Robust Regression on Clustered Data and Signature Based Online Arabic Handwriting Recognition

by

Daniel G. Wilson-Nunn

Thesis submitted in partial fulfilment
of the requirements for the degree of
Doctor of Philosophy in Statistics

The University of Warwick
Department of Statistics

—— & ——

The Alan Turing Institute

June 2021

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

I wish to thank my supervisors; firstly Dr Anastasia Papavasiliou whose great support and guidance brought me through both my masters dissertation and this PhD thesis; secondly, Professor Terry Lyons and Dr Hao Ni for all the support that they have provided during the course of this PhD. Furthermore, I extend many thanks to Jeremy Reizenstein, without whom the work on Arabic handwriting recognition would have taken significantly longer. I extend my thanks to all those at Warwick University, The Alan Turing Institute and the University of Oxford who provided guidance and support along the way.

I also wish to thank Professor Sabri A. Mahmoud for providing the segmented Online KHATT dataset that has been worked on and the Alan Turing Institute for providing the funding for my studies and travel to conferences.

Finally, I wish to thank my family for their support; my wife Adla whose support kept me going during the days when I was staring blankly at a sheet of paper, right through to the final submission of my thesis; my parents who encouraged me to pursue higher education and finally my newborn son Zachariah.

Declaration

The work presented in this thesis is the result of research carried out by myself with support from my supervisors. This thesis is submitted to the University of Warwick in support of my application for the degree of Doctor of Philosophy. It has been composed by myself and has not been submitted in any previous application for any degree.

The work presented (including data generated and data analysis) was carried out by the author except in the cases outlined below:

- n/a

Parts of this thesis have been published by the author:

- Wilson-Nunn, D. and Lyons, T. and Papavasiliou, A. and Ni, H. – A Path Signature Approach to Online Arabic Handwriting Recognition. – In: 2018 IEEE 2nd International Workshop on Arabic and Derived Script Analysis and Recognition (ASAR)

Code produced by the author is available at the following GitHub repositories:

- `https://github.com/wilson-nunn/mathematica_sigs_tensor_algebra`
- `https://github.com/wilson-nunn/mathematica_repr`

Abstract

In this thesis, we present two different methodologies; one for processing time series data for use in machine learning and the other, a robust linear regression for clustered data. The foundation in both of these methodologies is attempting to utilise time series data in ways in which have traditionally been prohibitive, owing to the ragged nature of such data.

In order to use standard machine learning tools to classify online Arabic handwritten characters, we develop a dyadic iterated integral path signature approach to processing the underlying time series data. The process developed transforms raw online Arabic handwritten character data in the form of multiple time series, into a single set of features that can be used as features for machine learning. When applied to the Online KHATT segmented character dataset, the methodology combined with both random forests and long short term memory (LSTM) neural networks demonstrates a dramatic improvement in recognition performance over the previously published best (using hidden Markov models). Furthermore, this processing methodology can be applied to any number of similar scenarios including other online handwritten scripts and even drawings on tablets.

Secondly, with the aim of carrying out polynomial regression using the iterated integral path log signature, we present a robust eigenvalue polynomial regression. This new form of regression is designed to significantly reduce the impact of clustered data on the fitting of a polynomial approximation to the data. Using knowledge of the location of the clusters of data in space, combined with the region over which we wish to obtain a robust estimate, this eigenvalue based method can be seen to have vast improvements over standard least squares linear regression. The methodology is demonstrated to result in a large decrease in the L_2 error of polynomial approximations to a number of functions.

Chapter 1

Introduction

This thesis is comprised of two distinct research projects; one focussing on machine learning using the dyadic iterated integral path signature, and the other introducing a novel robust polynomial regression methodology. Whilst these two projects may seem rather separate, they are intrinsically linked as the logarithm of the iterated integral path signature is a prime candidate for use in high dimensional polynomial regression and has been presented as such by both Ni & Levin [Ni17; LLN13].

1.1 Thesis Overview

The format of this thesis is as follows:

Chapters 2 & 3

These chapters provide an introduction to the theory of iterated integral path signatures and the space in which they lie. The iterated integral path signature is an incredibly powerful tool as, much like an individual's signature is unique to them, it uniquely identifies the underlying path.

The iterated integral path signature has been used as a tool for encoding time series data for machine learning in fields as diverse as handwriting recognition [Gra13; Xie+18], activity classification from short video clips [Yan+19] and

detecting psychiatric conditions in mental health patients [PA+18]. Such is the popularity of the path signature as a tool for machine learning, that Chevyrev & Kormilitzin have produced a primer for people wishing to use the path signature as a tool for machine learning [CK16] and two Python packages available from PyPI - `esig` [LM] and `iisignature` [RG18; Rei] - have been developed to compute the path signature, with work underway to integrate both into major machine learning libraries such as `keras`, `pytorch`, `theano` and `tensorflow`.

Chapter 4

This chapter's contents are based upon a paper presented at ASAR2018 [WN+18]. In this chapter, a methodology for preprocessing online Arabic handwriting data is presented. Online handwriting recognition is the study of recognising handwriting that is received as spatio-temporal data, i.e. data as a time series [PS00].

Online handwriting recognition receives significantly less study than the other major area of handwriting recognition; offline handwriting recognition which deals with spatio-luminescent data, i.e. image data. We present a methodology that is used to recognise online Arabic handwriting. Using the iterated integral path signature introduced in Chapters 2 & 3, we are able to reduce the error on a major dataset within the Arabic handwriting recognition field by more than 50%. Furthermore, the methodology developed can be implemented on numerous other spatio-temporal datasets such as online handwriting recognition in other scripts such as Roman script and has even been shown to perform strongly on simple drawings using the Google "Quick, Draw!" dataset [Fer21].

Chapter 5

The content of this chapter, presents and demonstrates the capabilities of a novel polynomial regression tool which is able to robustly deal with clustered data where the clustered nature of the observations can negatively impact the result of an approximation using least squares polynomial regression.

When attempting to use the iterated integral path signature for polynomial regression, it was observed that the signature was clustered within space and that often at least one of the clusters would contain only a few observations. These clusters containing small numbers of points were found to negatively impact the fit of a polynomial approximation to the data using least squares regression. The problem of a small number of points strongly influencing the fit of an approximation using least squares regression is present not just in high dimensional examples such as using the signature, but also in low dimensions such as $\mathbb{R}^2$. The methodology developed utilises knowledge of the regions where the clusters of data lie, the region of space where a polynomial estimate of the unknown function is desired and the data itself. The result of applying this methodology is a robust polynomial approximation that is not impacted by the small number of points in any given region.

Chapter 6

The final chapter of this thesis provides some concluding remarks and discussion regarding future work and study opportunities building on the methodologies developed.

Chapter 2

Tensor Algebra Spaces

In this chapter, we provide an introduction to the space known as the tensor algebra and a related Lie algebra. Knowledge of these spaces will form a vital basis for the understanding of the path signature which will be introduced in Chapter 3. The theory covered in this chapter is found in much greater depth in [Reu93], which is the key text in this field. In addition, the theory is presented in both [LCL07; FV10] which explain the role of these spaces in the context of the iterated integral path signature and rough path theory in general.

2.1 Tensor Algebras

Consider a space that contains all sequences such that each element is a successive tensor power of an element of V, equivalently

$$\left\{(a_0, a_1, \dots, a_n, \dots) \middle| \forall n \in \mathbb{N}_0, a_n \in (V)^{\otimes n}\right\}, \tag{2.1}$$

here, we follow the convention that $(V)^{\otimes 0} = \mathbb{R}$. This space is known as the tensor algebra of V.

Definition 2.1. For a vector space V defined over $\mathbb{R}$, define the *tensor algebra of* V, notated as $T((V))$, to be the direct sum of $V^{\otimes n}$ for $n = 0, 1, 2 \dots$

$$\begin{aligned} T((V)) &= \bigoplus_{n=0}^{\infty} V^{\otimes n} = V^{\otimes 0} \oplus V^{\otimes 1} \oplus V^{\otimes 2} \oplus \cdots \\ &= \mathbb{R} \oplus V \oplus V^{\otimes 2} \oplus \cdots \end{aligned} \tag{2.2}$$

alternatively, this can be represented as

$$= \left\{ (a_0, a_1, \dots, a_n, \dots) \middle| \forall n \in \mathbb{N}_0, a_n \in (V)^{\otimes n} \right\}. \tag{2.3}$$

Proposition 2.2. *The space $T((V))$ equipped with the following addition, multiplication and scalar multiplication operations is an algebra with unit and zero elements given by*

$$\mathbf{1} := (1, 0, 0, \dots) \in T((V)), \qquad \mathbf{0} := (0, 0, 0, \dots) \in T((V)). \tag{2.4}$$

Note. It is important to realise here that in $\mathbf{1}$ and $\mathbf{0}$ defined above, the 0 in the n-th position (for $n \geq 1$) is the zero element of $V^{\otimes n}$, and not simply $0 \in \mathbb{R}$.

Addition *For $\mathbf{a}, \mathbf{b} \in T((V))$,*

$$\mathbf{a} + \mathbf{b} = (a_0 + b_0, a_1 + b_1, a_2 + b_2, \dots) \tag{2.5}$$

Multiplication *For $\mathbf{a}, \mathbf{b} \in T((V))$,*

$$\mathbf{a} \otimes \mathbf{b} = (c_0, c_1, c_2, \dots), \tag{2.6}$$

with

$$c_n = \sum_{k=0}^{n} a_k \otimes b_{n-k}, \tag{2.7}$$

where $a_k \otimes b_{n-k}$ is the standard tensor multiplication between two tensors.

Scalar Multiplication *For $\mathbf{a} \in T((V))$ and $\alpha \in \mathbb{R}$,*

$$\alpha \cdot \mathbf{a} = (\alpha a_0, \alpha a_1, \alpha a_2, \dots) \tag{2.8}$$

Proof. A vector space V, equipped with addition ($+$), multiplication ($\otimes$) and scalar multiplication ($\cdot$) operations, is an algebra if the three following properties hold:

- Right distributivity: for $\mathbf{x}, \mathbf{y}, \mathbf{z} \in V$,

$$(\mathbf{x} + \mathbf{y}) \otimes \mathbf{z} = \mathbf{x} \otimes \mathbf{z} + \mathbf{y} \otimes \mathbf{z}. \tag{2.9}$$

Proof. By the definitions of $+$ and $\otimes$ in Proposition 2.2, we can see that for $\mathbf{x}, \mathbf{y}, \mathbf{z} \in T((V))$,

$$\begin{aligned}(\mathbf{x}+\mathbf{y})\otimes\mathbf{z} &= ((x_0, x_1, x_2, \dots) + (y_0, y_1, y_2 \dots)) \otimes \mathbf{z} \\ &= ((x_0+y_0, x_1+y_1, x_2+y_2 \dots)) \otimes \mathbf{z} \\ &= (x_0z_0 + y_0z_0, x_0z_1 + y_0z_1 + x_1z_0 + y_1z_0, \dots) \\ &= (x_0z_0, x_0z_1 + x_1z_0, \dots) + (y_0z_0, y_0z_1 + y_1z_0, \dots) \\ &= \mathbf{x}\otimes\mathbf{z} + \mathbf{y}\otimes\mathbf{z}\end{aligned}$$

□

- Left distributivity: for $\mathbf{x}, \mathbf{y}, \mathbf{z} \in V$,

$$\mathbf{x}\otimes(\mathbf{y}+\mathbf{z}) = \mathbf{x}\otimes\mathbf{y} + \mathbf{x}\otimes\mathbf{z}. \tag{2.10}$$

Proof. Owing to the reflexive nature of the definitions of $+$ and $\otimes$ in Proposition 2.2, this follows naturally from the proof of right distributivity.

□

- Compatibility with scalars: for $\alpha, \beta \in \mathbb{R}$ and $\mathbf{x}, \mathbf{y} \in V$

$$(\alpha\cdot\mathbf{x})\otimes(\beta\cdot\mathbf{y}) = (\alpha\beta)\cdot(\mathbf{x}\otimes\mathbf{y}). \tag{2.11}$$

Proof. By the definitions of $\otimes$ and $\cdot$ in Proposition 2.2, we can see that for $\alpha, \beta \in \mathbb{R}$ and $\mathbf{x}, \mathbf{y} \in T((V))$,

$$\begin{aligned}(\alpha\cdot\mathbf{x})\otimes(\beta\cdot\mathbf{y}) &= (\alpha x_0, \alpha x_1, \alpha x_2, \dots) \otimes (\beta y_0, \beta y_1, \beta y_2, \dots) \\ &= (\alpha\beta x_0y_0, \alpha\beta x_0y_1 + \alpha\beta x_1y_0, \dots) \\ &= \alpha\beta\cdot(x_0y_0, x_0y_1 + x_1y_0, \dots) \\ &= (\alpha\beta)\cdot(\mathbf{x}\otimes\mathbf{y})\end{aligned}$$

□

As each of these three properties has been satisfied, the space $T((V))$ equipped with $+$, $\otimes$ and $\cdot$ as defined is an algebra. □

Owing the the infinite nature of any given $\mathbf{a} \in T((V))$, one might wish to consider only a finite portion of this $\mathbf{a}$. In this case, we introduce the truncated tensor algebra of order N.

Definition 2.3. For a vector space V defined over $\mathbb{R}$ and some $N \in \mathbb{N}$, define the *truncated tensor algebra of order N of V*, notated as $T^N(V)$, to be the direct sum of $V^{\otimes n}$ for $n = 0, 1, 2 \ldots, N$

$$\begin{aligned} T^N(V) &= \bigoplus_{n=0}^{N} V^{\otimes n} = V^{\otimes 0} \oplus V^{\otimes 1} \oplus \cdots \oplus V^{\otimes N} \\ &= \mathbb{R} \oplus V \oplus V^{\otimes 2} \oplus \cdots \oplus V^{\otimes N} \end{aligned} \tag{2.12}$$

alternatively, this can be represented as

$$= \left\{ (a_0, a_1, \ldots, a_N) \middle| \forall n \in \{1, 2, \ldots, N\}, a_n \in V^{\otimes n} \right\}. \tag{2.13}$$

Recall the operations $+$, $\otimes$ and $\cdot$ as defined in 2.2, these work with $T^N(V)$ simply by truncating the operations; e.g.

Addition For $\mathbf{a}, \mathbf{b} \in T^N(V)$,

$$\mathbf{a} + \mathbf{b} = (a_0 + b_0, a_1 + b_1, a_2 + b_2, \ldots, a_N + b_N) \tag{2.14}$$

Multiplication For $\mathbf{a}, \mathbf{b} \in T^N(V)$,

$$\mathbf{a} \otimes \mathbf{b} = (c_0, c_1, c_2, \ldots, c_N), \tag{2.15}$$

with

$$c_n = \sum_{k=0}^{n} a_k \otimes b_{n-k}, \tag{2.16}$$

where $a_k \otimes b_{n-k}$ is the tensor multiplication between two tensors.

Scalar Multiplication For $\mathbf{a} \in T^N(V)$ and $\alpha \in \mathbb{R}$,

$$\alpha \cdot \mathbf{a} = (\alpha a_0, \alpha a_1, \alpha a_2, \ldots, \alpha a_N) \tag{2.17}$$

For the remainder of this chapter we consider only the truncated tensor algebra $T^N(V)$ for some N and V.

2.2 Some important subspaces

For the path signature, as will be seen in Chapter 3, we wish to consider two restricted versions of the truncated tensor algebra, namely the following two

subspaces

$$\begin{aligned} \mathfrak{t}_0^N(V) &= \left\{ \mathbf{a} \in T^N(V) \middle| \pi_0(\mathbf{a}) = 0 \right\} \\ &= \left\{ (0, a_1, \ldots, a_N) \middle| \forall n \in \{1, \ldots, N\}, a_n \in (V)^{\otimes n} \right\} \end{aligned} \tag{2.18}$$

and

$$\begin{aligned} \mathfrak{t}_1^N(V) &= \left\{ \mathbf{a} \in T^N(V) \middle| \pi_0(\mathbf{a}) = 1 \right\} \\ &= \left\{ (1, a_1, \ldots, a_N) \middle| \forall n \in \{1, \ldots, N\}, a_n \in (V)^{\otimes n} \right\}. \end{aligned} \tag{2.19}$$

Note. For completeness sake, define

$$\begin{aligned} \mathfrak{t}_0(V) = \mathfrak{t}_0^\infty(V) &= \{ \mathbf{a} \in T((V)) | \pi_0(\mathbf{a}) = 0 \} \\ &= \left\{ (0, a_1, \ldots, a_n, \ldots) \middle| \forall n \in \{1, \ldots, N\}, a_n \in (V)^{\otimes n} \right\} \end{aligned} \tag{2.20}$$

and

$$\begin{aligned} \mathfrak{t}_1(V) = \mathfrak{t}_1^\infty(V) &= \{ \mathbf{a} \in T((V)) | \pi_0(\mathbf{a}) = 1 \} \\ &= \left\{ (1, a_1, \ldots, a_n, \ldots) \middle| \forall n \in \{1, \ldots, N\}, a_n \in (V)^{\otimes n} \right\}. \end{aligned} \tag{2.21}$$

Where $\pi_i(\mathbf{a})$ is the projection of the i-th element of $\mathbf{a}$.

It is immediately obvious that $\mathfrak{t}_0^N(V)$ is a subspace of $T^N(V)$, and it is also clear that as $\mathfrak{t}_1^N(V) = \mathbf{1} + \mathfrak{t}_0^N(V)$, then $\mathfrak{t}_1^N(V)$ is an affine subspace of $T^N(V)$. First we shall look into some of the properties of $\mathfrak{t}_0^N(V)$.

Proposition 2.4. *The set $\mathfrak{t}_0^N(V)$ equipped with addition $(+)$, multiplication $(\otimes)$ and scalar multiplication $(\cdot)$ as defined before is an algebra.*

Proof. The proof of this is very similar to the proof that $T((V))$ is an algebra, so will not be repeated here unnecessarily. □

Recall the definition of a Lie algebra, from which we will go on to show that along with the standard commutator for an algebra, $\mathfrak{t}_0^N(V)$ is a Lie algebra.

Definition 2.5. A *Lie algebra* is an algebra $\mathfrak{g}$ defined over a field K equipped with an additional binary operation $[\cdot, \cdot] : \mathfrak{g} \times \mathfrak{g} \to \mathfrak{g}$ with the following properties

Anticommutativity for all $\mathbf{a}, \mathbf{b} \in \mathfrak{g}$,

$$[\mathbf{a}, \mathbf{b}] = -[\mathbf{b}, \mathbf{a}] \tag{2.22}$$

Jacobi identity for all $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathfrak{g}$,

$$[\mathbf{a}, [\mathbf{b}, \mathbf{c}]] + [\mathbf{c}, [\mathbf{a}, \mathbf{b}]] + [\mathbf{b}, [\mathbf{c}, \mathbf{a}]] = \mathbf{0}. \tag{2.23}$$

The operation $[\cdot, \cdot]$ is known as the *Lie bracket*.

As $\mathfrak{t}_0^N(V)$ forms an algebra, we can consider the commutator

$$[\mathbf{a}, \mathbf{b}] = \mathbf{a} \otimes \mathbf{b} - \mathbf{b} \otimes \mathbf{a}, \tag{2.24}$$

as the Lie bracket.

Proposition 2.6. *The algebra* $\left(\mathfrak{t}_0^N(V), +, \cdot, \otimes\right)$ *is a* Lie algebra *when equipped with the Lie bracket defined in* (2.24)

Proof. To show that $\left(\mathfrak{t}_0^N(V), +, \cdot, \otimes\right)$ with $[\mathbf{a}, \mathbf{b}] = \mathbf{a} \otimes \mathbf{b} - \mathbf{b} \otimes \mathbf{a}$ is a Lie algebra, we need to show that both anticommutativity and the Jacobi identity hold.

- Anticommutativity: for all $\mathbf{a}, \mathbf{b} \in \mathfrak{g}$, $[\mathbf{a}, \mathbf{b}] = -[\mathbf{b}, \mathbf{a}]$.

  Proof.

$$\begin{aligned}
[\mathbf{a}, \mathbf{b}] &= \mathbf{a} \otimes \mathbf{b} - \mathbf{b} \otimes \mathbf{a} \\
&= -(\mathbf{b} \otimes \mathbf{a} - \mathbf{a} \otimes \mathbf{b}) \\
&= -[\mathbf{b}, \mathbf{a}]
\end{aligned}$$

□

- Jacobi identity: for all $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathfrak{g}$, $[\mathbf{a}, [\mathbf{b}, \mathbf{c}]] + [\mathbf{c}, [\mathbf{a}, \mathbf{b}]] + [\mathbf{b}, [\mathbf{c}, \mathbf{a}]] = \mathbf{0}$.

  Proof.

$$\begin{aligned}
&[\mathbf{a}, [\mathbf{b}, \mathbf{c}]] + [\mathbf{c}, [\mathbf{a}, \mathbf{b}]] + [\mathbf{b}, [\mathbf{c}, \mathbf{a}]] \\
&\qquad = [\mathbf{a}, \mathbf{b} \otimes \mathbf{c} - \mathbf{c} \otimes \mathbf{b}] + [\mathbf{c}, \mathbf{a} \otimes \mathbf{b} - \mathbf{b} \otimes \mathbf{a}] + [\mathbf{b}, \mathbf{c} \otimes \mathbf{a} - \mathbf{a} \otimes \mathbf{c}] \\
&\qquad = \mathbf{a} \otimes (\mathbf{b} \otimes \mathbf{c} - \mathbf{c} \otimes \mathbf{b}) - (\mathbf{b} \otimes \mathbf{c} - \mathbf{c} \otimes \mathbf{b}) \otimes \mathbf{a} \\
&\qquad\quad + \mathbf{c} \otimes (\mathbf{a} \otimes \mathbf{b} - \mathbf{b} \otimes \mathbf{a}) - (\mathbf{a} \otimes \mathbf{b} - \mathbf{b} \otimes \mathbf{a}) \otimes \mathbf{c} \\
&\qquad\quad + \mathbf{b} \otimes (\mathbf{c} \otimes \mathbf{a} - \mathbf{a} \otimes \mathbf{c}) - (\mathbf{c} \otimes \mathbf{a} - \mathbf{a} \otimes \mathbf{c}) \otimes \mathbf{b} \\
&\qquad = \mathbf{0}.
\end{aligned}$$

□

Therefore as both conditions hold, $\left(\mathfrak{t}_0^N(V), +, \cdot, \otimes\right)$ equipped with $[\cdot, \cdot]$ is a Lie algebra. □

Now we look into some of the properties of $\mathfrak{t}_1^N(V)$.

Proposition 2.7. *The set $\mathfrak{t}_1^N(V)$ equipped with the operation $\otimes$ as defined before is a group.*

Proof. To show that $\mathfrak{t}_1^N(V)$ is a group, it is necessary to show that the following properties hold.

- Closure: if $\mathbf{a}, \mathbf{b} \in \mathfrak{t}_1^N(V)$, then $\mathbf{a} \otimes \mathbf{b} \in \mathfrak{t}_1^N(V)$

  Proof. From the definition of $\otimes$,

$$\begin{aligned}
\mathbf{a} \otimes \mathbf{b} &= (1, a_1, a_2, \ldots, a_N) \otimes (1, b_1, b_2, \ldots, b_N) \\
&= (1, a_1 + b_1, \ldots, a_N + \cdots + b_N) \in \mathfrak{t}_1^N(V)
\end{aligned}$$

□

- Associativity: for all $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathfrak{t}_1^N(V)$, the following result holds

$$(\mathbf{a} \otimes \mathbf{b}) \otimes \mathbf{c} = \mathbf{a} \otimes (\mathbf{b} \otimes \mathbf{c}) \tag{2.25}$$

Proof. By the definition of $\otimes$, we get that

$$\begin{aligned}(\mathbf{a}\otimes\mathbf{b})\otimes\mathbf{c} &= ((1,a_1,a_2,\ldots,a_N)\otimes(1,b_1,b_2,\ldots,b_N))\otimes\mathbf{c}\\ &= (1,a_1+b_1,\ldots,a_N+\cdots+b_N)\otimes(1,c_1,c_2,\ldots,c_N)\\ &= (1,a_1+b_1+c_1,\ldots,a_N+\cdots+b_N+\cdots+c_N)\\ &= (1,a_1,a_2,\ldots,a_N)\otimes(1,b_1+c_1,\ldots,b_N+\cdots+c_N)\\ &= \mathbf{a}\otimes((1,b_1,b_2,\ldots,b_N)\otimes(1,c_1,c_2,\ldots,c_N))\\ &= \mathbf{a}\otimes(\mathbf{b}\otimes\mathbf{c})\end{aligned}$$ □

- Identity element: there exists $\mathbf{e}\in\mathfrak{t}_1^N(V)$ such that $\mathbf{a}\otimes\mathbf{e}=\mathbf{a}$ for all $\mathbf{a}\in\mathfrak{t}_1^N(V)$

  Proof. Recall from Proposition 2.2 that the identity element was defined to be $\mathbf{1}=(1,0,0,\ldots,0)\in\mathfrak{t}_1^N(V)$. We have that

$$\begin{aligned}\mathbf{a}\otimes\mathbf{e} &= (1,a_1,a_2,\ldots,a_N)\otimes(1,0,0,\ldots,0)\\ &= (1,a_1\cdot 1+1\cdot 0,a_2\cdot 1+a_1\cdot 0+0\cdot 1,\ldots,\\ &\qquad a_N\cdot 1+a_{N-1}\cdot 0+\cdots+a_1\cdot 0+1\cdot 0)\\ &= (1,a_1,a_2,\ldots,a_N)\\ &= \mathbf{a}\end{aligned}$$ □

- Inverse element: for each $\mathbf{a}\in\mathfrak{t}_1^N(V)$ there exists $\mathbf{b}\in\mathfrak{t}_1^N(V)$ such that $\mathbf{a}\otimes\mathbf{b}=\mathbf{e}$ where $\mathbf{e}$ is the identity element

  Proof. The inverse of $\mathbf{a}$ is given by

$$\mathbf{b}=\mathbf{a}^{-1}=\sum_{k=0}^{N}(\mathbf{1}-\mathbf{a})^{\otimes k}=\sum_{k=0}^{N}(-1)^k(\mathbf{a}-\mathbf{1})^{\otimes k}. \tag{2.26}$$

  Recall that the Taylor series expansion of $(x+1)^{-1}$ is $\sum(-1)^n x^n$ for $n\geq 0$, then notice that $\mathbf{a}=(\mathbf{a}-\mathbf{1}+\mathbf{1})$, so from the Taylor expansion,

$$\begin{aligned}\mathbf{a}^{-1} &= (\mathbf{a}-\mathbf{1}+\mathbf{1})^{-1}\\ &= \sum_{k=0}^{N}(-1)^k(\mathbf{a}-\mathbf{1})^{\otimes k}\end{aligned}$$ □

Therefore, as each of the properties required are satisfied, the set $\mathfrak{t}_1^N(V)$ equipped with $\otimes$ is a group. $\square$

It is shown in [FV10] that $\left(\mathfrak{t}_1^N(V), \otimes\right)$ is not only a group, but also a Lie group, the definition of which is also found in the same source. Recall that when we have a Lie group and a Lie algebra associated with it, then we can define an exponential map.

Definition 2.8. For a vector space V, for the Lie group $\mathfrak{t}_1^N(V)$ and the associated Lie algebra $\mathfrak{t}_0^N(V)$, there exists an exponential map $\exp : \mathfrak{t}_0^N(V) \to \mathfrak{t}_1^N(V)$, defined as

$$\begin{aligned} \exp : \mathbf{a} &\mapsto \sum_{n=0}^{N} \frac{\mathbf{a}^{\otimes n}}{n!} \\ \mathbf{a} &\mapsto \mathbf{1} + \mathbf{a} + \frac{\mathbf{a}^{\otimes 2}}{2} + \cdots + \frac{\mathbf{a}^{\otimes N}}{N!}. \end{aligned} \tag{2.27}$$

Additionally, we can define the inverse of the exponential function, the logarithm function.

Definition 2.9. The function $\log : \mathfrak{t}_1^N(V) \to \mathfrak{t}_0^N(V)$ is defined as

$$\begin{aligned} \log : \mathbf{a} &\mapsto \sum_{n=1}^{N} (-1)^{n+1} \frac{(\mathbf{a}-\mathbf{1})^{\otimes n}}{n} \\ \mathbf{a} &\mapsto (\mathbf{a}-\mathbf{1}) - \frac{(\mathbf{a}-\mathbf{1})^{\otimes 2}}{2} + \cdots + (-1)^{N+1} \frac{(\mathbf{a}-\mathbf{1})^{\otimes N}}{N}. \end{aligned} \tag{2.28}$$

Thus ends this introduction to the tensor algebra space and some of its associated subspaces. These spaces will be used in the next chapter where an introduction to the iterated integral path signature is provided.

Chapter 3

Path Signatures

In the following chapter, we will introduce the theory of iterated integral path signatures. The path signature in a primitive state was proposed by Chen in 1958 [Che58] and was subsequently revisited by Lyons in the 1990s [Lyo98] as a tool for the study of controlled differential equations. Since the introduction and formalisation of the theory by Lyons, there has been a constant and steady increase in the use and study of both rough path theory and specifically the path signature. In the past 5-10 years, the path signature has received strong recognition for use as a tool for processing time series data for use in machine learning with Graham in 2013 [Gra13] being one of the pioneers of its usage. It is as a result of both the uniqueness of the signature and its proven performance for use in classification [Gra13; PA+18; Fer21; MLG19] that this interest has been so strong.

3.1 Motivation

The path signature has its origins in the study of differential equations of the form

$$\mathrm{d}Y_t = f(Y_t)\,\mathrm{d}X_t, \quad Y_0 = \xi, \tag{3.1}$$

with $X : [0,T] \to V_1$, $Y : [0,T] \to V_2$ and $f : V_2 \to \mathbf{L}(V_1, V_2)$ (where $\mathbf{L}(V,W)$ is the set of all continuous linear maps from the space V to W), when X is

highly oscillatory. From the Picard-Lindelöf theorem, it is known that if X has bounded variation and f is Lipschitz continuous, then $\forall \xi \in V_2$, the differential equation has a unique solution.

First consider the following simple ODE of first order

$$\mathrm{d}y = f(x, y)\ \mathrm{d}x, \qquad y(x_0) = y_0, \tag{3.2}$$

where $y : \mathbb{R} \to \mathbb{R}$ and $f(x, y)$ is continuous in x and uniformly Lipschitz continuous in y. This ODE satisfies the conditions of the Picard-Lindelöf theorem which implies that an approximate solution to (3.2) can be found using Picard's method. In order to find this approximate solution, the integral form of (3.2) is used:

$$y(x) = y(x_0) + \int_{x_0}^{x} f(t, y(t))\ \mathrm{d}t \tag{3.3}$$

Applying the fixed point theorem to (3.3), we obtain a solution to (3.2)

$$y(x) = \lim_{n\to\infty} y_n(x)\,, \tag{3.4}$$

where

$$y_i(x) = y(x_0) + \int_{x_0}^{x} f(\tau, y_{i-1}(\tau))\ \mathrm{d}\tau. \tag{3.5}$$

Example 3.1. Consider the following ODE of the same form as (3.2)

$$\mathrm{d}y = 2x\,(1 - y)\,\mathrm{d}x, \qquad y(0) = 2. \tag{3.6}$$

Applying the Picard iteration, the result is

$$\begin{aligned}
y_1(x) &= 2 - x^2 \\
y_2(x) &= 2 - x^2 + \frac{x^4}{2} \\
y_3(x) &= 2 - x^2 + \frac{x^4}{2} + \frac{x^6}{6} \\
&\vdots \\
y_n(x) &= 2 - x^2 + \frac{x^4}{2} + \ldots + (-1)^n\,\frac{x^{2n}}{n!}
\end{aligned}$$

One can see that $\lim_{n\to\infty} y_n = 1 + \mathrm{e}^{-x^2}$, which is equal to the exact solution to (3.6).

Extending this to (3.1), the integral form is

$$Y_t = \xi + \int_0^t f(Y_s)\ \mathrm{d}X_s, \tag{3.7}$$

then, the unique solution to the differential equation in (3.1) when X has bounded variation can be found by considering the Picard iteration method.

$$\begin{aligned} Y_t^{[1]} &= \xi \\ Y_t^{[2]} &= \xi + \int_0^t f\left(Y_{u_1}^{[1]}\right)\ \mathrm{d}X_{u_1} = \xi + f(\xi)\int_0^t \mathrm{d}X_{u_1} \\ Y_t^{[3]} &= \xi + \int_0^t f\left(Y_{u_2}^{[2]}\right)\ \mathrm{d}X_{u_2} \\ &= \xi + \int_0^t f(\xi) + f\left(f(\xi)\int_0^{u_2} \mathrm{d}X_{u_1}\right)\ \mathrm{d}X_{u_2} \\ &\vdots \end{aligned} \tag{3.8}$$

this converges to

$$Y_t = \xi + \sum_{n=0}^{\infty} f^{\otimes n}(\xi) \int\limits_{0\le u_1\le\cdots\le u_n\le T} \mathrm{d}X_{u_1}\otimes\cdots\otimes\mathrm{d}X_{u_n}, \tag{3.9}$$

where $f^{\otimes n}(\xi)$ can be defined as in [Lyo14].

3.2 Path Signatures

From above, it can be seen that the k-th step of the Picard iteration is a linear function of

$$\int\limits_{0\le u_1\le\cdots\le u_k\le T} \mathrm{d}X_{u_1}\otimes\cdots\otimes\mathrm{d}X_{u_k}. \tag{3.10}$$

It has been proven [Che58; HL10] that for piecewise smooth paths and paths of bounded variation, the collection of iterated integrals

$$\left(\int\limits_{0\le u_1\le\cdots\le u_k\le T} \mathrm{d}X_{u_1}\otimes\cdots\otimes\mathrm{d}X_{u_n}\right)_{n\ge 0}, \tag{3.11}$$

is well-defined and it is from that conclusion that we define the signature of the path as the collection of all such integrals.

Definition 3.2. Let V be some space defined over $\mathbb{R}$ and let $X : [0,T] \to V$ be a path with bounded variation. Define, for some interval $[s,t] \subseteq [0,T]$,

$$S^{(n)}_{[s,t]}(X) := \int\limits_{\substack{u_1 \leq \cdots \leq u_n \\ u_1,\ldots,u_n \in [s,t]}} \mathrm{d}X_{u_1} \otimes \cdots \otimes \mathrm{d}X_{u_n}, \tag{3.12}$$

then the *signature of* X *over* $[s,t]$ is notated $S_{[s,t]}(X)$ and is defined as

$$S_{[s,t]}(X) = \left(1, \int\limits_{u_1 \in [s,t]} \mathrm{d}X_{u_1}, \int\limits_{\substack{u_1 \leq u_2 \\ u_1,u_2 \in [s,t]}} \mathrm{d}X_{u_1} \otimes \mathrm{d}X_{u_2}, \ldots \right) \tag{3.13}$$

$$= \left(1, S^{(1)}_{[s,t]}(X), S^{(2)}_{[s,t]}(X), \ldots, S^{(n)}_{[s,t]}(X), \ldots\right). \tag{3.14}$$

The initial element of the signature is always equal to $1 \in \mathbb{R} = V^{\otimes 0}$, this is a convention as the first element is the empty integral which we define to be 1.

Note. For ease of notation, the interval over which the signature is computed is often omitted, for example, the signature over the whole domain of a path $X : [0,T] \to V$ will often be written as $S(X) := S_{[0,T]}(X)$.

Notice that the definition of $S^{(n)}_{[s,t]}(X)$ implies that

$$S^{(n)}_{[s,t]}(X) \in V^{\otimes n} = V \otimes \cdots \otimes V, \tag{3.15}$$

therefore the signature of a path $X : [0,T] \to V$ must lie in (possibly some subset of) $\mathfrak{t}_1(V)$. Let us define

$$G(V) := \left\{ S_{[a,b]}(X) \middle| \forall [a,b] \subseteq [s,t], X : [s,t] \to V \right\}. \tag{3.16}$$

It turns out (as we shall see later), that the space $\mathfrak{t}_1(V)$ is actually larger than $G(V)$. In §2.2 it was shown that it is possible to compute the logarithm of elements of $\mathfrak{t}_1(V)$. We therefore introduce the *log signature* of a path.

Definition 3.3. Let V be some space defined over $\mathbb{R}$ and let $X : [0,T] \to V$ be a path with bounded variation. If $S_{[s,t]}(X)$ is the signature of this path over the region $[s,t]$ then

$$s_{[s,t]}(X) = \log\Big(S_{[s,t]}(X)\Big) \tag{3.17}$$

is the *log signature of X over $[s,t]$*. The initial element of the log signature is always equal to $0 \in \mathbb{R} = V^{\otimes 0}$.

Note. As with the signature, for ease of notation, the interval over which the log signature is computed is often omitted, for example, the log signature over the whole domain of a path $X : [0,T] \to V$ will often be written as $s(X) := s_{[0,T]}(X)$.

As with tensor algebra spaces, there are truncated versions of the signature and log signature. These are what will be used for the work in this thesis and are what is used in practice as the infinite signature and log signature are clearly not suitable for use.

Definition 3.4. Let V be some space defined over $\mathbb{R}$ and let $X : [0,T] \to V$ be a path with bounded variation. Define, for some interval $[s,t] \subseteq [0,T]$ and some N the *step-N truncated signature of X over $[s,t]$* to be

$$S^N_{[s,t]}(X) = \left(1, \int_{u_1 \in [s,t]} \mathrm{d}X_{u_1}, \ldots, \int_{\substack{u_1 \leq \cdots \leq u_N \\ u_1,\ldots,u_N \in [s,t]}} \mathrm{d}X_{u_1} \otimes \cdots \otimes \mathrm{d}X_{u_N}\right) \tag{3.18}$$

$$= \left(1, S^{(1)}_{[s,t]}(X), S^{(2)}_{[s,t]}(X), \ldots, S^{(N)}_{[s,t]}(X)\right). \tag{3.19}$$

And similarly, the *step-N truncated log signature of X over $[s,t]$* is

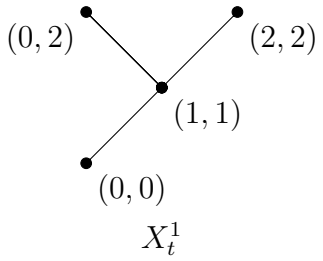
$$s^N_{[s,t]}(X) = \log\left(S^N_{[s,t]}(X)\right) \tag{3.20}$$

3.3 Uniqueness of the Signature

The signature possesses a number of properties, here we shall introduce what is possibly one of the most important features of the signature, uniqueness.

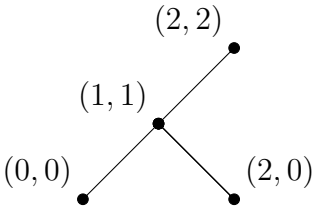
Theorem 3.5 (Lyons & Hambly [HL10])**.** *Let $X : [s,t] \to \mathbb{R}^d$ be a path with bounded p-variation for $p = 1$, then $S_{[s,t]}(X)$ determines X up to the tree-like equivalence.*

Tree-like paths are ones that "double back" on themselves, this means that there is a "redundant" part of the path. For example, in Figure 3.1, there are



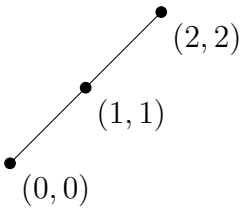
X_t^1

(a) Path with tree-like section from $(1,1)$ to $(0,2)$



X_t^2

(b) Path with tree-like section from $(1,1)$ to $(2,0)$



X_t^3

(c) Tree-reduced version of the above two paths

X_t^1 is the linear interpolation of the points $\{(0,0),(1,1),(0,2),(1,1),(2,2)\}$
X_t^2 is the linear interpolation of the points $\{(0,0),(1,1),(2,0),(1,1),(2,2)\}$
X_t^3 is the linear interpolation of the points $\{(0,0),(1,1),(2,2)\}$

Figure 3.1: Examples of three paths with tree-like equivalence and therefore equal signatures

three paths with the same signature. This is owing to their "tree-like equivalence". Paths X_t^1 and X_t^2 both have "tree-like" sections, whereby the path doubles back on itself. These paths are equivalent in signature to the tree-reduced version X_t^3. The level 3 signature (represented as an element of $\mathbb{R}^d$) of all three paths is

$$S^3\left(X_t^1\right) = S^3\left(X_t^2\right) = S^3\left(X_t^3\right) = \left(1, 2, 2, 2, 2, 2, 2, \frac{4}{3}, \frac{4}{3}, \frac{4}{3}, \frac{4}{3}, \frac{4}{3}, \frac{4}{3}, \frac{4}{3}, \frac{4}{3}\right). \tag{3.21}$$

Theorem 3.5 states that it is not just the level 3 signature that is equal for each path, but the whole signature is equal for all three paths. The following Lemma provides an excellent way of reducing the problem of tree-like

equivalence within data.

Lemma 3.6. *Let $X : [0,T] \to V$ be a path with bounded p-variation for $p = 1$, with a fixed starting point and where at least one coordinate of X is a monotone function, then $S_{[0,T]}(X)$ determines X uniquely.*

As a result of Lemma 3.6, taking the paths X_t^1, X_t^2 and X_t^3 and adding a time dimension, i.e.

$$\begin{aligned} X_t^1 &\text{ is the linear interpolation of the points } \{(0,0,0),(1,1,1),(0,2,2),\\ &\qquad\qquad (1,1,3),(2,2,4)\}\\ X_t^2 &\text{ is the linear interpolation of the points } \{(0,0,0),(1,1,1),(2,0,2),\\ &\qquad\qquad (1,1,3),(2,2,4)\}\\ X_t^3 &\text{ is the linear interpolation of the points } \{(0,0,0),(1,1,1),(2,2,2)\}, \end{aligned}$$

results in three different signatures, (here only the level 2 signature is shown for brevity)

$$\begin{aligned} S^3\left(X_t^1\right) &= (1,2,2,4,2,2,3,2,2,5,5,3,8)\\ S^3\left(X_t^2\right) &= (1,2,2,4,2,2,5,2,2,3,3,5,8)\\ S^3\left(X_t^3\right) &= \left(1,2,2,3,2,2,\frac{7}{2},2,2,\frac{7}{2},\frac{5}{2},\frac{5}{2},\frac{9}{2}\right) \end{aligned} \tag{3.22}$$

3.4 Calculation of Signatures

Recall the definition of the n-th level of the signature,

$$S_{[a,b]}^{(n)}(X) := \int_{\substack{u_1 \le \cdots \le u_n \\ u_1,\ldots,u_n \in [a,b]}} \mathrm{d}X_{u_1} \otimes \cdots \otimes \mathrm{d}X_{u_n}, \tag{3.23}$$

this is most easily calculated elementwise, i.e. by calculating

$$S_{[a,b]}^{\{\tau\}}(X) = S_{[a,b]}^{(\tau_1,\tau_2,\ldots,\tau_n)}(X) = \int_{\substack{u_1 \le \cdots \le u_n \\ u_1,\ldots,u_n \in [a,b]}} \mathrm{d}X_{u_1}^{(\tau_1)} \cdots \mathrm{d}X_{u_n}^{(\tau_n)}, \tag{3.24}$$

where $\{\tau\} = (\tau_1,\ldots,\tau_n)$ is a word of length n, with letters $\tau_i \in \{1,\ldots,n\}$, $i = 1,\ldots,n$, and $X_{u_j}^{(\tau_i)}$ denotes the τ_i-th element of X_{u_j}. Higher order iterated

integrals clearly involve a large amount of integration, in order to make these integrals easier to compute, we introduce a form of integration by parts that relies on the following definition.

Definition 3.7. The *shuffle product* of two words $\{\tau\}$ and $\{\eta\}$ of length m and n respectively, is given by $\{\tau\}\sqcup\!\sqcup\{\eta\}$ and is defined to be the set of the $\frac{(m+n)!}{m!n!}$ words, formed by considering all possible orderings of the letters contained within $\{\tau\}$ and $\{\eta\}$ such that the letters retain their original ordering.

Example 3.8. If $\{\tau\} = (1,5)$ and $\{\eta\} = (2,4,7)$ then $\{\tau\} \sqcup\!\sqcup \{\eta\}$ consists of $\frac{(2+3)!}{2!3!} = 10$ words,

$$\begin{aligned}\{\tau\} \sqcup\!\sqcup \{\eta\} = \{&(1,5,2,4,7), (1,2,5,4,7), (1,2,4,5,7), (1,2,4,7,5), (2,1,5,4,7),\\ &(2,1,4,5,7), (2,1,4,7,5), (2,4,1,5,7), (2,4,1,7,5), (2,4,7,1,5)\}.\end{aligned} \tag{3.25}$$

This is then used in the integration by parts that follows,

Proposition 3.9. *Let $X : [s,t] \to \mathbb{R}^d$ and let $\{\tau\}$ and $\{\eta\}$ be two words, then*

$$S^{\{\tau\}}_{[a,b]}(X)\, S^{\{\eta\}}_{[a,b]}(X) = \sum_{\{\rho\}\in\{\tau\}\sqcup\!\sqcup\{\eta\}} S^{\{\rho\}}_{[a,b]}(X) \tag{3.26}$$

Proof. By considering the individual coordinate signatures,

$$\begin{aligned} X^{\{\tau\}}_{[0,T]} = X^{(\tau_1,\dots,\tau_n)}_{[0,T]} &= \int_{\substack{u_1\le\cdots\le u_n\\ u_1,\dots,u_n\in[0,T]}} \mathrm{d}X^{(\tau_1)}_{u_1}\cdots\mathrm{d}X^{(\tau_n)}_{u_n}\\ X^{\{\eta\}}_{[0,T]} = X^{(\eta_1,\dots,\eta_m)}_{[0,T]} &= \int_{\substack{v_1\le\cdots\le v_m\\ v_1,\dots,v_m\in[0,T]}} \mathrm{d}X^{(\eta_1)}_{v_1}\cdots\mathrm{d}X^{(\eta_m)}_{v_m}, \end{aligned}$$

we see that

$$X^{\{\tau\}}_{[0,T]}X^{\{\eta\}}_{[0,T]} = \int_{\substack{u_1\le\cdots\le u_n\\ v_1\le\cdots\le v_m\\ u_1,\dots,u_n,v_1,\dots,v_m\in[0,T]}} \mathrm{d}X^{(\tau_1)}_{u_1}\cdots\mathrm{d}X^{(\tau_n)}_{u_n}\,\mathrm{d}X^{(\eta_1)}_{v_1}\cdots\mathrm{d}X^{(\eta_m)}_{v_m}$$

We need to find all the possible ways of ordering the integrals such that the u_i and v_j maintain the correct ordering, i.e. $u_i \leq u_j$ and $v_i \leq v_j$ for $i < j$. This is done using the shuffle product as it generates every single way of permuting the elements without changing the ordering within the original words.

$$= \sum_{\{\rho\} \in \{\tau\} \sqcup \{\eta\}} \int_{\substack{w_1 \leq \cdots \leq w_{m+n} \\ w_1, \ldots, w_m \in [0,T]}} \mathrm{d}X_{w_1}^{(\rho_1)} \cdots \mathrm{d}X_{w_{m+n}}^{(\rho_{m+n})}$$

$$= \sum_{\{\rho\} \in \{\tau\} \sqcup \{\eta\}} X_{[0,T]}^{\{\rho\}}. \qquad \square$$

As a result of this proposition, it is possible to compute the log signature and show that a number of the elements are reduced to zero. As an example, let us consider a path $X : [0,1] \to \mathbb{R}^2$, the level two signature of the path lies in (some subspace of) the space $\mathfrak{t}_1^2(\mathbb{R}^2)$, and can be represented as the following:

$$\mathbf{a} = S^2(X) = (1, a_1, a_2, a_{11}, a_{12}, a_{21}, a_{22}). \tag{3.27}$$

Computing $\log(\mathbf{a})$, one obtains

$$\log(\mathbf{a}) = \left(0, a_1, a_2, a_{11} - \frac{a_1^2}{2}, a_{12} - \frac{a_1 a_2}{2}, a_{21} - \frac{a_1 a_2}{2}, a_{22} - \frac{a_2^2}{2}\right). \tag{3.28}$$

We know from Proposition 3.9 that $a_1^2 = S^{\{1\}}(X)\, S^{\{1\}}(X) = 2S^{\{1,1\}}(X) = 2a_{11}$ so $a_{11} - \frac{a_1^2}{2} = 0$, and the same holds for $a_{22} - \frac{a_2^2}{2}$. Furthermore, notice that $a_1 a_2 = a_{12} + a_{21}$ and therefore, $a_{12} - \frac{a_1 a_2}{2} = \frac{a_{12}}{2} - \frac{a_{21}}{2} = -\left(\frac{a_{21}}{2} - \frac{a_1 a_2}{2}\right) = a_{21} - \frac{a_1 a_2}{2}$. This gives us

$$\log(\mathbf{a}) = \left(0, a_1, a_2, 0, \frac{1}{2}(a_{12} - a_{21}), \frac{-1}{2}(a_{12} - a_{21}), 0\right). \tag{3.29}$$

The calculation of signatures of streams of data is something that has been worked on significantly by two different groups. Terry Lyons' group in Oxford, through their CoRoPa (Computational Rough Path) project, have published the Python package `esig` [LM] and Jeremy Reizenstein during his time at Warwick produced the `iisignature` Python package [RG18], both of these packages are available on PyPI (Python Package Index).

3.5 Space of signatures and log signatures

We wish to concretely define the spaces in which the signature and the log signature lie. Recall that in (3.16), we loosely defined the space of signatures $G(V)$, let us similarly define the space $G^N(V)$ to be the set of all step-N signatures of paths in V.

$$G^N(V) := \left\{ S^N_{[a,b]}(X) \middle| \forall [a,b] \subseteq [s,t], X : [s,t] \to V \right\}. \tag{3.30}$$

For the moment, we will not define this space more concretely but will instead come back to it once we have first looked at the space of log signatures.

Now that we have an arbitrary space for the signatures, we shall define the space containing the log signatures as a particular subset of $\mathfrak{t}^N_0(V)$. First, define a sequence $(L_i)_{i=0}^N$ where $L_0 = 0$, $L_1 = V$ and

$$L_n = [V, L_{n-1}], \qquad n \in \{2, \dots, N\}. \tag{3.31}$$

For each n, L_n is defined as the space of *homogeneous Lie polynomials of degree* n and it is easy to see that for every n, L_n is a linear subspace of $V^{\otimes n}$. Using this, we define $\mathfrak{g}^N(V)$.

Definition 3.10. The *free N-step nilpotent Lie algebra* is defined as

$$\begin{aligned} \mathfrak{g}^N(V) &= \bigoplus_{n=0}^{N} L_n = L_0 \oplus L_1 \oplus L_2 \oplus \cdots \oplus L_N \\ &= V \oplus [V,V] \oplus [V,[V,V]] \oplus \cdots \oplus \overbrace{[V,[\dots,[V,V]]]}^{N-1 \text{ brackets}} \\ &= \{(l_0, l_1, \dots, l_N) | l_n \in L_n\} \end{aligned} \tag{3.32}$$

The bases of this space are known as the Hall bases. We will go on to discuss one specific Hall basis, the one that is used within both `esig` [**lyons__esig__nodate__pypi**; LM] and CoRoPa [Lyo] - the C++ package upon which `esig` is built. First, an example.

Example 3.11. Assume that V is of dimension 2, then we can write the basis of V as $\mathbf{e}_1, \mathbf{e}_2$. Now, If we consider $\mathfrak{g}^2(V)$, we have that

$$\mathfrak{g}^2(V) = V \oplus [V,V], \tag{3.33}$$

so naïvely, one might assume that a basis could be the following collection of elements $\mathbf{f}_1, \ldots, \mathbf{f}_8 \in \mathfrak{g}^1(V)$,

$$\begin{aligned}
\mathbf{f}_1 &= \mathbf{e}_1 + [\mathbf{e}_1, \mathbf{e}_1] & \mathbf{f}_2 &= \mathbf{e}_2 + [\mathbf{e}_1, \mathbf{e}_1] \\
&= \mathbf{e}_1 + \mathbf{e}_1 \otimes \mathbf{e}_1 - \mathbf{e}_1 \otimes \mathbf{e}_1 & &= \mathbf{e}_2 + \mathbf{e}_1 \otimes \mathbf{e}_1 - \mathbf{e}_1 \otimes \mathbf{e}_1 \\
&= \mathbf{e}_1 & &= \mathbf{e}_2 \\
\mathbf{f}_3 &= \mathbf{e}_1 + [\mathbf{e}_2, \mathbf{e}_1] & \mathbf{f}_4 &= \mathbf{e}_1 + [\mathbf{e}_1, \mathbf{e}_2] \\
&= \mathbf{e}_1 + \mathbf{e}_2 \otimes \mathbf{e}_1 - \mathbf{e}_1 \otimes \mathbf{e}_2 & &= \mathbf{e}_1 + \mathbf{e}_1 \otimes \mathbf{e}_2 - \mathbf{e}_2 \otimes \mathbf{e}_1 \\
\mathbf{f}_5 &= \mathbf{e}_2 + [\mathbf{e}_2, \mathbf{e}_1] & \mathbf{f}_6 &= \mathbf{e}_2 + [\mathbf{e}_1, \mathbf{e}_2] \\
&= \mathbf{e}_2 + \mathbf{e}_2 \otimes \mathbf{e}_1 - \mathbf{e}_1 \otimes \mathbf{e}_2 & &= \mathbf{e}_2 + \mathbf{e}_1 \otimes \mathbf{e}_2 - \mathbf{e}_2 \otimes \mathbf{e}_1 \\
\mathbf{f}_7 &= \mathbf{e}_1 + [\mathbf{e}_2, \mathbf{e}_2] & \mathbf{f}_8 &= \mathbf{e}_2 + [\mathbf{e}_2, \mathbf{e}_2] \\
&= \mathbf{e}_1 + \mathbf{e}_2 \otimes \mathbf{e}_2 - \mathbf{e}_2 \otimes \mathbf{e}_2 & &= \mathbf{e}_2 + \mathbf{e}_2 \otimes \mathbf{e}_2 - \mathbf{e}_2 \otimes \mathbf{e}_2 \\
&= \mathbf{e}_1 & &= \mathbf{e}_2
\end{aligned}$$

It is clear here that this collection of elements of $\mathfrak{g}^1(V)$ has a large amount of redundancy and therefore is not a basis, for example,

$$\mathbf{f}_1 = \mathbf{f}_7 \qquad \text{and} \qquad \mathbf{f}_2 = \mathbf{f}_8,$$

furthermore, if $\mathbf{g} = \mathbf{e}_1 \otimes \mathbf{e}_2 - \mathbf{e}_2 \otimes \mathbf{e}_1$, then

$$\begin{aligned}
\mathbf{f}_3 = \mathbf{f}_1 - \mathbf{g} \qquad &\text{and} \qquad \mathbf{f}_4 = \mathbf{f}_1 + \mathbf{g} \\
\mathbf{f}_5 = \mathbf{f}_2 - \mathbf{g} \qquad &\text{and} \qquad \mathbf{f}_6 = \mathbf{f}_2 + \mathbf{g}.
\end{aligned}$$

Therefore it is clear that in fact, a basis for $\mathfrak{g}^1(V)$ is simply

$$\mathbf{e}_{[1]} = \mathbf{e}_1, \qquad \mathbf{e}_{[2]} = \mathbf{e}_2, \qquad \mathbf{e}_{[1,2]} = \mathbf{e}_1 \otimes \mathbf{e}_2 - \mathbf{e}_2 \otimes \mathbf{e}_1.$$

The method of generating a basis of this form for $\mathfrak{g}^N(V)$ is a recursive one, and is presented in Algorithm 3.1.

Algorithm 3.1: Generating basis for $\mathfrak{g}^N(V)$.

Inputs: A basis for V given by $\mathbf{e}_1, \ldots, \mathbf{e}_d$

Step 1: Start with basis for V and then define the "first level" of the basis of $\mathfrak{g}^N(V)$ to be $\mathbf{e}_{[i]} = \mathbf{e}_i$ for $i = 1, \ldots, d$

Step 2: For the "second level" of the basis, generate all bracketed pairs (that is $[i, j]$) such that $i < j$ for $i, j \in \{1, \ldots, d\}$, then define for each bracketed pair $[i, j]$, $\mathbf{e}_{[i,j]} = [\mathrm{e}_i, \mathrm{e}_j] = \mathrm{e}_i \otimes \mathrm{e}_j - \mathrm{e}_j \otimes \mathrm{e}_i$ to be the "second level" of the basis

Step 3: For the "n-th level" ($3 \leq n \leq N$), obtain all possible bracketed pairs of the form $[i, A]$ where $i \in \{1, \ldots, d\}$ and A is a bracketed pair from the $(n-1)$-th level of the basis. The n-th level of the basis is then formed using each bracketed pair $[i, A]$, using $\mathbf{e}_{[i,A]} = \mathbf{e}_{[i,[\ldots,[j,k]]]} = [\mathbf{e}_i, [\ldots, [\mathbf{e}_j, \mathbf{e}_k]]]$.

Therefore we have concretely defined the space in which the log signature lies. The space of signatures can now be defined from the following theorem.

Theorem 3.12. *Where $G^N(V)$ is the set of all step-N signatures of paths in V and $\mathfrak{g}^N(V)$ is the free N-step nilpotent Lie algebra, the following relationship holds*

$$G^N(V) = \exp\Big(\mathfrak{g}^N(V)\Big). \tag{3.34}$$

$G^N(V)$ is known as the free n-step nilpotent group over V *and is a closed sub-Lie group of $\Big(\mathfrak{t}_1^N(V), \otimes\Big)$.*

Proof. The proof of this can be found in Fritz & Victoir [FV10] (pp. 143-144). □

From the way in which $\mathfrak{g}^N(V)$ is defined, it is clear that every element $\mathbf{a} \in \mathfrak{g}^N(V)$ is a log signature and by definition of $G^N(V)$, every $\mathbf{a} \in G^N(V)$ is a signature. However owing to the fact that the space $G^N(V)$ has no known closed form, we will consider the two main spaces used to be $\mathfrak{t}_1^N(V)$ and $\mathfrak{g}^N(V)$.

3.6 Polynomials in the log signature

It is a well known theorem within the study of Lie algebras that polynomials in the linear functionals of the log signature form the same space of functions on paths as linear functionals of the signature, each with a different basis [Reu93; LCL07]. We wish to investigate the relationship between these bases. In doing this, we will look at two bases for the linear functionals on the log signature, these are

- Monomials e.g. (in two dimensions up to degree 2) $1, x, y, x^2, xy, y^2$
- Chebyshev polynomials of the first kind e.g. (in two dimensions up to degree 2) $1, x, y, 2x^2 - 1, xy, 2y^2 - 1$

In order to define these polynomials, it is necessary to first understand the linear functionals on the log signature and signature.

3.6.1 Linear functionals on the signature and log signature

Given a normed vector space $(V, \|\cdot\|)$ defined over $\mathbb{R}$ of dimension d, let $\mathbf{e}_1, \ldots, \mathbf{e}_d$ be a basis for V. The space $V^{\otimes n}$ is then equipped with the basis

$$\mathbf{e}_I = \mathbf{e}_{(i_1,\ldots,i_n)} = \mathbf{e}_{i_1} \otimes \cdots \otimes \mathbf{e}_{i_N}, \qquad \forall I = (i_1, \ldots, i_n) \in \{1, \ldots, d\}^n, \tag{3.35}$$

by taking the direct sum of these bases for all n, we obtain a basis for the space $T((V))$. The dual of V is the set of all bounded linear functionals from V to $\mathbb{R}$,

$$V^* = \{\varphi : V \to \mathbb{R} | \varphi \text{ is bounded}\} \tag{3.36}$$

and has a basis $\mathbf{e}_1^*, \ldots, \mathbf{e}_d^*$, where

$$\langle \mathbf{e}_i^*, \mathbf{e}_j \rangle = \mathbf{e}_i^*(\mathbf{e}_j) = \delta_{ij}. \tag{3.37}$$

Note. δ_{ij} is the Kronecker delta function

$$\delta_{ij} = \begin{cases} 0 & \text{if } i \neq j, \\ 1 & \text{if } i = j. \end{cases} \tag{3.38}$$

The space $(V^{\otimes n})^* = (V^*)^{\otimes n}$ then has a basis

$$\mathbf{e}_i^* = \mathbf{e}_{(i_1,\ldots,i_n)}^* = \mathbf{e}_{i_1}^* \otimes \cdots \otimes \mathbf{e}_{i_n}^*, \qquad \forall I = (i_1, \ldots, i_n) \in \{1, \ldots, d\}^n . \tag{3.39}$$

It follows that $T((V))$ has a basis

$$\bigoplus_{n=0}^{\infty} \mathbf{e}_{i_1} \otimes \cdots \otimes \mathbf{e}_{i_n} \tag{3.40}$$

and $T(V^*)$ has a basis

$$\bigoplus_{n=0}^{\infty} \mathbf{e}_{i_1}^* \otimes \cdots \otimes \mathbf{e}_{i_n}^* . \tag{3.41}$$

Take $\sum a_I \mathbf{e}_I = \mathbf{a} \in T((V))$, then

$$\mathbf{e}_I^*(\mathbf{a}) = \mathbf{a}_I, \tag{3.42}$$

i.e. the $I = (i_1, \ldots, i_n)$-th coordinate of $\mathbf{a}$.

We now investigate what happens if we take $\mathbf{f}^*, \mathbf{g}^* \in T(V^*)$ and consider their pointwise product. We know that if $\mathbf{f}^*, \mathbf{g}^* : T((V)) \to \mathbb{R}$ then $\mathbf{f}^* \cdot \mathbf{g}^* : T((V)) \to \mathbb{R}$ and then

$$(\mathbf{f}^* \cdot \mathbf{g}^*)(\mathbf{a}) = \mathbf{f}^*(\mathbf{a}) \cdot \mathbf{g}^*(\mathbf{a}) \tag{3.43}$$

is a quadratic form on $T((V))$. It is known [LCL07] that there exists an additional, unique third element of $T(V^*)$ that we shall denote $\mathbf{f}^* \sqcup \mathbf{g}^*$ that has the property that $\mathbf{f}^* \cdot \mathbf{g}^*$ and $\mathbf{f}^* \sqcup \mathbf{g}^*$ are equal on a subset of $\mathfrak{t}_1(V)$. As $\mathbf{f}^*$ and $\mathbf{g}^*$ can be written as linear combinations of basis elements of $T(V^*)$, we shall define this third element in terms of basis elements. This third element is defined using as follows;

Definition 3.13. Given $\mathbf{e}_I^*, \mathbf{e}_J^* \in T(V^*)$ with the indices being the words $I = (i_1, \ldots, i_n)$ and $J = (j_1, \ldots, j_m)$ (for some $m, n \in \mathbb{N}$), define the *shuffle product of* $\mathbf{e}_I^*$ *and* $\mathbf{e}_J^*$, notated as $\mathbf{e}_I^* \sqcup \mathbf{e}_J^*$ to be

$$\mathbf{e}_I \sqcup \mathbf{e}_J = \sum_{K \in I \sqcup\!\sqcup J} \mathbf{e}_K, \tag{3.44}$$

where $I \sqcup\!\sqcup J$ is the *shuffle product of the two words* I *and* J where I consists of n letters and J consists of m letters. It is defined to be the set of the $\frac{(n+m)!}{n!m!}$ words of length $n+m$ formed by considering all possible orderings of the letters contained within I and J such that the letters retain their original ordering.

Example 3.14. Consider $\mathbf{e}_{11}$ and $\mathbf{e}_2$, then

$$(1,1) \sqcup\!\sqcup (2) = \{(1,1,2), (1,2,1), (2,1,1)\}, \tag{3.45}$$

and therefore

$$\mathbf{e}_{11} \sqcup \mathbf{e}_2 = \mathbf{e}_{112} + \mathbf{e}_{121} + \mathbf{e}_{211} \tag{3.46}$$

Using these properties, we are able to define polynomials on elements of $\mathfrak{g}^N(V)$. Let us consider the example of $\mathfrak{t}_1^2(V)$ where V is of dimension 2. Now this space has a basis $\{\mathbf{e}_0, \mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_{11}, \mathbf{e}_{12}, \mathbf{e}_{21}, \mathbf{e}_{22}\}$ and dual basis $\{\mathbf{e}_0^*, \mathbf{e}_1^*, \mathbf{e}_2^*, \mathbf{e}_{11}^*, \mathbf{e}_{12}^*, \mathbf{e}_{21}^*, \mathbf{e}_{22}^*\}$, and the corresponding space $\mathfrak{g}^N(V)$ has a basis $\{\mathbf{e}_{[0]}, \mathbf{e}_{[1]}, \mathbf{e}_{[2]}, \mathbf{e}_{[1,2]}\}$ and dual basis $\{\mathbf{e}_{[0]}^*, \mathbf{e}_{[1]}^*, \mathbf{e}_{[2]}^*, \mathbf{e}_{[1,2]}^*\}$. For some $\mathbf{a} \in \mathfrak{g}^N(V)$, we have the following monomials:

$$\text{degree 0: } \begin{cases} \mathbf{e}_{[0]}^*(\mathbf{a}) = \mathbf{e}_0^*(\mathbf{a}) \end{cases} \tag{3.47}$$

$$\text{degree 1: } \begin{cases} \mathbf{e}_{[1]}^*(\mathbf{a}) = \mathbf{e}_1^*(\mathbf{a}) \\ \mathbf{e}_{[2]}^*(\mathbf{a}) = \mathbf{e}_2^*(\mathbf{a}) \end{cases} \tag{3.48}$$

$$\text{degree 2: } \begin{cases} \mathbf{e}_{[1,2]}^*(\mathbf{a}) = \mathbf{e}_{12}^*(\mathbf{a}) - \mathbf{e}_{21}^*(\mathbf{a}) \\ (\mathbf{e}_{[1]}^* \cdot \mathbf{e}_{[2]}^*)(\mathbf{a}) = (\mathbf{e}_{[1]}^* \sqcup \mathbf{e}_{[2]}^*)(\mathbf{a}) = \mathbf{e}_{12}^*(\mathbf{a}) + \mathbf{e}_{21}^*(\mathbf{a}) \\ (\mathbf{e}_{[1]}^* \cdot \mathbf{e}_{[1]}^*)(\mathbf{a}) = (\mathbf{e}_{[1]}^* \sqcup \mathbf{e}_{[1]}^*)(\mathbf{a}) = \mathbf{e}_{11}^*(\mathbf{a}) + \mathbf{e}_{11}^*(\mathbf{a}) = 2\mathbf{e}_{11}^*(\mathbf{a}) \\ (\mathbf{e}_{[2]}^* \cdot \mathbf{e}_{[2]}^*)(\mathbf{a}) = (\mathbf{e}_{[2]}^* \sqcup \mathbf{e}_{[2]}^*)(\mathbf{a}) = \mathbf{e}_{22}^*(\mathbf{a}) + \mathbf{e}_{22}^*(\mathbf{a}) = 2\mathbf{e}_{22}^*(\mathbf{a}) \end{cases} \tag{3.49}$$

3.7 Wolfram Mathematica package

In order to assist with the investigation into polynomials of log signature, I have produced a Mathematica package. This section will list and demonstrate some of the main functions of interest in the package.

3.7.1 Generating Hall bases

In order to define the polynomials, one needs the Hall basis for $\mathfrak{g}^N(V)$ where $\dim(V) = d$ and $\dim(N) = m$; the function `GenerateStandardHallBasis[d,m]` is

designed to do just that. This function (and indeed most others in this package) rely on symbolic computation in Mathematica and some are based upon code produced as path of the `iisignature` Python package [**reizenstein_bottleriisignature_no**
Given `d` and `m`, Hall basis elements will be output in the form of bracketed expressions such as `{e1,{e1,e2}}` which is the representation of $[\mathbf{e}_1, [\mathbf{e}_1, \mathbf{e}_2]] = \mathbf{e}_{[1,[1,2]]}$ grouped by the depth of their brackets. In order to obtain a list of the Hall basis elements ungrouped, the list must be flattened by 1 level.

```
In[1]:= GenerateStandardHallBasis[2,3]

Out[1]= {{e1, e2}, {{e1, e2}},
          {{e1, {e1, e2}}, {e2, {e1, e2}}}}
```

It is important to note that, the way in which the bases are notated means that it is only possible to generate the Hall basis for up to `d=9`. After that, the notation methodology chosen causes the code to breakdown, this is owing to the fact that the notation relies on single digit numbers; `e10` could be considered to be either a level 1 or level 2 basis element.

3.7.2 Expanding Hall basis brackets

Once the basis has been obtained, we wish to expand the brackets to obtain the basis in terms of basis elements of $\mathfrak{t}_1^N(V)$. This is done using the `ExpandBracketedExp[x,d]` function. This function takes a bracketed expression `x` in the form produced by `GenerateStandardHallBasis[d,m]` (for example `{e1,{e1,e2}}`) and `d`, the dimension of the space V.

```
In[2]:= ExpandBracketedExp[{e1,{e1,e2}},2]

Out[2]= e112 - 2 e121 + e211
```

```
In[3]:= ExpandBracketedExp[{e3,{e1,{e2,e4}}},4]

Out[3]= - e1243 + e1423 + e2413 + e3124 - e3142
                - e3241 + e3421 - e4213
```

3.7.3 Shuffle product

Having expanded the Hall basis elements, it is necessary to find the monomials. The monomials are dependent upon the shuffle product. In the package I have provided two different shuffle product functions, `ShuffleW[s1,s2]` takes two words `s1` and `s2` in the form of lists and outputs a list containing all possible shuffles of these two words, i.e. `s1` ⧢ `s2`.

```
In[4]:= ShuffleW[{1, 2, 3}, {2, 5}]

Out[4]= {{1, 2, 3, 2, 5}, {1, 2, 2, 3, 5}, {1, 2, 2, 5, 3},
         {1, 2, 2, 3, 5}, {1, 2, 2, 5, 3}, {1, 2, 5, 2, 3},
           {2, 1, 2, 3, 5}, {2, 1, 2, 5, 3},
             {2, 1, 5, 2, 3}, {2, 5, 1, 2, 3}}
```

The other shuffle product function `Shuffle[a,b]` takes two expanded Hall basis elements (i.e. a linear combination of basis elements of $\mathfrak{t}_1^N(V)$) and outputs the shuffle product of the two, i.e. `a` ⊔ `b`.

```
In[5]:= Shuffle[e1, e2]

Out[5]= e12 + e21
```

```
In[6]:= Shuffle[4e2 - 3e1, -e22]

Out[6]= 3 e122 + 3 e212 + 3 e221 - 12 e222
```

3.7.4 Monomials and Chebyshev Polynomials

The `BasisMonomials[d,m]` function takes `d` and `m` as defined previously and generates all possible monomials (up to and including degree `m`) in terms of the expanded Hall basis elements.

```
In[7]:= BasisMonomials[2,3]

Out[7]= {1, e1, e2, 2 e11, e12 - e21, e12 + e21, 2 e22, 6 e111,
          2 e112 - 2 e211, e112 - 2 e121 + e211,
            2 e112 + 2 e121 + 2 e211, 2 e122 - 2 e221,
              -e122 + 2 e212 - e221,
                2 e122 + 2 e212 + 2 e221, 6 e222}


In[8]:= BasisMonomials[4,2]

Out[8]= {1, e1, e2, e3, e4, 2 e11, e12 - e21, e12 + e21, 2 e22,
          e13 - e31, e13 + e31, e23 - e32, e23 + e32, 2 e33,
            e14 - e41, e14 + e41, e24 - e42, e24 + e42,
              e34 - e43, e34 + e43, 2 e44}
```

For generating Chebyshev polynomials, `BasisChebyshevPolynomials[d,m]` takes `d` and `m` as defined previously and generates all possible Chebyshev polynomials in terms of the expanded Hall basis elements.

```
In[9]:= BasisChebyshevPolynomials[2,2]

Out[9]= {1, e1, e2, 4 e11 - 1, e12 - e21, e12 + e21, 4 e22 - 1}
```

3.7.5 Operations on tensor algebra structures

There are a handful of functions to carry out operations on tensor algebra structures which take as input representations of tensor objects such as `{1,{a1,a2},{{a11,a12},{a21,a22}}}`, below a handful of these functions are demonstrated.

- `TensorAlgebraProduct[a,b]` computes $\mathtt{a} \otimes \mathtt{b}$ where $\mathtt{a}, \mathtt{b} \in T^N((V))$
- `TensorAlgebraSum[a,b]` computes $\mathtt{a} + \mathtt{b}$ where $\mathtt{a}, \mathtt{b} \in T^N((V))$
- `TensorAlgebraScalarProduct[k,a]` computes $\mathtt{k} \cdot \mathtt{b}$ where $\mathtt{k} \in V$ and $\mathtt{b} \in T^N((V))$
- `TensorAlgebraLieBracket[a,b]` computes $[\mathtt{a}, \mathtt{b}]$ where $\mathtt{a}, \mathtt{b} \in \mathfrak{t}_0^N(V)$
- `TensorAlgebraInverse[a]` computes $\mathtt{a}^{-1}$ for $\mathtt{a} \in \mathfrak{t}_1^N(V)$
- `TensorAlgebraExp[a]` computes $\exp(\mathtt{a})$ for $\mathtt{a} \in \mathfrak{t}_0^N(V)$
- `TensorAlgebraLog[a]` computes $\log(\mathtt{a})$ for $\mathtt{a} \in \mathfrak{t}_1^N(V)$

```
In[10]:= TensorAlgebraProduct[
                  {1,{a1,a2},{{a11,a12},{a12,a22}}},
                  {1,{b1,b2},{{b11,b12},{b12,b22}}}]

Out[10]= {1, {a1 + b1, a2 + b2},
           {{a11 + a1 b1 + b11, a12 + b12 + a1 b2},
             {a21 + a2 b1 + b21, a22 + a2 b2 + b22}}}
```

```
In[11]:= TensorAlgebraSum[
                {0,{a1,a2},{{a11,a12},{a12,a22}}},
                {0,{b1,b2},{{b11,b12},{b12,b22}}}]

Out[11]= {0, {a1 + b1, a2 + b2},
          {{a11 + b11, a12 + b12}, {a21 + b21, a22 + b22}}}


In[12]:= TensorAlgebraScalarProduct[k,
                {1,{a1,a2},{{a11,b12},{b12,b22}}}]

Out[12]= {k, {k * a1, k * a2},
          {{k * a11, k * a12}, {k * a21, k * a22}}}


In[13]:= TensorAlgebraLieBracket[
                {0,{a1,a2},{{a11,a12},{a12,a22}}},
                {0,{b1,b2},{{b11,b12},{b12,b22}}}]

Out[13]= {0, {0, 0}, {{0, -a2 b1 + a1 b2}, {a2 b1 - a1 b2, 0}}}


In[14]:= TensorAlgebraInverse[
                {1,{a1,a2},{{a11,a12},{a12,a22}}}]

Out[14]= {1, {-a1, -a2}, {{a1^2 - a11, -a12 + a1 a2},
          {a1 a2 - a21, a2^2 - a22}}}


In[15]:= TensorAlgebraExp[
                {0,{a1,a2},{{a11,a12},{a12,a22}}}]

Out[15]= {1, {a1, a2}, {{a1^2/2 + a11, a12 + (a1 a2)/2},
          {(a1 a2)/2 + a21, a2^2/2 + a22}}}
```

```
In[16]:= TensorAlgebraLog[
                {1,{a1,a2},{{a11,a12},{a12,a22}}}]

Out[16]= {0, {a1, a2}, {{-(a1^2/2) + a11, a12 - (a1 a2)/2},
          {-((a1 a2)/2) + a21, -(a2^2/2) + a22}}}
```

By combining this Mathematica package [WN21] with the code developed by Jeremy Reizenstein [**reizenstein_bottleriisignature_nodate**] to compute signatures using Mathematica, these form a very powerful tool for computing and manipulating signatures and log signatures.

Chapter 4

Online Arabic Handwritten Character Recognition Using Dyadic Signatures

In this chapter, we present a methodology using the iterated integral path signature, to transform sequential data for use with machine learning, applied to online Arabic handwritten characters.

Signatures have been demonstrated to be an excellent tool for classifying sequential data in the form of Chinese online handwriting [Gra13; Xie+18]. Building on the successes of researchers at both The University of Warwick and South China University of Technology, we introduce a new methodology using the iterated integral path signature for the classification of online Arabic handwritten characters.

The contents of this chapter are based upon a paper [WN+18] presented at ASAR2018 - the 2018 IEEE 2nd International Workshop on Arabic and Derived Script Analysis and Recognition.

The structure of this chapter is as follows; §4.1-4.2 briefly introduce the study of online handwriting recognition and the segmented Online KHATT dataset that will be used in this chapter, §4.3 explains the methodology that has been

developed, §4.4 introduces the two different classification methodologies that will be used and finally, §4.5 & 4.6 present the results of applying the dyadic signature approach and provide a conclusion.

4.1 Online Handwriting Recognition

Handwriting recognition methods are of two distinct types; online and offline. Online handwriting recognition deals with data recorded "in time", i.e. data that is represented as a function of time. Offline handwriting recognition, on the other hand, deals with recognition of data that is in image format. Online handwriting recognition deals with the spatio-temporal resolution of the input whereas offline handwriting recognition deals with the spatio-luminance of an image [PS00]. Handwriting recognition has a number of important application areas, from converting handwriting to text on a tablet or touch screen to signature verification for bank fraud to digitisation of ancient manuscripts for historical study [Pri+16].

Recent advances in computing and deep learning have resulted in a large amount of interest in the area of handwriting recognition, with over 30,000 results for "handwriting recognition" appearing on Google Scholar since 2013. The popularity of the field has lead to significant developments and advancements in recent years. Online handwriting recognition is a topic that has been studied since the 1960s and has been greatly studied in the past decade, with competitions emerging for recognition of online handwriting at major conferences such as ICDAR (International Conference on Document Analysis and Recognition) and ICFHR (International Conference on Frontiers in Handwriting Recognition), in many languages and scripts ranging from Chinese to English to Arabic.

Figure 4.1: Online handwritten Arabic characters ($n\bar{u}n$ and $sh\bar{i}n$)

4.2 Online Handwritten Arabic Characters Data - KHATT

Online handwritten characters are sequences of (x, y) coordinates, possibly with multiple sequences per character to indicate multiple strokes within a single character. The general format of these characters is shown in (4.1), where Y_i is the class of the character whose data is contained within X_i, i.e. Y_i is the label of X_i.

$$X_i = \{\overbrace{\left[(x,y)_1, (x,y)_2, \ldots, (x,y)_{\ell_1}\right]_1}^{\text{Arbitrary number of coordinates per stroke}}, \ldots, \underbrace{\left[(x,y)_1, (x,y)_2, \ldots, (x,y)_{\ell_N}\right]_N}_{\text{Arbitrary number of strokes per character}}, Y_i\}, \tag{4.1}$$

The data used in this chapter comes from the Online KHATT dataset with individual characters segmented. This data has been produced and worked on by Al-Helali and Mahmoud [AHM16]. Fig. 4.1 shows two examples of characters from this dataset. The dataset contains 22,795 observations, each observation may fall into one of 59 different classes. Note that whilst the Arabic alphabet contains 28 characters, the number of classes is higher owing to the presence of ligatures, characters that the researchers were unable to separate and various punctuation marks. The dataset is split into training and testing sets as shown in Table 4.1, the remainder of the data is kept for use as a validation set. The testing set is chosen such that all the classes that appear in the testing set are ones that also appear in the training set. Of the 59 different classes in the data, 54 are represented in the training set. Appendix A shows an example character

from each class as well as information about the number of observations from that class in each of the training, testing and validation sets and the percentage of observations in each set that are from that specific class. No attempt was made to ensure that the class proportions were equal within the training, testing and validation sets but as can be seen in Appendix A, the proportions are mostly the same within each of the three sets.

Set	Size	No. Classes
Training	11397	54
Testing	6833	51
Validation	4559	49

Table 4.1: Breakdown of splitting of the dataset into training and testing sets

4.2.1 Background

Historically, the most commonly used tool for online handwriting recognition is Hidden Markov Models (HMM), this tool works by predicting an underlying distribution of a hidden time series, from which the handwritten data is sampled [Car+20]. The HMM method still receives a large amount of interest and was recently proposed as a method for classifying segmented Arabic characters by al Helali and Mahmoud [AHM16], resulting in an 82% recognition rate on a database consisting of segmented characters taken from the Online KHATT database. A large number of other papers propose HMM based classifiers for online Arabic handwriting [TKA13].

More recently, the breakthrough in computing abilities (particularly the use of GPUs) combined with the use of machine learning and deep learning has lead to more advanced Neural Networks being used for handwriting recognition [Car+20; AHM17]. Recently Tagougui et al. [Tag+14] implemented a multilayer perception classifier using neural networks and Assaleh et al. [ASH09] used a k Nearest Neighbours approach. These approaches have been shown to

significantly improve on the performance of HMMs and are regarded by many to be the only tool currently capable of accurately recognising handwriting.

1. *Hidden Markov Models*: An HMM is a tool that is used to express probability distributions on sequences of observations of a time series. These discrete observations of the time series form a discrete stochastic process Y_t, where the structure of Y_t is of any form (real valued, integer, letters etc.) with the restriction that it is possible to define a probability distribution on the process. The stochastic process Y_t is assumed to behave dependent on some unknown hidden Markov process X_t.

2. *Neural Networks*: Neural networks were originally an attempt to mimic the way in which the human brain works. Through multiplication of input data by a large number of weights at nodes, followed by an optimisation of the weights so as to achieve the best possible outcome, neural networks have achieved significant prominence in the field of classification owing to their abilities.

4.3 Methodology

The methodology presented below transforms the raw character data (assumed to lie in $[-1, 1]^2$) to a set of feature variables that can be used for classification with a tool such as neural networks.

4.3.1 Fixing dimensionality

Each character contains a number of strokes and each stroke contains a number of coordinates (x, y). The first step is to change the character from a list of strokes to a single list of coordinates. To do this, a third dimension is added, and a number of "jump points" are also added. This third dimension may be thought of as a discrete time dimension, encoding each stroke or jump as its own epoch. For ease of notation, let $(x, y)_i^k$ be the i-th coordinate of stroke k.

The third dimension, P, is added through the following mechanism:

$$(x, y)_i^k \mapsto (x, y, P)_i^k, \tag{4.2}$$

where

$$P = 2 \times (k - 1), \tag{4.3}$$

where k is the number of the stroke that contains the coordinate. For example, the coordinate $(0.1, 0.5)_2^2$ (i.e. the second coordinate in stroke 2) would become $(0.1, 0.5, 2)_2^2$. The data is now of the form:

$$\begin{aligned} X_i = \{&(x, y, 0)_1^1, \ldots, (x, y, 0)_{\ell_1}^1, \\ &(x, y, 2)_1^2, \ldots, (x, y, 2)_{\ell_2}^2, \\ &\ldots \\ &(x, y, 2N-2)_1^N, \ldots, (x, y, 2N-2)_{\ell_N}^N\}. \end{aligned} \tag{4.4}$$

The second step to this stage, is to add coordinates to indicate where the pen leaves the paper (tablet). These are added between coordinates where a new stroke begins and have the exact same x and y coordinate values as the end point of one stroke and beginning point of the next. The P value of these two coordinates that are added is equal to the odd integer that lies between the two even integers of the existing coordinates. For example, the data in (4.4) would become

$$\begin{aligned} & X_i = \{(x, y, 0)_1^1, \ldots, (x, y, 0)_{\ell_1}^1, \\ \text{new} \quad \rightarrow \quad & (x, y, 1)_{\ell_1}^1, (x, y, 1)_1^2, \\ & (x, y, 2)_1^2, \ldots, (x, y, 2)_{\ell_2}^2, \\ & \ldots \\ \text{new} \quad \rightarrow \quad & (x, y, 2N-3)_{\ell_{N-1}}^{N-1}, (x, y, 2N-3)_1^N, \\ & (x, y, 2N-2)_1^N, \ldots, (x, y, 2N-2)_{\ell_N}^N\}. \end{aligned} \tag{4.5}$$

A handful of other ways of fixing the dimensionality of the data have been tested, these included the following:

- Setting the third coordinate to simply be the stroke number and linearly interpolating between the end of one stroke and the beginning of the next.

- Adding coordinates in the same way as above however setting the third dimension to equal 0 when the pen/stylus is on the paper and 1 when the pen is off the paper/tablet

- Concatenating all the strokes and adding a third dimension as time/coordinate number normalised to $t \in [0, 1]$.

These alternate methods for dealing with the dimensionality of the data did not prove to be as powerful as the one presented earlier in this section. It is possible that the number of strokes is of significance and thus as using any of the other processes described removes this piece of information, performance is not as strong.

4.3.2 Signature of the Character

After the data has been converted to a single series of coordinates in $\mathbb{R}^3$, the data extraction method in preparation for the classifier is to be carried out. The method that will be used in this work is known as the dyadic signature, developed based on the iterated integral path signature, invented by Terry Lyons [CK16][1]. The signature of a path was introduced in Chapter 3.

The first step to using the signature to encode details of the characters is to take a linear interpolation - at constant speed - of the sequence of points in 3D that were produced in the first stage. By computing a linear interpolation of the points, this results in an interpretation that is independent of both the speed at which the user wrote the character and the sampling speed of the tablet. Second is to consider this as a path in 3D and to compute either the signature or the log signature of each character (up to a fixed level, say M). The resulting data is used as the input for a classifier.

Previous work using the path signature as a tool for preprocessing online Chinese handwriting has involved the use of a sliding window [Xie+18]. A sliding window of fixed length is applied to the character and the signature of

[1]note that this citation is not by Prof T Lyons, but provides a very easy to follow introduction for using the iterated integral path signature in machine learning

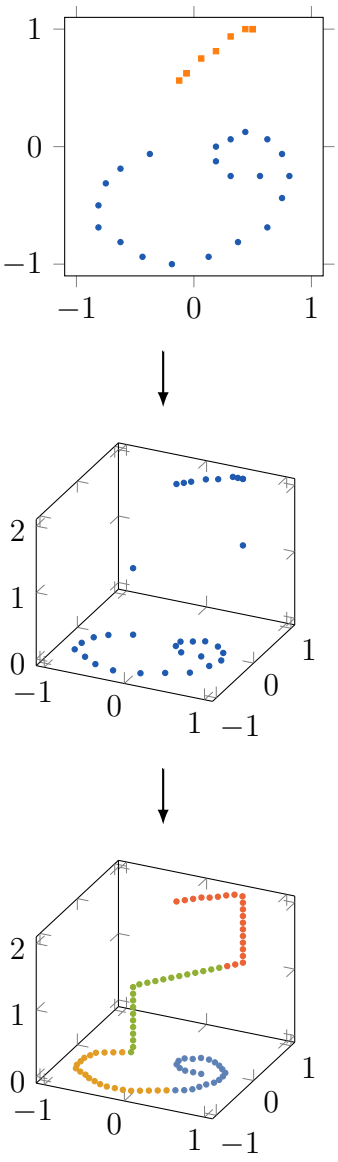


Figure 4.2: Transformation from original character to linearly interpolated character with "pen" dimension added (character *qāf*) and then split into 4 dyadic intervals

each snapshot of the character within the sliding window is computed. This results in a very large amount of data and it is possible for Arabic handwriting which is inherently less complex than Chinese handwriting, this level of detail is unnecessary. The methodology proposed herein attempts to mimic the sliding window but also to drastically decrease the amount of data and the number of computations necessary. Instead of using a sliding window, the character is split

into dyadic intervals.

Definition 4.1. For a given pair of natural numbers $j, k \in \mathbb{N}_0$, where $k < 2^j$, the *dyadic interval* $I_{j,k} \subseteq [0,1] \subset \mathbb{R}$ is the interval

$$I_{j,k} = \left[\frac{k}{2^j}, \frac{k+1}{2^j}\right]. \tag{4.6}$$

Normally, one would consider the partition of $[0,1]$ formed by $\{I_{j,k}\}_{k=0}^{j-1}$ for some j.

In order to use dyadic signatures, instead of calculating the signature of the whole path, the path is split into 2^k sections of equal length and the signature of each of these sections is calculated. The final step is to prepend each of the dyadic signatures with the (x, y, P) coordinates at the start of each dyadic interval. This provides the classification tool with details of the path shape (through the level N iterated integral path signature/log signature) and its location in space. It is this combination of location plus signature that is used as input for the classifier. Thus, when expressing the signature as en element of $\mathbb{R}^d$ (for an appropriate value of d), the resultant data for the first dyadic interval is of the format

$$(x, y, P, S^{(1)}_{[0,1/2^n]}(X), \ldots, S^{(3)}_{[0,1/2^n]}(X), S^{(1,1)}_{[0,1/2^n]}(X), \ldots) \tag{4.7}$$

4.4 Classification methodologies

Two different methodologies are used to classify the processed characters;

- Random Forests
- LSTM Neural Networks

4.4.1 Random Forests

Classification trees have been a tool used for classification informally since the 18th Century however the theory was only formalised in 1984 by Breiman et

al [Bre+84]. Owing to the comparatively small number of classes in the segmented Online KHATT dataset, using classification trees becomes a suitable methodology for use with this data.

Classification trees on continuous data have a property that there are infinitely many possible trees[2], each with different split points. Furthermore, owing to the greedy nature of the algorithm (picking the best initial split) the best possible classification tree is unlikely to be selected. In order to counteract this issue, N different random subsets of the variables x_j are picked and N different classification trees are produced based on these subsets. The resulting N classification trees are then combined into one tree through a process known as voting (whereby the split which occurs most often at a given point is picked). This is the foundation of the theory of random forests [Yiu19]. Random forests were introduced by Tin Kam Ho in 1995 [Tin95].

4.4.2 LSTM Neural Networks

Neural networks are a tool used in data science that have been around for a number of years, however have dramatically increased in usage over the past few years owing to the increase in size of datasets available and also owing to a significant increase in computing abilities, particularly parallel processing and GPU computing abilities. A basic neural network consists of different layers; an input layer, a number of hidden layers; and an output layer.

Building on the theory of neural networks, more complex networks can be employed for different data types; one such example is Recurrent Neural Networks (RNNs). RNNs have the property that the output of each node is fed back into itself, as a way of learning from itself. This is very important for when sequential data or data that has a temporal aspect is employed. One limitation of recurrent neural networks for use with sequential data is how these networks "forget" the past quickly, i.e. large changes in the input will result in sudden large changes in the model, thus forgetting the historical informa-

[2]indeed if one had data containing just 6 Boolean attributes, there would be 18,446,744,073,709,551,616 possible classification trees

tion. To counteract the "forgetfulness" of the network, it is common to add an extra "memory" state to the RNN. This additional state helps to improve the short term memory of the neural network. This is the foundation of a Long Short Term Memory (LSTM) network [Ola]. LSTMs were first introduced by Hochreiter and Schmidhuber in 1997 [HS97].

4.5 Results

Using the two classification methodologies introduced in 4.4, models are fitted to the training data. For this section, when dyadic level n is used, it implies that the following signatures are calculated and used for recognition:

$$\{S_{[0,1]}(X), S_{[0,1/2]}(X), S_{[1/2,1]}(X), \ldots, S_{[0,1/2^n]}(X), \ldots, S_{[(2^n-1)/2^n,1]}(X)\}, \quad (4.8)$$

i.e. the signature of the whole character ($[0,1]$), the signature of each half of the character ($[0,1/2]$, $[1/2,1]$) and all the way up to the signature of each dyadic interval of length $1/2^n$ of the character. The results of applying these classification methodologies to the data are presented in this section.

4.5.1 Random Forests

Using the `ExtraTreesClassifier` classifier in the Python package `scikit-learn`, based on the Extremely Randomised Trees concept developed by Geurts et al. [GEW06], the data is classified by adjusting various parameters. Fixing the function parameters `n_estimators=1000` and `max_depth=20`, whilst adjusting the signature level and the number of dyadic intervals, the results shown in Table 4.2 are obtained.

From the table, a number of conclusions can be drawn, first of all, that the recognition rate of 82% achieved by al Hilali and Mahmoud [AHM16] is easy to not only equal, but to better by approximately 8% using this methodology. Secondly, it is interesting to note that by fixing the dyadic level to be low and increasing the level of the signature, an increase in recognition can be obtained.

This is owing to the fact that more information is contained within higher levels of the signature, furthermore, once the random forest is able to select from more variables, it is more likely to be able to find a better solution. If on the other hand, the dyadic level is fixed to be large, increasing the signature level can be seen to result in a decrease in recognition rate. It is believed that this is a direct result of the greedy nature of the random forest methodology - in that it selects subsequently good splits, ignoring the potential for a better split later on at the expense of a bad split earlier on. Whilst this greediness has been decreased as a result of using random forests as opposed to classification trees, there is still an element of greediness in the algorithm. This is an issue that will be alleviated using neural networks - see §4.5.2. Thirdly, for lower level signatures, increasing the number of dyadic intervals has a direct increase in recognition rate, however increases at the lower dyadic levels are more impactful than at the higher level. In the higher level signatures, such as level 5, increasing the number of dyadic intervals results in a lower recognition rate. It is believed that this is again a result of the greedy nature of the random forest. Finally, it is noticeable that for lower signature level, using the log signature as opposed to the signature resulted in an increased recognition rate. This is likely as a result of the log signature removing a large amount of the redundancy within the signature. Determining the optimal combination of signature level and dyadic level is a hyperparameter choice problem.

4.5.2 LSTM Neural Networks

A custom LSTM was built using the `keras` and `tensorflow-gpu` Python packages. A custom LSTM was developed, as opposed to an (potentially significantly more complex) off the shelf LSTM, in order to highlight the performance of the preprocessing technique that has been developed as well as to ensure repeatability and reproducibility. This LSTM network has three hidden layers, each containing 50 nodes, and was trained using the training set assisted by an NVidia GTX 1080Ti graphics card. The results obtained after training the

Run	(Log) Sig	Signature Level	Dyadic Level	Avg Result % (st. dev.)
1	Sig	2	0	72.26 (0.0743)
2	Sig	2	1	85.44 (0.0438)
3	Sig	2	2	89.10 (0.0785)
4	Sig	2	3	90.15 (0.0418)
5	Sig	2	4	90.43 (0.0841)
6	Sig	2	5	90.37 (0.0749)
7	Log Sig	2	0	73.82 (0.0941)
8	Log Sig	2	1	86.01 (0.0991)
9	Log Sig	2	2	89.05 (0.0898)
10	Log Sig	2	3	90.06 (0.0503)
11	Log Sig	2	4	90.39 (0.0865)
12	Log Sig	2	5	90.32 (0.0365)
13	Sig	3	0	79.48 (0.0648)
14	Sig	3	1	87.65 (0.0450)
15	Sig	3	2	89.87 (0.0778)
16	Sig	3	3	90.58 (0.0825)
17	Sig	3	4	90.32 (0.0697)
18	Sig	3	5	90.13 (0.0399)
19	Log Sig	3	0	80.46 (0.0397)
20	Log Sig	3	1	87.94 (0.0520)
21	Log Sig	3	2	89.97 (0.0312)
22	Log Sig	3	3	90.48 (0.0468)
23	Log Sig	3	4	90.31 (0.0170)
24	Log Sig	3	5	90.11 (0.0397)
25	Sig	5	0	83.14 (0.110)
26	Sig	5	1	88.58 (0.0835)
27	Sig	5	2	89.59 (0.0843)
28	Sig	5	3	89.64 (0.0498)
29	Sig	5	4	89.15 (0.0574)
30	Sig	5	5	88.81 (0.117)
31	Log Sig	5	0	80.64 (0.825)
32	Log Sig	5	1	87.56 (0.107)
33	Log Sig	5	2	89.33 (0.0719)
34	Log Sig	5	3	89.57 (0.0529)
35	Log Sig	5	4	89.38 (0.0794)
36	Log Sig	5	5	89.22 (0.0861)

Table 4.2: Results of using random forest as classification methodology.

LSTM using varying signature signature levels and number of dyadic intervals are obtained and presented in Table 4.3.

The results show that we obtain a recognition rate of over 92.5% when using either the signature or the log signature up to level 5 and with 32 (2^5) dyadic intervals of the character. This rate is a significant improvement on the 82% achieved by Hilali and Mahmoud in their original paper, indeed this represents a 58.3% decrease in incorrect recognition rate. The results in the table demonstrate that by increasing both the signature level and the number of dyadic intervals, the recognition rate on average increases, however with the level 10 signature there was a decrease in the recognition rate. This decrease is likely owing to the dimensionality of the input and the relative simplicity of the LSTM, in order to deal with such a large input, more layers or more nodes per layer may be necessary. Furthermore, it can be seen that when using lower signature levels, the difference between the recognition results when using the signature and log signature is negligible and may possibly be different simply owing to different starting conditions for training the network, suggesting that the LSTM "learnt" about the redundancy within the signature and handled it accordingly. The final point to highlight is that the recognition rate when using the level 10 log signature was markedly higher than when using the level 10 signature. This is further evidence that the complexity of the LSTM used was not enough to deal with the incredibly high dimensionality of the input, indeed the level 10 signature of a path $X : \mathbb{R}^3 \to \mathbb{R}$ contains 88572 values whereas the level 10 log signature of the same path contains only 9382 values.

4.6 Conclusion

It has been demonstrated that the dyadic iterated integral path signature approach to online Arabic character recognition when combined with both random forests and LSTM neural networks, has the ability to perform very well. A number of opportunities present themselves as a result of this methodology, including

Run	(Log) Sig	Signature Level	Dyadic Level	Recognition Rate (%)
1	Sig	2	3	90.44
2	Sig	2	4	90.95
3	Sig	2	5	91.26
4	Log Sig	2	3	90.37
5	Log Sig	2	4	90.83
6	Log Sig	2	5	91.41
7	Sig	3	3	90.47
8	Sig	3	4	91.29
9	Sig	3	5	91.78
10	Log Sig	3	3	90.49
11	Log Sig	3	4	91.13
12	Log Sig	3	5	91.75
13	Sig	5	3	90.63
14	Sig	5	4	92.01
15	Sig	5	5	92.57
16	Log Sig	5	3	90.73
17	Log Sig	5	4	91.88
18	Log Sig	5	5	92.50
19	Sig	10	3	90.62
20	Sig	10	4	91.38
21	Sig	10	5	91.92
22	Log Sig	10	3	90.97
23	Log Sig	10	4	91.50
24	Log Sig	10	5	92.16

Table 4.3: Results of LSTM networks trained using various parameters

the ability for very fast recognition of new characters, owing to the fact that computation of the signature is very fast and capable of being carried out on smartphone processors. Furthermore, the methodology presented here for Arabic Handwriting recognition has been exploited further by Fermanian [Fer21] with success on other datasets including the Google "Quick Draw!" dataset.

Chapter 5

Robust Eigenvalue Polynomial Regression

In §3, it was seen that one can compute polynomials of log signatures. This provides an opportunity to investigate another methodology for building models based upon data and signatures; linear and polynomial regression. One of the biggest drawbacks of using signatures for linear and polynomial regression is that signatures are often clustered in space, and as such standard least squares regression can result in poor results. This is even more so the case when only a small number of points lie in one of these clusters. To highlight this, take the following example using data in $\mathbb{R}$ instead of signatures in signature space.

5.1 An Example

Consider the following scenario; one is presented with data that has been sampled from an unknown smooth function f, the data is contained within three distinct intervals of $\mathbb{R}$,

$$R_1 = [-5, -4], \qquad R_2 = \left[-\frac{1}{2}, \frac{1}{2}\right] \qquad R_3 = [4, 5]. \tag{5.1}$$

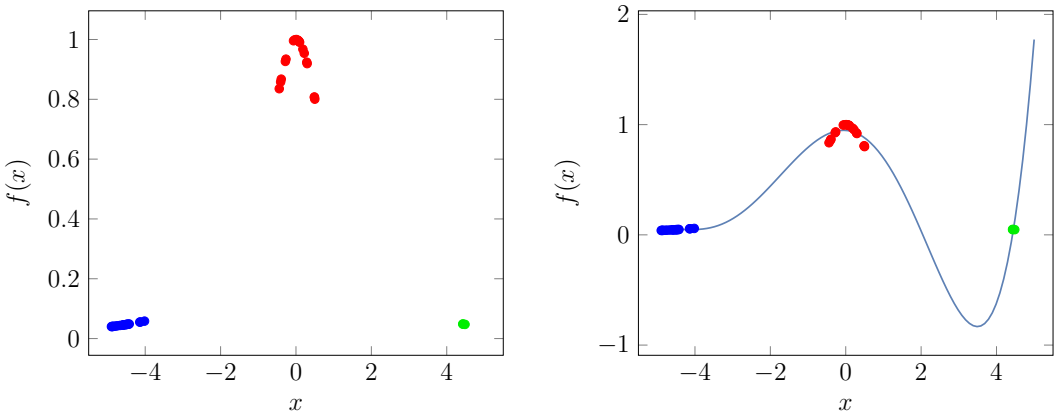
For the purposes of this example, the unknown function $f : \mathbb{R} \to \mathbb{R}$ is defined

as

$$f(x) = \frac{1}{1+x^2}. \tag{5.2}$$

Throughout this chapter, the following notation is used: data in R_i is represented as $X_i = \left\{x_1^{(i)}, \ldots, x_{M_i}^{(i)}\right\}$ and $Y_i = \left\{y_1^{(i)}, \ldots, y_{M_i}^{(i)}\right\}$ where $x_k^{(i)} \in R_i$ and $y_k^{(i)} = f(x_k^{(i)})$.

For this example, there are 20 points in each of R_1 and R_2 and 3 points in R_3, Figure 5.1a plots this data and the actual data can be found in Appendix B.



(a) Example data sampled from $\mathbb{R}$. (b) Degree 5 polynomial fitted to the data.

Figure 5.1: Data and fitted degree 5 polynomial using least squares polynomial regression.

With this data, we wish to obtain a degree 5 polynomial approximation of the underlying function f over the region where we have data, i.e. $[-5, 5]$. The standard approach to a situation such as this one is to use least square regression and find the unique set of parameters $c_0, \ldots, c_N$ (which thus describe the polynomial) that minimises

$$\sum_{j=1}^{M} \left(\sum_{k=0}^{N} c_k x_j^k - y_j \right)^2, \tag{5.3}$$

where all three X_1, X_2 and X_3 are joined and treated as a single dataset where $M = M_1 + M_2 + M_3$ and N is the dimension of the space of polynomials that we wish to use, e.g. for fitting a degree 5 polynomial (as we do in this example), $N = 6$. The result of this is demonstrated in 5.1b.

By definition, least squares regression will result in the least possible l_2 error of the fitted values from the polynomial approximation and the data from which

the approximation is derived is minimised. Unfortunately, upon comparing the approximation to the actual underlying function, it is clear that there is a large L_2 error between the two functions - this comparison can be seen in Figure 5.2. The three data points in region R_3 being clustered very close together results in the large variation between the actual function and the approximation. This is a demonstration of the concept of *points of high influence*.

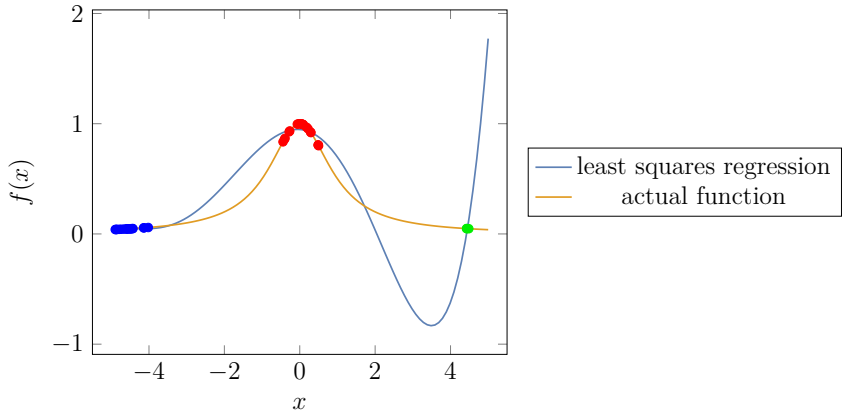


Figure 5.2: Comparison between fitted model from Figure 5.1b and underlying function.

Throughout this chapter, a novel methodology for producing robust polynomial approximations of smooth functions from clustered data with one cluster being highly influential such as in this example is presented. The results of using this new methodology are demonstrated throughout the rest of this example.

The intuition behind this novel methodology is to use a subspace of the space of polynomials to fit an approximation to each region, instead of using the full space of polynomials as is done in standard least squares regression. The polynomial approximation for the function is obtained using the following minimisation in each of the regions R_i, $i = 1 \ldots, K$:

$$\sum_{j=1}^{M_i} \left(\sum_{k=0}^{N_i} c_k^{(i)} \theta_k^{(i)}(x_j^{(i)}) - y_j^{(i)} \right)^2, \tag{5.4}$$

where M_i is the number of points in region R_i, N_i is the number of basis functions of the space of polynomials to use for the data in region R_i, $\theta_k^{(i)}$ are particular basis functions for the space of polynomials, $x_k^{(i)} \in R_i$ and $f(x_k^{(i)}) = y_k^{(i)}$. In this

example,

$$\begin{aligned} K = 3, \quad M_1 = 20, \\ M_2 = 20, \quad M_3 = 3, \end{aligned} \tag{5.5}$$

whilst N_i and $\theta_k^{(i)}$ will be explained further in this chapter. After applying this minimisation, an iterative process is used to improve the fit.

By applying the robust eigenvalue polynomial regression, the resultant approximation is shown in Figure 5.3. Note that the three points in R_3 have not resulted in very large perturbations away from the actual function.

In this chapter, to determine the performance of different models, we will consider two different error measures; one on the individual points (l_2) and one on the whole fitted models (L_2). For two continuous functions $f, g : U \mapsto \mathbb{R}$ where $U \subseteq \mathbb{R}^d$ for some d, the error measures are:

$$l_2 \text{ error:} \quad \sqrt{\sum_{i=1}^{N} \left(f(x_i) - g(x_i)\right)^2} \tag{5.6}$$

$$L_2 \text{ error:} \quad \sqrt{\int_V \left(f(\mathbf{x}) - g(\mathbf{x})\right)^2 \, \mathrm{d}\mathbf{x}}, \tag{5.7}$$

where N is the number of points used for fitting the model, x_i are the points used for fitting the model, $V \subseteq U$ is the subset over which we wish to consider the L_2 error and $\mathbf{x} \in V$ .Table 5.1 lists some statistics measuring the ability of the fitted model to approximate the underlying function.

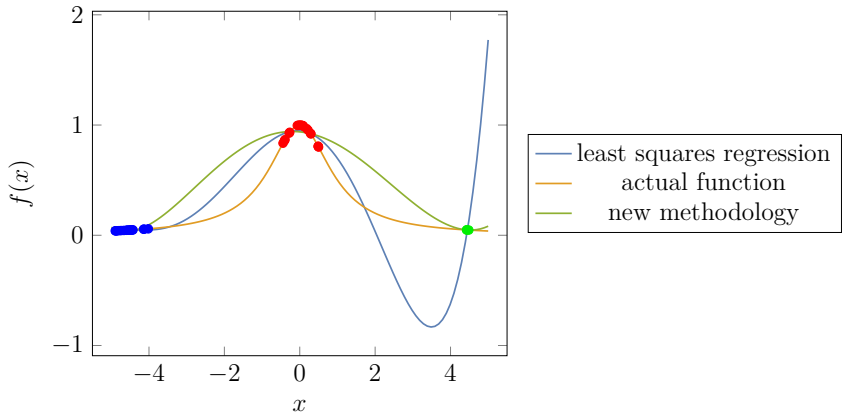


Figure 5.3: Example 1 (same example that has been used throughout this chapter).

Model	Statistic	Value
Linear regression	l_2 error	0.254942
	L_2 error over R	1.3125
	L_2 error over $\cup R_i$	0.79377
New methodology	l_2 error	0.288437
	L_2 error over R	0.847252
	L_2 error over $\cup R_i$	0.0867305

Table 5.1: l_2 and L_2 error statistics (as defined in (5.6) and (5.7)) for each model in Figure 5.3

From Table 5.1, it is clear that whilst there is some loss of accuracy in terms of l_2 error (i.e. the difference between the actual and fitted values using each approximation), there is a significant increase in accuracy in an L_2 sense, over both the whole region and the union of the individual regions.

5.2 Introduction

It is often the case that samples from an unknown function are received in different mutually exclusive intervals [Gua17]. These intervals could contain any number of points, and if the underlying, unknown, function is computationally expensive then there is a possibility that in one of these regions, the number of observations may be small. When fitting polynomial approximations to the data, the cluster containing a small number of points has the potential to aversely impact the fit of a polynomial approximation using standard least squares regression. These points are known as *influential observations* [Bri98].

Definition 5.1. An *influential observation* is an observation with the property that if it is removed, when a regression model is refitted, there is a large difference in the parameter estimates model.

Whilst in the case of noisy data sampled from a population, such influential observations may be considered outliers [Ken92], in the scenario outlined above

there is no concept of outliers as the data is known to be sampled from the underlying function.

A majority of the methodologies utilised in applied fields such as econometrics [Ken92] and geography [BBR09] focus on identification of influential points through commonly used tools such as cross validation or minimising the l_2 error of a fitted model through processes such as M-estimators [Hub64] and L-estimators [Sti73]; all of which are focussed on reducing the impact of the influential point(s) on the l_2 error of the rest of the model.

Work done by Guan [Gua17] introduces a methodology specific to obtaining mixture model approximations for probability density functions where samples come in clusters. Whilst the methodology proposed by Guan addresses the issue of clustered data and fitting polynomial approximations to clustered data, the methodology does not address the issue of influential observations.

The structure of this chapter follows the following format; §5.3 provides an overview of the method, §5.4-5.6 explains the details of the process to carry out the robust eigenvalue polynomial regression, §5.7 demonstrates the performance of the robust eigenvalue polynomial regression with a number of examples, showing scenarios in which this methodology performs well and finally §5.8 includes closing remarks and explains some limitations of this methodology.

5.3 Method

In this section, we introduce the details of the methodology demonstrated in §5.1 whereby one can generate single polynomial approximations for functions where an imbalance in the number of points in each region does not impact the resulting fitted model, i.e. the data is grouped and one of the groups is a cluster of high leverage points. As seen in the example in §5.1, a small number of high leverage points can adversely impact the fit of the polynomial approximation.

The intuition behind the methodology presented within lies in fitting an individual polynomial model to each of the N regions of data, computing the

sum of these models and iteratively adjusting the resultant model in order to find a stable solution. The N initial models are fitted using subspaces of the space of polynomials (of the chosen degree), where the subspace is chosen such that information is maximised within the region containing data and minimised outside of the region.

As mentioned above, the intuition is to fit a separate model (say $\widehat{f}_{R_k}$) to the data in each individual region R_k, and compute the sum of these individual models. In an ideal scenario, $\widehat{f}_{R_k}$ would be compactly supported on R_k for each k, however the aim is for the final model $\widehat{f} = \sum \widehat{f}_{R_k}$ to be a polynomial and the only way in which $\sum \widehat{f}_{R_k}$ is a polynomial is if $\widehat{f}_{R_k}$ are all polynomials. This leads to an impossible situation as polynomials are not compactly supported.

In order to obtain the subspaces of the space of polynomials mentioned above, the intuition is to find K bases (i.e. one for each R_i) for the space of polynomials (of the desired degree and dimension) where in each basis, each basis function is non-zero over the region where the data lies and as close to zero as possible outside of that region. As an example, Figure 5.4 demonstrates such a basis for the space of polynomials in one variable up to degree 3 (which shall be notated for ease as $\mathbb{R}_3[x_1]$ where in general the space of polynomials in d variables up to degree m is notated as $\mathbb{R}_m[x_1, \ldots, x_d]$), where the functions are attempting to match the following form:

$$\widehat{f}_{R_1,R}(x) = \begin{cases} g(x), & x \in R_1 \\ \approx 0, & x \in R \setminus R_1 \end{cases}, \tag{5.8}$$

where $R_1 = [0, 1]$ and $R = [0, 2.5]$. The basis functions in Figure 5.4 are produced using the methodology presented in Algorithm 5.1.

Notice in Figure 5.4 that the basis functions are ordered 1 through 4, with basis function γ_1 closest to resembling what we were looking for and γ_4 the furthest from the desired shape with γ_2 and γ_3 incrementally decreasing in their ability to match our desired properties. Table 5.2 shows the value of

$$\langle f, g \rangle_{L_2(R)} = \int_R (f(\mathbf{x}) \cdot g(\mathbf{x}))\, \mathrm{d}\mathbf{x}, \tag{5.9}$$

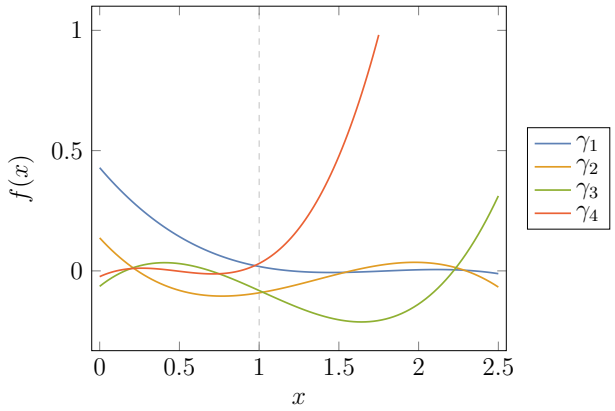


Figure 5.4: Example basis of polynomial space with desired properties

for $\gamma_1, \ldots, \gamma_4$ using R_1, $R \setminus R_1$ and R.

Function	$\langle \cdot, \cdot \rangle_{L_2(R_1)}$	$\langle \cdot, \cdot \rangle_{L_2(R \setminus R_1)}$	$\langle \cdot, \cdot \rangle_{L_2(R)}$	$\langle \cdot, \cdot \rangle_{L_2(R)} / \langle \cdot, \cdot \rangle_{L_2(R_1)}$
γ_1	0.0427862	0.0000468582	0.0428331	1.0011
γ_2	0.00689222	0.00248732	0.00937954	1.36089
γ_3	0.00114839	0.0405525	0.0417009	36.3124
γ_4	0.000105745	4.97664	4.97674	47063.8

Table 5.2: Comparison between $\langle \cdot, \cdot \rangle_{L_2(\cdot)}$ for different R for each basis function in Figure 5.4

By taking K (one for each R_i) such bases and using only a portion of each, it is possible to obtain K subspaces of the space of polynomials. By fitting a least squares optimised polynomial approximation using the data in each region, one obtains K polynomials and these are the closest possible polynomials to matching the desired form shown in (5.8). Notice that it is necessary to use only a portion of the basis for each region as using the whole basis would be no different to doing standard least squares regression in each region.

5.4 Obtaining the basis

When defining the K different bases for the space of polynomials, the basis that we pick is dependent upon two regions, the region where the data lies (for each

given cluster $R_1, R_2, \ldots, R_K$) and also the region, R, over which we wish to obtain an estimate for the function.

The process to obtain the basis for the space of polynomials with the desired properties as explained above, for each region $R_1, R_2, \ldots, R_n$ is presented in Algorithm 5.1.

Algorithm 5.1: Computing basis for the space of polynomials where the basis functions exhibit the desired properties.

Inputs: Degree of polynomial $= m$; dimension of polynomial $= d$; interval containing cluster $= R_k$; interval over which estimate is desired $= R$; any basis of the space of polynomials $\mathbb{R}_m[x_1, \ldots, x_d]$, $\beta = \{\beta_1(x_1, \ldots, x_d), \ldots, \beta_N(x_1, \ldots, x_d)\}$

Step 1: Define the matrices $A = \{a_{i,j}\}_{i,j\geq 1}$ and $B = \{b_{i,j}\}_{i,j\geq 1}$ where

$$\begin{aligned} a_{i,j} &= \langle \beta_i, \beta_j \rangle_{L_2(R_k)} \\ b_{i,j} &= \langle \beta_i, \beta_j \rangle_{L_2(R)} \end{aligned} \tag{5.10}$$

this computes the inner product of each pair of basis functions over each region.

Step 2: Let $[T]_\beta = A^{-1}B$ be the representation with respect to the basis β of the linear transformation T between the space of polynomials defined over R_i to the space of polynomials defined over R

Step 3: Compute the eigenvalues ($\alpha = \{\alpha_1, \ldots, \alpha_N\}$) [where for ease of notation, $\alpha_i \leq \alpha_j$ for $i < j$] and eigenvectors ($\phi = \{\phi_1, \ldots, \phi_N\}$) of the matrix $[T]_\beta$

Step 4: Treat β as a vector and compute the dot product $\beta.\phi_i$ for each i to obtain $\gamma = \{\gamma_1(x_1, \ldots, x_n), \ldots, \gamma_N(x_1, \ldots, x_n)\}$, this γ forms a basis for the space of polynomials with the desired property. γ is known as the eigenfunctions of $[T]_\beta$.

Notes to Algorithm 5.1:

1. Whilst the eigenfunctions/eigenvectors of $[T]_\beta$ depend on the choice of β, the eigenvalues will remain the same

2. $N = \binom{d+m}{d}$ is the dimension of the space $\mathbb{R}_m[x_1, \ldots, x_d]$

5.4.1 The meaning of these basis functions

The γ obtained from Algorithm 5.1 form the basis that we have been looking for. The basis function based upon the eigenvector associated with the smallest magnitude eigenvalue is the basis function that most closely matches our desired form and the function based upon the eigenvector associated with the largest magnitude eigenvalue is the one that least matches our desired format as presented in (5.8). More concisely, for low values of i, γ_i closely matches the form in (5.8) but as i increases, the functions are increasingly different in shape to the objective format. This is precisely as a result of the fact that the eigenvalues are the ratio of the inner product of the eigenfunctions, i.e.

$$\alpha_i = \frac{\langle \gamma_i, \gamma_i \rangle_{L_2(R)}}{\langle \gamma_i, \gamma_i \rangle_{L_2(R_k)}}. \tag{5.11}$$

If the eigenvalue is close to 1, then the majority of the weight of the basis function is centred over R_k whereas a larger eigenvalue indicates that a larger portion of the weight of the basis function is in $R \setminus R_k$ than in R_k.

5.5 Fitting the model

Having computed the basis functions for each R_k using the methodology described in Algorithm 5.1, the next step in fitting a model using our proposed methodology is to do a "recursive regression", this is done in three steps:

1. For each region R_k, fit an initial model, $\widehat{f}_{R_k}$, from the data in R_k, using a subspace of the space of polynomials defined using the basis functions computed in Algorithm 5.1. See §5.5.1 and Algorithm 5.2.

2. Obtain an initial polynomial approximation $\widehat{f} = \sum \widehat{f}_{R_k}$. See **Step 3** of Algorithm 5.2.

3. Iteratively alter $\widehat{f}$ using the data in each of the regions to adjust for the impact of $\widehat{f}_{R_k}$ in $R_1, \ldots, R_{k-1}, R_{k+1}, \ldots, R_K$. See §5.5.2 and Algorithm 5.3.

The final step here is necessary as each $\widehat{f}_{R_k}$ contains information within $R \backslash R_k$ which therefore, as a result of computing the sum, will have a detrimental impact on the fit of $\widehat{f}$ for $i \neq k$.

5.5.1 Fitting the initial model

The first step is relatively simple, and that is to fit K initial models using the data in each of the K regions and some subset of the basis functions. The methodology through which this subset is chosen is discussed further on in this chapter. Recall equations (5.3) and (5.4) in the introduction to this chapter. Whereas in standard polynomial regression, a model is fitted by choosing the parameters $c_0, c_1, \ldots, c_N$ that minimise

$$\sum_{j=1}^{M} \left(\sum_{k=0}^{N} c_k x_j^k - y_j \right)^2, \tag{5.3}$$

we instead wish to minimise

$$\sum_{j=1}^{M_i} \left(\sum_{k=0}^{N_i} c_k^{(i)} \theta_k^{(i)}(x_j^{(i)}) - y_j^{(i)} \right)^2. \tag{5.4}$$

for each R_i, where the parameters are as explained previously. The steps to produce the initial model are presented in Algorithm 5.2.

Notes to Algorithm 5.2: **Step 2** of this algorithm involves selecting the number of basis functions to use in each region, this will be discussed in §5.6. It is important to note that the number of basis functions in each region must not exceed the number of points in that region, i.e. $N_i \leq M_i$ for all i.

5.5.2 Iteratively altering the model

Having fitted the initial model using the methodology presented in Algorithm 5.2, the next key part of this methodology is to iteratively alter the model. The idea behind this is as follows; we have fitted K linear models and taken the sum of these linear models, whilst we have used subsets of the space of polynomials that were optimised to minimise interaction between the component parts of

Algorithm 5.2: Fitting the initial model

Inputs: Degree of polynomial = m; dimension of polynomial = d;
K intervals containing clusters = $\{R_i\}_{i=1}^{K}$; interval over which estimate is desired = R;
K sets of data $X_i = \{\mathbf{x}_1^{(i)}, \ldots, \mathbf{x}_{M_i}^{(i)}\}$ and associated $Y_i = \{y_1^{(i)}, \ldots, y_{M_i}^{(i)}\}$

Step 1: Compute the basis functions for each R_i using Algorithm 5.1 and let $\gamma^{(i)} = \{\gamma_0^{(i)}(\mathbf{x}), \ldots, \gamma_N^{(i)}(\mathbf{x})\}$ be the basis functions associated with the region R_i

Step 2: For each R_i, pick N_i; the number of basis functions to use to fit the model, and find $c_0^{(i)}, \ldots, c_{N_i}^{(i)}$ such that

$$\sum_{j=1}^{M_i} \left(\sum_{k=0}^{N_i} c_k^{(i)} \gamma_k^{(i)}(\mathbf{x}_j^{(i)}) - y_j^{(i)} \right)^2 \tag{5.12}$$

is minimised. This gives

$$\widehat{f}_{R_i,N_i}(\mathbf{x}) = \sum_{k=0}^{N_i} c_k^{(i)} \gamma_k^{(i)}(\mathbf{x}) \tag{5.13}$$

Step 3: Obtain the whole initial fitted model as

$$\begin{aligned} \widehat{f}_{1,\mathbf{N}}(\mathbf{x}) &= \sum_{i=1}^{K} \widehat{f}_{R_i,N_i}(\mathbf{x}) \\ &= \sum_{i=1}^{K} \sum_{k=0}^{N_i} c_k^{(i)} \gamma_k^{(i)}(\mathbf{x}) \end{aligned} \tag{5.14}$$

where $\mathbf{N} = \{N_1, \ldots, N_K\}$.

the sum, this is far from perfect as the non-zero nature of the basis functions outside of their intended regions will result in interactions which degrade the fitting (in an l_2 sense) of the model. By iteratively altering the model, it is the aim that these interactions between the models are minimised and a well fitting (in an l_2 sense) single polynomial model over the entire of the region R is the result. The methodology for the iterative improvements are presented in Algorithm 5.3.

Algorithm 5.3: Iteratively altering the initial model to reduce interference between submodels.

Inputs: K sets of data $X_i = \{\mathbf{x}_1^{(i)}, \ldots, \mathbf{x}_{M_i}^{(i)}\}$ and associated $Y_i = \{y_1^{(i)}, \ldots, y_{M_i}^{(i)}\}$; initial model $\widehat{f}_{1,\mathbf{N}}$ from Algorithm 5.2

For ***k*=1, ...**

Step 1: Apply $\widehat{f}_{k,\mathbf{N}}$ to the data in X_i for each i and subtract the correct value to obtain residuals:

$$\begin{aligned} E_{i,k} &= \{y_1^{(i)} - \widehat{f}_{k,\mathbf{N}}(\mathbf{x}_1^{(i)}), \ldots, y_{M_i}^{(i)} - \widehat{f}_{k,\mathbf{N}}(\mathbf{x}_{M_i}^{(i)})\} \\ &= \{\varepsilon_{1,\mathbf{N},k}^{(i)}, \ldots, \varepsilon_{M_i,\mathbf{N},k}^{(i)}\} \end{aligned} \tag{5.15}$$

Step 2: Repeat Algorithm 5.2 using $E_{i,k}$ instead of Y_i. The result of Algorithm 5.2 when applied to the residuals is the correction term $\widehat{C}_{k,\mathbf{N}}$

Step 3: The updated full model is then given as

$$\widehat{f}_{k+1,\mathbf{N}} = \widehat{f}_{k,\mathbf{N}} + \widehat{C}_{k,\mathbf{N}}. \tag{5.16}$$

Step 4: Continue the loop until the sequence of functions has converged or the sequence has exploded. The sequence converges if:

$$\sqrt{\int_R \left(\widehat{f}_{k+1,\mathbf{N}}(\mathbf{x}) - \widehat{f}_{k,\mathbf{N}}(\mathbf{x})\right)^2 \, \mathrm{d}\mathbf{x}} \to 0 \tag{5.17}$$

as k increases. Using eigenfunctions associated with smaller eigenvalues will converge whilst those associated with larger values will explode.

The final result of applying Algorithms 5.1, 5.2 & 5.3 will be a linear combi-

nation of fitted polynomials forming a single polynomial approximation of the data.

5.6 Subspace selection

Thus far, we have introduced the process for implementing the methodology. In Algorithm 5.2 where the initial model is fitted to the data, **Step 2** requires selecting an N_i for each R_i, i.e. selecting the number of basis functions (and therefore the dimension) of the space of polynomials to use for each region. From the explanation in §5.4.1, it is seen that the higher the eigenvalue, the worse the associated basis function is in terms of maximising information over R_i when compared with the rest of R. On the other hand, by including only one basis function (for example) then the number of degrees of freedom is not high and therefore the fit of the model will be poor. It is therefore necessary to determine what the optimal N_i is for each region.

5.6.1 Different selection schemata

We were unable to determine a single rule to always select the optimal eigenfunction combination. This was, in the main part, owing to the variability of scenarios, i.e. different number of regions, different lengths of regions, differing number of points in each region etc, all impacting the optimal choice for the number of basis functions per region. We therefore propose to fit a number of different models to the data and select the best performing model of those returned based on the l_2 error of the resulting polynomial approximations. In order to determine what may be the best process to select the basis functions for each region, a number of different schemata were tested. The three criteria that remained constant throughout each schema were:

- the number of basis functions within each region cannot be more than the number of points in that region, this is because it is not possible to fit a

stable model using more degrees of freedom than there are data points – this would introduce multicollinearity

- the collection of all basis functions used across all the regions must be linearly independent; this is because adding linearly dependant basis functions results in no convergence of the model during the iteration stage – this is once again as a result of multicollinearity
- if within region k, a basis function associated with an eigenvalue $\alpha_i^{(k)}$ is included, then all basis functions that are associated with $\alpha_j^{(k)}$ for $j < i$ must also be included (recall that $\alpha_i < \alpha_j$ for $i < j$)

The three selection schemata are as follows, where each has the caveat that the three criteria above must be satisfied:

- Exhaustive search – try every single combination
- Same number from each region – search with much fewer options, selecting exactly n from each region for $n = 1, \ldots, N$
- Eigenvalue based filter – select basis functions from each region based upon their eigenvalue, see Algorithm 5.4

Each of these had various advantages and disadvantages.

Exhaustive search: Using a full exhaustive search and selecting the basis function combination that results in the lowest l_2 error will always give the best result in an l_2 sense that is possible with this methodology. However, it is computationally intensive and a vast amount of unnecessary calculation is done. The complexity of this methodology is $O(N^K)$ where N is the dimension of the space of polynomials in which the model is being approximated and K is the number of regions R_i that are used. Furthermore, the selection which results in the lowest l_2 error may be one that uses a large number of basis functions from one region, thus reducing the ability of the methodology to select a model which is not overly impacted by the data in one particular region.

Same number from each region: Reducing the number of possible options to choose from in the exhaustive search is clearly a favourable option as it decreases the complexity from $O(N^K)$ to $O(NK)$. Upon testing this selection schema, it was found that often the better basis function selection were not included and thus not able to be chosen; this was as a result of the eigenvalues in each region following different distributions.

Eigenvalue based filter: This schema is equal in terms of computational complexity to the previous one however improves upon it as it uses information about the basis functions, i.e. the associated eigenvalue. By selecting basis functions based upon the eigenvalue, the impact of differing eigenvalue distributions associated with each region was lessened. The schema is described in Algorithm 5.4. The intuition behind this selection schema is that if a basis function is added in one region, another need not be added in another unless they are similarly behaved in terms of information both in and outside of the region.

Notes to Algorithm 5.4:

- Given that one of the main aims of this methodology is to work when a region of data contains a small number of highly influential points, care must be taken with the region containing these points. It is not possible to fit a model using more basis functions than there are points in the region, therefore it is only necessary to look at Λ_k up to the point whereby there are at most M_i eigenvalues selected in region R_i where $i = \arg\min(M_1, \ldots, M_K)$.

- The filtering of eigenvalues based on some real valued threshold in (5.18) is possible because the eigenvalues are all real-valued. Indeed, each eigenvalue is a ratio of two inner products - see (5.11).

Algorithm 5.4: Schema for selecting number of basis functions for each region.

Inputs: Eigenvalues $\alpha^{(i)} = \{\alpha_1^{(i)}, \ldots, \alpha_N^{(i)}\}$ from Algorithm 5.2 for each R_i

Step 1: Define $H(\alpha^{(i)}, \ell)$ to be the number of eigenvalues in $\alpha^{(i)}$ that are less than or equal to $\ell \in \mathbb{R}$, that is

$$H(\alpha^{(i)}, \ell) = \left|\{\alpha \in \alpha^{(i)} | \alpha \leq \ell\}\right| \tag{5.18}$$

Step 2: Let Λ_k be defined as

$$\begin{aligned}\Lambda_k = \min\{\ell \in \mathbb{R} | \max(H(\alpha^{(1)}, \ell), \ldots, H(\alpha^{(K)}, \ell)) = k \text{ and} \\ \min(H(\alpha^{(1)}, \ell), \ldots, H(\alpha^{(K)}, \ell)) \geq 1\},\end{aligned} \tag{5.19}$$

that is, let Λ_k be the smallest value of ℓ such that at least one eigenvalue is less than or equal to ℓ in each $\alpha^{(i)}$ and there is at least one $\alpha^{(i)}$ containing exactly k eigenvalues less than or equal to ℓ and none of the $\alpha^{(i)}$ contain more than k eigenvalues greater than ℓ.

Step 3: For each value of Λ_k select the eigenvalues in each $\alpha^{(i)}$ such that $\alpha_j^{(i)} \leq \Lambda_k$. These eigenvalues correspond to eigenfunction basis elements and these are the combinations for which to compute the models to compare.

Step 4: With the collections of basis functions chosen in **Step 3**, select only those such that the following two conditions hold:

- The number of basis functions selected in each region must be less than or equal to the number of points in that region
- The basis functions selected must all be linearly independent

5.6.2 Demonstration of three selection schemata

Using the example in §5.1, the three different selection schemata are demonstrated here.

Exhaustive search

By looking at every single possible model that satisfies the three criteria, in this instance there are 19 different possible basis function selections. These are presented in Figure 5.5.

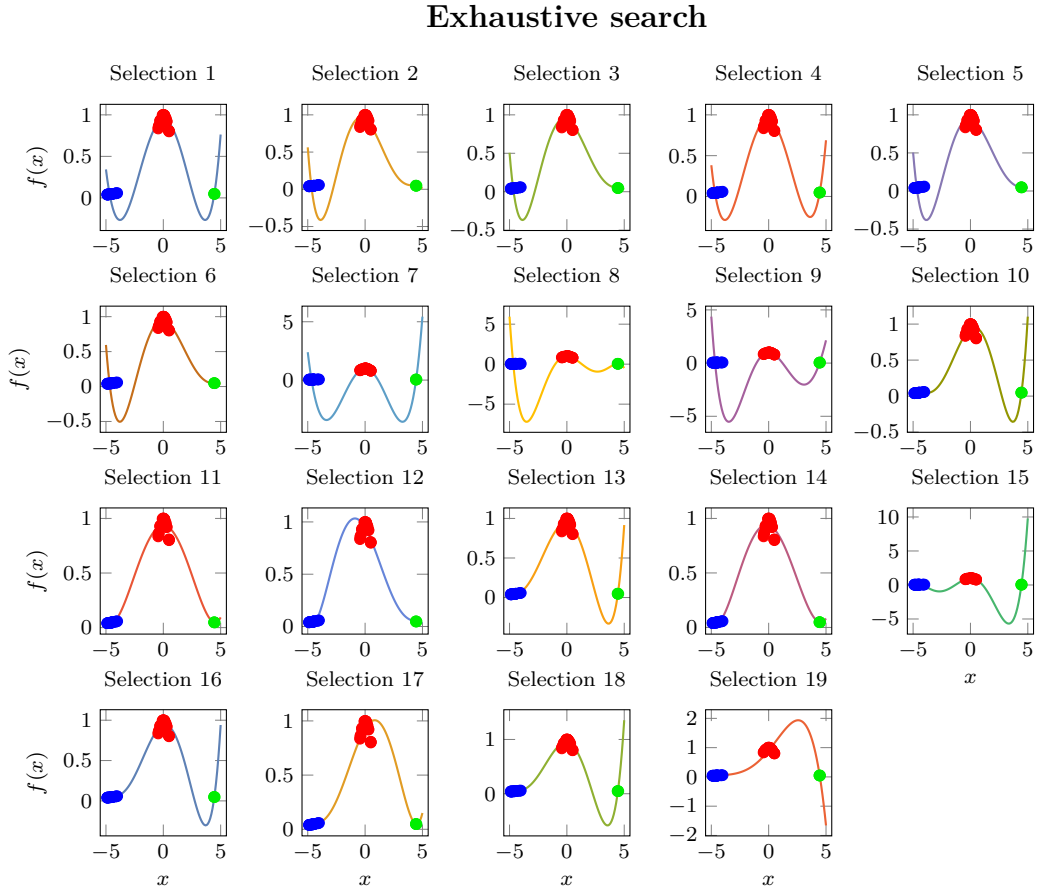


Figure 5.5: Model selection using exhaustive search.

The l_2 errors for each model are shown in Table 5.3. It can be seen from this table that "Selection 18" offers the lowest l_2 error and thus this is the one that is selected. When looking at the plot of the fitted model associated with this selection, one can see that it is very similar to the result provided by standard

least squares regression - see §5.1. Therefore it is possible that this selection has not benefited from the basis functions that have been generated.

Selection	l_2 error	Eigenvalues			Selection	l_2 error	Eigenvalues		
		R_1	R_2	R_3			R_1	R_2	R_3
1	0.761147	1	1	1	11	0.289988	2	1	2
2	1.18314	1	1	2	12	0.357073	2	1	3
3	1.06604	1	1	3	13	0.263927	2	2	1
4	0.831548	1	2	1	14	0.288437	2	2	2
5	1.07444	1	2	2	15	0.690424	2	3	1
6	1.30776	1	2	3	16	0.278391	3	1	1
7	6.00071	1	3	1	17	0.386383	3	1	2
8	14.1242	1	3	2	18	0.257126	3	2	1
9	10.5149	1	4	1	19	0.748469	4	1	1
10	0.316847	2	1	1					

Table 5.3: l_2 error for each model in Figure 5.5

Same number from each region

Owing to the fact that there are only three points contained within region R_3, there are a maximum of three possible models using this selection schema. Notice, however, that if three basis functions are selected in each region, it is not possible for all 9 basis functions to be linearly independent as the dimension of the space of polynomials of degree 5 in one variable is only 6. We therefore have only two choices of basis selection. These are presented in Figure 5.6.

The l_2 errors for each model are shown in Table 5.4. Clearly the number of models has been massively reduced (from 19 to just 2 in this case), Furthermore, it can be seen from the table that "Selection 2" offers the lowest l_2 error and thus this is the one that is selected.

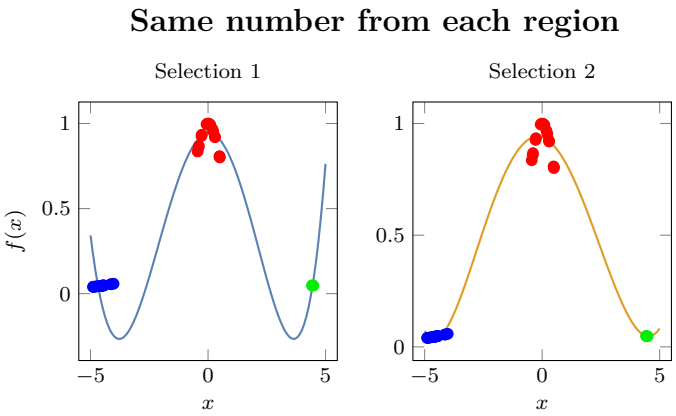


Figure 5.6: Model selection using same number from each region.

Selection	l_2 error	Eigenvalues		
		R_1	R_2	R_3
1	0.761147	1	1	1
2	0.288437	2	2	2

Table 5.4: l_2 error for each model in Figure 5.6

Eigenvalue based filter

By adding basis functions in each region based upon the eigenvalues associated with each basis function, three models are generated. In this particular example, two remain the same as when selecting an equal number from each region, however an additional one is also included. The three different fitted models are presented in Figure 5.7.

The l_2 errors for each model are shown in Table 5.5. Notice that whilst Selection 2 has been added, when compared to using the same number of basis functions in each region. In this example, "Selection 3" (which is the same as "Selection 2" from Table 5.4) has been chosen. Indeed this model is the one that is shown in §5.1.

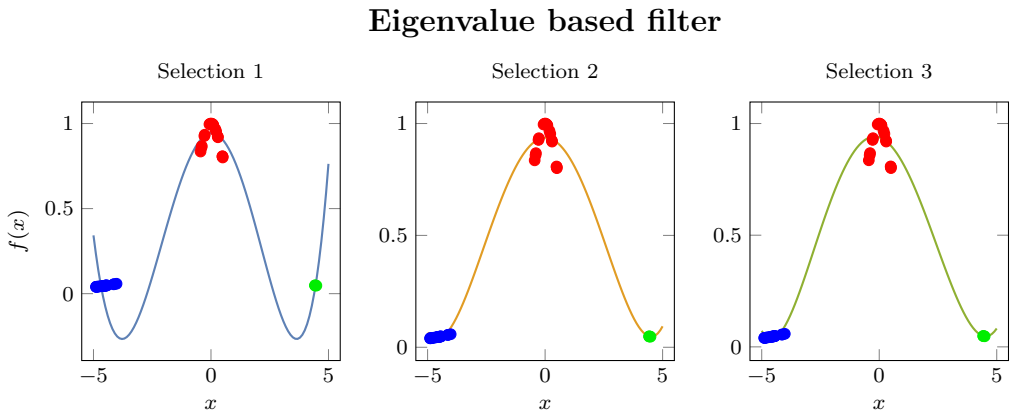


Figure 5.7: Model selection using eigenvalue based filter described in Algorithm 5.4.

Selection	l_2 error	Eigenvalues		
		R_1	R_2	R_3
1	0.761147	1	1	1
2	0.289988	2	1	2
3	0.288437	2	2	2

Table 5.5: l_2 error for each model in Figure 5.7

5.7 Examples

In this section, a handful of examples are provided to demonstrate the abilities of the methodology.

5.7.1 Example 1

For this first example, the underlying function is one for which no closed form exists, and is computationally expensive to compute values, therefore obtaining a polynomial estimate can be useful.

The data is clustered in two regions:

$$R_1 = [-20, -15], \qquad R_2 = [-5, 5], \tag{5.20}$$

with 3 points in R_1 and 20 points in R_2. We aim to find a degree 6 polynomial approximation to the underlying function over the region $[-20, 5]$. Figure 5.8

shows the data, the fitted polynomial using both standard least squares regression as well as our methodology and the actual function. Statistics about the capability of the approximations are presented in Table 5.6.

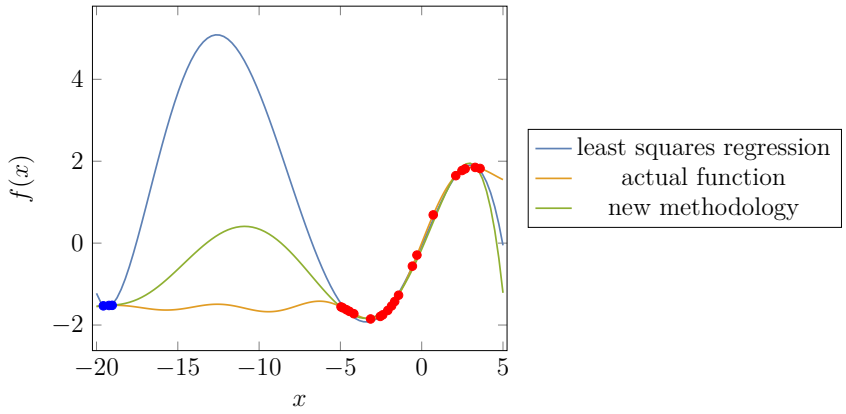


Figure 5.8: Data and degree 6 polynomials fitted to the data.

The underlying function here is

$$f(x) = \begin{cases} \int_0^x \frac{\sin(t)}{t}\,\mathrm{d}t, & x \geq 0 \\ \int_x^0 \frac{\sin(t)}{t}\,\mathrm{d}t, & x < 0 \end{cases} \tag{5.21}$$

Model	Statistic	Value
Linear regression	l_2 error	0.235091
	L_2 error over R	16.4546
	L_2 error over $\cup R_i$	6.90063
New methodology	l_2 error	0.338999
	L_2 error over R	5.00754
	L_2 error over $\cup R_i$	2.5914

Table 5.6: l_2 and L_2 error statistics for each model in Figure 5.8

5.7.2 Example 2

This example once again uses an underlying function that has no closed form.

$$f(x) = \begin{cases} 1 - \frac{2}{\pi}\int_0^x \mathrm{e}^{-t^2}\,\mathrm{d}t, & x \geq 0 \\ 1 - \frac{2}{\pi}\int_x^0 \mathrm{e}^{-t^2}\,\mathrm{d}t, & x < 0 \end{cases}, \tag{5.22}$$

and data has been generated in the intervals

$$R_1 = \left[-\frac{5}{2}, -\frac{9}{4}\right], \qquad R_2 = \left[-\frac{1}{2}, 0\right], \qquad R_3 = \left[\frac{1}{2}, 1\right], \tag{5.23}$$

thus giving $R = \left[-\frac{5}{2}, 1\right]$ with 3 points in R_1, 15 in R_2 and 25 points in R_3. The data and results of standard least squares regression and our methodology are demonstrated in Figure 5.9 and the summary statistics are provided in Table 5.7.

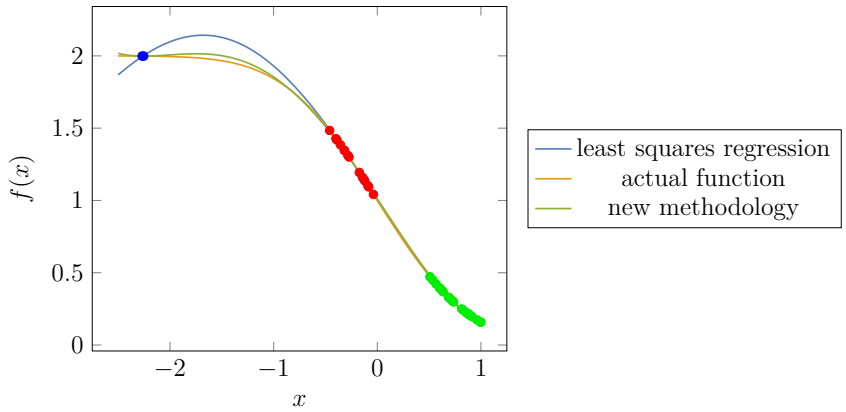


Figure 5.9: Data and degree 4 polynomials fitted to the data.

Model	Statistic	Value
Linear regression	l_2 error	0.0176658
	L_2 error over R	0.152454
	L_2 error over $\cup R_i$	0.0397447
New methodology	l_2 error	0.0329347
	L_2 error over R	0.0307476
	L_2 error over $\cup R_i$	0.0115115

Table 5.7: l_2 and L_2 error statistics for each model in Figure 5.9

5.7.3 Example 3

Thus far, all of the examples that have been presented use data that is sampled directly from the underlying function. There is, however, no reason why this methodology cannot be used on noisy samples from the underlying function. In

this example, noisy data is sampled from

$$f(x) = \sin(\sqrt{x}) \tag{5.24}$$

in the regions

$$R_1 = [0, 5]\,, \qquad R_2 = [18, 20]\,, \tag{5.25}$$

with 30 points sampled in R_1 and 2 in R_2. The data and the results of fitting polynomial approximations using standard least squares regression and the robust eigenvalue polynomial regression are shown in Figure 5.10 with summary statistics shown in Table 5.8.

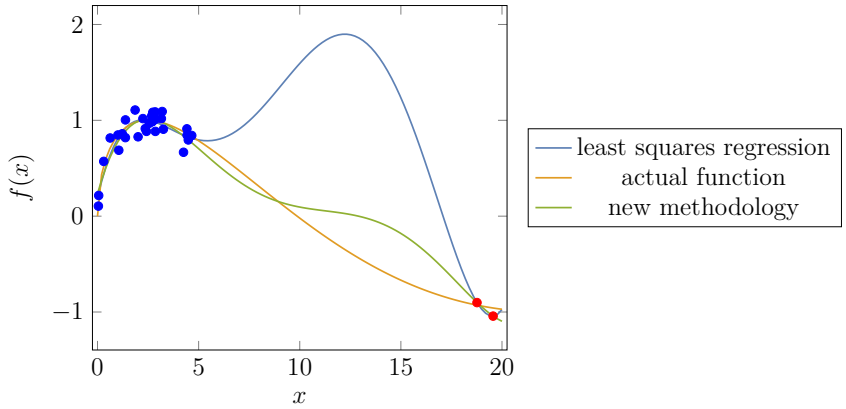


Figure 5.10: Data and degree 5 polynomials fitted to the data.

Model	Statistic	Value
Linear regression	l_2 error	0.51663
	L_2 error over R	5.51128
	L_2 error over $\cup R_i$	0.26299
New methodology	l_2 error	0.534192
	L_2 error over R	1.10391
	L_2 error over $\cup R_i$	0.2258

Table 5.8: l_2 and L_2 error statistics for each model in Figure 5.10

5.7.4 Example 4

This final example demonstrates that whilst so far we have only presented examples working in $\mathbb{R}$, there is no reason why one cannot consider $\mathbb{R}^d$ for $d > 1$. In this example, data is sampled from a function very similar to that used in the example in §5.1 but defined instead over $\mathbb{R}^2$.

$$f(x_1, x_2) = \frac{-1}{1 + x_1^2 + x_2^2} \tag{5.26}$$

in the regions

$$R_1 = [-3, -1]^2, \qquad R_2 = [0, 1]^2, \tag{5.27}$$

with 35 points sampled in R_1 and 3 in R_2. The data and the results of fitting polynomial approximations using standard least squares regression and the new robust eigenvalue polynomial regression are shown in Figure 5.11 with summary statistics shown in Table 5.9. Whilst in 3D, it is significantly harder to determine the differences from the plot, the statistics clearly indicate an approx 12% increase in l_2 error with a more than 53% decrease in the L_2 error over R which in this case is equal to $[-3, 1]^2$.

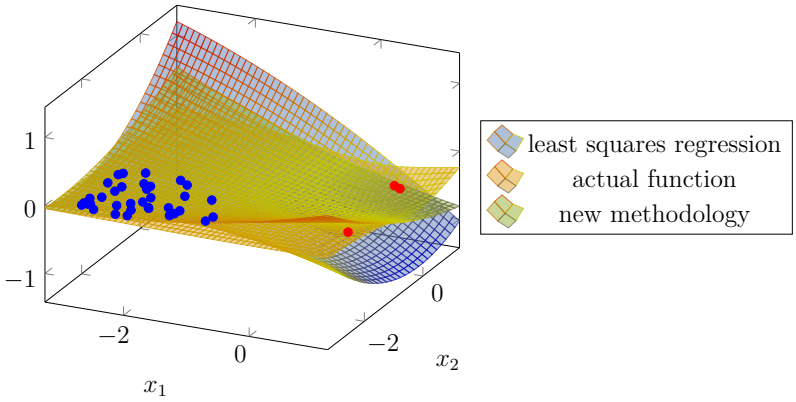


Figure 5.11: Data and degree 2 polynomials fitted to the data.

5.8 Conclusion

Throughout the course of this chapter, the robust eigenvalue polynomial regression methodology has been introduced and its performance shown on an

Model	Statistic	Value
Linear regression	l_2 error	0.48321
	L_2 error over R	1.23667
	L_2 error over $\cup R_i$	0.903843
New methodology	l_2 error	0.539906
	L_2 error over R	0.660852
	L_2 error over $\cup R_i$	0.310041

Table 5.9: l_2 and L_2 error statistics for each model in Figure 5.11

initial example in §5.1 and four further examples in §5.7. The examples have highlighted the performance of the methodology when compared to standard least squares regression in different situations circumstances. Throughout these examples, the robust eigenvalue polynomial regression methodology developed has shown to result in a significant improvement in performance over standard least squares regression.

5.8.1 Issues

It should be noted, however, that this methodology is not better than standard least squares regression in all situations and indeed standard least squares regression will *always* result in a model with a better l_2 error as a result of the least squares optimisation. Other issues which arise when using this methodology are the following:

Speed: Standard least squares regression is very fast and computationally easy to do – it is simply matrix algebra which is optimised to work on computer processors. The methodology proposed in this report requires significantly more computing time to yield a result. Whilst the times are not long (< 1 sec), this is magnitudes slower than standard least squares regression.

Computational limitations: Further to the speed issue highlighted above, the most significant issue with this methodology is the requirement to

compute eigenvectors and eigenvalues of potentially large and sparse matrices. With standard computing techniques, the numbers involved tend to be both very large and very small, this results in matrices which are difficult to compute the eigenvectors and eigenvalues for without running into numerical errors.

Single solution: Least squares regression will always result in a single solution (assuming no issues with multicollinearity etc), and whilst the proposed methodology does result in a single solution, it is the result of computing a number of solutions and picking the best amongst those. Whilst the number of solutions returned is small and hence it is possible to generate all of them and choose the one that is "best", this is not as simple as standard least squares regression.

Chapter 6

Summary & Future Work

In this thesis, we have presented two processes; one for processing time series data for machine learning and the other for obtaining robust polynomial approximations to data. In addition, we have provided an introduction into the theory of iterated integral path signatures and log signatures upon which time series data preparation process is built and the robust polynomial regression had its origin.

In this chapter, possible further areas of study and research for each of the two methodologies are discussed. Given the nature of the two processes, there are vastly more directions for further study for the robust eigenvalue polynomial regression methodology than the time series preprocessing methodology.

6.1 Further Work for Chapter 4

The possible further work for Chapter 4 is relatively limited however there are still a few areas where improvements can be made.

More advanced neural networks: As was discussed in §4.5.2, it was shown that when using an LSTM, the recognition rate was increasing as the level of the truncated signature used also increased. It was, however, also highlighted that when the level 10 truncated iterated integral path signature or log signature was used, the recognition rate was lower than

may have been expected. Owing to the relative simplicity of the LSTM used, it is possible that by increasing the complexity of the LSTM, a higher recognition rate can be achieved.

Higher level signature: If the work suggested above to use a more complex LSTM is completed, then it would be a logical step to test the performance of the recognition methodology with higher still levels of the truncated iterated integral path signature and log signature. This is because more detail is contained about the path as the level of the signature increases.

Higher resolution data: One key point to note is that whilst the data used had sufficiently high resolution to work well with this recognition methodology, if the data were higher resolution then it would be possible to use a larger number of dyadic intervals over the character. Currently we have limited the number of dyadic intervals to $2^5 = 32$ owing to the fact that some characters contained only a few points more than that. If one were to use finer resolution data then there would be no issue in increasing the number of dyadic intervals.

6.2 Further Work for Chapter 5

It is important to highlight that whilst the methodology presented within Chapter 5 has demonstrated its performance in the five examples provided, there are ways in which this methodology can be enhanced through further work or by combining with other tools.

Higher dimension data: It was discussed at the beginning of Chapter 5 that the original aim was to use log signatures as the underlying data for polynomial regression. Owing to the computational limitations, it was unfortunately not possible to compute the eigenvalues and eigenvectors necessary to use the robust eigenvalue polynomial regression in dimensions large enough for the methodology to be usable. An area of further work would

be to determine an optimal way to compute the eigenvalues and eigenvectors such that the basis functions can be derived for higher dimension data and higher degree polynomials.

Initial basis: Note that in Algorithm 5.1, the input specified "any basis of the space of polynomials $\mathbb{R}_m[x_1, \ldots, x_d]$". Throughout each of the examples provided, the basis that was chosen was the Chebyshev polynomials of the first kind, scaled and translated into the correct regions where necessary. Another basis that was used during development of the algorithm was the standard monomial basis. Whilst both bases provided identical results in terms of final result, it was seen that when using the monomial basis, computation errors were more likely to occur within the computation of the eigenvalues and eigenfunctions (see §5.8.1).

Choosing the regions: In each of the examples presented, the regions were first chosen and the points subsequently generated within these regions. This scenario matches what may occur in real world statistics whereby a computationally expensive function can be sampled within a region and the results returned. Another common scenario is to be presented with data and then to have to find an approximation using this data. The way in which this methodology can be utilised in this situation is to apply a clustering algorithm such as k-means clustering and then use the regions obtained to carry out the methodology.

Number of points: The robust eigenvalue polynomial regression methodology is shown to be a powerful tool for use when one of the regions of data contains a very small number of points, resulting in a highly influential group of points. In the examples provided, this small number was always 3 except one example where the number of points was only 2. An open question is the number of points at which the benefits of using the robust eigenvalue polynomial regression method are no longer enough to warrant use as opposed to standard least squares regression.

More regions: In the scenario where data is being sampled from an unknown underlying function that is computationally expensive, one may consider the option of sampling a small number of additional observations either in one of the regions where the observations already exist or in a completely new region in space. Using knowledge of the eigenvalues, it may be possible to strategically pick the new region to obtain data in such a way that the model may be improved. As an example, if adding data in a specific region would necessitate adding basis functions associated with large eigenvalues than it is likely that this would impact the quality of the fit in other regions. On the other hand, if one were able to find a region where an additional basis function did not negatively impact the fit in other regions to a large degree and improved the fit in the new region; this would be valuable.

Bibliography

[AHM16] Baligh M. Al-Helali and Sabri A. Mahmoud. "A Statistical Framework for Online Arabic Character Recognition". In: *Cybernetics and Systems* 47.6 (Aug. 17, 2016), pp. 478–498. ISSN: 0196-9722. DOI: 10.1080/01969722.2016.1206768.

[AHM17] Baligh M. Al-Helali and Sabri A. Mahmoud. "Arabic Online Handwriting Recognition (AOHR): A Survey". In: *ACM Computing Surveys* 50.3 (June 29, 2017), 33:1–33:35. ISSN: 0360-0300. DOI: 10.1145/3060620.

[ASH09] Khaled Assaleh, Tamer Shanableh, and Husam Hajjaj. "Recognition of handwritten Arabic alphabet via hand motion tracking". In: *Journal of the Franklin Institute* 346.2 (Mar. 1, 2009), pp. 175–189. ISSN: 0016-0032. DOI: 10.1016/j.jfranklin.2008.08.005.

[BBR09] James E. Burt, Gerald M. Barber, and David L. Rigby. *Elementary Statistics for Geographers*. Guilford Press, Mar. 19, 2009. ISBN: 978-1-57230-484-0.

[Bre+84] Leo Breiman, Jerome Friedman, Charles J. Stone, and R. A. Olshen. *Classification and Regression Trees*. Taylor & Francis, Jan. 1, 1984. ISBN: 978-0-412-04841-8.

[Bri98] Brian Everitt. *The Cambridge Dictionary of Statistics*. Cambridge University Press, 1998. ISBN: 978-0-521-59346-5.

[CK16] Ilya Chevyrev and Andrey Kormilitzin. "A Primer on the Signature Method in Machine Learning". In: *arXiv:1603.03788 [cs, stat]* (Mar. 11, 2016). arXiv: 1603.03788. URL: http://arxiv.org/abs/1603.03788.

[Car+20] Victor Carbune, Pedro Gonnet, Thomas Deselaers, Henry A. Rowley, Alexander Daryin, Marcos Calvo, Li-Lun Wang, Daniel Keysers, Sandro Feuz, and Philippe Gervais. "Fast multi-language LSTM-based online handwriting recognition". In: *International Journal on Document Analysis and Recognition (IJDAR)* 23.2 (June 1, 2020), pp. 89–102. ISSN: 1433-2825. DOI: 10.1007/s10032-020-00350-4.

[Che58] Kuo-Tsai Chen. "Integration of Paths–A Faithful Representation of Paths by Noncommutative Formal Power Series". In: *Transactions of the American Mathematical Society* 89.2 (1958). Publisher: American Mathematical Society, pp. 395–407. ISSN: 0002-9947. DOI: 10.2307/1993193. URL: https://www.jstor.org/stable/1993193.

[FH14] Peter K. Friz and Martin Hairer. *A Course on Rough Paths: With an Introduction to Regularity Structures*. Universitext. Springer International Publishing, 2014. ISBN: 978-3-319-08332-2. DOI: 10.1007/978-3-319-08332-2. URL: https://www.springer.com/gp/book/9783319083322.

[FV10] Peter K. Friz and Nicolas B. Victoir. *Multidimensional Stochastic Processes as Rough Paths: Theory and Applications*. Cambridge Studies in Advanced Mathematics. Cambridge: Cambridge University Press, 2010. ISBN: 978-0-521-87607-0.

[Fer21] Adeline Fermanian. "Embedding and learning with signatures". In: *Computational Statistics & Data Analysis* 157 (May 1, 2021), pp. 107–148. ISSN: 0167-9473. DOI: 10.1016/j.csda.2020.107148.

[GEW06] Pierre Geurts, Damien Ernst, and Louis Wehenkel. "Extremely randomized trees". In: *Machine Learning* 63.1 (Apr. 1, 2006), pp. 3–42. ISSN: 1573-0565. DOI: 10.1007/s10994-006-6226-1.

[Gra13] Benjamin Graham. "Sparse arrays of signatures for online character recognition". In: *arXiv:1308.0371 [cs]* (Dec. 1, 2013). arXiv: 1308.0371. URL: http://arxiv.org/abs/1308.0371.

[Gua17] Zhong Guan. "Bernstein polynomial model for grouped continuous data". In: *Journal of Nonparametric Statistics* 29.4 (Oct. 2, 2017), pp. 831–848. ISSN: 1048-5252. DOI: 10.1080/10485252.2017.1374384.

[HL10] Ben Hambly and Terry Lyons. "Uniqueness for the signature of a path of bounded variation and the reduced path group". In: *Annals of Mathematics* 171.1 (Mar. 17, 2010), pp. 109–167. ISSN: 0003-486X. DOI: 10.4007/annals.2010.171.109. arXiv: math/0507536. URL: http://arxiv.org/abs/math/0507536.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (Nov. 1, 1997). Publisher: MIT Press, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.

[Hub64] Peter J. Huber. "Robust Estimation of a Location Parameter". In: *The Annals of Mathematical Statistics* 35.1 (Mar. 1964). Publisher: Institute of Mathematical Statistics, pp. 73–101. ISSN: 0003-4851, 2168-8990. DOI: 10.1214/aoms/1177703732.

[Ken92] Peter Kennedy. *A guide to econometrics*. Cambridge, Mass. : MIT Press, 1992. ISBN: 978-0-262-11160-7.

[LCL07] Terry J. Lyons, Michael J. Caruana, and Thierry Lévy. *Differential Equations Driven by Rough Paths: Ecole d'Eté de Probabilités de Saint-Flour XXXIV-2004*. École d'Été de Probabilités de Saint-Flour. Berlin Heidelberg: Springer-Verlag, 2007. ISBN: 978-3-

540-71284-8. DOI: 10.1007/978-3-540-71285-5. URL: https://www.springer.com/gp/book/9783540712848.

[LLN13] Daniel Levin, Terry Lyons, and Hao Ni. "Learning from the past, predicting the statistics for the future, learning an evolving system". In: (Sept. 1, 2013). arXiv: 1309.0260. URL: http://arxiv.org/abs/1309.0260.

[LM] Terry Lyons and David Maxwell. *The esig Python Package — esig 0.6 documentation*. URL: https://esig.readthedocs.io/en/latest/.

[LQ02] Terry Lyons and Zhongmin Qian. *System Control and Rough Paths*. Oxford Mathematical Monographs. Oxford, New York: Oxford University Press, Dec. 19, 2002. ISBN: 978-0-19-850648-5.

[Lej03] Antoine Lejay. "An Introduction to Rough Paths". In: *Séminaire de Probabilités XXXVII*. Ed. by Jacques Azéma, Michel Émery, Michel Ledoux, and Marc Yor. Lecture Notes in Mathematics. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 1–59. ISBN: 978-3-540-40004-2. DOI: 10.1007/978-3-540-40004-2_1.

[Lyo] Terry J. Lyons. *Computational Rough Paths - CoRoPa*. Computational Rough Paths - CoRoPa. URL: https://coropa.sourceforge.io/.

[Lyo14] Terry Lyons. "Rough paths, Signatures and the modelling of functions on streams". In: *arXiv:1405.4537 [math, q-fin, stat]* (May 18, 2014). arXiv: 1405.4537. URL: http://arxiv.org/abs/1405.4537.

[Lyo98] Terry J. Lyons. "Differential equations driven by rough signals". In: *Revista Matemática Iberoamericana* 14.2 (Aug. 31, 1998), pp. 215–310. ISSN: 0213-2230. DOI: 10.4171/RMI/240. URL: https://www.ems-ph.org/journals/show_abstract.php?issn=0213-2230&vol=14&iss=2&rank=1.

[MA12] Volker Märgner and Haikal El Abed, eds. *Guide to OCR for Arabic Scripts*. London: Springer-Verlag, 2012. ISBN: 978-1-4471-4071-9. DOI: 10.1007/978-1-4471-4072-6. URL: https://www.springer.com/gp/book/9781447140719.

[MLG19] P. J. Moore, T. J. Lyons, and J. Gallacher. "Using path signatures to predict a diagnosis of Alzheimer's disease". In: *PLoS ONE* 14.9 (Sept. 19, 2019). ISSN: 1932-6203. DOI: 10.1371/journal.pone.0222212. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6752804/.

[Ni17] Hao Ni. "The Signature-Based Learning and its Application". In: LMS-EPSRC Durham Symposium Stochastic Analysis. Durham University, July 19, 2017.

[Ola] Christopher Olah. *Understanding LSTM Networks – colah's blog*. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[PA+18] Imanol Perez Arribas, Guy M. Goodwin, John R. Geddes, Terry Lyons, and Kate E. A. Saunders. "A signature-based machine learning model for distinguishing bipolar disorder and borderline personality disorder". In: *Translational Psychiatry* 8.1 (Dec. 13, 2018). Number: 1 Publisher: Nature Publishing Group, pp. 1–7. ISSN: 2158-3188. DOI: 10.1038/s41398-018-0334-0. URL: https://www.nature.com/articles/s41398-018-0334-0.

[PS00] R. Plamondon and S. N. Srihari. "Online and off-line handwriting recognition: a comprehensive survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.1 (Jan. 2000). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 63–84. ISSN: 1939-3539. DOI: 10.1109/34.824821.

[Pri+16] A. Priya, S. Mishra, S. Raj, S. Mandal, and S. Datta. "Online and offline character recognition: A survey". In: *2016 International Con-*

ference on Communication and Signal Processing (ICCSP). 2016 International Conference on Communication and Signal Processing (ICCSP). Apr. 2016, pp. 0967–0970. DOI: 10.1109/ICCSP.2016.7754291.

[RG18] Jeremy Reizenstein and Benjamin Graham. "The iisignature library: efficient calculation of iterated-integral signatures and log signatures". In: *arXiv:1802.08252 [cs, math]* (Feb. 22, 2018). arXiv: 1802.08252. URL: http://arxiv.org/abs/1802.08252.

[Rei] Jeremy Reizenstein. *iisignature: Iterated integral signature calculations*. URL: https://github.com/bottler/iisignature.

[Reu93] Christophe Reutenauer. *Free Lie Algebras*. London Mathematical Society Monographs. Oxford, New York: Oxford University Press, May 6, 1993. ISBN: 978-0-19-853679-6.

[Sti73] Stephen M. Stigler. "Simon Newcomb, Percy Daniell, and the History of Robust Estimation 1885-1920". In: *Journal of the American Statistical Association* 68.344 (1973). Publisher: [American Statistical Association, Taylor & Francis, Ltd.], pp. 872–879. ISSN: 0162-1459. DOI: 10.2307/2284515. URL: https://www.jstor.org/stable/2284515.

[TKA13] Najiba Tagougui, Monji Kherallah, and Adel M. Alimi. "Online Arabic handwriting recognition: a survey". In: *International Journal on Document Analysis and Recognition (IJDAR)* 16.3 (Sept. 1, 2013), pp. 209–226. ISSN: 1433-2825. DOI: 10.1007/s10032-012-0186-8.

[Tag+14] Najiba Tagougui, Houcine Boubaker, Monji Kherallah, and Adel M. ALIMI. "A Hybrid NN/HMM Modeling Technique for Online Arabic Handwriting Recognition". In: *arXiv:1401.0486 [cs]* (Jan. 2, 2014). arXiv: 1401.0486. URL: http://arxiv.org/abs/1401.0486.

[Tin95] Tin Kam Ho. "Random decision forests". In: *Proceedings of 3rd International Conference on Document Analysis and Recognition*. Proceedings of 3rd International Conference on Document Analysis and Recognition. Vol. 1. Aug. 1995, 278–282 vol.1. DOI: 10.1109/ICDAR.1995.598994.

[WN+18] D. Wilson-Nunn, T. Lyons, A. Papavasiliou, and H. Ni. "A Path Signature Approach to Online Arabic Handwriting Recognition". In: *2018 IEEE 2nd International Workshop on Arabic and Derived Script Analysis and Recognition (ASAR)*. 2018 IEEE 2nd International Workshop on Arabic and Derived Script Analysis and Recognition (ASAR). Mar. 2018, pp. 135–139. DOI: 10.1109/ASAR.2018.8480300.

[WN21] Daniel Wilson-Nunn. *mathematica_sigs_tensor_algebra*. original-date: 2021-04-20T20:28:38Z. Apr. 21, 2021. URL: https://github.com/wilson-nunn/mathematica_sigs_tensor_algebra.

[Xie+18] Z. Xie, Z. Sun, L. Jin, H. Ni, and T. Lyons. "Learning Spatial-Semantic Context with Fully Convolutional Recurrent Network for Online Handwritten Chinese Text Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.8 (Aug. 2018). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1903–1917. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2017.2732978.

[Yan+19] W. Yang, T. Lyons, H. Ni, C. Schmid, and L. Jin. "Developing the Path Signature Methodology and its Application to Landmark-based Human Action Recognition". In: *arXiv [cs]* (Dec. 12, 2019). arXiv: 1707.03993. URL: http://arxiv.org/abs/1707.03993.

[Yiu19] Tony Yiu. *Understanding Random Forest*. Medium. Aug. 14, 2019. URL: https://towardsdatascience.com/understanding-random-forest-58381e0602d2.

Appendix **A**

Online KHATT Data

The plots in this chapter provide an example from each class of the data that is used in Chapter 4. Each plot demonmstrates one randomly selected observation of the class. The title of each plot is of the following format:

name - **character** (**shape**)

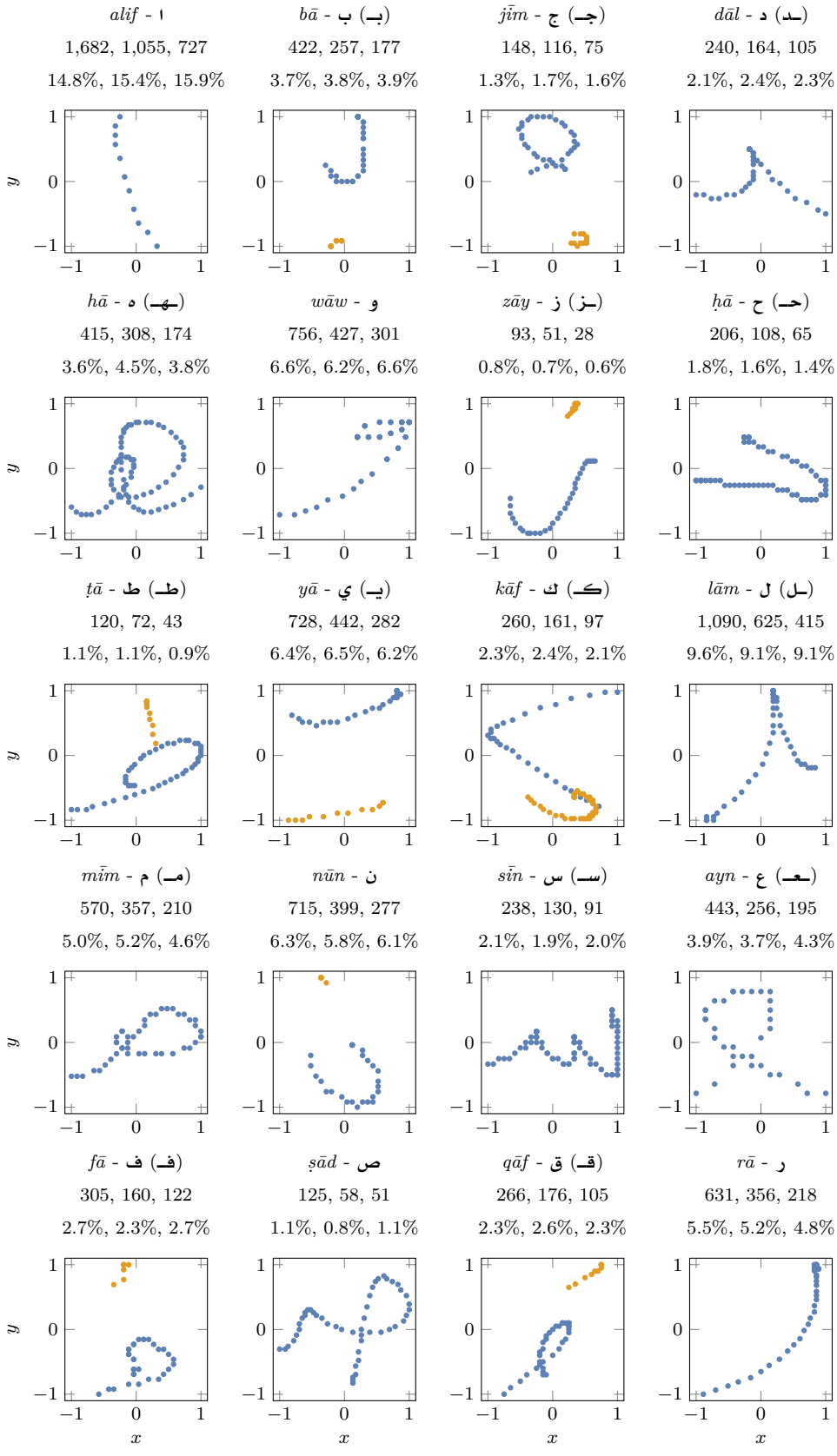
n_{train}, n_{test}, n_{validate}

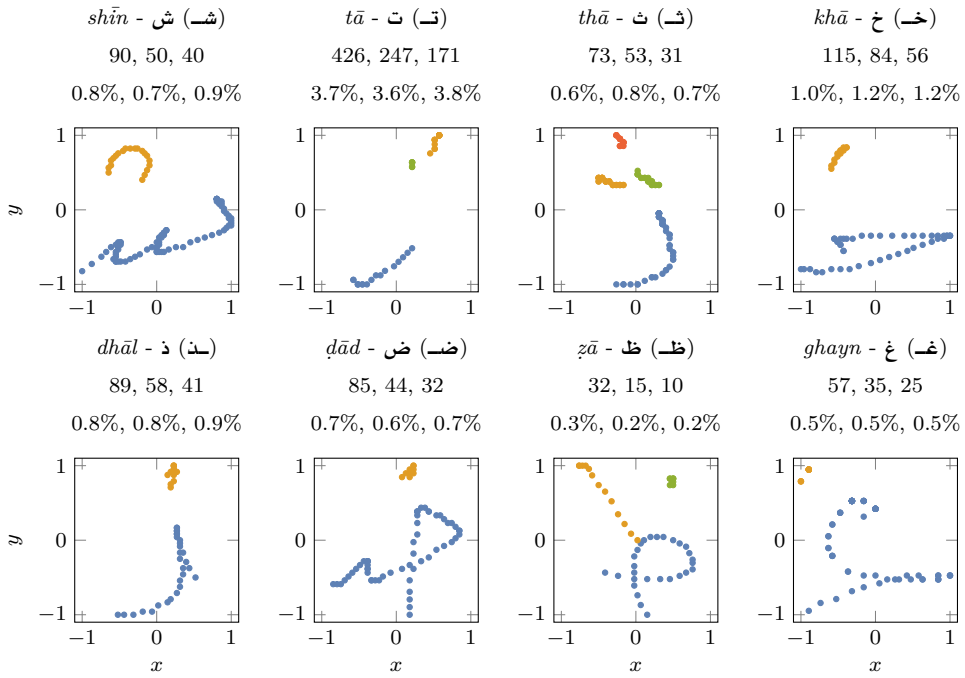
$x_{\text{train}}\%$, $x_{\text{test}}\%$, $x_{\text{validate}}\%$

Where:

- "*name*" is an English transliteration of the Arabic name for the character. in the class
- "**character**" is the Arabic character in its isolated form.
- "**shape**" is the typed representation of the shape which the character has taken in the plot. This is only included if the character is *not* shown in its isolated form.
- "n_{set}" is the number of observations of this class in the set.
- "$x_{\text{set}}\%$" is the percentage (to 1 decimal place) of the set that is formed of observations of this class.

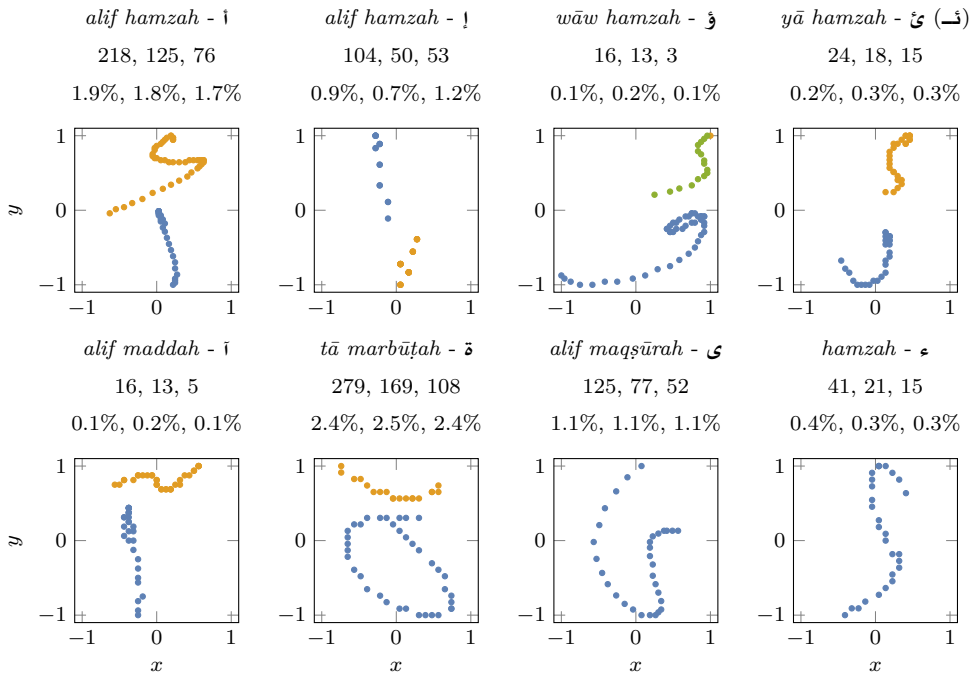
A.1 Arabic Alphabet





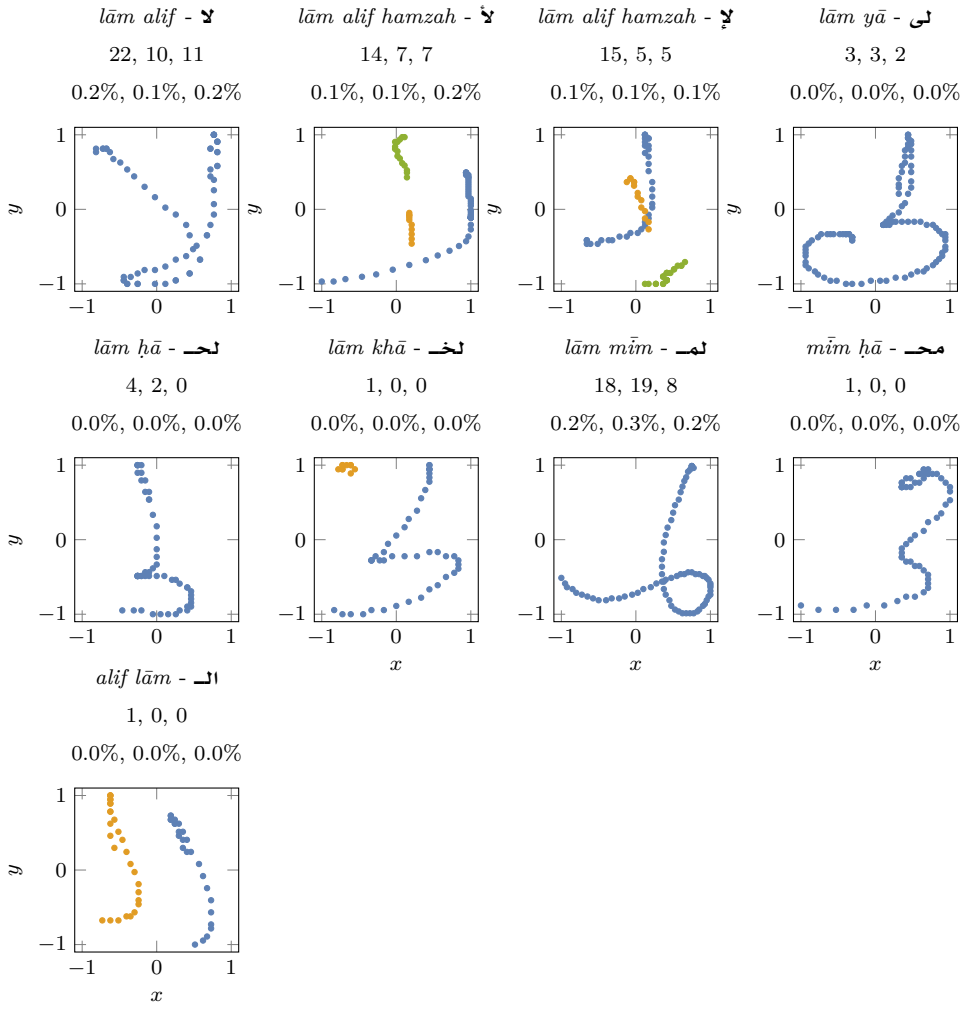
A.2 Special characters

In Arabic, there are a number of additional letters which are variations of those in the standard alphabet, these are presented in this section.



A.3 Ligatures

Note that a majority of ligatures in Arabic occur only in handwriting and therefore when typed do not accurately represent what would occur when written by hand.

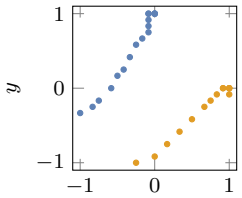


A.4 Punctuation and other symbols

spacial symbol

3, 2, 5

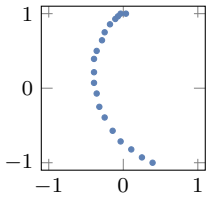
0.0%, 0.0%, 0.1%



bracket - (

5, 1, 2

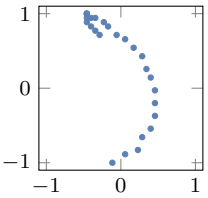
0.0%, 0.0%, 0.1%



bracket -)

4, 3, 1

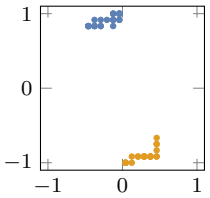
0.0%, 0.0%, 0.0%



colon - :

7, 1, 1

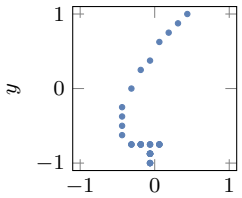
0.1%, 0.0%, 0.0%



comma - ،

36, 22, 14

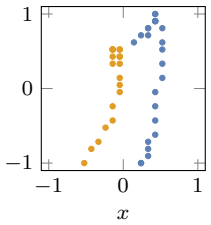
0.3%, 0.3%, 0.3%



quotation mark - ”

10, 1, 5

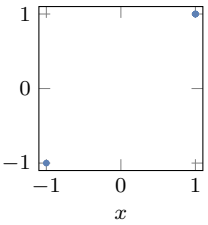
0.1%, 0.0%, 0.1%



full stop - .

8, 5, 6

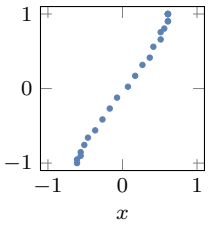
0.1%, 0.1%, 0.1%



forward slash - /

1, 1, 1

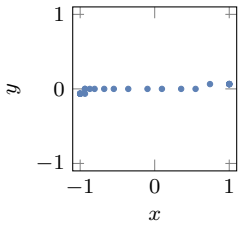
0.0%, 0.0%, 0.0%



hyphen - -

1, 1, 0

0.0%, 0.0%, 0.0%



Appendix B

Example Data

In this appendix, label `a` denotes R_1, `b` denotes R_2 and `c` denotes R_3.

B.1 Introductory Example

Below is the data used in the example in §5.1.

x	y	label	x	y	label
-4.51004	0.0468594	a	0.0103023	0.999894	b
-4.48342	0.047391	a	0.490886	0.805822	b
-4.79215	0.0417281	a	-0.406876	0.857965	b
-4.57794	0.0455425	a	-0.443821	0.835438	b
-4.49993	0.0470602	a	0.0634227	0.995994	b
-4.42451	0.0485997	a	0.499321	0.800435	b
-4.0184	0.0583175	a	0.0967509	0.990726	b
-4.88039	0.0402929	a	0.219663	0.953969	b
-4.57577	0.0455837	a	-0.28129	0.926678	b
-4.8954	0.0400562	a	-0.0653653	0.995746	b
-4.4432	0.0482113	a	0.0752992	0.994362	b
-4.60079	0.0451116	a	-0.0320785	0.998972	b
-4.14449	0.0550153	a	0.286541	0.924124	b
-4.66597	0.043915	a	0.296307	0.919288	b
-4.86161	0.0405923	a	-0.390914	0.867443	b
-4.53697	0.0463304	a	0.487641	0.807889	b
-4.12746	0.0554449	a	0.0821692	0.993293	b
-4.4667	0.0477296	a	-0.265171	0.934304	b
-4.73865	0.0426352	a	0.0479974	0.997702	b
-4.84687	0.0408294	a	0.183241	0.967513	b
			4.43123	0.0484596	c
			4.48459	0.0473674	c
			4.42554	0.0485781	c

B.2 Example 1

Below is the data used in the example in §5.7.1.

x	y	label	x	y	label
-19.5794	-1.53121	a	-3.14275	-1.85194	b
-19.036	-1.51895	a	-2.54853	-1.78965	b
-19.2322	-1.52182	a	2.09531	1.64677	b
-1.85204	-1.53339	b	-4.44296	-1.66653	b
-1.66025	-1.42608	b	-0.293555	-0.292153	b
2.67347	1.81387	b	0.7085	0.689037	b
3.28817	1.84863	b	-4.9618	-1.55733	b
-2.08577	-1.64281	b	-0.572734	-0.562399	b
-4.62815	-1.62639	b	-2.39542	-1.75119	b
-4.17362	-1.72382	b	-4.84946	-1.57979	b
3.59051	1.82311	b	-1.42253	-1.27199	b
2.47811	1.77318	b			

B.3 Example 2

Below is the data used in the example in §5.7.2.

x	y	label	x	y	label
-2.25334	1.99856	a	0.81459	0.24932	c
-2.26924	1.99867	a	0.566669	0.422906	c
-0.459342	1.48405	b	0.872033	0.217486	c
-0.120105	1.13488	b	0.962596	0.173414	c
-0.283791	1.31183	b	0.862627	0.222488	c
-0.144307	1.16171	b	0.997513	0.158334	c
-0.130239	1.14613	b	0.534353	0.449835	c
-0.354951	1.38432	b	0.864101	0.221699	c
-0.389228	1.41799	b	0.687979	0.330579	c
-0.0370687	1.04181	b	0.88655	0.209925	c
-0.399107	1.42753	b	0.82863	0.241253	c
-0.0911024	1.10251	b	0.690375	0.328897	c
-0.272839	1.30039	b	0.887096	0.209645	c
-0.316528	1.34559	b	0.508658	0.471925	c
-0.0831479	1.09361	b	0.736472	0.297631	c
-0.317302	1.34638	b	0.63577	0.368591	c
-0.173468	1.19379	b	0.998846	0.157779	c
0.728326	0.303006	c	0.601218	0.395185	c
0.627961	0.374502	c	0.889481	0.208422	c
0.604277	0.392786	c	0.712035	0.313949	c
0.877644	0.214541	c	0.915622	0.19536	c

B.4 Example 3

Below is the data used in the example in §5.7.3.

x	y	label	x	y	label
1.00904	0.847063	a	2.71278	0.987273	a
3.19856	1.09173	a	2.67932	1.04598	a
4.48808	0.795198	a	1.38145	1.00429	a
2.41996	0.884731	a	1.22429	0.859805	a
2.85276	1.08747	a	0.0583265	0.215705	a
1.05255	0.687788	a	2.86061	0.884412	a
4.65956	0.841669	a	4.42914	0.844983	a
2.36021	0.91336	a	0.626463	0.815657	a
3.25704	0.907092	a	1.37816	0.819531	a
2.73276	1.08366	a	1.85951	1.10738	a
2.70135	0.9866	a	2.55588	0.968816	a
4.42041	0.911129	a	2.85025	1.00925	a
2.00586	0.827967	a	0.307521	0.571447	a
4.25295	0.66669	a	3.15784	1.01819	a
2.24483	1.01792	a	18.7636	-0.901753	b
0.0429018	0.105066	a	19.5602	-1.04276	b

B.5 Example 4

Below is the data used in the example in §5.7.4. Here the `l` column is the same as the `label` column elswhere in this appendix.

x	y	z	l	x	y	z	l
-2.32127	-1.64043	-0.110141	a	-2.91149	-1.475	-0.0858192	a
-1.02182	-2.54771	-0.117166	a	-1.01311	-2.30823	-0.135975	a
-2.39537	-2.63449	-0.0731082	a	-2.77582	-2.78774	-0.0606918	a
-2.8509	-2.77388	-0.0594459	a	-2.66855	-2.11083	-0.0795117	a
-1.96741	-2.47314	-0.0910158	a	-2.08382	-2.9417	-0.0714495	a
-1.94977	-2.49606	-0.090646	a	-1.37389	-1.57975	-0.185764	a
-2.25771	-2.45335	-0.0825342	a	-2.60647	-1.19388	-0.108471	a
-2.114	-2.07318	-0.102385	a	-1.52789	-2.34188	-0.113394	a
-2.444	-1.59863	-0.104945	a	-2.90008	-1.85598	-0.0777898	a
-2.84311	-2.86889	-0.0577574	a	-2.75218	-2.76586	-0.0616353	a
-1.71943	-2.3154	-0.107324	a	-1.91978	-1.24985	-0.160059	a
-2.65795	-1.89652	-0.0857521	a	-2.72699	-2.43081	-0.069709	a
-2.87362	-1.39154	-0.0893329	a	-2.13557	-2.30209	-0.0920788	a
-2.07456	-1.14052	-0.15141	a	-1.74137	-1.70961	-0.143779	a
-1.56229	-2.61998	-0.0970399	a	-2.25927	-2.97338	-0.0669106	a
-2.618	-2.9481	-0.0604404	a	-2.85726	-2.56083	-0.063606	a
-2.31632	-1.77943	-0.104913	a	0.0210433	0.0806178	-0.993106	b
-2.13557	-2.7018	-0.0777581	a	0.507136	0.810484	-0.522447	b
-1.53766	-2.50082	-0.103966	a	0.385014	0.880368	-0.519944	b