mathcal{H}}] \\
&= \|f\|_{\mathcal{H}}\,\mathbb{E}_{X\sim\mathbb{P}}[\sqrt{k(X,X)}]
\end{aligned}$$

where the first inequality above is Jensen's inequality and the second inequality follows from the Cauchy-Schwarz inequality together with the preservation of ordering property

[12] A Bochner integral is in general an integral of a function which takes values in some Banach space.

of expectation. We also used the reproducing property twice, first to write $f(X)$ as $\langle f, k(\cdot, X)\rangle_{\mathcal{H}}$ and secondly to write $\|k(\cdot, X)\|_{\mathcal{H}} = \langle k(\cdot, X), k(\cdot, X)\rangle_{\mathcal{H}}^{1/2} = \sqrt{k(X, X)}$. Now, by the assumption we have $\mathbb{E}_{X\sim\mathbb{P}}\left[\sqrt{k(X, X)}\right] < \infty$ and so we have showed that $T_{\mathbb{P}}$ is a bounded operator and so by Theorem 1.3.1 we have that $T_{\mathbb{P}}$ is a continuous linear functional. We can thus invoke the Riesz representation theorem to assert that there exists $h \in \mathcal{H}$ such that $T_{\mathbb{P}}[f] = \langle f, h\rangle_{\mathcal{H}}$ for all $f \in \mathcal{H}$. If we now take $f = k(\cdot, x)$ for some $x \in \mathcal{X}$ we obtain the following,

$$h(x) = \langle k(\cdot, x), h\rangle_{\mathcal{H}} = T_{\mathbb{P}}[k(\cdot, x)] = \int_{\mathcal{X}} k(t, x)\mathrm{d}\mathbb{P}(t)$$

and so $h = \int_{\mathcal{X}} k(\cdot, t)\mathrm{d}\mathbb{P}(t) = \Pi[\mathbb{P}]$ as required. ■

Remark. When the conditions of the above Lemma are satisfied we have that $\mathbb{E}_{X\sim\mathbb{P}}[f(X)] = \langle f, \Pi[\mathbb{P}]\rangle_{\mathcal{H}}$ for any $f \in \mathcal{H}$. This can be viewed as an analogue of the reproducing property of the evaluation operator in the **RKHS** for the expectation operator.

Having introduced Hilbert Space Embeddings of probability measures we can now given an alternative characterisation of **MMD** for a particular set of probability measures; we will show that for this set $\gamma_k(\mathbb{P}, \mathbb{Q})$ is in fact the **RKHS** norm of the difference of the Hilbert Space Embeddings of $\mathbb{P}$ and $\mathbb{Q}$. We present the result in the following theorem (which can be found in [14, see p.1525]):

Theorem 1.3.6. Let $\mathscr{P}_k := \{\mathbb{P} \in \mathscr{P} : \mathbb{E}_{X\sim\mathbb{P}}[\sqrt{k(X, X)}] < \infty\}$, where k is measurable on $\mathcal{X}$. Then for any $\mathbb{P}, \mathbb{Q} \in \mathscr{P}_k$ we have,

$$\gamma_k(\mathbb{P}, \mathbb{Q}) = \left\| \int_{\mathcal{X}} k(\cdot, x)\mathrm{d}\mathbb{P}(x) - \int_{\mathcal{X}} k(\cdot, x)\mathrm{d}\mathbb{Q}(x) \right\|_{\mathcal{H}_k} = \|\Pi[\mathbb{P}] - \Pi[\mathbb{Q}]\|_{\mathcal{H}_k} \tag{1.13}$$

Proof. Noting that the conditions placed on probability measures in $\mathscr{P}_k$ are the same conditions from Lemma 1.3.5 we can conclude that for any $\mathbb{P} \in \mathscr{P}_k$ we have that $\mathbb{E}_{X\sim\mathbb{P}}[f(X)] = \langle f, \Pi[\mathbb{P}]\rangle_{\mathcal{H}}$ for any $f \in \mathcal{H}_k$. From the definition (1.6) of **MMD** we can then write:

$$\begin{aligned}
\gamma_k(\mathbb{P}, \mathbb{Q}) &= \sup_{f\in\mathcal{F}_k} \left| \int_{\mathcal{X}} f\mathrm{d}\mathbb{P} - \int_{\mathcal{X}} f\mathrm{d}\mathbb{Q} \right| \\
&= \sup_{\|f\|_{\mathcal{H}_k}\leq 1} |\mathbb{E}_{X\sim\mathbb{P}}[f(X)] - \mathbb{E}_{X\sim\mathbb{Q}}[f(X)]| \\
&= \sup_{\|f\|_{\mathcal{H}_k}\leq 1} |\langle f, \Pi[\mathbb{P}]\rangle_{\mathcal{H}_k} - \langle f, \Pi[\mathbb{Q}]\rangle_{\mathcal{H}_k}| \\
&= \sup_{\|f\|_{\mathcal{H}_k}\leq 1} |\langle f, \Pi[\mathbb{P}] - \Pi[\mathbb{Q}]\rangle_{\mathcal{H}_k}|
\end{aligned}$$

In order to proceed we must now deal with the supremum in the final equality above. For convenience we will denote the **RKHS** element $\Pi[\mathbb{P}] - \Pi[\mathbb{Q}]$ by g. We now can see that in order to simplify our expression for **MMD** we must figure out the following maximization problem:

$$\sup_{\|f\|_{\mathcal{H}_k}\leq 1} |\langle f, g\rangle_{\mathcal{H}_k}|$$

We do so by noting the following:

$$|\langle f, g\rangle_{\mathcal{H}_k}| \leq \|f\|_{\mathcal{H}_k}\|g\|_{\mathcal{H}_k} \leq \|g\|_{\mathcal{H}_k}$$

where the first inequality is the Cauchy-Schwarz inequality and the second inequality follows from the constraint of the maximization problem above. Now the above holds for any $f \in \mathcal{F}_k$ and we can achieve equality in the above bounds iff f is chosen to be proportional to g (this is the condition for equality in Cauchy-Schwarz) and $\|f\|_{\mathcal{H}_k} = 1$ (for equality in second inequality). In order to satisfy these two conditions we must take $f = g/\|g\|_{\mathcal{H}_k}$ and thus we can finally simplify our expression for **MMD** as:

$$\begin{aligned}\gamma_k(\mathbb{P}, \mathbb{Q}) &= \sup_{\|f\|_{\mathcal{H}_k} \leq 1} |\langle f, g \rangle_{\mathcal{H}_k}| \\ &= |\langle g/\|g\|_{\mathcal{H}_k}, g \rangle_{\mathcal{H}_k}| \\ &= \|g\|_{\mathcal{H}_k} \\ &= \|\Pi[\mathbb{P}] - \Pi[\mathbb{Q}]\|_{\mathcal{H}_k}\end{aligned}$$

as required. ■

Remark. The result (1.13) in Theorem 1.3.6 holds only for measures $\mathbb{P}, \mathbb{Q} \in \mathscr{P}_k$. In most applications however, we do not know the measures $\mathbb{P}, \mathbb{Q}$, as mentioned earlier, and so it will not be possible to check whether the measures lie in $\mathscr{P}_k$. In view of this, it would be ideal if we had a kernel k such that $\mathbb{E}_{X \sim \mathbb{P}}[\sqrt{k(X, X)}] < \infty$ for all $\mathbb{P} \in \mathscr{P}$. We present in the following proposition (taken from [14, see p. 1526]) a result which will give us a condition on the kernel which is equivalent to this requirement.

Proposition 1.3.2. Let f be a measurable function on $\mathcal{X}$. Then $\mathbb{E}_{X \sim \mathbb{P}}[f(X)] < \infty$ for all $\mathbb{P} \in \mathscr{P}$ iff f is bounded.

Proof. See Appendix A for proof. ■

We can see from the above proposition that if we take a kernel k which is bounded then we satisfy the requirement $\mathbb{E}_{X \sim \mathbb{P}}[\sqrt{k(X, X)}] < \infty$ for all $\mathbb{P} \in \mathscr{P}$. So if k is bounded we thus have that $\gamma_k(\mathbb{P}, \mathbb{Q}) = \|\Pi[\mathbb{P}] - \Pi[\mathbb{Q}]\|_{\mathcal{H}_k}$ for all $\mathbb{P}, \mathbb{Q} \in \mathscr{P}$. We can thus see that the question of whether or not γ_k is a metric on $\mathscr{P}$ becomes equivalent to the question of whether or not the embedding Π is injective, when k is bounded. This topic will be discussed shortly. We first briefly present a number of equivalent representations of γ_k which will help improve our grasp on **MMD** and help with its computation.

The first new representation is a simple use of the reproducing property of k (which we assume is bounded). We can write:

$$
\begin{aligned}
\gamma_k^2(\mathbb{P},\mathbb{Q}) &= \|\Pi[\mathbb{P}]-\Pi[\mathbb{Q}]\|_{\mathcal{H}_k}^2 \\
&= \langle \Pi[\mathbb{P}]-\Pi[\mathbb{Q}], \Pi[\mathbb{P}]-\Pi[\mathbb{Q}]\rangle_{\mathcal{H}_k} \\
&= \langle \Pi[\mathbb{P}], \Pi[\mathbb{P}]\rangle_{\mathcal{H}_k} + \langle \Pi[\mathbb{Q}], \Pi[\mathbb{Q}]\rangle_{\mathcal{H}_k} - 2\langle \Pi[\mathbb{P}], \Pi[\mathbb{Q}]\rangle_{\mathcal{H}_k} \\
&= \left\langle \int_{\mathcal{X}} k(\cdot,x)\mathrm{d}\mathbb{P}(x), \int_{\mathcal{X}} k(\cdot,y)\mathrm{d}\mathbb{P}(y)\right\rangle_{\mathcal{H}_k} + \left\langle \int_{\mathcal{X}} k(\cdot,x)\mathrm{d}\mathbb{Q}(x), \int_{\mathcal{X}} k(\cdot,y)\mathrm{d}\mathbb{Q}(y)\right\rangle_{\mathcal{H}_k} \\
&- 2\left\langle \int_{\mathcal{X}} k(\cdot,x)\mathrm{d}\mathbb{P}(x), \int_{\mathcal{X}} k(\cdot,y)\mathrm{d}\mathbb{Q}(y)\right\rangle_{\mathcal{H}_k} \\
&= \int_{\mathcal{X}}\int_{\mathcal{X}} \langle k(\cdot,x),k(\cdot,y)\rangle_{\mathcal{H}_k}\,\mathrm{d}\mathbb{P}(x)\mathbb{P}(y) + \int_{\mathcal{X}}\int_{\mathcal{X}} \langle k(\cdot,x),k(\cdot,y)\rangle_{\mathcal{H}_k}\,\mathrm{d}\mathbb{Q}(x)\mathbb{Q}(y) \\
&- 2\int_{\mathcal{X}}\int_{\mathcal{X}} \langle k(\cdot,x),k(\cdot,y)\rangle_{\mathcal{H}_k}\,\mathrm{d}\mathbb{P}(x)\mathbb{Q}(y) \\
&= \int_{\mathcal{X}}\int_{\mathcal{X}} k(x,y)\mathrm{d}\mathbb{P}(x)\mathrm{d}\mathbb{P}(y) + \int_{\mathcal{X}}\int_{\mathcal{X}} k(x,y)\mathrm{d}\mathbb{Q}(x)\mathrm{d}\mathbb{Q}(y) \\
&- 2\int_{\mathcal{X}}\int_{\mathcal{X}} k(x,y)\mathrm{d}\mathbb{P}(x)\mathrm{d}\mathbb{Q}(y) \\
&= \mathbb{E}_{X\sim\mathbb{P},Y\sim\mathbb{P}}[k(X,Y)] + \mathbb{E}_{X\sim\mathbb{Q},Y\sim\mathbb{Q}}[k(X,Y)] - 2\,\mathbb{E}_{X\sim\mathbb{P},Y\sim\mathbb{Q}}[k(X,Y)] \qquad (1.14)
\end{aligned}
$$

We thus can see from the last equalThusity above that in the case of bounded k, the squared **MMD** between two probability measures $\mathbb{P},\mathbb{Q}$ is simply a linear combination of expectations of the kernel k, as mentioned previously. The representation (1.14) will be used later to provide an unbiased estimator of squared **MMD** in our applications where we will be unable to explicitly compute the expectations.

The second equivalent representation of **MMD** we will present applies when the kernel k is a translation invariant, bounded, continuous positive definite function. In order to derive this representation we require the well-known Bochner Theorem which we quote from [14, see p. 1527]:

Theorem 1.3.7. (Bochner) A continuous function $\psi : \mathbb{R}^d \to \mathbb{R}$ is positive definite iff it is the Fourier transform of a finite non-negative Borel measure Λ on $\mathbb{R}^d$, i.e.,

$$\psi(x) = \int_{\mathbb{R}^d} e^{-ix^T\omega}\mathrm{d}\Lambda(\omega),\ \ x\in\mathbb{R}^d \qquad (1.15)$$

Having stated this theorem we are now in a position to present and prove the following corollary (which we follow from [14, see p. 1527]):

Corollary 1.3.7.1. Let $\mathcal{X}=\mathbb{R}^d$ and $k(x,y)=\psi(x-y)$, where $\psi:\mathcal{X}\to\mathbb{R}$ is a bounded, continuous positive definite function. Then for any $\mathbb{P},\mathbb{Q}\in\mathscr{P}$ we have,

$$\gamma_k(\mathbb{P},\mathbb{Q}) = \sqrt{\int_{\mathbb{R}^d} |\phi_{\mathbb{P}}(\omega)-\phi_{\mathbb{Q}}(\omega)|^2\,\mathrm{d}\Lambda(\omega)} =: \|\phi_{\mathbb{P}}-\phi_{\mathbb{Q}}\|_{L^2(\mathbb{R}^d,\Lambda)}\,, \qquad (1.16)$$

where $\phi_{\mathbb{P}},\phi_{\mathbb{Q}}$ represent the characteristic functions of $\mathbb{P}$ and $\mathbb{Q}$ respectively.

Proof. Since the kernel $k(x,y) = \psi(x-y)$ is bounded we can use the representation (1.14). We write this succinctly[13] as,

$$\begin{aligned}
\gamma_k^2(\mathbb{P},\mathbb{Q}) &= \int_{\mathbb{R}^d}\int_{\mathbb{R}^d} \psi(x-y)\mathrm{d}(\mathbb{P}-\mathbb{Q})(x)\mathrm{d}(\mathbb{P}-\mathbb{Q})(y) \\
&\overset{(a)}{=} \int_{\mathbb{R}^d}\int_{\mathbb{R}^d}\int_{\mathbb{R}^d} e^{-i(x-y)^T\omega}\mathrm{d}\Lambda(\omega)\mathrm{d}(\mathbb{P}-\mathbb{Q})(x)\mathrm{d}(\mathbb{P}-\mathbb{Q})(y) \\
&\overset{(b)}{=} \int_{\mathbb{R}^d}\int_{\mathbb{R}^d} e^{-ix^T\omega}\mathrm{d}(\mathbb{P}-\mathbb{Q})(x)\int_{\mathbb{R}^d} e^{iy^T\omega}\mathrm{d}(\mathbb{P}-\mathbb{Q})(y)\mathrm{d}\Lambda(\omega) \\
&\overset{(c)}{=} \int_{\mathbb{R}^d} (\phi_{\mathbb{P}}(\omega)-\phi_{\mathbb{Q}}(\omega))\overline{(\phi_{\mathbb{P}}(\omega)-\phi_{\mathbb{Q}}(\omega))}\mathrm{d}\Lambda(\omega) \\
&= \int_{\mathbb{R}^d} |\phi_{\mathbb{P}}(\omega)-\phi_{\mathbb{Q}}(\omega)|^2\,\mathrm{d}\Lambda(\omega),
\end{aligned}$$

as required, where in (a) we used Bochner's Theorem and in (b) we used Fubini's theorem to interchange the order of the integrals. In (c) we used the definition of the characteristic function of a probability measure.

■

We note that there are more interpretations of **MMD** under different conditions. We direct the interested reader to [14, see p.1527-1530] for a discussion of these interpretations. We now move on to the next subsection to discuss how the choice of kernel determines whether **MMD** is a metric on probability distributions.

1.3.3 Characteristic Kernels

We will now discuss the question of whether or not **MMD** is a metric on $\mathscr{P}$ (or on some proper subset of $\mathscr{P}$). The answer to this question depends on the choice of kernel k, and we define those kernels for which the resulting **MMD** is a metric as being characteristic. We present the formal definition (which we take from [14, see p.1530]) below:

Definition 1.3.6. (Characteristic Kernel) A bounded measurable positive definite kernel k is said to be *characteristic* to a set $\mathscr{D} \subseteq \mathscr{P}$ of probability measures defined on $(\mathcal{X},\mathcal{A})$ if for $\mathbb{P},\mathbb{Q} \in \mathscr{D}, \gamma_k(\mathbb{P},\mathbb{Q}) = 0 \iff \mathbb{P} = \mathbb{Q}$. k is simply said to be *characteristic* if it is *characteristic* to $\mathscr{P}$. The **RKHS** $\mathcal{H}_k$ induced by such a k is called a *characteristic* **RKHS**.

Remark. As discussed previously, in the case that k is bounded, the question of whether or not γ_k is a metric on $\mathscr{P}$ is equivalent to the question of whether or not the embedding Π is injective. In light of the above definition we see that it is precisely for characteristic kernels that this embedding is injective. The name "characteristic" comes from the fact that the embedding of a measure $\mathbb{P}$ into a characteristic **RKHS** can be viewed as a generalization of the characteristic function associated with $\mathbb{P}$, $\phi_{\mathbb{P}} = \int_{\mathbb{R}^d} e^{i\langle\cdot,x\rangle}\mathrm{d}\mathbb{P}(x)$. The uniqueness theorem for characteristic functions ensures us that $\phi_{\mathbb{P}} = \phi_{\mathbb{Q}} \implies \mathbb{P} = \mathbb{Q}$, which is the same as saying that the mapping $\mathbb{P} \longmapsto \int_{\mathbb{R}^d} e^{i\langle\cdot,x\rangle}\mathrm{d}\mathbb{P}(x)$ is injective. We can thus see that it is possible to view $e^{i\langle y,x\rangle}$ as a characteristic kernel; however this is not technically true as this function is not positive definite, as required by definition 1.3.6.

[13]The notation $\mathrm{d}(\mathbb{P}-\mathbb{Q})(x)\mathrm{d}(\mathbb{P}-\mathbb{Q})(y)$ in the integral is shorthand for the linear combination of integrals in the first representation of **MMD** mentioned on the previous page.

We will now briefly present some results taken from the literature, without proof, which characterize whether or not a given kernel k is characteristic. In order to present the first characterization we must first define what an integrally strictly p.d. (positive definite) kernel is. We follow the definition from [14, see p.1523]:

Definition 1.3.7. (Integrally strictly positive definite kernels) Let $\mathcal{X}$ be a topological space. A measurable and bounded kernel k is said to be *integrally strictly positive definite* if,

$$\int_{\mathcal{X}}\int_{\mathcal{X}} k(x,y)\mathrm{d}\mu(x)\mathrm{d}\mu(y) > 0$$

for all finite non-zero signed Borel measures μ defined on $\mathcal{X}$.

Remark. It is worth noting that the definition given above is not the same as the definition of strictly pd kernels. We have that if k is integrally strictly pd then it is strictly pd. The other direction is not true.

Having defined what an integrally strictly positive definite kernel is, we are now in the position to present the first characterization of whether or not a given kernel k is characteristic. We present this in the following Theorem (taken from [14, see p.1532]):

Theorem 1.3.8. (Integrally strictly pd kernels are characteristic) Let k be an integrally strictly positive definite kernel on a topological space $\mathcal{X}$. Then k is characteristic to $\mathscr{P}$.

Remark. For a proof of the above theorem please consult [14, p. 1532].

The above characterization is relatively easy to understand especially compared to the following two characterizations found in the literature (these characterizations are referred to on p.1520 of [14]):

1. In the case of compact $\mathcal{X}$, k is characteristic if it is *universal* (as defined by Steinwart in [16, see p.72]), which essentially means that the **RKHS** $\mathcal{H}_k$ induced by k is dense in the Banach space of bounded continuous function (w.r.t the supremum norm).

2. The characterization above was extended to non-compact $\mathcal{X}$ in [17, see p.4] and [18, see p.1881] where it is shown that k is characteristic iff the direct sum of the induced **RKHS** $\mathcal{H}_k$ and $\mathbb{R}$ is dense in the Banach space of r-integrable functions (for some $r \geq 1$).

The above two characterizations are not as easy to understand as the one presented in Theorem 1.3.8 and it is quite difficult to verify the conditions of denseness required in both of the above. While the integrally strictly pd characterization is easier to understand, it can still be very difficult to verify if a given kernel satisfies it. This leads us to present the next characterization which can be found in [14, see p.1533] which is much simpler (in principle) to verify. We now assume we have $\mathcal{X} = \mathbb{R}^d$ and that the kernel k satisfies the following assumption:

Assumption 1 $k(x,y) = \psi(x-y)$ where ψ is a bounded continuous real-valued positive definite function on $\mathcal{X} = \mathbb{R}^d$.

We now present in the following theorem[14] (adopted from [14, see p.1533]) a complete characterization for all translation invariant kernels on $\mathbb{R}^d$:

[14]supp here denotes the support of a Borel measure: $\mathrm{supp}(\mu) := \mathcal{X} \backslash \bigcup\{U \subseteq \mathcal{X} : U \text{ is open}, \mu(U) = 0\}$.

Theorem 1.3.9. Suppose k satisfies Assumption 1. Then k is characteristic iff $\text{supp}(\Lambda) = \mathbb{R}^d$, where Λ is defined as in (1.15).

Remark. For a proof of the above theorem please consult the proof of Theorem 9 in [14].

We note that the above theorem essentially states that a translation invariant kernel on $\mathbb{R}^d$ is characteristic iff the support of its Fourier Transform is the whole of $\mathbb{R}^d$ which is, in principle, a relatively easy condition to check, provided the Fourier transform can be computed by some means. A corollary to Theorem 1.3.9 is presented in [14, see p.1535] which sates that any compactly supported translation invariant continuous bounded kernel on $\mathbb{R}^d$ is characteristic:

Corollary 1.3.9.1. Suppose $k \neq 0$ satisfies Assumption 1 and $\text{supp}(\psi)$ is compact. Then k is characteristic.

A further corollary to Theorem 1.3.9 is given in [14, see p.1535] which allows us to construct new characteristic kernels from a given characteristic kernel. We state this corollary here:

Corollary 1.3.9.2. Let k, k_1 and k_2 satisfy Assumption 1. Suppose k is characteristic and $k_2 \neq 0$. Then $k + k_1$ and $k \cdot k_2$ are characteristic.

Remark. Note that in the above corollary we do note require k_1 or k_2 to be characteristic; we can therefore obtain all sorts of characteristic kernels provided we start with one which is characteristic.

We now present several examples of kernels.

Example 1 (Trivial kernel) Let $k(x, y) = C \ \forall x, y \in \mathbb{R}^d$, where $C > 0$ is a constant. Using this in the representation (1.14) we obtain,

$$\begin{aligned}\gamma_k^2(\mathbb{P}, \mathbb{Q}) &= \mathbb{E}_{X\sim\mathbb{P},Y\sim\mathbb{P}}[k(X,Y)] + \mathbb{E}_{X\sim\mathbb{Q},Y\sim\mathbb{Q}}[k(X,Y)] - 2\mathbb{E}_{X\sim\mathbb{P},Y\sim\mathbb{Q}}[k(X,Y)] \\ &= C + C - 2C = 0\end{aligned}$$

for any $\mathbb{P}, \mathbb{Q} \in \mathscr{P}$. We can thus see that this is an example of a bounded, measurable positive definite kernel which is NOT characteristic.

Example 2 (polynomial kernel of degree 1) Let $k(x, y) = xy$, $x, y \in \mathbb{R}$. We can compute,

$$\begin{aligned}\gamma_k^2(\mathbb{P}, \mathbb{Q}) &= \int_{\mathcal{X}}\int_{\mathcal{X}} xy \mathrm{d}\mathbb{P}(x)\mathrm{d}\mathbb{P}(y) + \int_{\mathcal{X}}\int_{\mathcal{X}} xy \mathrm{d}\mathbb{Q}(x)\mathrm{d}\mathbb{Q}(y) - 2\int_{\mathcal{X}}\int_{\mathcal{X}} xy \mathrm{d}\mathbb{P}(x)\mathrm{d}\mathbb{Q}(y) \\ &= \mu_{\mathbb{P}}^2 + \mu_{\mathbb{Q}}^2 - 2\mu_{\mathbb{P}}\mu_{\mathbb{Q}} \\ &= (\mu_{\mathbb{P}} - \mu_{\mathbb{Q}})^2\end{aligned}$$

where $\mu_{\mathbb{P}}, \mu_{\mathbb{Q}}$ denote the means of the probability measures $\mathbb{P}, \mathbb{Q}$ respectively, i.e., $\mu_{\mathbb{P}} := \int_{\mathbb{R}} x \mathrm{d}\mathbb{P}(x)$. We can thus see that the polynomial kernel of degree 1 leads to the squared **MMD** being simply the squared difference in the means of the two distributions under consideration. We also note that this kernel is NOT characteristic as $\gamma_k(\mathbb{P}, \mathbb{Q}) = 0 \iff \mu_{\mathbb{P}} = \mu_{\mathbb{Q}} \implies \mathbb{P} = \mathbb{Q}$ for general $\mathbb{P}, \mathbb{Q} \in \mathscr{P}$.

Example 3 (polynomial kernel of degree 2) Let $k(x, y) = xy + x^2y^2,\ \forall x, y \in \mathbb{R}$. We have,

$$\begin{aligned}\gamma_k^2(\mathbb{P}, \mathbb{Q}) = &\int_{\mathbb{R}}\int_{\mathbb{R}}(xy + x^2y^2)\mathrm{d}\mathbb{P}(x)\mathrm{d}\mathbb{P}(y) + \int_{\mathbb{R}}\int_{\mathbb{R}}(xy + x^2y^2)\mathrm{d}\mathbb{Q}(x)\mathrm{d}\mathbb{Q}(y)\\ &- 2\int_{\mathbb{R}}\int_{\mathbb{R}}(xy + x^2y^2)\mathrm{d}\mathbb{P}(x)\mathrm{d}\mathbb{Q}(y)\end{aligned}$$

We deal with the first integral:

$$\begin{aligned}\int_{\mathbb{R}}\int_{\mathbb{R}}(xy + x^2y^2)\mathrm{d}\mathbb{P}(x)\mathrm{d}\mathbb{P}(y) &= \int_{\mathbb{R}}(y\mu_{\mathbb{P}} + y^2v_{\mathbb{P}})\mathrm{d}\mathbb{P}(y)\\ &= \mu_{\mathbb{P}}^2 + v_{\mathbb{P}}^2\end{aligned}$$

where $v_{\mathbb{P}}$ denotes the second un-centered moment associated with $\mathbb{P}$, i.e., $v_{\mathbb{P}} := \int_{\mathbb{R}} x^2\mathrm{d}\mathbb{P}(x)$. The second integral can be similarly computed as,

$$\int_{\mathbb{R}}\int_{\mathbb{R}}(xy + x^2y^2)\mathrm{d}\mathbb{Q}(x)\mathrm{d}\mathbb{Q}(y) = \mu_{\mathbb{Q}}^2 + v_{\mathbb{Q}}^2$$

Similarly we also can compute the final integral as,

$$\begin{aligned}\int_{\mathbb{R}}\int_{\mathbb{R}}(xy + x^2y^2)\mathrm{d}\mathbb{P}(x)\mathrm{d}\mathbb{Q}(y) &= \int_{\mathbb{R}}(y\mu_{\mathbb{P}} + y^2v_{\mathbb{P}})\mathrm{d}\mathbb{Q}(y)\\ &= \mu_{\mathbb{P}}\mu_{\mathbb{Q}} + v_{\mathbb{P}}v_{\mathbb{Q}}\end{aligned}$$

Putting this all together we obtain:

$$\begin{aligned}\gamma_k^2(\mathbb{P}, \mathbb{Q}) &= \mu_{\mathbb{P}}^2 + v_{\mathbb{P}}^2 + \mu_{\mathbb{Q}}^2 + v_{\mathbb{Q}}^2 - 2\mu_{\mathbb{P}}\mu_{\mathbb{Q}} - 2v_{\mathbb{P}}v_{\mathbb{Q}}\\ &= (\mu_{\mathbb{P}} - \mu_{\mathbb{Q}})^2 + (v_{\mathbb{P}} - v_{\mathbb{Q}})^2\\ &= (\mu_{\mathbb{P}} - \mu_{\mathbb{Q}})^2 + (\sigma_{\mathbb{P}}^2 - \sigma_{\mathbb{Q}}^2 + \mu_{\mathbb{P}}^2 - \mu_{\mathbb{Q}}^2)^2\end{aligned}$$

where $\sigma_{\mathbb{P}}^2$ denotes the variance associated with $\mathbb{P}$. We thus note that k is also NOT characteristic as $\gamma_k^2(\mathbb{P}, \mathbb{Q}) = 0 \iff \mu_{\mathbb{P}} = \mu_{\mathbb{Q}}$ and $\sigma_{\mathbb{P}}^2 = \sigma_{\mathbb{Q}}^2 \implies \mathbb{P} = \mathbb{Q}$ for general $\mathbb{P}, \mathbb{Q} \in \mathscr{P}$.

Example 4 (Gaussian kernel) Let $k(x, y) = e^{-\sigma\|x-y\|_2^2},\ \forall x, y \in \mathbb{R}^d$, where $\|\cdot\|_2$ denotes the standard Euclidean norm in $\mathbb{R}^d$, and $\sigma > 0$ is a constant which determines the bandwidth of the Gaussian. For this example we do not explicitly compute the resulting **MMD** as this cannot be computed in closed form in general. We instead note that we can write $k(x, y) = \psi(x - y)$ where $\psi(x) = e^{-\sigma\|x\|_2^2}$. The Gaussian kernel thus satisfies Assumption 1. We proceed to compute the Fourier transform of ψ, which we denote by $\widehat{\psi}$, on the next page as follows:

$$\begin{aligned}
\widehat{\psi}(\omega) &:= \frac{1}{(2\pi)^{d/2}} \int_{\mathbb{R}^d} e^{-i\omega^T x} \psi(x) \mathrm{d}x \\
&= \frac{1}{(2\pi)^{d/2}} \int_{\mathbb{R}^d} e^{-i\omega^T x} e^{-\sigma x^T x} \mathrm{d}x \\
&= \frac{1}{(2\pi)^{d/2}} \int_{\mathbb{R}^d} e^{-\sum_{j=1}^d (\sigma x_j + i\omega_j) x_j} \mathrm{d}x \\
&= \frac{1}{(2\pi)^{d/2}} \int_{\mathbb{R}^d} \prod_{j=1}^d e^{-(\sigma x_j^2 + i\omega_j x_j)} \mathrm{d}x \\
&= \prod_{j=1}^d \frac{1}{\sqrt{2\pi}} \int_{\mathbb{R}} e^{-(\sigma x_j^2 + i\omega_j x_j)} \mathrm{d}x_j
\end{aligned}$$

From the above computation we can see that the Fourier transform of ψ is in fact a product of Fourier transforms of 1-D Gaussians, whose formula is well-known. We thus have,

$$\widehat{\psi}(\omega) = \prod_{j=1}^d \frac{1}{\sqrt{2\sigma}} e^{-\omega_j^2/4\sigma} = \frac{1}{(2\sigma)^{d/2}} e^{-\|\omega\|_2^2/4\sigma}, \ \forall \omega \in \mathbb{R}^d$$

We thus observe that the support of the Fourier transform of ψ is the whole of $\mathbb{R}^d$ and so by Theorem 1.3.9 we have that k is characteristic.

Having discussed characteristic kernels and presented several characterizations for them we will now conclude this subsection by making several remarks:

Remark. In many applications it is important that we take the associated kernel for **MMD** to be characteristic so that γ_k is a metric on $\mathscr{P}$. While choosing a characteristic kernel does ensure that $\gamma_k(\mathbb{P}, \mathbb{Q}) = 0 \iff \mathbb{P} = \mathbb{Q}$ we can still nonetheless come into problems in distinguishing two different distributions. In section 4 of [14] Sriperumbudur et al. show that under certain conditions there can exist probability measures $\mathbb{P} \neq \mathbb{Q}$ on $\mathcal{X}$ such that $\gamma_k(\mathbb{P}, \mathbb{Q}) < \epsilon$ for arbitrarily small $\epsilon > 0$. Essentially, under these conditions, even though the kernel is characteristic it is unable to "properly" distinguish distributions which differ at sufficiently high frequencies. This can cause problems in many applications, especially when we use empirical estimates of **MMD** based on finite samples.

Remark. The characterization for translation invariant kernels involved the support of the Fourier transform of ψ to be the whole of $\mathbb{R}^d$. When the support is instead a proper subset of $\mathbb{R}^d$ it is still possible to show, under certain conditions, that k is characteristic to a proper subset of $\mathscr{P}$. We direct the interested reader to section 3.2 of [14].

Remark. Section 3.3 of [14] investigates a similar characterization for translation invariant kernels on a d-dimensional torus $\mathbb{T}^d$. The interested reader is directed to this reference.

Chapter 2

Parameter Estimation for ODEs using MMD

In this chapter we will explain how **MMD** can be used to perform parameter inference in generative models based on **ODEs**. We begin this chapter by outlining the general idea behind all our estimation procedures. We follow the procedure outlined in section 2 of [1].

2.1 Minimum MMD estimators

Throughout this section we will assume, unless otherwise stated, that we are working with the complex generative model described in section 1.1 (see equation (1.1)). We denote by $\mathscr{P}_\Theta$ the collection of distributions this generative model can yield samples from. We shall assume that the true data generating distribution is unknown to us; this distribution shall be denoted by $\mathbb{Q}$. Further, we will also make the assumption that $\mathbb{Q} \in \mathscr{P}_\Theta$, i.e. we are in the *M-closed* setting, and we shall denote by $\boldsymbol{\theta}_0$ the parameter which is associated with $\mathbb{Q}$ (so that $\mathbb{Q} = \mathbb{P}_{\boldsymbol{\theta}_0}$). Upon observing i.i.d. samples $\{y_i\}_{i=1}^m \overset{i.i.d.}{\sim} \mathbb{Q}$ our goal is to use these samples to infer the true parameter vector $\boldsymbol{\theta}_0$. Throughout this project we will seek to estimate the true parameter by using the following minimum **MMD** estimator ([1, see p.5]):

$$\hat{\boldsymbol{\theta}}_m \in \arg\min_{\theta \in \Theta} \gamma_k^2(\mathbb{P}_{\boldsymbol{\theta}}, \mathbb{Q}^m) \tag{2.1}$$

where $\mathbb{Q}^m(\mathrm{d}y) = \frac{1}{m}\sum_{i=1}^m \delta_{y_i}(\mathrm{d}y)$ is the empirical measure associated with the samples $\{y_i\}_{i=1}^m$. We note that in (2.1) we have used the belongs symbol instead of an equal sign as it might be the case that this optimization problem does not have a unique minimum.

We note that the parameter estimation procedure we are adopting essentially involves using the squared MMD as a cost function which will then be optimized to estimate the true parameter value. As discussed in section 1.3 if the kernel k is chosen to be characteristic then **MMD** is able to discriminate between distributions. The hope is that this should be enough to discriminate between parameters[1]. This optimization problem is in general non-convex and can often not be solved analytically. As such it will be necessary to turn to a computational approach in order to obtain an approximation of

[1]Of course there is the possibility that there is an identifiability problem; namely that two different parameter vectors yield the same distribution (i.e. $\exists \boldsymbol{\theta}_1 \neq \boldsymbol{\theta}_2$ such that $\mathbb{P}_{\boldsymbol{\theta}_1} = \mathbb{P}_{\boldsymbol{\theta}_2}$). In this case our procedure might give us a different parameter vector which also generates the same distribution. This problem does not occur when the map $\boldsymbol{\theta} \mapsto \mathbb{P}_{\boldsymbol{\theta}}$ is injective.

the minimum **MMD** estimator (2.1). Before we proceed with outlining this approach we must first state a few further assumption we will make. First we shall assume that $\mathscr{P}_\Theta \subseteq \mathscr{P}_k$. This is to ensure that for any choice of $\boldsymbol{\theta} \in \Theta$ we can use the representation (1.14) for $\gamma_k^2(\mathbb{P}_{\boldsymbol{\theta}}, \mathbb{Q}^m)$.

Remark. If we choose the kernel k to be bounded than we immediately satisfy this assumption as in this case we have $\mathscr{P}_k = \mathscr{P}$ as discussed following Proposition 1.3.2.

The second assumption we shall need is that the generator $G_{\boldsymbol{\theta}}$ is differentiable with respect to $\boldsymbol{\theta}$ with a computable Jacobian matrix. This assumption implies that the minimum **MMD** estimator will be a fixed point of the equation

$$\dot{\boldsymbol{\theta}} = -\nabla_{\boldsymbol{\theta}} \gamma_k^2(\mathbb{P}_{\boldsymbol{\theta}}, \mathbb{Q}^m) \tag{2.2}$$

where $\nabla_{\boldsymbol{\theta}} = (\partial_{\theta_1}, \ldots, \partial_{\theta_p})$. In this project we will make use of gradient descent algorithms for finding the fixed points of the equation (2.2). In this section we shall investigate a classical stochastic gradient descent method as well as a variant of this. We note that other optimizers exist such as the ADAM optimizer [19] (which we will utilize in our numerical experiments in Chapter 4) as well as gradient descent with a momentum term [20]. In order to do this we must first derive an expression for the gradient term in (2.2), which we now proceed to do. This will be done by utilizing the representation (1.14) for the squared **MMD** together with an additional assumption that the Jacobian $\nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}$ is $\mathbb{U}$-integrable (which will allow us to interchange the order of the differentiation and expectation operations when computing the gradient). We are now in a position to derive the gradient. We first note that under all of our assumptions the squared **MMD** can be written as follows:

$$\begin{aligned}
\gamma_k^2\left(\mathbb{P}_{\boldsymbol{\theta}}, \mathbb{Q}^m\right) &= \mathbb{E}_{X\sim\mathbb{P}_{\boldsymbol{\theta}}, Y\sim\mathbb{P}_{\boldsymbol{\theta}}}[k(X,Y)] - 2\mathbb{E}_{X\sim\mathbb{P}_{\boldsymbol{\theta}}, Y\sim\mathbb{Q}^m}[k(X,Y)] \\
&\quad + \mathbb{E}_{X\sim\mathbb{Q}^m, Y\sim\mathbb{Q}^m}[k(X,Y)] \\
&= \int_{\mathcal{X}}\int_{\mathcal{X}} k(x,y)\mathrm{d}\mathbb{P}_{\boldsymbol{\theta}}(x)\mathrm{d}\mathbb{P}_{\boldsymbol{\theta}}(y) - 2\int_{\mathcal{X}}\int_{\mathcal{X}} k(x,y)\mathrm{d}\mathbb{P}_{\boldsymbol{\theta}}(x)\mathrm{d}\mathbb{Q}^m(y) \\
&\quad + \mathbb{E}_{X\sim\mathbb{Q}^m, Y\sim\mathbb{Q}^m}[k(X,Y)] \\
&= \int_{\mathcal{U}}\int_{\mathcal{U}} k\left(G_{\boldsymbol{\theta}}(u), G_{\boldsymbol{\theta}}(v)\right)\mathbb{U}(\mathrm{d}u)\mathbb{U}(\mathrm{d}v) - \frac{2}{m}\sum_{j=1}^{m}\int_{\mathcal{U}} k(G_{\boldsymbol{\theta}}(u), y_j)\mathbb{U}(\mathrm{d}u) \\
&\quad + \mathbb{E}_{X\sim\mathbb{Q}^m, Y\sim\mathbb{Q}^m}[k(X,Y)]
\end{aligned}$$

Only the first two terms in the lines above depend on $\boldsymbol{\theta}$ and so only they will contribute to the gradient. We now use the assumptions to interchange the derivative operator with the integrals to obtain:

$$\nabla_{\boldsymbol{\theta}}\gamma_k^2(\mathbb{P}_{\boldsymbol{\theta}}, \mathbb{Q}^m) = \int_{\mathcal{U}}\int_{\mathcal{U}} \nabla_{\boldsymbol{\theta}} k\left(G_\theta(u), G_\theta(v)\right)\mathbb{U}(\mathrm{d}u)\mathbb{U}(\mathrm{d}v) - \frac{2}{m}\sum_{j=1}^{m}\int_{\mathcal{U}} \nabla_{\boldsymbol{\theta}} k\left(G_\theta(u), y_j\right)\mathbb{U}(\mathrm{d}u)$$

We now work out the two different gradients in the integrands above separately:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} k(G_{\boldsymbol{\theta}}(u), G_{\boldsymbol{\theta}}(v)) &= \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(u)^T \nabla_1 k(G_{\boldsymbol{\theta}}(u), G_{\boldsymbol{\theta}}(v)) + \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(v)^T \nabla_2 k(G_{\boldsymbol{\theta}}(u), G_{\boldsymbol{\theta}}(v)) \\
\nabla_{\boldsymbol{\theta}} k(G_{\boldsymbol{\theta}}, y_j) &= \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(u)^T \nabla_1 k(G_{\boldsymbol{\theta}}(u), y_j)
\end{aligned}$$

where we have utilized the chain rule and where $\nabla_1 k, \nabla_2 k$ denote the partial derivative w.r.t. the first and second arguments respectively. Plugging these gradients into the integrals above yields:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \gamma_k^2 \left(\mathbb{P}_{\boldsymbol{\theta}}, \mathbb{Q}^m\right) &= \int_{\mathcal{U}} \int_{\mathcal{U}} \left(\nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(u)^T \nabla_1 k(G_{\boldsymbol{\theta}}(u), G_{\boldsymbol{\theta}}(v)) + \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(v)^T \nabla_2 k(G_{\boldsymbol{\theta}}(u), G_{\boldsymbol{\theta}}(v))\right) \mathbb{U}(\mathrm{d}u)\mathbb{U}(\mathrm{d}v) \\ &\quad - \frac{2}{m} \sum_{j=1}^{m} \int_{\mathcal{U}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(u)^T \nabla_1 k(G_{\boldsymbol{\theta}}(u), y_j) \mathbb{U}(\mathrm{d}u) \\ &= 2 \int_{\mathcal{U}} \int_{\mathcal{U}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(u)^T \nabla_1 k(G_{\boldsymbol{\theta}}(u), G_{\boldsymbol{\theta}}(v)) \mathbb{U}(\mathrm{d}u)\mathbb{U}(\mathrm{d}v) \\ &\quad - \frac{2}{m} \sum_{j=1}^{m} \int_{\mathcal{U}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(u)^T \nabla_1 k(G_{\boldsymbol{\theta}}(u), y_j) \mathbb{U}(\mathrm{d}u) \end{aligned} \tag{2.3}$$

where in the last equality we essentially use the symmetry of the kernel; for a more detailed justification of this see Appendix A.

In general the integrals in (2.3) are not able to be done analytically. We thus will utilize a U-statistic[2] approximation for the gradient:

$$\begin{aligned}\hat{J}_{\boldsymbol{\theta}}\left(\mathbb{Q}^m\right) &= \frac{2 \sum_{i \neq i'} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}\left(u_i\right)^T \nabla_1 k\left(G_{\boldsymbol{\theta}}\left(u_i\right), G_{\boldsymbol{\theta}}\left(u_{i'}\right)\right)}{n(n-1)} \\ &\quad - \frac{2 \sum_{j=1}^{m} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}\left(u_i\right)^T \nabla_1 k\left(G_{\boldsymbol{\theta}}\left(u_i\right), y_j\right)}{nm} \end{aligned} \tag{2.4}$$

where $\{u_i\}_{i=1}^n \overset{i.i.d.}{\sim} \mathbb{U}$. (2.4) is an unbiased estimator of the gradient (2.3) in the sense that $\mathbb{E}\left[\hat{J}_{\boldsymbol{\theta}}\left(\mathbb{Q}^m\right)\right] = \nabla_{\boldsymbol{\theta}} \gamma_k^2\left(\mathbb{P}_{\boldsymbol{\theta}}, \mathbb{Q}^m\right)$ where here the expectation is taken w.r.t. the independent realizations of the $\{u_i\}_{i=1}^n$.

Having set-up our notation and introduced an unbiased estimator of the gradient (2.3) we are now in a position to present the first gradient descent algorithm we will be considering. We present this in Algorithm 1 below (see [1, p. 5]):

Algorithm 1: Classical Stochastic Gradient Descent

Input: data $\{y_j\}_{j=1}^m$, initial parameter vector $\widehat{\boldsymbol{\theta}}^{(0)} \in \Theta$, step-size sequence $(\eta_r)_{r \in \mathbb{N}}$

Output: Sequence of parameter vectors $(\widehat{\boldsymbol{\theta}}^{(r)})_{r \in \mathbb{N}}$

1 **repeat**

2 Sample $\{u_i\}_{i=1}^n \overset{i.i.d.}{\sim} \mathbb{U}$ and compute $\widetilde{y}_i = G_{\widehat{\boldsymbol{\theta}}^{(r-1)}}(u_i)$ for $i = 1, \ldots, n$;

3 Compute U-statistic estimate of gradient $\hat{J}_{\widehat{\boldsymbol{\theta}}^{(r-1)}}\left(\mathbb{Q}^m\right)$;

4 Update parameter vector $\widehat{\boldsymbol{\theta}}^{(r)} = \widehat{\boldsymbol{\theta}}^{(r-1)} - \eta_r \hat{J}_{\widehat{\boldsymbol{\theta}}^{(r-1)}}\left(\mathbb{Q}^m\right)$;

5 **until** *convergence or max number of steps reached*;

We note that the step-size sequence $(\eta_r)_{r \in \mathbb{N}}$ above should be chosen so as to guarantee convergence[3]. If we have $\mathcal{X} \subseteq \mathbb{R}^d$ then upon counting the number of multiplications in

[2]For a definition of U-statistics see for instance Definition 3.4 of [15, p.51-52].

[3]In our numerical experiments we will however also set a threshold on the maximum number of steps we can take.

each iteration of Algorithm 1 we observe that the cost per iteration is $\mathcal{O}((n^2 + nm)dp)$. The cost is thus linear in the number of data points m yet quadratic in the number of simulated points n. For large enough values of n the algorithm should approach the minimum **MMD** estimator provided we start at a good location; it is very likely that our choice of starting vector will land us in a local minimum of $\gamma_k^2\left(\mathbb{P}_{\boldsymbol{\theta}}, \mathbb{Q}^m\right)$ where we will subsequently be trapped. To try to alleviate this problem we will endeavour to make multiple runs from different locations. For each run we will add an extra step to our algorithm which will involve computing a U-statistic approximation[4] for the squared **MMD** (see [1, p.6] or [15, p.52-53]) at each iteration, namely:

$$\mathrm{MMD}^2(\mathbb{P}_{\boldsymbol{\theta}}^n, \mathbb{Q}^m) = \frac{\sum_{i \neq i'} k\left(\widetilde{y}_i, \widetilde{y}_{i'}\right)}{n(n-1)} - \frac{2\sum_{j=1}^m \sum_{i=1}^n k\left(\widetilde{y}_i, y_j\right)}{mn} + \frac{\sum_{j \neq j'} k\left(y_j, y_{j'}\right)}{m(m-1)} \tag{2.5}$$

where $\mathbb{P}_{\boldsymbol{\theta}}^n = \frac{1}{n}\sum_{i=1}^n \delta_{\widetilde{y}_i}$ is the empirical distribution associated with the samples $\{\widetilde{y}_i\}_{i=1}^n$ computed in line 2 of Algorithm 1. Our estimate for the minimum **MMD** estimator will then be the vector which achieves the lowest score (i.e. the estimate from (2.5)) across all runs.

While the algorithm above should converge to a local minimum of the cost function it is not in general true that the negative of the gradient $-\nabla_{\boldsymbol{\theta}}\gamma_k^2\left(\mathbb{P}_{\boldsymbol{\theta}}, \mathbb{Q}^m\right)$ is in the steepest descent direction of the cost function in **MMD** space; therefore it is possible that the gradient will be making huge moves in parameter space without moving much in **MMD** space and vice-versa. In such circumstances it may instead be better to modify the descent direction using information regarding the local curvature of the **MMD** space. This leads us to consider what is called "natural gradient adaptation" (see [21] and [22]), which essentially involves modifying the parameter update step to be:

$$\widehat{\boldsymbol{\theta}}^{(r)} = \widehat{\boldsymbol{\theta}}^{(r-1)} - \eta_r F^{-1}\left(\widehat{\boldsymbol{\theta}}^{(r-1)}\right) \hat{J}_{\widehat{\boldsymbol{\theta}}^{(r-1)}}\left(\mathbb{Q}^m\right) \tag{2.6}$$

where here F is the Riemannian metric on Θ which essentially contains information regarding the local curvature of the **MMD** space at each point. For a more detailed discussion see [21] and [22]. For the complex generative model and the squared **MMD** loss we consider here this Riemannian metric tensor can be expressed as (taken from [1, see p.7]):

$$F(\boldsymbol{\theta}) = \int_{\mathcal{U}}\int_{\mathcal{U}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(u)^\top \nabla_2 \nabla_1 k\left(G_{\boldsymbol{\theta}}(u), G_{\boldsymbol{\theta}}(v)\right) \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(v) \mathbb{U}(\mathrm{d}u)\mathbb{U}(\mathrm{d}v) \tag{2.7}$$

where the matrix $\nabla_1\nabla_2 k(x,y)$ has ij-th entry $\partial_{x_i}\partial_{y_j}k(x,y)$ for $i,j = 1,\ldots,d$. As was the case with the gradient, the integrals in (2.7) are often intractable and so instead we will consider a U-statistic approximation for the metric tensor found in [1, p. 8]:

$$F_U(\boldsymbol{\theta}) = \frac{1}{n(n-1)} \sum_{i \neq j} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}\left(u_i\right)^\top \nabla_2 \nabla_1 k\left(G_{\boldsymbol{\theta}}\left(u_i\right), G_{\boldsymbol{\theta}}\left(u_j\right)\right) \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}\left(u_j\right) \tag{2.8}$$

where again the $\{u_i\}_{i=1}^n$ are i.i.d. samples from $\mathbb{U}$. This is an unbiased estimator for F and with this we can now present in Algorithm 2 below (see [1, p. 8]) a natural gradient descent algorithm for minimizing the squared **MMD**:

[4] This estimator is an unbiased estimator for the squared **MMD**.

Algorithm 2: Natural Stochastic Gradient Descent

Input: data $\{y_j\}_{j=1}^m$, initial parameter vector $\widehat{\boldsymbol{\theta}}^{(0)} \in \Theta$, step-size sequence $(\eta_r)_{r\in\mathbb{N}}$

Output: Sequence of parameter vectors $(\widehat{\boldsymbol{\theta}}^{(r)})_{r\in\mathbb{N}}$

1 **repeat**

2 Sample $\{u_i\}_{i=1}^n \overset{i.i.d.}{\sim} \mathbb{U}$ and compute $\widetilde{y}_i = G_{\widehat{\boldsymbol{\theta}}^{(r-1)}}(u_i)$ for $i = 1, \ldots, n$;

3 Compute U-statistic estimate of gradient $\hat{J}_{\widehat{\boldsymbol{\theta}}^{(r-1)}}(\mathbb{Q}^m)$;

4 Compute U-statistic estimate of Riemann metric tensor $F_U\left(\widehat{\boldsymbol{\theta}}^{(k-1)}\right)$;

5 Update parameter vector $\widehat{\boldsymbol{\theta}}^{(r)} = \widehat{\boldsymbol{\theta}}^{(r-1)} - \eta_r F_U^{-1}\left(\widehat{\boldsymbol{\theta}}^{(r-1)}\right) \hat{J}_{\widehat{\boldsymbol{\theta}}^{(r-1)}}(\mathbb{Q}^m)$

6 **until** *convergence or max number of steps reached*;

Due to the inversion of the U-statistic approximation F_U, which is in general a dense matrix, and the additional matrix vector multiplication in the update step, Algorithm 2 has a higher cost (see [1, p. 8]) of $\mathcal{O}((n^2 + nm)p^2 d + p^3)$ per iteration. We thus see that in cases where the dimension p of the parameter space is high that this inversion at every step will make the computational cost of Algorithm 2 restrictive. For moderate values of p however this algorithm is not that much more computationally costly and as discussed above should provide faster convergence as the steps taken are now in the direction of steepest descent of the cost function in **MMD** space.

We will now move on to the next subsection where we will discuss ways to compute the Jacobian $\nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}$ which is required in both algorithms above.

2.2 Computing the Jacobian $\nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}$

In both Algorithm 1 and 2 of the previous subsection the Jacobian $\nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}$ is required. As such, we will need a way of computing it. For the generative model we are considering the generator $G_{\boldsymbol{\theta}}(u)$ is a collection of observations of the solution of the IVP (1.1). Thus, computing the Jacobian $\nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}$ will involve computing partial derivatives of the state vector $\mathbf{x}$, where we have dropped the superscripts (i) in (1.1) as now we are considering 1 particular sample, w.r.t. the parameters $\boldsymbol{\theta}$. Such partial derivatives quantify the sensitivity of the solution with respect to changes in the parameters, and the study of such quantities is known as 'sensitivity analysis'. Since we are dealing with systems of first order differential equations we shall only deal here with such systems. For clarity we will simplify our notation and write the system we are considering as:

$$\dot{\mathbf{x}} = f_{\boldsymbol{\theta}}(\mathbf{x}), \quad \mathbf{x}(0; \boldsymbol{\theta}) = \mathbf{x}_0 \tag{2.9}$$

where we have essentially absorbed any randomness from $\boldsymbol{\xi}, \boldsymbol{\epsilon}$ into the function $f_{\boldsymbol{\theta}}$ and the vector $\mathbf{x}_0$ respectively and where we have made explicit the dependence of the solution on the parameters by writing $\mathbf{x} = \mathbf{x}(t; \boldsymbol{\theta})$. We are interested in computing $j(t, \boldsymbol{\theta}) := \nabla_{\boldsymbol{\theta}} \mathbf{x}(t; \boldsymbol{\theta})$. The entries of this matrix j are what is referred to in the literature as 'first-order' sensitivity coefficients, see for instance [23]. The most straightforward method to computing the entries of this matrix is through utilizing a finite-difference approximation (e.g. see [23, p. 46]):

$$j_{rs}(t; \boldsymbol{\theta}) = \frac{\partial x_r}{\partial \theta_s} \approx \frac{1}{\Delta \theta_s}(x_r(t; \boldsymbol{\theta} + \Delta\theta_s \mathbf{e}_s) - x_r(t; \boldsymbol{\theta})) \tag{2.10}$$

where $\Delta\theta_s$ is a small perturbation and $\mathbf{e}_s$ is the s-th standard basis vector. The use of finite-differences here however has two potential problems. The first is that for each entry of the matrix j we are required to solve the IVP (2.9) once more in order to compute the first term in the numerator of (2.10). This can become restrictive when p is large. The second problem comes from numerical inaccuracies in the finite-difference approximation. As such we will not be using this method and we will instead use, for this chapter[5], the method we will now describe which essentially involves deriving a system of equations for the time evolution of j.

We will now derive this system which can then be coupled with (2.9) and solved simultaneously via the same method we use to solve (2.9). In order to derive this system we compute the time derivative of j as follows:

$$\begin{aligned}\frac{\mathrm{d}j}{\mathrm{d}t} &= \frac{\mathrm{d}}{\mathrm{d}t}\nabla_{\boldsymbol{\theta}}\mathbf{x} \\ &= \nabla_{\boldsymbol{\theta}}\dot{\mathbf{x}} \\ &= \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{x}) \\ &= \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{x}) + \left(\nabla_{\mathbf{x}} f_{\boldsymbol{\theta}}(\mathbf{x})\right) j\end{aligned}$$

where in the second equality above we exchanged the order of differentiation and in the final equality we utilized the chain rule. This is justified if we assume that both f and $\mathbf{x}$ are continuously differentiable.

We now need to determine the initial condition for j. In general, the parameters may appear in the initial condition. We shall assume that the random vector $\mathbf{x}_0$ can be written as $\mathbf{x}_0 = h(\boldsymbol{\theta}; \boldsymbol{\zeta})$ where h is some deterministic function and $\boldsymbol{\zeta}$ is a random vector which does not depend on $\boldsymbol{\theta}$. Such an assumption is what is known in the machine learning community as the 'Reparametrization Trick' (see for instance [24] and [25]). In this case we have,

$$\begin{aligned} j(0, \boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \mathbf{x}(0; \boldsymbol{\theta}) \\ &= \nabla_{\boldsymbol{\theta}} h(\boldsymbol{\theta}; \boldsymbol{\zeta}) \end{aligned} \tag{2.11}$$

Combining all of this we can solve simultaneously the following system:

$$\begin{aligned} &\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = f_{\boldsymbol{\theta}}(\mathbf{x}), \quad \mathbf{x}(0; \boldsymbol{\theta}) = \mathbf{x}_0 = h(\boldsymbol{\theta}; \boldsymbol{\zeta}) \\ &\frac{\mathrm{d}j}{\mathrm{d}t} = \nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(\mathbf{x}) + \left(\nabla_{\mathbf{x}} f_{\boldsymbol{\theta}}(\mathbf{x})\right) j, \quad j(0, \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} h(\boldsymbol{\theta}; \boldsymbol{\zeta}) \end{aligned} \tag{2.12}$$

Remark. If the initial condition does not depend on the parameters $\boldsymbol{\theta}$ then we have $j(0, \boldsymbol{\theta}) = 0$.

We note that the dimension of the system we need to solve has increased by $p \times d$. For problems with either large p or large d this method can thus become quite computationally expensive if no effort is made to find efficient methods of solving (2.12). This shall be address partly in Chapter 3 when we discuss the 'adjoint' method.

[5] In the next chapter we will be considering the use of the so called Adjoint Method to compute the gradient of the squared **MMD** and as such we will not require the Jacobian $\nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}$.

For a more detailed discussion of this method the interested reader is directed to [26]. This approach will be utilized in Numerical Experiment 1 which will be presented in Chapter 4. We note that similar expressions for the Jacobian can also be derived for Stochastic Differential Equations, see for example [27].

Chapter 3

Reverse Differentiation of ODEs

In this Chapter the use of the so called Adjoint Method will be investigated to help with the parameter estimation problem. As was noted in Chapter 2 when the dimension of the parameter space is high Algorithm 2 can become infeasible. Further, even in Algorithm 1 the computation of the sensitivity matrix j can also quickly become infeasible as the dimension of the **ODE** system needed to be solved (for each simulated system) increases by $p \times d$. It will thus prove useful to consider the Adjoint Method (also known as the Pontraygin Principle) which will provide us with a way of obtaining the gradient of the squared **MMD** ***without*** having to compute the sensitivity matrix j. The Adjoint Method will be derived in this chapter and then it will be applied to some numerical examples which will be presented in Chapter 4.

3.1 Derivation of the Adjoint Method

In this section the Adjoint Method will be derived. The method can be applied to many different problems and frameworks and so it will be presented first for a more general one. This will be done in the first subsection of this section. After having done this a worked example will be presented. Following this worked example the method shall be developed for the particular types of problems this project looks at.

3.1.1 Adjoint Method for Time-dependent Problems

For the remainder of this subsection it shall be assumed that the gradient of a scalar-valued function is a row-vector, this is to make the derivation clearer[1]. The treatment in section 2 of [28] is followed. In this subsection we will consider the following problem:

$$\begin{array}{ll} \underset{\boldsymbol{\theta}\in\Theta}{\text{minimize}} & C(\mathbf{x},\boldsymbol{\theta}), \quad \text{where} \quad C(\mathbf{x},\boldsymbol{\theta}) \equiv \int_0^T f(\mathbf{x},\boldsymbol{\theta},t)\mathrm{d}t \\ \text{subject to} & h(\mathbf{x},\dot{\mathbf{x}},\boldsymbol{\theta},t) = 0 \\ & g(\mathbf{x}(0),\boldsymbol{\theta}) = 0 \end{array} \tag{3.1}$$

where again $\boldsymbol{\theta}$ is a vector of unknown parameters, $\mathbf{x}$ is a (possibly vector-valued) function of time, $h(\mathbf{x},\dot{\mathbf{x}},\boldsymbol{\theta},t) = 0$ is an **ODE** in implicit form[2], and $g(\mathbf{x}(0),\boldsymbol{\theta}) = 0$ represents the initial condition on the **ODE**. It shall be assumed that the Jacobian $\nabla_{\mathbf{x}} g$ is everywhere

[1] The ij-th entry of a Jacobian matrix $\nabla_{\mathbf{x}}\mathbf{y}$ is still $\partial_{x_j} y_i$.

[2] If we have an **ODE** in explicit form, $\dot{\mathbf{x}} = \bar{h}(\mathbf{x},\boldsymbol{\theta},t)$, this can be converted into implicit form by setting $h(\mathbf{x},\dot{\mathbf{x}},\boldsymbol{\theta},t) = \dot{\mathbf{x}} - \bar{h}(\mathbf{x},\boldsymbol{\theta},t)$.

non-singular. If one wishes to solve the optimization problem (3.1) via a gradient-based algorithm then the gradient $\nabla_{\boldsymbol{\theta}} C$ is required. If we assume that the function f is sufficiently well-behaved so that the order of integration and differentiation can be exchanged then one obtains,

$$\nabla_{\boldsymbol{\theta}} C(\mathbf{x}, \boldsymbol{\theta}) = \int_0^T \left[\nabla_{\boldsymbol{\theta}} f(\mathbf{x}, \boldsymbol{\theta}, t) + \nabla_{\mathbf{x}} f(\mathbf{x}, \boldsymbol{\theta}, t) \nabla_{\boldsymbol{\theta}} \mathbf{x}\right] \mathrm{d}t$$

where we have utilized the chain rule. As discussed in section 2.2 computation of $\nabla_{\boldsymbol{\theta}} \mathbf{x}$ can become quite computationally expensive for large p. The Adjoint Method aims to avoid having to compute this sensitivity matrix by developing a second **ODE** for the so called adjoint vector $\boldsymbol{\lambda}$ which is then used to compute the required gradient $\nabla_{\boldsymbol{\theta}} C$. This method will now be derived. The first thing to do is to introduce the Lagrangian associated with the constrained optimization problem (3.1) namely,

$$\mathcal{L} \equiv \int_0^T \left[f(x, \boldsymbol{\theta}, t) + \boldsymbol{\lambda}^T h(\mathbf{x}, \dot{\mathbf{x}}, \boldsymbol{\theta}, t)\right] \mathrm{d}t + \boldsymbol{\mu}^T g(\mathbf{x}(0), \boldsymbol{\theta}) \tag{3.2}$$

where the vector of Lagrange multipliers $\boldsymbol{\lambda}$ is a function of time and $\boldsymbol{\mu}$ is a second vector of multipliers corresponding to the initial condition constraints. Since the two constraints $h = 0$ and $g = 0$ are satisfied everywhere we have the flexibility to set the values of the multipliers $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$ to any values we like and further we have $\nabla_{\boldsymbol{\theta}} \mathcal{L} = \nabla_{\boldsymbol{\theta}} C$. Taking the gradient yields:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} \mathcal{L} = \int_0^T & \left[\nabla_{\boldsymbol{\theta}} f + \nabla_{\mathbf{x}} f \nabla_{\boldsymbol{\theta}} \mathbf{x} + \boldsymbol{\lambda}^T \left(\nabla_{\boldsymbol{\theta}} h + \nabla_{\mathbf{x}} h \nabla_{\boldsymbol{\theta}} \mathbf{x} + \nabla_{\dot{\mathbf{x}}} h \nabla_{\boldsymbol{\theta}} \dot{\mathbf{x}}\right)\right] \mathrm{d}t \\ & + \boldsymbol{\mu}^T \left(\nabla_{\boldsymbol{\theta}} g + \nabla_{\mathbf{x}(0)} g \nabla_{\boldsymbol{\theta}} \mathbf{x}(0)\right) \end{aligned} \tag{3.3}$$

where again we have utilized the chain rule. The integrand contains terms in both $\nabla_{\boldsymbol{\theta}} \mathbf{x}$ and $\nabla_{\boldsymbol{\theta}} \dot{\mathbf{x}}$. Integration by parts is used to eliminate the second one as follows:

$$\int_0^T \boldsymbol{\lambda}^T \nabla_{\dot{\mathbf{x}}} h \nabla_{\boldsymbol{\theta}} \dot{\mathbf{x}} \mathrm{d}t = \left[\boldsymbol{\lambda}^T \nabla_{\dot{\mathbf{x}}} h \nabla_{\boldsymbol{\theta}} \mathbf{x}\right]_0^T - \int_0^T \left[\dot{\boldsymbol{\lambda}}^T \nabla_{\dot{\mathbf{x}}} h + \boldsymbol{\lambda}^T \mathrm{d}_t \nabla_{\dot{\mathbf{x}}} h\right] \nabla_{\boldsymbol{\theta}} \mathbf{x} \mathrm{d}t$$

where d_t denotes the time derivative $\frac{\mathrm{d}}{\mathrm{d}t}$. Using this in (3.3) and collecting terms yields:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} \mathcal{L} = \int_0^T & \left[\left(\nabla_{\mathbf{x}} f + \boldsymbol{\lambda}^T \left(\nabla_{\mathbf{x}} h - \mathrm{d}_t \nabla_{\dot{\mathbf{x}}} h\right) - \dot{\boldsymbol{\lambda}}^T \nabla_{\dot{\mathbf{x}}} h\right) \nabla_{\boldsymbol{\theta}} \mathbf{x} + \nabla_{\boldsymbol{\theta}} f + \boldsymbol{\lambda}^T \nabla_{\boldsymbol{\theta}} h\right] \mathrm{d}t \\ & + \boldsymbol{\lambda}^T \nabla_{\dot{\mathbf{x}}} h\big|_T \nabla_{\boldsymbol{\theta}} \mathbf{x}(T) + \left(-\boldsymbol{\lambda}^T \nabla_{\dot{\mathbf{x}}} h + \boldsymbol{\mu}^T \nabla_{\mathbf{x}(0)} g\right)\big|_0 \nabla_{\boldsymbol{\theta}} \mathbf{x}(0) + \boldsymbol{\mu}^T \nabla_{\boldsymbol{\theta}} g \end{aligned} \tag{3.4}$$

The idea behind the Adjoint Method is to choose $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ appropriately so as to make all terms involving $\nabla_{\boldsymbol{\theta}} \mathbf{x}$ in (3.4) vanish. Thus, if one chooses to set $\boldsymbol{\lambda}(T) = 0$ as well as $\boldsymbol{\mu}^T = \boldsymbol{\lambda}^T \nabla_{\dot{\mathbf{x}}} h|_0 \left(\nabla_{\mathbf{x}(0)} g\right)^{-1}$ the terms $\nabla_{\boldsymbol{\theta}} \mathbf{x}(T)$ and $\nabla_{\boldsymbol{\theta}} \mathbf{x}(0)$ are cancelled out. Then in order to avoid having to compute $\nabla_{\boldsymbol{\theta}} \mathbf{x}$ at all other times in $(0, T)$ one can set,

$$\nabla_{\mathbf{x}} f + \boldsymbol{\lambda}^T \left(\nabla_{\mathbf{x}} h - \mathrm{d}_t \nabla_{\dot{\mathbf{x}}} h\right) - \dot{\boldsymbol{\lambda}}^T \nabla_{\dot{\mathbf{x}}} h = 0 \tag{3.5}$$

These computations yield the Adjoint Method algorithm for computing $\nabla_{\boldsymbol{\theta}} C$ as follows:

Algorithm 3: Adjoint Method for Computing $\nabla_{\boldsymbol{\theta}} C$

1 Integrate $h(\mathbf{x}, \dot{\mathbf{x}}, \boldsymbol{\theta}, t) = 0$ from $t = 0$ to T with initial conditions $g(\mathbf{x}(0), \boldsymbol{\theta}) = 0$ to obtain $\mathbf{x}$;

2 Integrate $\nabla_{\mathbf{x}} f + \boldsymbol{\lambda}^T \left(\nabla_{\mathbf{x}} h - \mathrm{d}_t \nabla_{\dot{\mathbf{x}}} h\right) - \dot{\boldsymbol{\lambda}}^T \nabla_{\dot{\mathbf{x}}} h = 0$ from $t = T$ to 0 with initial conditions $\boldsymbol{\lambda}(T) = 0$ to obtain the adjoint vector $\boldsymbol{\lambda}$;

3 Set $\nabla_{\boldsymbol{\theta}} C = \int_0^T \left[\nabla_{\boldsymbol{\theta}} f + \boldsymbol{\lambda}^T \nabla_{\boldsymbol{\theta}} h\right] \mathrm{d}t + \boldsymbol{\lambda}^T \nabla_{\dot{\mathbf{x}}} h\big|_0 \left(\nabla_{\mathbf{x}(0)} g\right)^{-1} \nabla_{\boldsymbol{\theta}} g$

This algorithm thus achieves the computation of the gradient $\nabla_{\boldsymbol{\theta}} C$ by solving a single additional **ODE** of the same dimension as our original **ODE**. This is to be compared with the original method which was used in Chapter 2 where a system of $p \times d$ **ODEs** was required to be solved. Having presented the method a worked example will now be presented in the following subsection in order to get a better grasp on how the method works.

3.1.2 A Simple Worked Example

A simple closed-form problem will now be considered to demonstrate the Adjoint Method. This example is taken from [28, see p.4]. Suppose one is interested in computing the gradient of:

$$\int_0^T x \mathrm{d}t$$

subject to the constraints $\dot{x} = bx$ and $x(0) - a = 0$. The parameters for this example are $\boldsymbol{\theta} = (a, b)^T$. The function g is $g(x(0), \boldsymbol{\theta}) = x(0) - a$, the function h is $h(x, \dot{x}, \boldsymbol{\theta}, t) = \dot{x} - bx$ and the function f is $f(x, \boldsymbol{\theta}, t) = x$. The steps in Algorithm 3 are now followed:

1. Integrating the **ODE** gives $x(t) = ae^{bt}$, where the initial condition has been used.

2. The following can be computed: $\partial_x f = 1$, $\partial_x h = -b$, $\partial_{\dot{x}} h = 1$ and so the adjoint **ODE** which must be solved is:

   $$1 - b\lambda - \dot{\lambda} = 0$$

   subject to the initial condition $\lambda(T) = 0$. Solving this backwards in time yields $\lambda(t) = b^{-1}\left(1 - e^{b(T-t)}\right)$.

3. The following can be computed: $\nabla_{\boldsymbol{\theta}} f = (0, 0)$, $\nabla_{\boldsymbol{\theta}} h = (0, -x)$, $\partial_{x(0)} g = 1$, and $\nabla_{\boldsymbol{\theta}} g = (-1, 0)$. Thus, the first component of the gradient is:

   $$\lambda(0) \cdot (1)^{-1} \cdot (-1) = b^{-1}(e^{bT} - 1);$$

   while the second component is,

   $$\int_0^T -\lambda x \mathrm{d}t = \int_0^T b^{-1}(e^{b(T-t)} - 1) a e^{bt} \mathrm{d}t = \frac{a}{b} T e^{bT} - \frac{a}{b^2}(e^{bT} - 1)$$

As a check that the method works one can compute the total derivative directly by first computing the objective function as so:

$$\int_0^T x\mathrm{d}t = \int_0^T ae^{bt}dt = \frac{a}{b}(e^{bT} - 1).$$

Taking the gradient of this expression with respect to $\boldsymbol{\theta} = (a, b)^T$ yields the same results obtained above.

Having gone through a simple worked example the next step is to discuss how the Adjoint Method can be applied to the types of parameter estimation problems considered in this project; this is done in the next subsection.

3.1.3 Application of the Adjoint Method to Computing the MMD gradient

In this subsection the Adjoint Method will be adapted to help with the parameter estimation problem discussed in Chapter 2. In particular it shall be shown how the optimization problem (2.1) can be moulded to resemble the problem (3.1).

The vector of unknown parameters will still be $\boldsymbol{\theta}$. For each simulated trajectory the governing **ODE** and the corresponding initial condition is given by (1.1). In order to fit this in the framework of (3.1) the individual state vectors $\{\mathbf{x}^{(i)}\}_{i=1}^n$ can be stacked on top of each other as so:

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}^{(1)} \\ \vdots \\ \mathbf{x}^{(n)} \end{pmatrix}$$

By doing so the new state vector $\mathbf{x}$ satisfies the following **IVP**:

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \boldsymbol{\phi}(\mathbf{x}(t); \boldsymbol{\xi}), \quad \mathbf{x}(0) = \boldsymbol{\epsilon} \tag{3.6}$$

where $\boldsymbol{\xi}$, $\boldsymbol{\epsilon}$, $\boldsymbol{\phi}$ are defined by:

$$\boldsymbol{\xi} = \begin{pmatrix} \boldsymbol{\xi}^{(1)} \\ \vdots \\ \boldsymbol{\xi}^{(n)} \end{pmatrix}, \quad \boldsymbol{\epsilon} = \begin{pmatrix} \boldsymbol{\epsilon}^{(1)} \\ \vdots \\ \boldsymbol{\epsilon}^{(n)} \end{pmatrix}, \quad \boldsymbol{\phi}(\mathbf{x}(t); \boldsymbol{\xi}) = \begin{pmatrix} \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}^{(1)}(t); \boldsymbol{\xi}^{(1)}) \\ \vdots \\ \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}^{(n)}(t); \boldsymbol{\xi}^{(n)}) \end{pmatrix}$$

The **ODE** can be specified in the implicit form in (3.1) by defining h to be:

$$h(\mathbf{x}, \dot{\mathbf{x}}, \boldsymbol{\theta}, t) = \dot{\mathbf{x}} - \boldsymbol{\phi}(\mathbf{x}; \boldsymbol{\xi}) \tag{3.7}$$

while the initial condition is specified by $g = 0$ where g is defined as:

$$g(\mathbf{x}(0), \boldsymbol{\theta}) = \mathbf{x}(0) - \boldsymbol{\epsilon} \tag{3.8}$$

The only thing which remains to be explicitly specified is the cost function C. In (2.1) the squared **MMD** was the cost function which needed to be minimised. In Algorithms 1 and 2 of Chapter 2 the U-statistic approximation (2.5) was used a proxy for the squared **MMD** since this can not in general be computed analytically. Since it is this proxy which

the algorithm attempts to minimize it shall be taken as the cost function C in (3.1), i.e. C will be defined as follows:

$$C(\mathbf{x}, \boldsymbol{\theta}) = \frac{1}{n(n-1)} \sum_{i \neq j} k\left(\mathbf{v}^{(i)}, \mathbf{v}^{(j)}\right) - \frac{2}{mn} \sum_{i=1}^{n} \sum_{j=1}^{m} k\left(\mathbf{v}^{(i)}, \tilde{\mathbf{v}}^{(j)}\right) + \frac{1}{m(m-1)} \sum_{i \neq j} k\left(\tilde{\mathbf{v}}^{(i)}, \tilde{\mathbf{v}}^{(j)}\right) \tag{3.9}$$

where the $\{\mathbf{v}^{(i)}\}_{i=1}^{n}$ and $\{\tilde{\mathbf{v}}^{(j)}\}_{j=1}^{m}$ are defined by,

$$\mathbf{v}^{(i)} = \begin{pmatrix} \mathbf{x}^{(i)}(t_0) \\ \vdots \\ \mathbf{x}^{(i)}(t_{N-1}) \end{pmatrix}, \quad \tilde{\mathbf{v}}^{(j)} = \begin{pmatrix} \tilde{\mathbf{x}}^{(j)}(t_0) \\ \vdots \\ \tilde{\mathbf{x}}^{(j)}(t_{N-1}) \end{pmatrix}$$

i.e. $\mathbf{v}^{(i)}$ is the vector of observations of the ith simulated state on the time-grid, $0 = t_0 < t_1 < \cdots < t_{N-1} = T$, upon which we are observing the data and $\tilde{\mathbf{v}}^{(j)}$ is the vector of observations of the jth true state on the time-grid.

The cost function C can not easily be written in the form given in (3.1). It is thus necessary to slightly modify the derivation of the Adjoint Method given in subsection 3.1.1. The only difference will be how the cost function term is dealt with. First, one can note that the cost function does not depend explicitly on the parameters $\boldsymbol{\theta}$ and so one can write $C(\mathbf{x}, \boldsymbol{\theta}) = C(\mathbf{x})$. The next thing to note is that the cost function only depends on the state vector $\mathbf{x}$ on the time grid $t_0 < \cdots < t_{N-1}$ and so one can write $C \equiv C(\mathbf{x}(t_0), \ldots, \mathbf{x}(t_{N-1}))$. So instead of the Lagrangian given by (3.2) one instead has:

$$\mathcal{L} \equiv C(\mathbf{x}(t_0), \ldots, \mathbf{x}(t_{N-1})) + \int_0^T \left[\boldsymbol{\lambda}^T h(\mathbf{x}, \dot{\mathbf{x}}, \boldsymbol{\theta}, t)\right] \mathrm{d}t + \boldsymbol{\mu}^T g(\mathbf{x}(0), \boldsymbol{\theta}) \tag{3.10}$$

Thus, the derivation in subsection 3.1.1 follows through almost exactly the same except for the first term in (3.10). Taking the gradient[3] of this term w.r.t the parameters $\boldsymbol{\theta}$ yields, via the chain rule:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} C &= \sum_{r=0}^{N-1} \left(\nabla_{\mathbf{x}(t_r)} C\right) \nabla_{\boldsymbol{\theta}} \mathbf{x}(t_r) \\ &= \int_0^T \left[\sum_{r=0}^{N-1} \left(\nabla_{\mathbf{x}(t_r)} C\right) \delta(t - t_r)\right] \nabla_{\boldsymbol{\theta}} \mathbf{x}(t) \mathrm{d}t \end{aligned}$$

where $\delta(\cdot)$ denotes the Dirac Delta function here. Having written the gradient as an integral of a function times $\nabla_{\boldsymbol{\theta}} \mathbf{x}(t)$ the remainder of the derivation in subsection 3.1.1 can be repeated and one can obtain the corresponding adjoint equation to be:

$$\sum_{r=0}^{N-1} \left(\nabla_{\mathbf{x}(t_r)} C\right) \delta(t - t_r) + \boldsymbol{\lambda}^T \left(\nabla_{\mathbf{x}} h - \mathrm{d}_t \nabla_{\dot{\mathbf{x}}} h\right) - \dot{\boldsymbol{\lambda}}^T \nabla_{\dot{\mathbf{x}}} h = 0$$

Utilizing the definition of h one can thus obtain the adjoint **ODE** to be:

$$\sum_{r=0}^{N-1} \left(\nabla_{\mathbf{x}(t_r)} C\right) \delta(t - t_r) - \boldsymbol{\lambda}^T \left(\nabla_{\mathbf{x}} \boldsymbol{\phi}\right) - \dot{\boldsymbol{\lambda}}^T = 0 \tag{3.11}$$

[3]This is not the gradient which takes into account the constraints $h = 0$ and $g = 0$ and just corresponds to taking the gradient of f w.r.t. $\boldsymbol{\theta}$ in the derivation in subsection 3.1.1.

together with the initial condition $\boldsymbol{\lambda}(T) = 0$. This **ODE** can be coupled with the **ODE** $h = 0$, using the observed state $\mathbf{x}(T)$ as the corresponding initial condition for $\mathbf{x}$ and solved backwards to give the adjoint vector $\boldsymbol{\lambda}$. Once this adjoint vector is obtained the gradient of the cost is given by:

$$\nabla_{\boldsymbol{\theta}} C = -\int_0^T \boldsymbol{\lambda}^T \nabla_{\boldsymbol{\theta}} \boldsymbol{\phi} \mathrm{d}t + \boldsymbol{\lambda}^T \nabla_{\boldsymbol{\theta}} g \tag{3.12}$$

In the numerical experiments which shall follow the initial condition $\boldsymbol{\epsilon}$ will be independent of $\boldsymbol{\theta}$ and as such the second term in (3.12) will vanish. Before presenting the numerical experiments demonstrating the Adjoint Method in the next chapter four brief remarks will be made:

1. In the method described above the adjoint **ODE** is coupled with the state vector **ODE** and solved backwards in time. For the **ODE** models considered in this project integrating backwards in time using as initial condition the final observation should theoretically yield the same state trajectory $\mathbf{x}(t)$ as obtained via integrating forwards the **IVP**. However, there can often be numerical issues which mean that integrating backwards in time yields ***drastically*** different results. In other words, the numerical **ODE** integrator might not be time-reversible. In future work this issue should be investigated and more advanced numerical **ODE** solvers which are time-reversible should be considered. For the numerical experiments presented here this was not an issue.

2. The forcing terms involving Dirac Deltas which appear in the adjoint **ODE** (3.11) essentially mean that this **ODE** must be solved separately on sub-intervals starting with (t_{N-1}, t_{N-2}) and working backwards to (t_1, t_0); on each sub-interval the term $\nabla_{\mathbf{x}(t_r)} C$ acts as the initial condition and is then removed from consideration, leaving the unforced **ODE** to be solved on each sub-interval. To illustrate this more clearly we start with the first sub-interval (t_{N-1}, t_{N-2}) and set $\boldsymbol{\lambda}(t_{N-1}) = 0$ (recall $t_{N-1} = T$). However, we shall pass $\nabla_{\mathbf{x}(t_{N-1})} C$ as the initial condition for this sub-interval to the numerical **ODE** solver (*note: this initial condition is NOT stored in the array holding the adjoint vector*) which will solve the unforced **ODE**. On the next sub-interval (t_{N-2}, t_{N-3}) the term $\nabla_{\mathbf{x}(t_{N-2})} C$ is passed to the numerical **ODE** solver as the initial condition and again the unforced **ODE** is solved. This process is repeated until we reach the last sub-interval (t_1, t_0).

3. The term $\nabla_{\mathbf{x}} \boldsymbol{\phi}$ has a block diagonal structure due to the form of $\mathbf{x}$ and $\boldsymbol{\phi}$. This structure is shown below:

$$\nabla_{\mathbf{x}} \boldsymbol{\phi} = \begin{pmatrix} \nabla_{\mathbf{x}^{(1)}} \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}^{(1)}; \boldsymbol{\xi}^{(1)}) & & & \\ & \nabla_{\mathbf{x}^{(2)}} \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}^{(2)}; \boldsymbol{\xi}^{(2)}) & & \\ & & \ddots & \\ & & & \nabla_{\mathbf{x}^{(n)}} \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x}^{(n)}; \boldsymbol{\xi}^{(n)}) \end{pmatrix}$$

   So the unforced adjoint **ODE** which will be solved on each sub-interval can be seen to decouple into n 'independent' systems. Thus, on each sub-interval the solution of these n systems can be done in parallel leading to computational speed-ups.

4. Once the adjoint vector $\boldsymbol{\lambda}$ has been computed the integral in (3.12) must be calculated. Since the integrand is only known on a discrete time-grid a simple Riemann sum approximation of the integral will be made using these values.

A brief discussion will be made in the next subsection on the computational cost of the Adjoint Method.

3.1.4 Computational Cost of the Adjoint Method

Following the derivation given in the preceding subsections one can see that the Adjoint Method involves, for each simulated system, the solution of a system of **ODEs** of dimension $d + d = \mathcal{O}(d)$. Thus, the dimension of the new system of **ODEs** which needs to be solved for each simulated system is independent of the number of parameters. This is to be compared to the system needed to be solved in Algorithms 1 and 2 of Chapter 2 which had dimension $d + p \times d$, i.e. which was dependent on the number of parameters. Algorithm 1 also involved $\mathcal{O}((n^2+nm)dp)$ multiplications at each iteration of the gradient descent procedure. In contrast, if one uses the Adjoint Method to obtain the gradient of the cost at each iteration the number of multiplications required is of order $\mathcal{O}(ndp)$. While the Adjoint Method is computationally faster (theoretically) it is still an open question of whether this approach can be used to compute the Riemann Metric Tensor as well so as to yield an adjoint-method natural gradient descent algorithm. This is an interesting question which may be researched further at a later stage.

Chapter 4

Numerical Experiments

In this chapter we will present our numerical experiments. These were coded in Python 3 and the GitHub page mentioned in the Abstract contains Jupyter Notebooks for these experiments.

4.1 Numerical Experiment 1: The Schnakenberg Model

In this section we shall present the first numerical experiment we undertook which will demonstrate Algorithms 1 and 2 presented in section 2.1. We will first set up the problem, then we will have a brief discussion on some coding aspects of our implementation of the algorithms, followed finally by a presentation of our results.

4.1.1 Problem Setup

For our first numerical experiment we shall focus on a particular system of first order differential equations called the Schnakenberg Model [29] which is a system of two coupled first order differential equations given below:

$$\begin{aligned} \dot{x}_1 &= \theta_1 x_1^2 x_2 + \theta_2 - \theta_3 x_1 \\ \dot{x}_2 &= -\theta_1 x_1^2 x_2 + \theta_4 \end{aligned} \tag{4.1}$$

This is a well-known model which models a chemical reaction between two species; x_1 is the concentration of the first and x_2 is the concentration of the second. The parameters for this system are $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3, \theta_4)^T$ all of which are assumed to be positive, i.e. $\theta_i > 0$ for $i = 1, 2, 3, 4$. The dimension of our parameter space is thus $p = 4$. We will assume that the parameters $\boldsymbol{\theta}$ are unknown and that our initial conditions are noisy. To be more specific we make the following assumptions about our initial conditions: $x_1(0) = \epsilon_1$, $x_2(0) = \epsilon_2$ where the distribution of ϵ_q is a truncated normal distribution with mean $\mu_q > 0$ and standard deviation $\sigma_q > 0$ truncated to the interval $(0, \infty)$ for $q = 1, 2$. Since our unknown parameters are constrained to be positive it will prove useful for the implementation of the gradient descent procedures that we reparametrize the model so as to ensure that the parameters are real-valued. Here we will choose[1] to make the reparametrization as follows: $\theta_i = e^{w_i}$ for $i = 1, 2, 3, 4$. We now have $w_i \in \mathbb{R}$ and so our parameter vector $\mathbf{w} \in \mathcal{W} = \mathbb{R}^4$.

[1]The choice of reparametrization can affect the performance of Algorithm 1 whereas Algorithm 2 is not affected by this choice due to the natural gradient adaption (at least theoretically).

We can now explicitly formulate the generative model in terms of the setup in section 1.1. Our vector $\mathbf{x}^{(i)}$ is now the vector $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)})^T$. $\boldsymbol{\xi}^{(i)}$ is dropped and $\boldsymbol{\epsilon}^{(i)} = (\epsilon_1^{(i)}, \epsilon_2^{(i)})^T$ where for each i the components of $\boldsymbol{\epsilon}^{(i)}$ are independent and distributed according to the truncated normal distributions mentioned above. We thus have $\mathcal{U} = \mathbb{R}^2$ and $\mathbb{U}$ is the probability measure corresponding to a 2-dimensional random vector of independent components the q-th of which has a truncated normal distribution on $(0, \infty)$ with mean μ_q and standard deviation σ_q for $q = 1, 2$. The observation function $\mathbf{h}$ is the identity here and the white noise terms $\mathbf{e}^{(i)}$ are dropped. The function on the RHS of the **ODE** is given by:

$$f_{\mathbf{w}}(\mathbf{x}) = \begin{pmatrix} e^{w_1} x_1^2 x_2 + e^{w_2} - e^{w_3} x_1 \\ -e^{w_1} x_1^2 x_2 + e^{w_4} \end{pmatrix} \tag{4.2}$$

We shall assume that given an initial condition for $\mathbf{x}^{(i)}$ we can solve the Schnakenberg model via a numerical **ODE** solver and that we observe the solution on a time grid $0 = t_0 < t_1 < \cdots < t_{N-1} = T$ for some fixed $T \in (0, \infty)$. The codomain of our generator is thus $\mathcal{X} = \mathbb{R}^{2N}$ as we have 2 components in the state vector and we observe each on a time grid with N grid-points. We thus have that given the sampled $u_i \sim \mathbb{U}$ where $u_i = \boldsymbol{\epsilon}^{(i)}$ the generator outputs:

$$y_i = G_{\boldsymbol{w}}(u_i) = \begin{pmatrix} x_1^{(i)}(t_0) \\ x_1^{(i)}(t_1) \\ \vdots \\ x_1^{(i)}(t_{N-1}) \\ x_2^{(i)}(t_0) \\ x_2^{(i)}(t_1) \\ \vdots \\ x_2^{(i)}(t_{N-1}) \end{pmatrix} \in \mathcal{X}$$

We note that we have chosen to stack the observations by first listing the first components followed by the second components. Our goal as discussed earlier is to infer the true parameter vector $\boldsymbol{\theta}_0$, which is related to $\mathbf{w}_0$ via the reparametrization above, by using the samples $\{y_i\}_{i=1}^m$. Before we can begin the parameter estimation procedure we must now consider the choice of kernel function k which will specify the squared **MMD**. We choose to take k to be the Gaussian kernel:

$$k(x, y) = \exp\left(-\sigma \|x - y\|^2\right) \tag{4.3}$$

where $\|\cdot\|$ is the standard Euclidean norm on $\mathcal{X} = \mathbb{R}^{2N}$ and σ is a parameter controlling the bandwidth of the Gaussian. This choice of kernel is made as it is a characteristic kernel (as demonstrated in section 1.3) and it is also relatively easy to work with. We note that due to the nature of the Euclidean norm in (4.3) the order in which we stack the observations in the y_i's does not play a role in the results.

Having set up all our notation the only thing remaining before we have everything ready for the implementation is to derive everything which is needed to compute the U-statistic approximations of the gradient of the squared **MMD** and of the metric tensor. We do this now. First we need to derive an expression for the Jacobian $\nabla_{\mathbf{w}} G_{\mathbf{w}}$. Due to

how we stacked the components in $G_{\mathbf{w}}(u)$ this Jacobian takes the following form:

$$\nabla_{\mathbf{w}} G_{\mathbf{w}}(u) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ \partial_{w_1} x_1(t_1) & \partial_{w_2} x_1(t_1) & \partial_{w_3} x_1(t_1) & \partial_{w_4} x_1(t_1) \\ \partial_{w_1} x_1(t_2) & \partial_{w_2} x_1(t_2) & \partial_{w_3} x_1(t_2) & \partial_{w_4} x_1(t_2) \\ \vdots & \vdots & \vdots & \vdots \\ \partial_{w_1} x_1(t_{N-1}) & \partial_{w_2} x_1(t_{N-1}) & \partial_{w_3} x_1(t_{N-1}) & \partial_{w_4} x_1(t_{N-1}) \\ 0 & 0 & 0 & 0 \\ \partial_{w_1} x_2(t_1) & \partial_{w_2} x_2(t_1) & \partial_{w_3} x_2(t_1) & \partial_{w_4} x_2(t_1) \\ \partial_{w_1} x_2(t_2) & \partial_{w_2} x_2(t_2) & \partial_{w_3} x_2(t_2) & \partial_{w_4} x_2(t_2) \\ \vdots & \vdots & \vdots & \vdots \\ \partial_{w_1} x_2(t_{N-1}) & \partial_{w_2} x_2(t_{N-1}) & \partial_{w_3} x_2(t_{N-1}) & \partial_{w_4} x_2(t_{N-1}) \end{pmatrix} \tag{4.4}$$

where the two rows of 0's in the matrix above are due to the fact that the initial conditions for the state vectors are independent of the parameters. As mentioned in section 2.2 the Jacobian can be determined by finding the system of equations governing the time evolution of the first-order sensitivity coefficients. We now derive this system for the Schnakenberg model. We compute,

$$\nabla_{\mathbf{w}} f_{\mathbf{w}}(\mathbf{x}) = \begin{pmatrix} e^{w_1} x_1^2 x_2 & e^{w_2} & -e^{w_3} x_1 & 0 \\ -e^{w_1} x_1^2 x_2 & 0 & 0 & e^{w_4} \end{pmatrix} \tag{4.5}$$

$$\nabla_{\mathbf{x}} f_{\mathbf{w}}(\mathbf{x}) = \begin{pmatrix} 2e^{w_1} x_1 x_2 - e^{w_3} & e^{w_1} x_1^2 \\ -2e^{w_1} x_1 x_2 & -e^{w_1} x_1^2 \end{pmatrix} \tag{4.6}$$

Plugging (4.5) and (4.6) into the system of equations (2.12) yields the following system which must be solved:

$$\begin{aligned} \dot{x}_1 &= e^{w_1} x_1^2 x_2 + e^{w_2} - e^{w_3} x_1 \\ \dot{x}_2 &= -e^{w_1} x_1^2 x_2 + e^{w_4} \\ \dot{g}_{11} &= e^{w_1} x_1^2 x_2 + g_{11}(2e^{w_1} x_1 x_2 - e^{w_3}) + g_{21} e^{w_1} x_1^2 \\ \dot{g}_{12} &= e^{w_2} + g_{12}(2e^{w_1} x_1 x_2 - e^{w_3}) + g_{22} e^{w_1} x_1^2 \\ \dot{g}_{13} &= -e^{w_3} x_1 + g_{13}(2e^{w_1} x_1 x_2 - e^{w_3}) + g_{23} e^{w_1} x_1^2 \\ \dot{g}_{14} &= g_{14}(2e^{w_1} x_1 x_2 - e^{w_3}) + g_{24} e^{w_1} x_1^2 \\ \dot{g}_{21} &= -e^{w_1} x_1^2 x_2 - 2g_{11} e^{w_1} x_1 x_2 - g_{21} e^{w_1} x_1^2 \\ \dot{g}_{22} &= -2g_{12} e^{w_1} x_1 x_2 - g_{22} e^{w_1} x_1^2 \\ \dot{g}_{23} &= -2g_{13} e^{w_1} x_1 x_2 - g_{23} e^{w_1} x_1^2 \\ \dot{g}_{24} &= e^{w_4} - 2g_{14} e^{w_1} x_1 x_2 - g_{24} e^{w_1} x_1^2 \end{aligned} \tag{4.7}$$

where the g_{ij} are the components of the sensitivity matrix g (*note*: we call the sensitivity matrix g instead of j here). This system will then be solved with a numerical **ODE** solver to yield solutions on the time grid we are considering, which will allow us to compute the matrix in (4.4).

Having detailed a method for computing the Jacobian $\nabla_{\mathbf{w}} G_{\mathbf{w}}$ we now move onto deriving an expression for $\nabla_1 k(x, y)$. We work component-wise first:

$$\partial_{x_q} k(x, y) = -2\sigma(x_q - y_q) k(x, y)$$

We can thus see that we have,

$$\nabla_1 k(x,y) = -2\sigma k(x,y)(x-y) \tag{4.8}$$

The final piece we need is the matrix $\nabla_2 \nabla_1 k(x,y)$. We proceed component-wise as before:

$$\begin{aligned}
\partial_{y_p} \partial x_q k(x,y) &= \partial_{y_p}(-2\sigma(x_q - y_q)k(x,y)) \\
&= 2\sigma\delta_{pq}k(x,y) - 2\sigma(x_q - y_q)\partial_{y_p}k(x,y) \\
&= 2\sigma\delta_{pq}k(x,y) - 2\sigma(x_q - y_q)(2\sigma(x_p - y_p)k(x,y)) \\
&= 2\sigma k(x,y)(I - 2\sigma(x-y)(x-y)^T)_{pq}
\end{aligned}$$

where δ_{pq} is the Kroenecker delta and I is a $d \times d$ identity matrix. We can thus see that we have,

$$\nabla_2 \nabla_1 k(x,y) = 2\sigma k(x,y)(I - 2\sigma(x-y)(x-y)^T) \tag{4.9}$$

We now have all the various bits needed to implement Algorithms 1 and 2 from section 2.1. We will now move onto the next subsection where we make some brief remarks on some coding aspects of our implementation of the algorithms.

4.1.2 Some Comments on the Implementation

In this subsection we make some brief remarks regarding our implementation of the Algorithms 1 and 2 for the Schnakenberg model. For this numerical experiment we choose to make 6 brief remarks regarding some aspects of the code we think are worth mentioning:

1. In order to solve the combined system of **ODEs** (4.7) we utilized the numerical **ODE** solver `odeint` from the Python package `scipy.integrate`.

2. When simulating the n trajectories from the Schnakenberg model we originally used a `for` loop to iterate over all the initial conditions and solve each **IVP** separately. This approach however is not the most efficient, especially in light of the fact that each simulated trajectory is independent from the other. We thus utilized the functions `Parallel` and `delayed` from the Python package `joblib` [30] together with the Python package `multiprocessing` to utilize all the CPU cores available to us to parallelize the simulation of the independent trajectories.

3. For both algorithms it will be necessary at each iteration to compute the kernel over all pairs of points from both the simulated set and true data set together with all pairs between the two sets. In order to carry out these computations both efficiently and simply we utilized the function `cdist` from the Python package `scipy.spatial.distance`. This function essentially takes in two arrays and then returns an array of all pairwise distances between the points specified by the rows of the two arrays. This allows us to use vectorization to quickly evaluate the kernel over all these pairs and then sum all the terms up according to (2.5).

4. In each iteration it is necessary to compute expressions like (2.4) and (2.8). These expressions can be expressed in Einstein summation convention and so for simplicity of coding as well as computational efficiency we utilize the function `einsum` found in the Python package `numpy` to compute these terms.

5. In Algorithm 2 it is necessary to invert the U-statistic approximation F_U at each iteration. This matrix is theoretically symmetric and invertible, but due to numerical issues can seem both non-symmetric and singular which will cause the code to break or to not give sensible results. To get around these issues we symmetrize the computed U-statistic by averaging it with its transpose and we then add a small positive diagonal perturbation term to it to ensure its invertiblity.

6. In our code for both Algorithms we include a tolerance parameter which will stop the gradient descent algorithm if the next parameter vector is within this tolerance from the previous vector (where the distance is computed using the Euclidean norm). We however, decide to set this tolerance to 0 as such a stopping criterion is very sensitive to the scale of the particular problem and so is hard to tune.

4.1.3 Results

We will now present our results in this subsubsection. We chose to set the true parameter vector for the model to be $\boldsymbol{\theta}_0 = (1, 2, 3, 4)^T$. Taking the natural logarithm of each component yields $\mathbf{w}_0$. We tried to recover this vector. The time grid upon which we make observations was determined by setting $T = 1$, $N = 10$ and taking a uniformly spaced grid. For the initial conditions we set the parameters of the truncated normal distributions to be $\mu_q = 1, \sigma_q = 0.5$ for both $q = 1, 2$. For the implementations of both Algorithms 1 and 2 we decided to take $m = 1000$ true data points and $n = 100$ simulated data points at each iteration of the gradient descent. Our step-size sequence for both algorithms was taken to be a constant sequence with entries 0.1 (i.e. we took a constant step-size of 0.1). We also chose to make no more than 4000 steps in our gradient descent algorithms. As discussed after presenting Algorithm 1 we made multiple runs from different starting locations. We chose the starting vectors at random; in particular we drew 30 vectors at random where each component was independently drawn from a uniform distribution on $[0.01, 10.01]$. The logarithm of each component was then taken to obtain our 30 starting points for $\widehat{\mathbf{w}}^{(0)}$. We used the same vectors for both Algorithms.

As expected, not all of the randomly chosen starting points gave results which converged to the true parameter vector within the maximum number of iterations. Some runs did converge to the truth. In general Algorithm 2 converged faster than Algorithm 1 and also converged to a vector 'closer' to the true vector. We include in Figure 4.1 on the next page plots of the results for one particular starting point, $\widehat{\mathbf{w}}^{(0)} = (2.05, -1.70, 1.82, 0.75)^T$ which corresponds to $\widehat{\boldsymbol{\theta}}^{(0)} = (7.73, 0.18, 6.19, 2.12)^T$ (each component is rounded to 2 decimal places). We include the results of both algorithms in the same plot for ease of comparison. We chose this starting point as it shows that Algorithm 2 can converge much faster to the truth than Algorithm 1. We note that within 4000 steps Algorithm 1 converges to a vector not too far from the truth yet this takes longer than the convergence of Algorithm 2 which manages to converge right at the truth in under 500 steps.

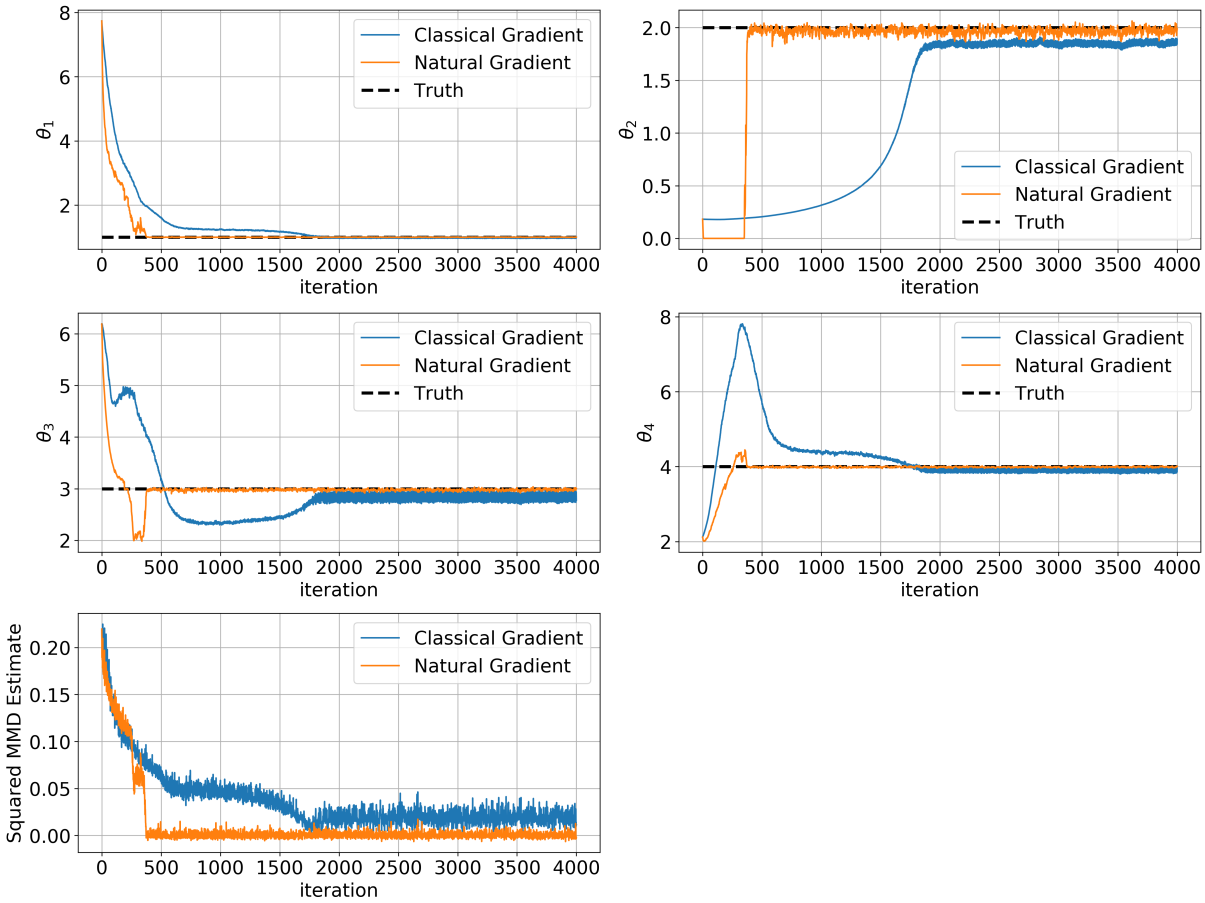

Figure 4.1: Results for a particular run starting from the vector $(7.73, 0.18, 6.19, 2.12)^T$

It might be the case that Algorithm 1 does not converge to the exact truth in this run but instead to something close to it due to the discretisation of both the **ODE** and the gradient of the squared **MMD**. However, it is important to stress that Algorithm 1 did converge closer to the true parameter vector in some of the other runs we made, though Algorithm 2 typically got closer. Even in the run presented above Algorithm 1 does not do that bad and the parameter vector which achieved the lowest score, while differing a bit from the truth, still produces trajectories whose distribution is quite similar to the true distribution. Figure 4.2 on the next page shows plots of simulated trajectories using the true vector, the resulting vector from Algorithm 1 and the one from Algorithm 2. These plots show that the distributions are quite similar.

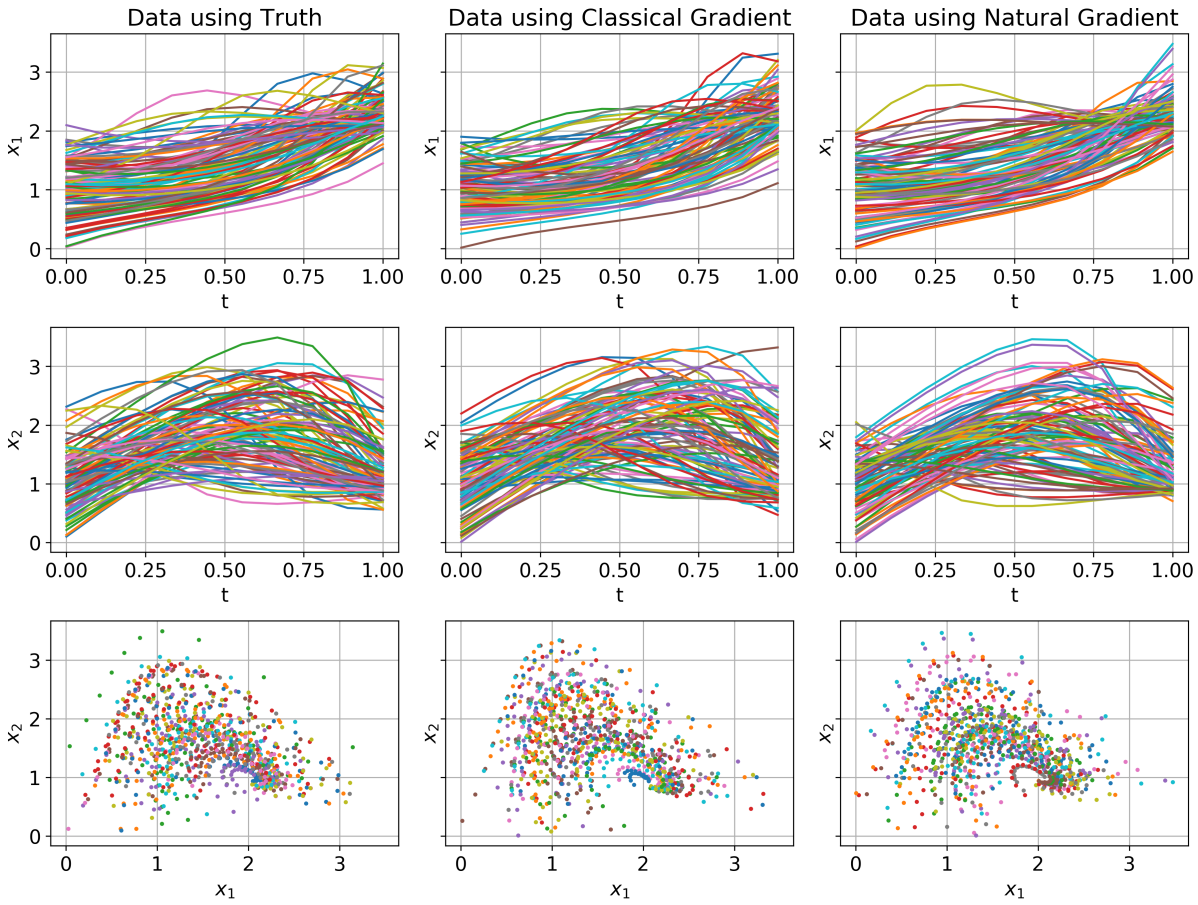

Figure 4.2: Sampled trajectories using the truth and the results from the particular run of both algorithms shown in Figure 4.1. The first row are plots of the first component of the state vector against time, the second row shows the second component against time and the third row shows phase plots.

The added computational cost of Algorithm 2 is negligible here, and the cost of the algorithm is completely dominated by the generation step. As such, the time to run both algorithms was not that different.

4.2 Numerical Experiment 2: The Schnakenberg Model

In this section the Adjoint Method shall be applied to the Schnakenberg Model which was studied in Numerical Experiment 1. This problem had $p = 4$ and so the dimension of the parameter space is not large, however it shall be presented as a check that the method works. A few brief remarks on the implementation will now be made in the next subsection.

4.2.1 Some Comments on the Implementation

In this subsection some brief remarks will be made about the implementation of the Adjoint Method for the Schnakenberg Model. For this numerical experiment two brief remarks regarding the implementation will now be made:

1. Deriving an expression for the terms $\nabla_{\mathbf{x}(t_r)}C$ in (3.11) is possible, however the derivation is quite messy and involved. This derivation was undertaken and then translated into code. The resulting algorithm was very slow as the derivatives involved a double sum and it was very difficult to avoid using a double for-loop for this. As such, the technique of Automatic Differentiation was deployed to help with this problem. The Python package PyTorch [31] was chosen to implement this. PyTorch is a Python-based scientific computing package which works with `tensors`, which are similar to NumPy's `ndarrays`, with the exception that `tensors` can utilize a GPU to speed up computations. This project will not, however utilize a GPU, but instead will rely on PyTorch's `autograd` package to automatically compute derivatives [32]. In particular, when requested PyTorch is able to track almost any operation performed on a `tensor` and at the end a simple call can be made to back-track along this operation history and compute a gradient automatically. Utilizing this functionality enables vector-Jacobian products to be computed simply and efficiently. For more details see the tutorial [33].

2. The ADAM gradient descent optimizer [19] was applied to this problem so as to produce more stable results. This optimizer works well for stochastic objective functions like the one under consideration and also has the nice property that it naturally performs a form of automatic step-size adjustment which can be different for each component of the parameter. For more details the interested reader is directed to the reference [19] mentioned above. Pseudo-code for the ADAM optimizer is included in Appendix B.

4.2.2 Results

The results obtained will now be presented. The true parameter vector was again set to be $\boldsymbol{\theta}_0 = (1, 2, 3, 4)^T$. The time grid this time around was determined by setting $T = 1,\ N = 20$ and again taking a uniformly spaced grid. The parameters of the truncated normal distributions for the initial conditions was taken to be the same as in Numerical Experiment 1. The values $m = 1000$ and $n = 100$ were again set to be the number of true and simulated data points respectively. The step-size parameter, α, for the ADAM optimizer was set to $\alpha = 0.75$ while all the other tuning parameters needed for the optimizer were set to the defaults given in [19] (see the pseudo-code given in [19, p. 2]). An upper limit of 1000 steps was set for the gradient descent. Again multiple runs from random[2] starts were performed. The results of one particular run which converged to the truth will now be presented. This run started from $\widehat{\boldsymbol{\theta}}^{(0)} = (2.59, 0.57, 3.21, 7.08)^T$ (each component is rounded to 2 decimal places). Plots of the resulting parameters and the scores are presented in Figure 4.3 on the next page.

[2]These starts were drawn randomly in same way as described in Numerical Experiment 1.

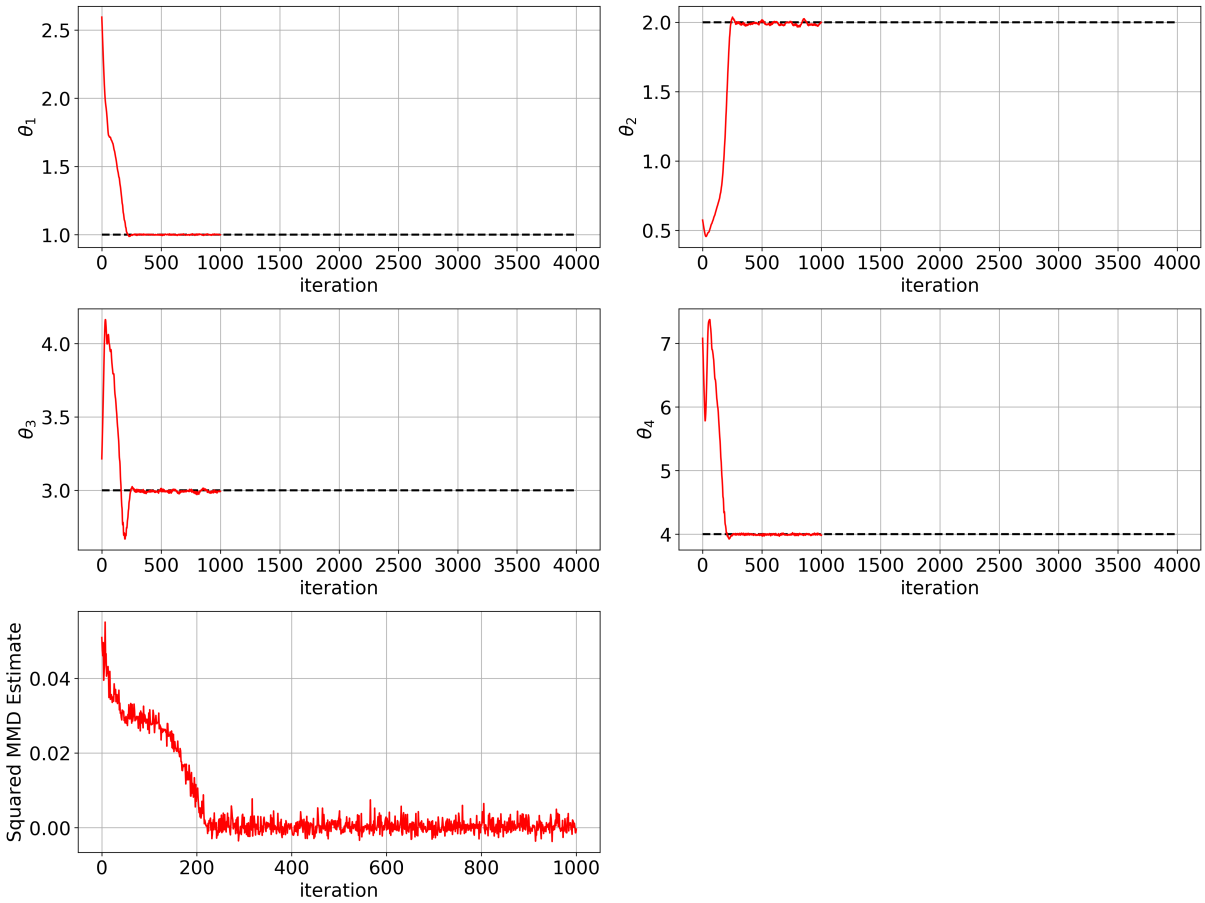

Figure 4.3: Results for a particular run starting from the vector $(2.59, 0.57, 3.21, 7.08)^T$. The dashed horizontal black lines are at the true parameter values.

It can be seen from Figure 4.3 above that the Adjoint-method gradient descent with ADAM managed to converge to the true parameter vector in around 200 iterations and it subsequently remained close to the truth for the remaining steps.

On the next page Figure 4.4 shows plots of simulated trajectories using the true vector and the resulting vector which obtained the minimum estimated squared **MMD** on this run. These plots show that the distributions are quite similar.

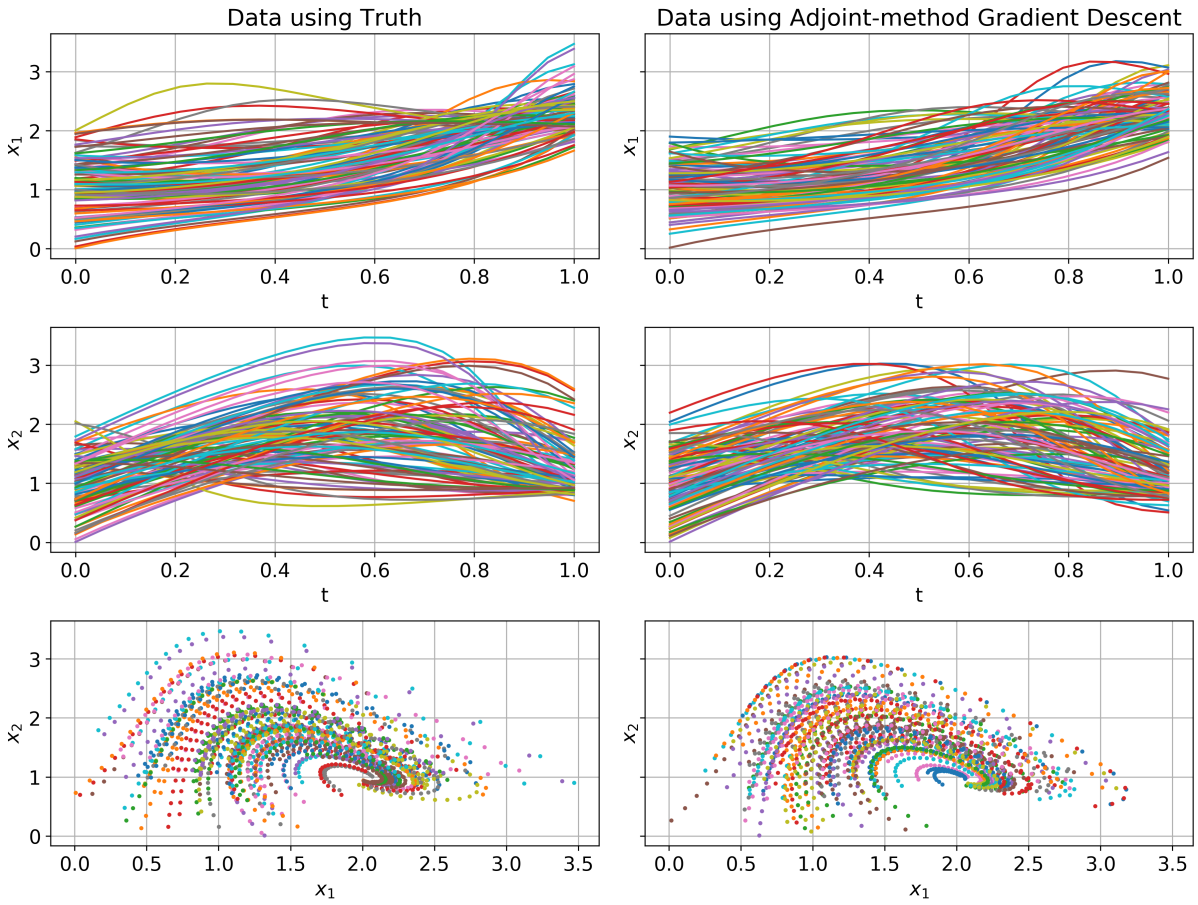

Figure 4.4: Sampled trajectories using the truth and the results from the particular run of the algorithm shown in Figure 4.3. The first row are plots of the first component of the state vector against time, the second row shows the second component against time and the third row shows phase plots.

It can thus be seen that the Adjoint-Method seems to be working and producing good results. The dimension of the parameter space for this model is not that large and so another experiment shall be considered where the paremeter space is higher dimensional. This experiment will be presented in the next section.

4.3 Numerical Experiment 3: Neural Nets

4.3.1 Problem Setup

In this section **ODEs** whose RHS are given by feed-forward neural networks shall be considered as a means of transforming distributions. To be more specific systems of first order **ODEs** of the following form will be considered:

$$\frac{\mathrm{d}\mathbf{x}(t)}{\mathrm{d}t} = \mathbf{f}(\mathbf{x}(t)) \tag{4.10}$$

where here $\mathbf{x} \in \mathbb{R}^d$ and the function $\mathbf{f} : \mathbb{R}^d \mapsto \mathbb{R}^d$ is a feed-forward neural net. Such **ODEs** have been considered in the literature before; see for instance [34] where such systems are referred to as 'Neural ODEs'. In this paper these Neural ODEs are motivated

by considering a limit where more layers and smaller step steps are taken in the following sequence of transformations:

$$\mathbf{h}_{t+1} = \mathbf{h}_t + f(\mathbf{h}_t, \theta_t)$$

where $t \in 0, \ldots, T$ and $\mathbf{h}_t \in \mathbb{R}^d$. Such sequences of transformations are often found in models such as residual networks, recurrent neural network decoders and normalizing flows. The paper discusses how the parameters of the neural net can be learned via optimizing a loss function by using an Adjoint-method which is very similar to the approach discussed in this chapter.

ODEs of the form (4.10) shall be considered as a means of transforming probability distributions. The initial condition for (4.10) shall be taken to be random: $\mathbf{x}(0) = \boldsymbol{\epsilon} \sim \mathcal{D}$ where $\mathcal{D}$ is some known distribution. The initial condition is then fed into the system (4.10) for a certain amount of time T and the resulting solution $\mathbf{x}(T)$ will yield the output by taking some (or all) of its components. By repeatedly drawing initial conditions from $\mathcal{D}$ and feeding them into (4.10) a potentially different distribution will be obtained as the output. The goal of this section will be to learn the neural net $\mathbf{f}$ which can transform the distribution $\mathcal{D}$ into a specified target distribution $\mathcal{D}_T$ at time T.

Remark. Since in this numerical experiment the observations might only involve some components of the state vector the Adjoint Method presented in subsection 3.1.3 must be slightly modified: the forcing terms involving Dirac Delta's in (3.11) is adjusted so that these forcing terms only affect the initial conditions for the components being observed. For the components which aren't being observed no forcing acts.

The neural net $\mathbf{f}$ considered here shall have one hidden layer with M nodes. The input layer connects to the hidden layer via an affine transformation, each node in the hidden layer then has an activation function applied to its output before connecting to the output layer via another affine transformation. No activation function is applied at the nodes of the output layer. This set-up is motivated by the Universal Approximation Theorem [35, see p. 11] which tells us that such a neural net can approximate arbitrarily well any given continuous function[3] provided we use enough nodes in the hidden layer.

The set up of the neural net considered above means that the net is specified by a total of $2Md + M + d$ parameters, $2Md$ of which are weights and $M + d$ of which are biases. Thus, the dimension of the parameter space for this model is $p = 2Md + M + d$.

It will now be made explicit how this model fits into the framework of the generative model discussed in section 1.1. The vector $\mathbf{x}^{(i)}$ corresponds to the ith simulated state vector. $\boldsymbol{\xi}^{(i)}$ is dropped and $\boldsymbol{\epsilon}^{(i)}$ is the ith sampled initial condition. Thus $\mathcal{U} = \mathbb{R}^d$ and $\mathbb{U} = \mathcal{D}$. The observation function $\mathbf{h}$ now picks out the necessary components of the solution of the **ODE** and the $\mathbf{e}^{(i)}$ are dropped. The function on the RHS of (1.1) is given by the neural net $\mathbf{f}$ whose structure is specified as above. The unknown parameters $\boldsymbol{\theta}$ are the collection of weights and biases needed to specify the neural net. Again it shall be assumed that given an initial condition for $\mathbf{x}^{(i)}$ it is possible to solve the **ODE** (4.10) via a numerical solver and that a collection of some (or possibly all) components of the

[3]The function does, however, have to technically be on a compact set; for our applications the input distribution will be a multivariate Gaussian, the support of which is the whole of $\mathbb{R}^d$ and so is not compact. However, the Gaussian density does decay exponentially fast and so this fact should mean that good results can be obtained using such neural nets.

output at time T will form the output from our generative model. Thus, given a sampled $u_i \sim \mathbb{U}$ where $u_i = \boldsymbol{\epsilon}^{(i)}$ the generator outputs:

$$y_i = G_{\boldsymbol{\theta}}(u_i) = (x_j^{(i)}(T))_{j \in \mathcal{A}}$$

where in the above the notation $(x_j^{(i)})_{j \in \mathcal{A}}$ is intended to denote a selection of components of $\mathbf{x}^{(i)}(T)$ where the set $\mathcal{A}$ is the collection of indices of these chosen components. Intuitively it should be easier to 'hit' the target distribution if the initial distribution is of higher dimension than the target and also if the target distribution has less entropy. For example, if we want to 'hit' a target distribution in 2 dimensions the initial distribution should be in a higher dimension say, $d = 4$, and then the first two components of the output from the neural net **ODE**, $\mathbf{x}(T)$, can be taken as the output of the generative model.

Two different target distributions will be considered. The results of these will be presented in the final two subsections of this chapter. Before this presentation some brief comments will be made on the implementation in the next subsection

4.3.2 Some Comments on the Implementation

In this subsection some brief remarks will be made about the implementation of the Adjoint Method for the neural net **ODE** model. In particular four brief remarks regarding the implementation will now be made:

1. The PyTorch package `torch.nn` will be utilised to efficiently implement the neural nets in these numerical experiments. No built-in optimizer from PyTorch is utilized in these experiments; instead the gradient descent procedure utilizing the Adjoint Method for computation of the gradient shall be implemented.

2. PyTorch's automatic differentiation facilities will again be utilized to compute the various vector-Jacobian products needed. This facility will be utilized even more in these numerical experiments as opposed to Numerical Experiment 2 as it is much more tedious to derive expressions for gradients of a neural net w.r.t. the parameters than it was for the function on the RHS of the Schnakenberg Model.

3. The ADAM optimizer was again adopted in the gradient descent procedure.

4. The use of PyTorch is not guaranteed to be completely reproducible and as such running the algorithm again can yield different results (even if the seed is set). This is an issue which warrants further investigation, which is not within the scope or time-frame of this project. This is a documented issue, see [36]. The notebooks given on GitHub thus might give different results and may need to be rerun several times to obtain good results. To counteract this there will be some files uploaded on the GitHub page with the learned parameters for the neural nets which gave good results.

4.3.3 Numerical Experiment 3.1: Gaussian Mixture

In this experiment the target distribution $\mathcal{D}_T$ is taken to be a Gaussian Mixture in 1 dimension. To be more specific in this experiment a 50/50 mixture of an $\mathcal{N}(1, 0.5^2)$ and

$\mathcal{N}(5,1)$ was set to be the target. $m = 1000$ samples are drawn from this mixture as the true data set. The initial distribution $\mathcal{D}$ was taken to be a bivariate-Gaussian distribution with independent components each of which have mean 1 and standard deviation 0.5. The neural **ODE** is solved up to time $T = 1$ using a numerical solver (`odeint`) on a uniform time grid with 10 grid points. At time $T = 1$ the first component of the state vector is observed as the output. The activation function for the nodes in the hidden layer is taken to be tanh and $M = 10$ nodes are used in the hidden layer. Thus, the dimension of the parameter space is $p = 52$, and so this experiment is a case where the parameter space is high-dimensional. The initial parameters were set randomly by drawing p i.i.d. samples from $\mathcal{N}(0,1)$. In each iteration of the gradient descent algorithm $n = 100$ samples were produced from the generative model. In Figure 4.5 below the estimated squared **MMD** is plotted against iteration number.

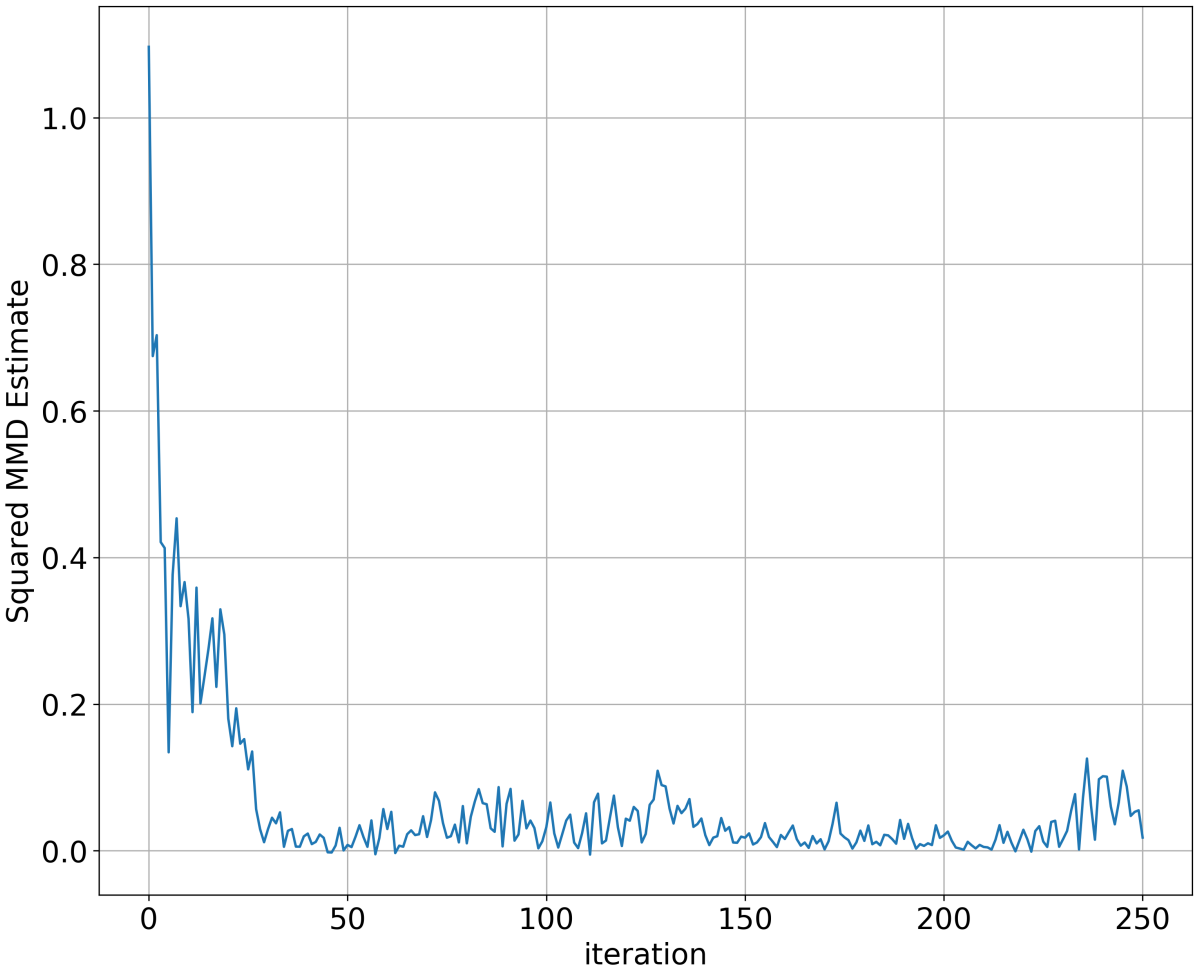

Figure 4.5: Estimated Squared MMD against iteration for Gaussian Mixture target

From the figure above it can be seen that the score has quickly converged to near zero in around 50 iterations. Even though the estimated score is close to zero it can still be the case that the target distribution is not achieved. To investigate this the parameter values which achieved the lowest score are used to simulate 1000 samples and these are compared to the true data via a histogram and density estimation, the results of which are shown in Figure 4.6 on the next page.

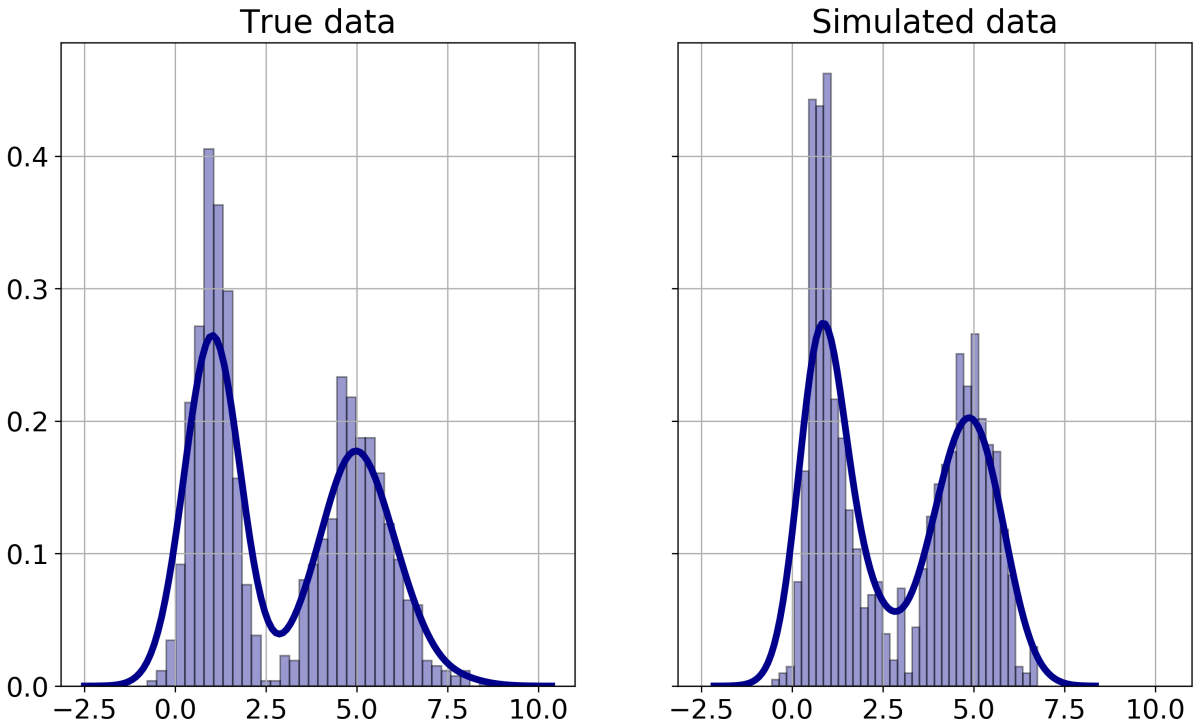

Figure 4.6: Histograms and density estimates for true data and simulated data

From the Figure 4.6 above it can be seen that the simulated samples have a distribution which is quite similar (at least visually) to the Gaussian Mixture target.

4.3.4 Numerical Experiment 3.2: The Make Circles Dataset

In this experiment the true data set (coming from the target distribution $\mathcal{D}_T$) is generated by utilizing the `make_circles` function from the Python package `sklearn` [37]. This function generates a dataset consisting of two circles; a large and a small one. There is the option of adding noise to jitter the points on these two circles. In this experiment this option is used with the level of noise set by the value 0.1 in order to obtain samples which when plotted look like a ring ($\mathcal{D}_t$ is a distribution in 2-dimensional space). $m = 1000$ samples are drawn using this function to act as the true data set. The initial distribution $\mathcal{D}$ was taken to be a multivariate Gaussian distribution in $d = 4$ dimensions with the 4 components being independent $\mathcal{N}(1, 0.5^2)$ random variables. The neural **ODE** is solved up to $T = 2$ using a numerical solver (`odeint`) on a uniform time grid with 20 grid points. At time $T = 2$ the first two components of the state vector is observed as the output. The activation function and number of nodes in the hidden layer remains the same as in Numerical Experiment 3.2. Thus, the dimension of the parameter space is $p = 94$, which is even higher than in the previous experiment. Again the initial parameters were set randomly by drawing p *i.i.d.* standard normal variables. The number of simulated points at each iteration of the gradient descent was again taken to be $n = 100$.

In Figure 4.7 on the next page the estimated squared **MMD** is plotted against iteration number. It can be seen from this Figure that the estimated squared **MMD** gets very close to zero for quite some time but then jumps up suddenly at around the 200th iteration. This might be due a stability issue and is not something which we will investi-

gate in this project.

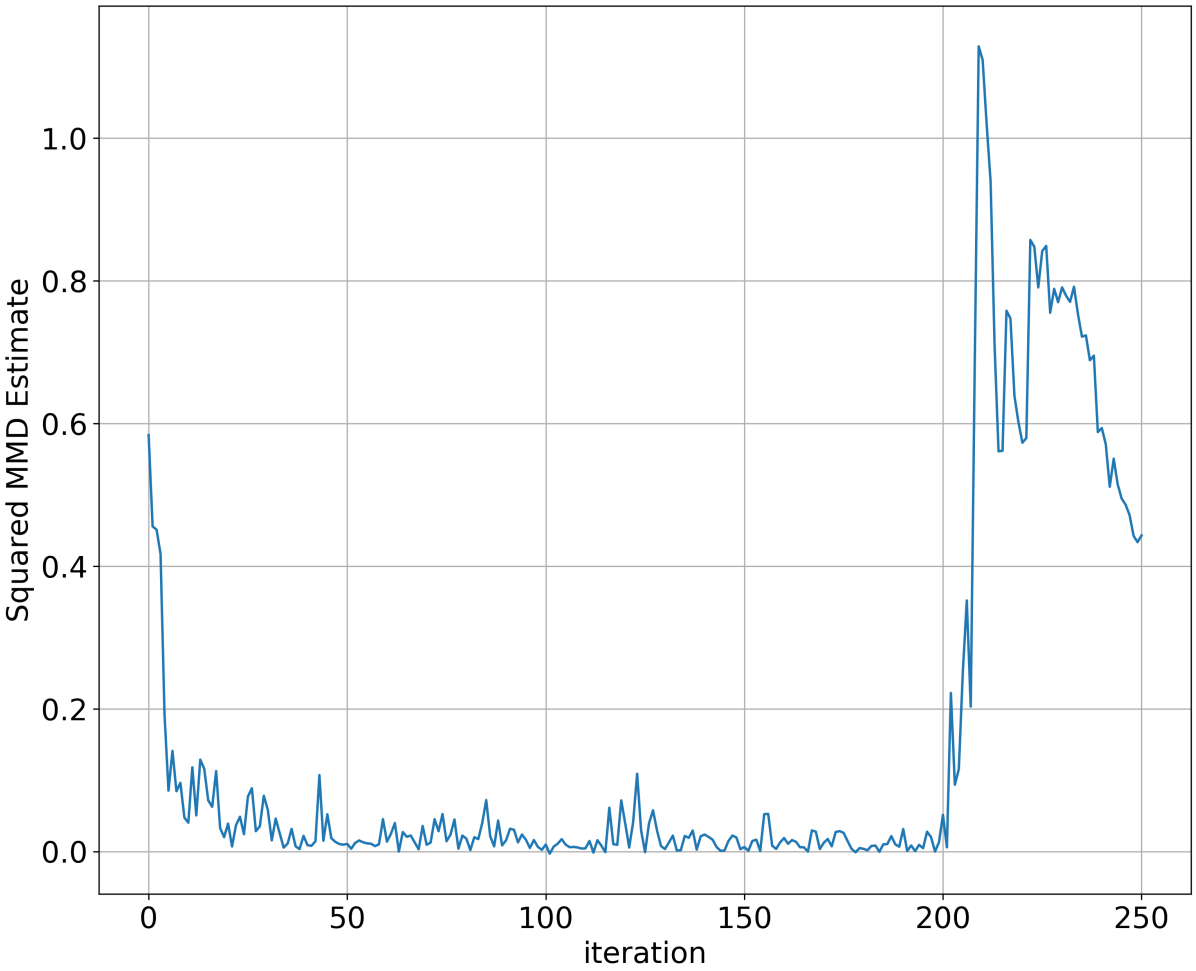

Figure 4.7: Estimated Squared MMD against iteration for noisy Make Circles target

The parameters which obtained the lowest score were then used to simulate 1000 points from the trained generative model. These points are plotted on a scatter plot next to the same plot for the true data. This is shown in Figure 4.8 on the next page. It can be seen that on this run the algorithm produced parameter values which gave resulting samples which are quite similar to the true data-set. The results are not perfect however as the ring is a bit less thick than the true data and not as symmetrical either. However, the results are promising and extra tuning of the bandwidth of the kernel might yield even better results; this is something left for further research.

Remark. It should be stressed that the issue of non-reproducibility is especially a problem for this experiment as repeated runs of the same code often gave very different samples. For instance, often the cost would get very close to 0 and the parameters corresponding to this minimum cost produced samples which formed a ring which was much thinner than the one presented here. This issue should be investigated further, for instance by increasing T to see how the **ODE** will continue to move the 'mass' or letting the gradient descent run for longer to see whether it will converge to somewhere else (the spike in the cost at the end might indicate that convergence has yet to occur).

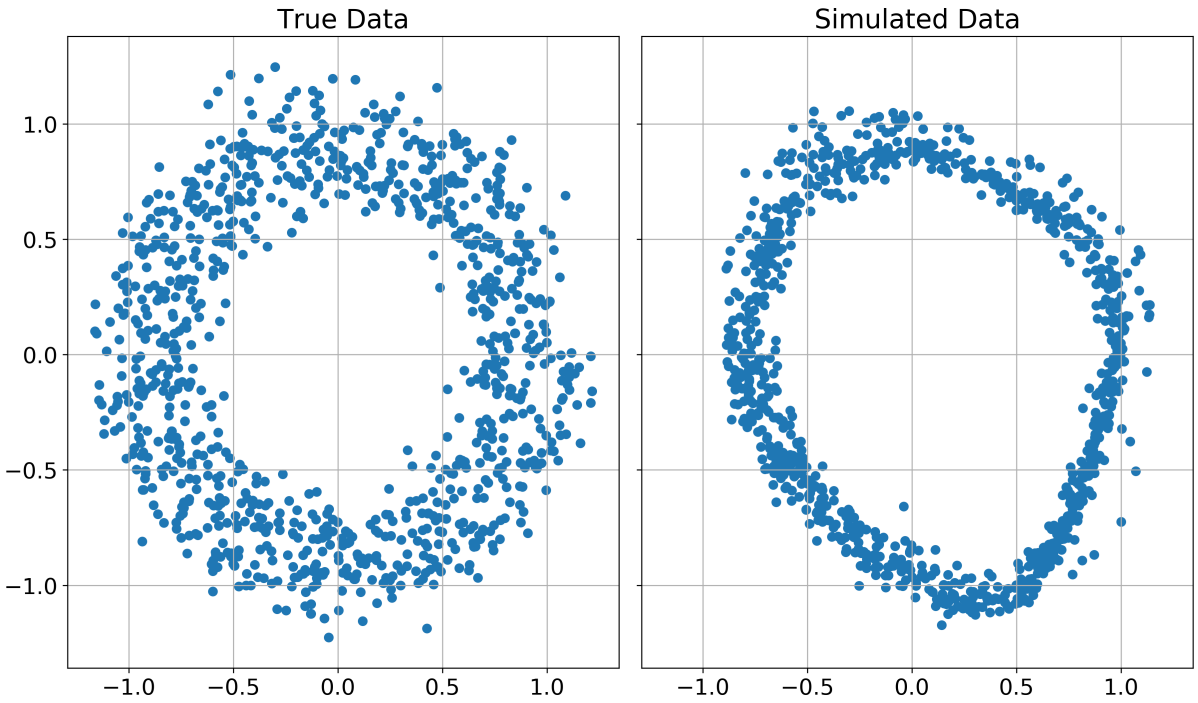

Figure 4.8: Scatter plots of the true data and the simulated data.

These experiments conclude our investigation into the use of the Adjoint Method.

Chapter 5

Conclusion

5.1 Summary

In the preceding chapters the use of **MMD** as a means of performing inference of generative statistical models based on **ODEs** was investigated. In Chapter 1 the focus was on going into the relevant background theory needed for one to have a good grasp on **MMD**, in particular the theory of Reproducing Kernel Hilbert Spaces was discussed with a concentration on the role of such spaces for **MMD**. Particular attention was also paid to characteristic kernels.

In Chapter 2 the parameter estimation problem was tackled by using the squared **MMD** as a cost function quantifying how close two distributions are. The aim was to minimize this cost (in actuality an unbiased estimator of it was minimized) in order to infer the true parameter vector. Two algorithms were presented in order to solve this optimization problem. This chapter focused especially on explaining how the Jacobian $\nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}$, needed in both algorithms, can be computed.

In Chapter 3 a detailed derivation of the Adjoint Method for time-dependent problems was presented. This method was then adapted for the parameter estimation problems dealt with in this project.

In Chapter 4 numerical experiments were presented to demonstrate the algorithms discussed in the preceding chapters. A numerical experiment on the Schnakenberg model was first presented to demonstrate Algorithms 1 and 2 from Chapter 2. Algorithm 2 performed better, albeit being more computationally expensive. Both Algorithms however were seen to become infeasible in terms of computational cost as the dimension of the parameter space increases, and this lead us to consider the use of the Adjoint Method. Two more numerical experiments were then presented to demonstrate this method; the first on the Schnakenberg model acted as a check that the method works, and the second was used to demonstrate the use of the Adjoint Method for high-dimensional parameter spaces.

5.2 Issues Faced

In this section we shall comment briefly on several of the main issues which were faced while working on this project:

1. The biggest issue faced was in tuning all of the additional parameters required by the various algorithms. For the first numerical experiment this was especially an issue. A lot of trial and error had to be performed in order to get the gradient descent algorithm to converge. The key insight which helped in getting the algorithms to work for the Schnakenberg model was to restrict the final time T to a single unit of time: for the particular parameter vector chosen $\boldsymbol{\theta}_0 = (1, 2, 3, 4)^T$ the trajectories of the **ODE** settled into an attractive fixed point. Thus, if one waited for too long every trajectory would, in the eyes of **MMD**, look similar to every other, yielding our approach almost useless. By restricting the final time T to be $T = 1$ we avoided this issue.

2. The next big issue we faced was the time the algorithms took to run. In the beginning the generator was not parallelized and so the code took a lot of time to run. Parallelisation helped massively in terms of computation time. A further time gain was made through the use of PyTorch's automatic differentiation facility.

3. The issue with reproducibility due to the use of PyTorch in Numerical Experiment 3.2 is another problem we had. A simple explanation for this issue is not apparent to us and further research into this is needed.

5.3 Potential Directions Forward

In this section we shall comment briefly on potential areas where further research can be done for this project. In particular, 2 directions appear to be most promising: performance of kernel based methods when the dimension of the state space d is large and the use of the Adjoint Method for natural gradient descent. These shall be briefly discussed in the following two subsections.

5.3.1 The Curse of Dimensionality

It is well known that in high-dimensional spaces that the Gaussian kernel becomes less effective as a similarity measure. See for instance [38]. In the numerical experiments considered here the highest dimension for the state space was $d = 40$ (in Numerical Experiment 2), and here the Gaussian kernel seemed to work well. It would be interesting to investigate the performance of the Gaussian kernel in much higher-dimensional state spaces and also in **PDE** models. The kernel might need to be modified, especially for **PDE** models, where the intrinsic geometry of the solution space will become very important. Further work will aim to address this.

5.3.2 Adjoint Method for Natural Gradient Descent

To the best of our knowledge the Adjoint Method has yet to be applied to help compute the Riemann Metric Tensor (2.7) which is needed to perform natural gradient descent. If this can be done this more powerful method will hopefully be able to be applied to problems with high-dimensional parameter spaces. This is a very interesting problem to consider and one which will be addressed in the future.

5.3.3 Other potential directions

Further areas related to this topic which might prove fruitful to research include generative models based on **PDEs** and on **SDEs**.

List of Notation and Abbreviations

The following list describes notation and abbreviations that are used frequently in this work:

$G_{\boldsymbol{\theta}}$ Generator for a generative model

$G_{\boldsymbol{\theta}}^{\#}\mathbb{U}$ Pushforward of the probability measure $\mathbb{U}$ w.r.t. the generator $G_{\boldsymbol{\theta}}$

RKHS Reproducing Kernel Hilbert Space

MMD Maximum Mean Discrepancy

$\mathcal{X}$ A non-empty set

$\mathcal{H}$ A Hilbert space

$\mathbb{R}^{\mathcal{X}}$ The vector space of all functions $f : \mathcal{X} \to \mathbb{R}$

$\langle \cdot, \cdot \rangle_{\mathcal{H}}$ Inner product in a Hilbert space $\mathcal{H}$

$\| \cdot \|_{\mathcal{H}}$ Norm associated to inner product in a Hilbert space $\mathcal{H}$

δ_x (Dirac) Evaluation functional in a Hilbert space of function $\mathcal{H} \subseteq \mathbb{R}^{\mathcal{X}}$; also used for the probability measure which places a unit point mass at x

ϕ A feature map $\phi : \mathcal{X} \to \mathcal{H}$

$\mathscr{P}$ set of all Borel probability measures on a topological space $(\mathcal{X}, \mathcal{A})$

IPM Integral Probability Metric

$\mathcal{H}_k$ A **RKHS** with reproducing kernel k

γ_k **MMD** associated with the **RKHS** $\mathcal{H}_k$

$\Pi[\mathbb{P}]$ Hilbert Space Embedding of the probability measure $\mathbb{P}$ into an **RKHS** $\mathcal{H}$ with reproducing kernel k

Appendix A

Proofs/Justifications

Proof of Proposition 1.3.2 (based on proof in [14, p. 1526]):

Proof. Suppose f is bounded. This means that there exists a constant $M > 0$ such that $|f(x)| \leq M \ \forall x \in \mathcal{X}$. Using Jensen's inequality and the preservation of ordering property of expectation, we have, for any $\mathbb{P} \in \mathscr{P}$ that,

$$\begin{aligned} |\mathbb{E}_{X\sim\mathbb{P}}[f(X)]| &\leq \mathbb{E}_{X\sim\mathbb{P}}\,|f(X)| \\ &\leq \mathbb{E}_{X\sim\mathbb{P}}\,M \\ &= M < \infty \end{aligned}$$

We thus have $\mathbb{E}_{X\sim\mathbb{P}}[f(X)] < \infty$ for all $\mathbb{P} \in \mathscr{P}$ as required.

For the converse, suppose that f is not bounded. It then follows that we can find a sequence $\{x_n\}$ in $\mathcal{X}$ such that $\lim_{n\to\infty} f(x_n) = \infty$. We can then assume w.l.o.g. (by taking a subsequence if necessary) that $f(x_n) > n^2$ for all n. It then follows that $A := \sum_{n=1}^{\infty} \frac{1}{f(x_n)} < \infty$. We can thus define a probability measure $\mathbb{P}$ on $\mathcal{X}$ by $\mathbb{P} = \frac{1}{A}\sum_{n=1}^{\infty} \frac{1}{f(x_n)}\delta_{x_n}$. We then have,

$$\begin{aligned} \mathbb{E}_{X\sim\mathbb{P}}[f(X)] &= \int_{\mathcal{X}} f(x)\mathrm{d}\mathbb{P}(x) \\ &= \frac{1}{A}\sum_{n=1}^{\infty}\frac{f(x_n)}{f(x_n)} = \infty \end{aligned}$$

This shows that if f is not bounded we can find a measure $\mathbb{P} \in \mathscr{P}$ such that $\mathbb{E}_{X\sim\mathbb{P}}[f(X)] = \infty$ as required. ∎

Justification of final equality in (2.3)

As mentioned in footnote 5 of Chapter 1 the reproducing kernel k is symmetric, i.e. $k(x, y) = k(y, x) \ \ \forall x, y \in \mathcal{X}$. Differentiating both sides of this equation w.r.t x_i (component i of vector x) yields the following equality:

$$\begin{aligned} \partial_{x_i} k(x, y) &= \frac{\partial y}{\partial x_i}^T \nabla_1 k(y, x) + \frac{\partial x}{\partial x_i}^T \nabla_2 k(y, x) \\ \partial_{x_i} k(x, y) &= \partial_{y_i} k(y, x) \\ \implies \nabla_1 k(x, y) &= \nabla_2 k(y, x) \end{aligned}$$

Using this equality in the first integral in the line preceding (2.3) yields the following:

$$\int_{\mathcal{U}}\int_{\mathcal{U}} \left(\nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(u)^T \nabla_1 k(G_{\boldsymbol{\theta}}(u), G_{\boldsymbol{\theta}}(v)) + \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(v)^T \nabla_2 k(G_{\boldsymbol{\theta}}(u), G_{\boldsymbol{\theta}}(v))\right) \mathbb{U}(\mathrm{d}u)\mathbb{U}(\mathrm{d}v) =$$
$$= \int_{\mathcal{U}}\int_{\mathcal{U}} \left(\nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(u)^T \nabla_1 k(G_{\boldsymbol{\theta}}(u), G_{\boldsymbol{\theta}}(v)) + \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(v)^T \nabla_1 k(G_{\boldsymbol{\theta}}(v), G_{\boldsymbol{\theta}}(u))\right) \mathbb{U}(\mathrm{d}u)\mathbb{U}(\mathrm{d}v)$$

Focusing on the second term in this integral we can write:

$$\begin{aligned}\int_{\mathcal{U}}\int_{\mathcal{U}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(v)^T \nabla_1 k(G_{\boldsymbol{\theta}}(v), G_{\boldsymbol{\theta}}(u))\mathbb{U}(\mathrm{d}u)\mathbb{U}(\mathrm{d}v) &= \int_{\mathcal{U}}\int_{\mathcal{U}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(v)^T \nabla_1 k(G_{\boldsymbol{\theta}}(v), G_{\boldsymbol{\theta}}(u))\mathbb{U}(\mathrm{d}v)\mathbb{U}(\mathrm{d}u) \\ &= \int_{\mathcal{U}}\int_{\mathcal{U}} \nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}(u)^T \nabla_1 k(G_{\boldsymbol{\theta}}(u), G_{\boldsymbol{\theta}}(v))\mathbb{U}(\mathrm{d}u)\mathbb{U}(\mathrm{d}v)\end{aligned}$$

where in the first equality we used Fubini's Theorem[1] to interchange the order of integration and in the second equality we relabeled the dummy integration variables. This change yields the result (2.3) as required.

[1]We are assuming here that $\nabla_{\boldsymbol{\theta}} G_{\boldsymbol{\theta}}$ is L^1 integrable and that the kernel is bounded, and these assumptions justify the use of Fubini's Theorem.

Appendix B

Miscellaneous

Pseudo-code for ADAM

We now include pseudo-code for the ADAM optimizer taken from [19, p. 2]:

Algorithm 4: ADAM optimizer. Notes: g_t^2 denotes the element-wise square here, all operations on vectors are element-wise and β_1^t, β_2^t denote β_1, β_2 to the power t.

Input: Stepsize: α, Exponential decay rates for the moment estimates: $\beta_1, \beta_2 \in [0, 1)$, Stochastic objective function with parameters θ: $f(\theta)$, Initial parameter vector: θ_0

Output: Resulting parameters θ_t

$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector);
$v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector);
$t \leftarrow 0$ (Initialize time step);
while θ_t *not converged* **do**
  $t \leftarrow t + 1$;
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t);
  $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate);
  $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate);
  $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate);
  $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate);
  $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters);
end

Bibliography

[1] Francois-Xavier Briol & Alessandro Barp & Andrew B. Duncan & Mark Girolami. *Statistical Inference for Generative Models with Maximum Mean Discrepancy*. 2019. URL: https://arxiv.org/pdf/1906.05944v1.pdf.

[2] A.A. Poyton et al. "Parameter estimation in continuous-time dynamic models using principal differential analysis". In: *Computers Chemical Engineering* 30 (2006), pp. 698–708. URL: https://doi.org/10.1016/j.compchemeng.2005.11.008.

[3] M. Benson. "Parameter fitting in dynamic models". In: *Ecological Modelling* 6 (1979), pp. 97–115. URL: https://doi.org/10.1016/0304-3800(79)90029-2.

[4] Yonathon Bard. *Nonlinear parameter estimation*. eng. New York ; London: Academic Press, 1974. ISBN: 0120782502.

[5] J. Varah. "A Spline Least Squares Method for Numerical Parameter Estimation in Differential Equations". In: *SIAM Journal on Scientific and Statistical Computing* 3.1 (1982), pp. 28–46. DOI: 10.1137/0903003. eprint: https://doi.org/10.1137/0903003. URL: https://doi.org/10.1137/0903003.

[6] Hua Liang and Hulin Wu. "Parameter estimation for differential equation models using a framework of measurement error in regression models". In: *Journal of the American Statistical Association* 103.484 (2008), pp. 1570–1583.

[7] Sophie Donnet and Adeline Samson. "Estimation of parameters in incomplete data models defined by dynamical systems". In: *Journal of Statistical Planning and Inference* 137.9 (2007), pp. 2815–2831.

[8] Sophie Donnet and Adeline Samson. "Parametric inference for mixed models defined by stochastic differential equations". In: *ESAIM: Probability and Statistics* 12 (2008), pp. 196–218.

[9] Bo Wang and Wayne Enright. "Parameter estimation for odes using a cross-entropy approach". In: *SIAM Journal on Scientific Computing* 35.6 (2013), A2718–A2737.

[10] M Peifer and J Timmer. "Parameter estimation in ordinary differential equations for biochemical processes using the method of multiple shooting". In: *IET Systems Biology* 1.2 (2007), pp. 78–88.

[11] Ben Calderhead, Mark Girolami, and Neil D Lawrence. "Accelerating Bayesian inference over nonlinear differential equations with Gaussian processes". In: *Advances in neural information processing systems*. 2009, pp. 217–224.

[12] Markus Heinonen et al. "Learning unknown ODE models with Gaussian processes". In: *arXiv preprint arXiv:1803.04303* (2018).

[13] Dino Sejdinovic & Arthur Gretton. *What is an RKHS?* 2014. URL: http://www.stats.ox.ac.uk/~sejdinov/teaching/atml14/Theory_2014.pdf.

[14] Bharath K. Sriperumbudur et al. *Hilbert Space Embeddings and Metrics on Probability Measures*. 2010. URL: http://www.jmlr.org/papers/volume11/sriperumbudur10a/sriperumbudur10a.pdf.

[15] Krikamol Muandet et al. *Kernel Mean Embedding of Distributions: A Review and Beyond*. Tech. rep. 2017. arXiv: 1605.09522v3. URL: https://arxiv.org/pdf/1605.09522.pdf.

[16] Ingo Steinwart. *On the Influence of the Kernel on the Consistency of Support Vector Machines*. 2001. URL: http://www.jmlr.org/papers/volume2/steinwart01a/steinwart01a.pdf.

[17] Kenji Fukumizu et al. "Kernel Measures of Conditional Dependence". In: *Advances in Neural Information Processing Systems 20*. Ed. by J. C. Platt et al. Curran Associates, Inc., 2008, pp. 489–496. URL: http://papers.nips.cc/paper/3340-kernel-measures-of-conditional-dependence.pdf.

[18] Kenji Fukumizu, Francis R. Bach, and Michael I. Jordan. "Kernel dimension reduction in regression". In: *Ann. Statist.* 37.4 (Aug. 2009), pp. 1871–1905. DOI: 10.1214/08-AOS637. URL: https://doi.org/10.1214/08-AOS637.

[19] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[20] Ning Qian. "On the momentum term in gradient descent learning algorithms". In: *Neural networks* 12.1 (1999), pp. 145–151.

[21] Shun-Ichi Amari and Scott C Douglas. "Why natural gradient?" In: *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP'98 (Cat. No. 98CH36181)*. Vol. 2. IEEE. 1998, pp. 1213–1216.

[22] Shun-Ichi Amari. "Natural gradient works efficiently in learning". In: *Neural computation* 10.2 (1998), pp. 251–276.

[23] Jorge R Leis and Mark A Kramer. "The simultaneous solution and sensitivity analysis of systems described by ordinary differential equations". In: *ACM Transactions on Mathematical Software (TOMS)* 14.1 (1988), pp. 45–60.

[24] URL: http://stillbreeze.github.io/REINFORCE-vs-Reparameterization-trick/.

[25] URL: http://blog.shakirm.com/2015/10/machine-learning-trick-of-the-day-4-reparameterisation-tricks/.

[26] Robert P Dickinson and Robert J Gelinas. "Sensitivity analysis of ordinary differential equation systems—a direct method". In: *Journal of computational physics* 21.2 (1976), pp. 123–143.

[27] Hiroshi Kunita. *Stochastic flows and stochastic differential equations*. Vol. 24. Cambridge university press, 1997.

[28] Andrew M Bradley. "Pde-constrained optimization and the adjoint method". In: (2010).

[29] J Schnakenberg. "Simple chemical reaction systems with limit cycle behaviour". In: *Journal of theoretical biology* 81.3 (1979), pp. 389–400.

[30] URL: https://joblib.readthedocs.io/en/latest/.

[31] URL: https://pytorch.org/.

[32] Adam Paszke et al. “Automatic differentiation in PyTorch”. In: (2017).

[33] URL: `https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html#sphx-glr-beginner-blitz-autograd-tutorial-py`.

[34] Tian Qi Chen et al. “Neural ordinary differential equations”. In: *Advances in neural information processing systems*. 2018, pp. 6571–6583.

[35] Balázs Csanád Csáji. “Approximation with artificial neural networks”. In: *Faculty of Sciences, Etvs Lornd University, Hungary* 24 (2001), p. 48.

[36] URL: `https://pytorch.org/docs/stable/notes/randomness.html`.

[37] URL: `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_circles.html`.

[38] Damien Francois, Vincent Wertz, Michel Verleysen, et al. “About the locality of kernels in high-dimensional spaces”. In: *International Symposium on Applied Stochastic Models and Data Analysis*. Citeseer. 2005, pp. 238–245.