The Signature Transform in Numerics and Machine Learning

Dissertation
zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von
Biagio Paparella
aus
Terlizzi (Italien)

Bonn 2024

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn

Betreuer: Prof. Dr. Michael Griebel
Gutachter: Prof. Dr. Jochen Garcke

Tag der Promotion: 19.04.2024
Erscheinungsjahr: 2024

Abstract

In this work we study the *signature transform* from the viewpoint of applied and numerical mathematics.

The theoretical background is established in the first part, where the signature is defined as a map going from continuous paths of bounded variations to ordered tensor algebras. Approximation theorems and computational considerations are clarified, together with explicit and well commented examples. Only selected essential properties are pointed out, useful for non-linear approximation of functionals, dimension reduction and extension to the probabilistic setting.

In the second part we use all the previously introduced theory to design numerical experiments of interest in data science and machine learning, targeting problems like time series classification, clustering, correlation detection and generation of artificial samples. A small section on agents classification for reinforcement learning is also included.

Finally, the reader is given a list of possible connections to other areas of mathematics like PDE, kernel theory, jump processes and even algebraic geometry. We did our best to keep the exposition clear and compact.

Contents

Introduction

Nowadays we are surrounded by *data* in different forms, like pictures, videos and texts. Part of the current research aims at using them to build *predictions*, at least to some extent. This goal has essentially been true for the whole history of science, but nowadays we experience the capability of collecting and processing *huge* amount of data as never done before. Digitalization, either on small (smartphones) and bigger scale (infrastructures, digital services) combined with the increase of processing speed and available storage, prepares the ground for modern *data science* and *machine learning* technologies.

In these fields where science and engineering are mixed together, there is a balance between classical statistics (e.g. Bayesian methods), optimization (e.g. gradient strategies), computer science (e.g. parallel computing) to build software capable of hopefully finding patterns in big collections of data, sometimes for the pure benefit of society (e.g. in medicine), sometimes for profit (e.g. marketing and advertisement), and of course for situations in between (e.g. pharmacology or robotics).

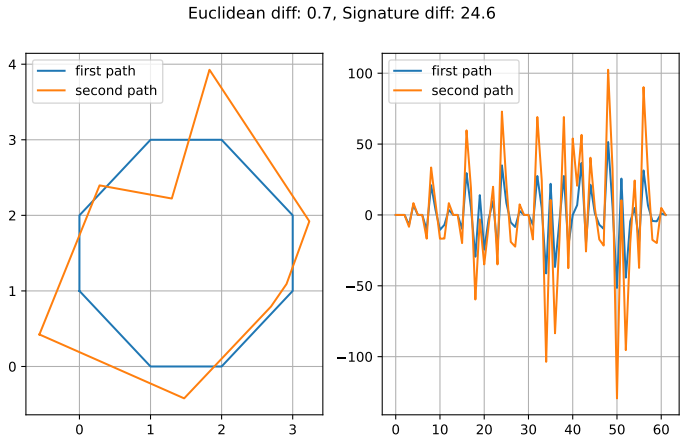
The nature of the data under analysis, the associated models, the chosen optimization methods, usually determine the level of mathematics required to understand (and justify) the strategies into play. In some case they are relatively accessible, while sometime quite advanced and mainly for the experts.

The purpose of this work is to focus on a technique called the *signature transform* potentially very useful to analyze *numerical time series* data, i.e. collections of measurements organized in time. Examples can be found everywhere: daily temperature records, stock price oscillations, or even the number of infected during a pandemic as we sadly experienced recently.

In their generality, data algorithms need a criteria to estimate whether two data points are "similar" or not. This can be very easy for some contexts (like numbers), but very hard on others (what is the "similarity" between two videos?). Since we interpret time series as (discrete) *curves* having x-axis given by timestamps, and y-axis given by the corresponding values, it is possible to simply compute their ordinary Euclidean norm to check their differences. Unfortunately, this method has serious limitations, for instance when two time series have a different number of points or when the sampling size is so big to represent a computational obstacle.

Both these issues are reasonably "fixed" when using the *signature transform*, which then offers an additional method to measure distances between paths. It

Figure 1: Two paths having small Euclidean distance and bigger signature difference (in absolute norm).



is *not* a replacement for the Euclidean metric, but rather a further supporting tool that should be used with awareness and caution.

For instance, we suggest having a look at the simple plot above. The two portrayed curves are maybe "similar" to our eyes: they indeed have a very "small" Euclidean distance. On the other hand, their signature transforms (right plot) are "enough" different. One should technically argue about the meaning of the absolute error in such a context, and we will. But in this introductory section the goal is simply to provide a simple intuitive example, and to point out how there is no universal "better" or "worse" way to measure these distances, rather it depends on the goals and the algorithms into play.

In the first part of this thesis we try to offer a solid and clear theoretical foundation. The theory of the signature transform can quickly became quite advanced from a mathematical perspective, which means to be less accessible and less prone to be actually used in applications. We did our best to select only the strongest and essential properties, always trying to be self-contained and then referencing to appropriate papers for important theorems and proofs.

In the chapter number 1, our intention is to refresh the notion of *tensor*. Indeed, there is usually a potential confusion around this term because of its appearance across multiple fields from pure mathematics to really applied computer science. We construct a solid algebraic setting and point out the *combinatorial* nature of tensors.

In the next section we revise the notion of path integral, starting from the smooth case and generalizing up to piecewise linear maps. It is here that we revise the definition of *bounded variation* functions and Riemann-Stieltjes integration, both of fundamental importance in order to later define the signature transform.

Indeed, given a curve $X : [0, 1] \to \mathbb{R}^d$ of appropriate regularity, the signature transform is related to the idea of performing iterated integrals between its components. The regularity of the curve determines the kind of "integration" under usage. And since we introduce bounded variation maps, that type will be the mentioned Riemann-Stieltjes.

Furthermore, the choice of the components can be described by corresponding multi-indices, which can be algebraically sorted so to facilitate their interpretation and even mirror analytical properties. In this operations of combinatorial arrangements tensor algebras will emerge as a powerful tool.

Moving further, in the second chapter the signature transform is defined, possible since there are no more mathematical prerequisites. The notion is introduced step by step, carefully checking all the details and including some remarks about slightly alternative definitions sometimes present in the literature.

The chapter proceeds with the problem of actually *computing* the signature of a curve. We start with simple segments and then extend to piecewise-linear interpolations, exploiting a "concatenation" property capable of strongly simplify the computations. Many explicit examples are provided in order to preserve a clear exposition and guide the reader step-by-step. When moving to the computational aspects, the *Signatory* Python library is used and of course referenced with its main paper, where the computational costs are better discussed and completed with remarks about parallelization (using OpenMP) and comparison with other algorithms (*esig* and *iisignature*).

We finally conclude the theoretical section with a chapter about the main properties of the signature transform. If it is true that we selected only few of them, on the other hand we used *all and only them* in our experiments without involving anything else. The single exception is maybe when we deal with geometric Brownian motions, where we tacitly still work with piecewise linear approximations and avoid to properly extend the signature to stochastic integration. This operation is legit (and regularly pointed out in the given references), but ultimately we limit ourselves on analyzing samples as piecewise linear maps without discussing asymptotic properties, mainly by reason of time and motivated by real world data where the discretization is fixed and not chosen by the user.

Summing up, the first part of the thesis has the goal of clarifying any possible theoretical ambiguity, preparing the ground for the second half where we really develop our original ideas. We remark how despite being mathematically "simple", the first part required a meticulous check between numerous papers of different level of expertise, viewpoints and approaches. Part of our effort was indeed to well organize the theoretical foundation in a self-contained and linear way hopefully constructing a "bridge" between this area of pure mathematics and the language of machine learning.

Machine learning is indeed the content of part two, where we conduct a series of numerical experiments so to use the signature transform to target typical problems in the field, like clustering and data visualization.

The second part opens with chapter 4, where our favorite technique is de-

veloped. A full explanation of the problems of *clustering* and *data visualization* is offered. This is done either in the context of "macro-clustering" (aimed at measuring how different two stochastic processes are), as well as in the case of "micro-clustering" (where we analyze the differences between the paths sampled by a single process). We apply these ideas in the setting of reinforcement learning where we try to better understand the behaviors of three popular agents (PPO, DQN, A2C) trained to solve the "cart pole problem".

In chapter 5 we numerically validate the idea of approximating a nonlinear functional with a *linear* one, using data appropriately preprocessed. This is a very important idea and we perform the experiments either in the deterministic setting (max operator), as well as in the stochastic case when computing expected values (this connects to the field of "uncertainty quantification", where this operation is usually called "quantity of interest"). This chapter has a great potential because "non-linearity" commonly constitutes a problem when training models, as recently proved by the explosion of neural networks methodologies.

When working with time series on a practical viewpoint, the detection of *correlation* is always appreciated, since it allows (for instance) to greatly improve prediction methods. This topic is also expanded here with corresponding numerical experiments.

The final chapter (number 7) is then a step towards an even more intuitive understanding of what the signature transform "really does". We choose very familiar shapes (straight lines, sinusoids, impulses) and look at how they are processed by such a transform, commenting the results and always trying to offer an explanation for what observed. In this section we also develop the idea of "reversing" the signature, and the combination of all these remarks will allow to implement a simple algorithm for the generation of artificial data (of limited applicability, but with a great development potential).

We finally conclude the work with a list of further references and connections to other fields of mathematics, without pretending to be exhaustive, but hoping to offer a nice overview of the available landscape.

The author would like to remark how the experiments were on purpose short and compact, aimed at quickly illustrate principles and completely executable on common laptops. The source code is available on request and has been arranged so to offer hopefully good readability and reproducibility. This is very important since accessibility and transparency of code must not be underestimated.

We express our gratitude to the reader and hope that the following work can offer a satisfying experience.

Part I

Foundation

Chapter 1

Background

1.1 Tensor algebras

The goal of this section is to build a solid algebraic architecture to host the main object of this work: the *signature transform*.

We start by revising concepts like tensor products and tensor algebras, including remarks on vectorization and computational aspects.

1.1.1 Tensor products: algebraic definition

Let E be a real vector space, generally assumed to be $E = \mathbb{R}^d$. Most of the following theory extends to the infinite-dimensional case. Nevertheless, we prefer this simpler setting to fix the essentials.

Definition 1.1.1. [Tensor product] Let E, F be two vector spaces with basis B_E and B_F. A pair $(E \otimes F, \phi)$ is called a tensor product iff:

i) $E \otimes F$ is a vector space;

ii) the map $\phi : E \times F \to E \otimes F$ is bilinear;

iii) a basis of $E \otimes F$ is given by $\{\phi(e_i, f_j) | e_i \in B_E, f_j \in B_F\}$ and called the *canonical tensor basis*;

The images $\phi(e_i, f_i)$ are written as $e_i \otimes f_j$.

Definition 1.1.2 (Tensor)**.** Any element of the space $E \otimes F$ is called a tensor.

For more comments concerning the existence and uniqueness of tensor products, we refer to [Hac19].

By definition it follows that $\dim(E \otimes F) = \dim(E)\dim(F)$ revealing the combinatorial nature of tensor products. For instance, consider $\mathbb{R}^2$ and $\mathbb{R}^3$. A basis for their tensor product has length 6 and is given by:

$$\{e_1 \otimes e_1, e_1 \otimes e_2, e_1 \otimes e_3, e_2 \otimes e_1, e_2 \otimes e_2, e_2 \otimes e_3\} \tag{1.1}$$

The order matters: $e_1 \otimes e_2 \neq e_2 \otimes e_1$.

This is clearly in contrast to ordinary Cartesian products, or direct sums, where we would have any element of $\mathbb{R}^2 \oplus \mathbb{R}^3$ written as tuple of length 5.

This difference is mathematically straightforward but nevertheless important to be clarified.

Tensor products emerge when objects are considered distinct and then mixed with a combinatorial perspective, direct sums when they are "glued" together.

1.1.2 Connection to words and alphabets

Let $\Sigma = \{1, \dots, d\}$ be a set of integers. A *word of length* n is a formal concatenation of n digits, like for instance the string "112" for the case $n = 3$. It is possible to rigorously define products between words (as suitable symbolic concatenation) and study their properties.

Let $E = \mathbb{R}^d$ a vector space of the same dimension as the set above, d. Then any word of length n can be seen as a tensor belonging to the product of E with itself, repeated n times. For instance for $n = 3$, the word 112 corresponds to the tensor $e_1 \otimes e_1 \otimes e_2 \in E \otimes E \otimes E$.

This interpretation establishes a connection between the algebraic setting that we are going to further develop, and a more formal combinatorial approach available in the literature.

1.1.3 Connection to non-commutative polynomials

Let's consider *non-commutative* real polynomials, say of degree 3, with 2 indeterminates, called x and y. Each polynomial is therefore written in the form:

$$\begin{aligned} P(a_1, \dots, a_8) = a_1(xxx) + a_2(xxy) + a_3(xyx) + \\ + a_4(xyy) + a_5(yxx) + a_6(yxy) + \\ + a_7(yyx) + a_8(yyy) \end{aligned} \tag{1.2}$$

for some real $2^3 = 8$ coefficients $(a_1, \dots, a_8)$.

Let $E = \mathbb{R}^2$. Each tensor in the space $E \otimes E \otimes E$ can be written in the form:

$$\begin{aligned} T(b_1, \dots, b_8) = b_1(e_1 \otimes e_1 \otimes e_1) + b_2(e_1 \otimes e_1 \otimes e_2) + \\ b_3(e_1 \otimes e_2 \otimes e_1) + b_4(e_1 \otimes e_2 \otimes e_2) + \\ b_5(e_2 \otimes e_1 \otimes e_1) + b_6(e_2 \otimes e_1 \otimes e_2) + \\ b_7(e_2 \otimes e_2 \otimes e_1) + b_8(e_2 \otimes e_2 \otimes e_2) \end{aligned} \tag{1.3}$$

for some real $2^3 = 8$ coefficients.

We can identify the indeterminates x and y with e_1 and e_2, interpreting non-commutative polynomials as tensors, and the other way around. This allows to extend the theory of tensors with results from non-commutative algebra.

1.1.4 On the tensor notation

We develop our work in a pure algebraic tensor notation, but the reader should keep in mind the two given alternative interpretations (formal words and non-commutative polynomials) since they nicely connect to other branches of mathematics and are quite common in the literature.

1.1.5 Vectorization, P-notation and scalar products

Let's come back to the definition of tensor products. If we start from two spaces of dimensions d_1 and d_2, their tensor product has dimension $d_1 d_2$ and can therefore be identified with $\mathbb{R}^{d_1 d_2}$.

Let's fix $E = \mathbb{R}^d$, and define $E^{\otimes n} = E \otimes \cdots \otimes E$, product done n times. For convention set $E^{\otimes 0} = \mathbb{R}$. An element of $E^{\otimes n}$ is called an n-tensor. For every integer n, the resulting space has dimension d^n and can be identified with $\mathbb{R}^{d^n}$.

For our computations, the choice of that isomorphism plays a role. We call *vectorization* any mapping of *canonical* tensor basis to *canonical* euclidean basis.

Definition 1.1.3 (Vectorization)**.** Let $G = \mathbb{R}^d$ with canonical basis $\{g_1, \ldots, g_d\}$. Let $\{e_1, \ldots, e_{d^2}\}$ be the canonical basis of $\mathbb{R}^{d^2}$. A *vectorization* v of $G \otimes G$ is any bijection:

$$v : \{g_i \otimes g_j\} \mapsto \{e_k\} \tag{1.4}$$

linearly extended so to define an isomorphism between $G \otimes G$ and $\mathbb{R}^{d^2}$.

Different choices for the *order* of the indices k generate different vectorization maps, leading to multiple *representations* of the same tensor in $\mathbb{R}^{d^n}$. This opens the way to optimization strategies according to the applications: indeed, the explosion of dimensionality d^n represents a challenge from a computational viewpoint. Mathematical theories like "sparse tensors" and related are connected with this observation.

It is time to introduce some notation that will strongly simplify all the computations from now on.

Definition 1.1.4 (P-notation)**.** For any two positive integers d and n, we call $P(d, n)$ the set of all possible multi-indexes I, $I = (i_1, \ldots, i_n)$, such that i_k is chosen from $\{1, \ldots, d\}$. When the dimension d is completely understood and clear from the context, we write $P(n)$ instead of $P(d, n)$.

Since repetitions are allowed, $P(d, n)$ has cardinality d^n, motivating our choice of P for "powers". For instance, choosing $d = 3$ and $n = 2$ leads to:

$$P(3,2) = \{(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)\} \tag{1.5}$$

of cardinality $3^2 = 9$.

Let's come back to $E = \mathbb{R}^d$ and the tensor space $E^{\otimes n}$. For each possible multi-index $I = \{i_1, \ldots, i_n\}$ for $I \in P(d, n)$, we abbreviate the basis tensor $e_{i_1} \otimes \ldots \otimes e_{i_n}$ simply by $e_I \in E^{\otimes n}$.

Since (by definition) a basis for E is given by all the possible e_I, each element $a \in E^{\otimes n}$ can be uniquely written as:

$$a = \sum_{I \in P(d,n)} a_I e_I \tag{1.6}$$

for suitable real coefficients $a_I \in \mathbb{R}$. One advantage of this notation is that it allows to introduce scalar products quite easily.

Definition 1.1.5 (Scalar product in $\mathbb{E}^{\otimes n}$)**.** Let $E = \mathbb{R}^d$ and $a, b \in E^{\otimes n}$. Then:

$$(a, b)_E \doteq \sum_{I \in P(d,n)} a_I b_I \in \mathbb{R} \tag{1.7}$$

When $E^{\otimes n}$ is identified with $\mathbb{R}^{d^n}$ by using any vectorization, we have by construction that the operation above $(\cdot, \cdot)_{E^{\otimes n}}$ coincides with the standard Euclidean product $(\cdot, \cdot)_{\mathbb{R}^{d^n}}$. This happens because the real coefficients of the basis are only re-arranged when converting tensors into vectors. Since on computers we usually represent tensors as "long arrays" implicitly using a vectorization map, this remark ensures compatibility.

Finally, we recall how given two elements $e_I \in E^{\otimes n}$ and $e_J \in E^{\otimes m}$, their *tensor product* is simply the tensor $e_{i_1} \otimes \cdots \otimes e_{i_n} \otimes e_{j_1} \otimes \cdots \otimes e_{j_m} \in E^{\otimes n+m}$. Extending this expression linearly we obtain the definition of tensor products between two generic elements of $E^{\otimes n}$ and $E^{\otimes m}$.

1.1.6 The tensor algebras $T^\infty(E)$, $T^N(E)$

As usual we fix $E = \mathbb{R}^d$. The tensor algebra associated to E is the direct sum of all the tensor products of the space with itself. More precisely:

Definition 1.1.6 (Tensor algebra)**.** For a vector space E, its tensor algebra is defined as:

$$T^\infty(E) \doteq \bigoplus_{n=0}^{\infty} E^{\otimes n} \tag{1.8}$$

Equivalently, it is the space of formal *infinite* tensor sequences:

$$T^\infty(E) = \{(a_0, \dots, a_n, \dots) | a_n \in E^{\otimes n}\} \tag{1.9}$$

Similarly, we have the:

Definition 1.1.7 (Truncated tensor algebra)**.** For $N \in \mathbb{N}$, the corresponding truncated tensor algebra is defined as:

$$T^N(E) \doteq \bigoplus_{n=0}^{N} E^{\otimes n} \tag{1.10}$$

Equivalently, this is the space of *finite* sequences of length $N + 1$:

$$T^N(E) = \{(a_0, \dots, a_N) | a_n \in E^{\otimes n}\} \tag{1.11}$$

Both $T^\infty(E)$ and $T^N(E)$ are vector spaces by pointwise extensions of the original operations on E. For instance, if $a,\, b \in T^\infty(E)$, their sum is defined as the infinite sequence having in position n the n-tensor $(a+b)_n \doteq a_n + b_n \in E^{\otimes n}$, and similarly for the scalar multiplication.

Every element $(a) \in T^N(E)$ can be identified with the sequence $(a, 0, 0, \dots)$ (same as a in the first $N+1$ components, followed my infinitely many zeroes) in $T^\infty(E)$, offering the linear embedding $\iota : T^N(E) \hookrightarrow T^\infty(E)$.

Conversely, any sequence $b \in T(E)$ can be truncated after $N+1$ terms giving the surjective linear projection $\pi^N : T^\infty(E) \to T^N(E)$, and such that $\pi^N \circ \iota(x) = x$ for each $x \in T^N(E)$.

Tensor products can be extended on tensor algebras.

Definition 1.1.8 (Tensor product on $T^\infty(E)$)**.** For two elements $a, b \in T^\infty(E)$, their tensor product is defined as:

$$a \otimes b \doteq (c_0, c_1, \dots, c_n, \dots) \in T^\infty(E) \tag{1.12}$$

where $c_n = \sum_{k=0}^{n} a_k \otimes b_{n-k} \in E^{\otimes n}$

We remark how this operation resembles classic polynomial multiplication. This insight that can be used to develop more efficient computational implementations.

Finally, by simply ignoring any emerging term with order higher than N, the tensor product can be defined between elements of $T^N(E)$.

1.1.7 The Hilbert space $T(E)$

For two elements $a, b \in T^N(E)$ is it directly possible to define their scalar product:

$$(a, b)_{T^N(E)} = \sum_{n=0}^{N} (a_n, b_n)_{E^{\otimes n}} \in \mathbb{R} \tag{1.13}$$

When trying to extend the definition to the larger algebra $T^\infty(E)$, we can encounter infinite sums. This is at the base of the following definition:

Definition 1.1.9 (Hilbert tensor algebra)**.** We call $T(E)$ the Hilbert vector space $T(E) \subset T^\infty(E)$ of summable tensors:

$$T(E) \doteq \{a \in T^\infty(E) \text{ such that } \|a\|_{T^\infty(E)} < \infty\} \tag{1.14}$$

To prove its completeness, let's consider the classic Hilbert space of summable sequences:

$$l^2 = \{a \in \mathbb{R}^\infty, a = (a_0, a_1, \dots) | \sum_{i=0}^{\infty} a_i^2 < \infty\} \tag{1.15}$$

Our strategy is to construct a linear isomorphism between l^2 and $T(E)$.

Recall the structure $T^\infty(E) = \bigoplus E^{\otimes n}$. For each member we can select a vectorization $v_n : E^{\otimes n} \to \mathbb{R}^{d^n}$. By concatenating them we obtain a linear map:

$$v : T^\infty(E) \to \mathbb{R}^d \oplus \mathbb{R}^{d^2} \oplus \mathbb{R}^{d^3} ... \tag{1.16}$$

given by

$$(a_0, \dots, a_n, \dots) \in T^\infty(E) \mapsto (v_0(a_0), v_1(a_1), \dots) \tag{1.17}$$

Observe that if $\|a\|^2_{T^\infty(E)} < \infty$, then:

$$\|a\|^2_{T^\infty(E)} = \sum_{i\in\mathbb{N}} \|a_i\|_{E^{\otimes n}} = \sum_{i\in\mathbb{N}} \|v_i(a_i)\|_{\mathbb{R}^{d^n}} = \sum_{k\in\mathbb{N}} b_k^2 \in l^2 \tag{1.18}$$

where the last equality holds since every single addend $\|v_i(a_i)\|_{\mathbb{R}^{d^n}}$ can be expressed as a finite sum of d^n squared reals, and we repeat these sums a infinite-countable number of times. For simplicity we grouped all these squared terms under the notation b_k.

This shows that the map above can be restricted to a linear isometry $v : T(E) \to l^2$. Surjectivity can also be verified: for any infinite sequence b, we can always group the ordered coefficients into parts of length d, d^2, ..., then send them back by using the vectorization maps, obtaining a tensor in $T^\infty(E)$ with the same (finite) norm as the squared sums of b, therefore belonging to $T(E)$.

In conclusion, we have the following chain of *strict* inequalities:

$$\mathbb{R} = T^0(E) \subset T^1(E) \subset \dots \subset T^N(E) \subset \dots \subset T(E) \subset T^\infty(E) \tag{1.19}$$

where each set is a closed vector space w.r.t. the next one and $T(E)$ is also an Hilbert space.

1.1.8 Tensors in computer science

In this section we defined the notion of tensors and tensor algebra from an applied mathematical perspective. We would like to invest few lines to point our connections with other fields, hopefully clarifying any possible ambiguity.

On a more abstract mathematical level, tensor spaces are usually defined by using a specific *universal characterization* for bilinear maps, a property we did not mention, which is actually still true in our definition (and explained in the mentioned reference). By insisting on developing connections with other algebraic or analytical structures, tensors are defined slightly differently for instance in the contexts of category theory or differential geometry, to mention two examples.

Our definition aims at something more "practical": combinatorial computations. We defined tensors essentially as ordered couples endowed with polynomial-like operations. For instance, a $3-$tensor is just an element of $E \otimes E \otimes E$, which is completely characterized by multi-indices of depth 3, $a_{ijk}(e_i \otimes e_j \otimes e_k)$. Collections of tensors of various ranks can be multiplied (tensor products) in a way similar to polynomials. All these operations are made explicit, scalar products and norms can be computed.

Finally, we have the classic informal computer science viewpoint. Here a tensor of depth n is simply defined as a collection of numbers arranged via a multi-index of length n. For instance, a picture can be seen as a 2-tensor, with a_{ij} being the color value at position i, j into a Cartesian axis. A video can be seen as a 3-tensor, basically adding the time coordinate to a picture, and similarly for other cases. No further mathematical structure is required.

Note that our formal definition of tensor is *also* a tensor in the sense of computer science, on which we add multiple mathematical operations that enable to develop a more solid theory connected with multiple branches.

In all our numerical experiments we use the Python programming language, more specifically the PyTorch library. We suggest the very interesting book [ES20] (in particular chapters 3 and 4) for everyone interested in understanding more on tensors in the world of programming and their implementations in PyTorch. We finally remark how PyTorch enables a complete scientific computing framework where it is very easy to take advantage of GPU computing, gaining huge performance boosts in multiple scenarios, as well as "semi-parallel" organization of processes (the so-called PyTorch "vectorization"). All the experiments in this work are reproducible in seconds or minutes and have been tested even on the author's machine (i7-12700H, 16 GB Ram, NVIDIA 3060, Debian Linux 12).

1.2 Elementary path integrals

We introduce the notion of path integral, a necessary ingredient for the definition of the *signature transform*. When combined with tensor algebras from the section before, this notion completes the required background.

1.2.1 Path integrals for smooth curves

Let's consider two smooth paths, X and $Y : [a, b] \to \mathbb{R}$. The path integral of X with respect to Y is defined as:

$$\int_a^b X(t)dY(t) \doteq \int_a^b X(t)\dot{Y}(t)dt \tag{1.20}$$

Sometimes we write path integrals more succinctly as $\int_a^b X dY$. Path integrals will play a central role in the whole work therefore it is useful to refresh some key properties.

1.2.2 Invariances

First of all, we point out how path integrals do not vary under time reparametrizations. We recall a change of variable formula, consequence of the chain rule:

Proposition 1.2.0.1 (Change of variables for Riemann integrals)**.** *Let $g : [a,b] \to [c,d]$ be a diffeomorphism, $f : [c,d] \to \mathbb{R}$ a continuous function. Then:*

$$\int_c^d f(t)dt = \int_{g(a)}^{g(b)} f(t)dt = \int_a^b f(g(u))g'(u)du \tag{1.21}$$

It directly follows that:

Proposition 1.2.0.2 (Invariance under time reparametrization)**.** *Let X, Y be two smooth paths $[c,d] \to \mathbb{R}$ and $\phi : [a,b] \to [c,d]$ a diffeomorphism. If $\hat{X} = X \circ \phi$ and $\hat{Y} = Y \circ \phi$, then:*

$$\int_c^d X dY = \int_a^b \hat{X} d\hat{Y} \tag{1.22}$$

Proof. Let $f(u) \doteq X(u)Y'(u)$ and $g(u) \doteq \phi(u)$. Using the proposition above:

$$\begin{gathered} \int_c^d X dY = \int_{\phi(a)}^{\phi(b)} f(t)dt = \int_a^b f(g(u))g'(u)du = \\ \int_a^b X(\phi(u))Y'(\phi(u))\phi'(u)du = \int_a^b \hat{X}(u)(Y \circ \phi)'(u)du = \int_a^b \hat{X} d\hat{Y} \end{gathered} \tag{1.23}$$

□

It is particularly convenient to use the map $\phi : [0,1] \to [c,d]$ defined as $\phi(t) = c + t(d-c)$ to transport the setting on the canonical unit interval $[0,1]$. When fixing a value of $t \in [0,1]$ and restricting ϕ on the domain $[0,t]$ the formula above becomes:

$$\int_0^t \hat{X} d\hat{Y} = \int_c^{c+t(d-c)} X dY \tag{1.24}$$

which will be useful later.

The results above clarify how path integration does not depend on the "speed" at which trajectories are traversed, as long as it is done with enough regularity. The invariance is preserved also when we add *constants* $r \in \mathbb{R}$, since integrating with respect to $Y + r$ cancels r when taking the derivative.

From a computational perspective these changes of domains are very useful for *data normalization*, for instance when making paths starting with value 0 (e.g. by translating $\hat{X} = X - X(0)$), and then compressing them into $[0,1]$ by using the mentioned ϕ map.

Another important remark is that path integrals are still paths themselves! For instance, defining $Z(t)$ as $Z(t) \doteq \int_c^t X dY$, the integration of it against another smooth path $Q(t)$ is totally meaningful.

Let's conclude this section by establishing a notation. If $X : [0,1] \to \mathbb{R}^d$ is a smooth multi-dimensional curve, each **component** is indicated by an **upper** index $X^i(t)$.

Note that we can path-integrate any component X^i with respect to another component X^j, and that we can repeat the process recursively on the obtained curve. This is a key remark before introducing the signature transform, which can be seen as a combinatorial object emerging by performing these cross-integrals among all possible (infinite) combinations of available components.

1.2.3 Bounded variation functions

The previous section was more focused around the hypothesis of smooth regularity. For our applications it is not enough, since practical data usually comes in form of discrete entries. We need to extend the theory.

We remark a brief abuse of notation present in our whole work. When indicating the paths domains (compact intervals in $\mathbb{R}$), we generally denote them with $[a, b]$ or $[c, d]$. In the the latter case, the letter d has of course no relationship with the codomain dimension $\mathbb{R}^d$.

The symbol $C([c, d]; \mathbb{R}^d)$ indicates the Banach space of continuous maps $X : [c, d] \to \mathbb{R}^d$, equipped with the uniform norm $\|X\|_{\sup} = \sup_{t\in[c,d]} \|X(t)\|$, the latter being the canonical Euclidean norm $\|X(t)\|^2 = \sum_{i=1}^{d} (X^i(t))^2$. The shortcut $C([c, d])$ is used when the target set is simply $\mathbb{R}$.

We aim at extending the notion of path integration to less regular cases. We can start with an intuitive idea. Given two curves X and Y, their path integral $\int X dY$ should relate to the sums "$X(t_i)(Y(t_{i+1}) - Y(t_i))$" ($t_i$ being part of a time interval partitioning). Indeed, the choice of Y as the identity function gives back the classic Riemannian sums, confirming this intuition.

Before proceeding in a more precise way, we need to clarify some definitions.

Definition 1.2.1 (Dissection/mesh)**.** For any integer N, an ordered collection of N values t_i such that $c = t_1 < \cdots < t_j < \cdots < t_N = d$ is called a *dissection* (or alternatively, *mesh*) of the interval $[c, d]$. It is generically indicated by the symbol $D_{[c,d]}$, and sometimes only D if the context allows no ambiguity. The number of its elements, N, is also referred as $\sharp D_{[c,d]}$.

Definition 1.2.2 (δ-dissection)**.** Given a dissection D with N elements, its meshsize $|D|$ is the value $|D| \doteq \max_{i\in\{1,\dots,N-1\}} |t_{i+1} - t_i|$. A δ-dissection is a dissection with meshsize δ.

A classic situation is to study convergence properties of functions approximated on some dissections whose meshsizes approach zero progressively.

Example 1.2.0.1 (The uniform dissection)**.** *A typical $h-$dissection is given by choosing an integer M and $h = \frac{d-c}{M}$. Then, $M + 1$ points $\{t_0, \dots, t_M\}$ are generated as $t_i = c + ih$, for $i \in \{0, \dots, M\}$. Note that $|t_i - t_{i+1}| = h$.*

We are now ready to introduce the notion of $1-$variation which will allow to generalize path integrals over less regular paths.

Definition 1.2.3 (1-variation)**.** The 1-variation of a continuous map $X : [c, d] \to \mathbb{R}^d$ is defined as:

$$[X] = \sup_{(t_1,\dots,t_N)\in D} \sum_{i=1}^{N-1} \|X(t_{i+1}) - X(t_i)\| \in \mathbb{R} \cup \{\infty\} \tag{1.25}$$

where the sup, with an abuse of notation, is taken among *all possible dissections* of the interval $[c, d]$.

Any continuous path X having finite $1-$variation is called of *bounded variation*. The $1-$variation is a seminorm, but we can slightly modify it to obtain a proper norm.

Definition 1.2.4 (Buonded variation norm)**.** The bounded variation norm for a continuous function of bounded variation $X : [c, d] \to \mathbb{R}^d$ is

$$\|X\|_{bv} \doteq [X] + \|X(c)\| \tag{1.26}$$

Theorem 1.2.1 (Bounded variation norm and BV spaces)**.** *Let $BV^d([c, d])$ be the set of continuous path of bounded variation from $[c, d]$ to $\mathbb{R}^d$, $BV_0^d([c, d])$ its subset of maps starting at zero. Then, both spaces are Banach when equipped with the bounded variation norm. The set $BV_0^d([c, d])$ is a closed subspace of $BV^d([c, d])$.*

Proof. See theorem 1.25 of [FV10], page 31. □

We sometimes abbreviate $BV^1([a, b])$ and $BV_0^1([a, b])$ simply by $BV([a, b])$ and $BV_0([a, b])$ respectively.

The following proposition will help in having quantitative bounds on paths of bounded variation.

Proposition 1.2.1.1 (1-variation of components)**.** *For every continuous map $X : [a, b] \to \mathbb{R}^d$, we have $X \in BV^d([a, b])$ if and only if for each component X^k, one has $X^k \in BV([a, b])$ for all $k \in \{1, \dots, d\}$. Additionally, $[X^k] \le [X]$ and $[X] \le \sum_{k=1}^d [X^k]$.*

Proof. Let's suppose $X \in BV^d$. Then for every component k we have

$$\begin{gathered} |X^k(t_{i+1}) - X^k(t_i)| \le \\ \sqrt{\sum_{j=1}^{d} |X^j(t_{i+1}) - X^j(t_i)|^2} = \\ \|X(t_{i+1}) - X(t_i)\| \end{gathered} \tag{1.27}$$

which implies $[X^k] \le [X]$ when passing to the sup on all possible interval meshes ($a = t_1 < \cdots < t_N = b$).

For the other implication, observe that for a generic vector $v \in \mathbb{R}^d$ the triangle inequality implies:

$$\|v\| = \|v^1 e_1 + \cdots + v^d e_d\| \leq \sum_{k=1}^{d} \|v^k e_k\| = \sum_{k=1}^{d} |v^k| \tag{1.28}$$

which translates into:

$$\begin{gathered} \sum_{i=0}^{N-1} \|X(t_{i+1}) - X(t_i)\| \leq \\ \sum_{i=0}^{N-1} \sum_{k=1}^{d} |X^k(t_{i+1}) - X^k(t_i)| = \\ \sum_{k=1}^{d} \sum_{i=0}^{N-1} |X^k(t_{i+1}) - X^k(t_i)| \end{gathered} \tag{1.29}$$

implying $[X] \leq \sum_{k=1}^{d} [X^k]$ by taking the sup with respect to the meshsize indexed by i.

□

We clarify now the relationship between $(C([c,d];\mathbb{R}^d), \|\cdot\|_{sup})$, the Banach space of continuous functions with uniform norm, and $(BV^d([c,d]), \|\cdot\|_{bv})$, the Banach space of maps with bounded variation as above.

We start with a lemma in that regard.

Lemma 1.2.2. *For any continuous function $X : [c,d] \to \mathbb{R}^d$, we have:*

$$\sup_t \|X(t)\| \leq \|X\|_{bv} \tag{1.30}$$

Proof. Let t^* be a point of maximum of the continuous X on the compact $[c,d]$. Assume $t^* \in (c,d)$. Then:

$$\|X(t^*)\| - \|X(c)\| \leq \|X(t^*) - X(c)\| \leq \|X(c) - X(t^*)\| + \|X(t^*) - X(d)\| \leq [X] \tag{1.31}$$

where we used the triangle inequality and finally the sum on the three-points dissection $\{c < t^* < d\}$ combined with the definition of $1-$variation. Therefore:

$$\sup_t \|X(t)\| = \|X(t^*)\| \leq [X] + \|X(c)\| = \|X\|_{bv} \tag{1.32}$$

The cases for $t^* = d$ or c follow the same logic, this time on the simpler two-points dissection $\{c, d\}$. □

Lemma 1.2.3 (Equivalence of norms)**.** *The bounded variation norm, $\|X\|_{bv} = [X] + \|X(c)\|$, is equivalent to the alternative norm:*

$$\|X\|_A \doteq [X] + \sup_t \|X(t)\| \tag{1.33}$$

Proof. Directly by construction

$$\|X\|_{bv} = [X] + \|X(c)\| \leq [X] + \sup_t \|X(t)\| = \|X\|_A \tag{1.34}$$

Conversely, using the lemma above:

$$\begin{aligned} \|X\|_A = [X] + \sup_t \|X(t)\| \leq [X] + \|X\|_{bv} = \\ = [X] + [X] + \|X(c)\| \leq 2[X] + 2\|X(c)\| = 2\|X\|_{bv} \end{aligned} \tag{1.35}$$

□

Corollary 1.2.3.1. *The set $BV^d([c,d])$ is a linear subspace of $C([c,d];\mathbb{R}^d)$ and the inclusion $BV^d([c,d]) \subset C([c,d];\mathbb{R}^d)$ is continuous and strict. Convergence in the bounded variation norm implies uniform convergence, but the converse is not necessarily true.*

Proof. Since we proved the equivalence between the two norms, we can directly apply proposition 1.7 from [Lyo07]. □

We have connected BV functions to continuous maps. The next step is to check possible relationships when differentiability comes into play.

Proposition 1.2.3.1. *Let $X : [c,d] \to \mathbb{R}^d$ be a C^1 path (differentiable with continuous derivatives). Then, it is of bounded variation, $X \in BV^d([c,d])$, and*

$$[X] = \int_c^d \|\dot{X}(t)\| dt < \infty \tag{1.36}$$

Proof. See Proposition 1.24 of [FV10], page 30. □

Sometimes the bounded variation norm can be explicitly interpreted.

Example 1.2.3.1. *Let $f : [0,1] \to \mathbb{R}$ be a C^1 function and X the planar curve $X(t) = (t, f(t))$. Then:*

$$[X] = \int_0^1 \|\dot{X}(t)\| dt = \int_0^1 \sqrt{1 + f'(t)^2} dt \tag{1.37}$$

which is the length *of the function graph! This is to highlight the interpretation of bounded variation curves as paths of finite length.*

Given a curve $X(t)$, the corresponding map $\hat{X}(t) = (t, X(t))$ will also play an important role later better explained. We call this operation a *time augmentation*. Note that if $X(t)$ is in BV_0^{d-1}, then $\hat{X} \in BV_0^d$ and $\|\hat{X}\|_{bv} \leq 1 + \|X\|_{bv}$.

To conclude this section, we provide an example of a two dimensional planar curve *not* of bounded variation: the trajectory of a Brownian motion. This result is classic and available in usual textbooks of stochastic analysis. The fact that it is not of bounded variation constitutes the premise on which the development of more general stochastic integration theories (like Itō and Stratonowich) are built.

1.2.4 Linear piecewise approximation

In this section we define the method of piecewise linear approximation and relate it to bounded variation functions. Let $X : [c, d] \to E = \mathbb{R}^d$ be a continuous path, and $D_\delta = (t_1, \ldots, t_k)$ a δ-mesh of $[c, d]$.

Definition 1.2.5 (Piecewise linear approximation)**.** The piecewise linear approximation of X w.r.t. D_δ is the map $X_\delta : [c, d] \to E$ defined as:

$$X_\delta(t) \doteq X(t_i) + \frac{t - t_i}{t_{i+1} - t_i}(X(t_{i+1}) - X(t_i)) \tag{1.38}$$

on every $t_i \leq t \leq t_{i+1}$. The values t_i in the mesh are called *nodes*.

Piecewise approximations are always of bounded variation:

Proposition 1.2.3.2. *Any map X_δ corresponding to a piecewise approximation in the sense of the definition above is of bounded variation. In symbols, $X_\delta \in BV^d([c, d])$.*

Proof. Proposition 1.28 [FV10], page 32. □

We are actually able to compute the bounded variation norm of a piecewise linear path. Let's start by a preparatory lemma.

Lemma 1.2.4. *Let $X : [c, d] \to \mathbb{R}^d$ be a continuous map, and D^1 and D^2 two meshes on $[c, d]$ such that $D^1 \subseteq D^2$. Then:*

$$\sum_{t_i \in D^1} \|X(t_{i+1}) - X(t_i)\| \leq \sum_{t_k \in D^2} \|X(t_{k+1}) - X(t_k)\| \tag{1.39}$$

Proof. Consider $t^* \in D^2 \setminus D^1$. By construction (since D^1 is a mesh) we can always find points t_i and t_{i+1} in D^1 such that $t_i < t^* < t_{i+1}$. They are also contained in D^2 (since $D^1 \subseteq D^2$). Using then the triangle inequality:

$$\|X(t_{i+1}) - X(t_i)\| \leq \|X(t_{i+1} - t^*)\| + \|X(t^*) - X(t_i)\| \tag{1.40}$$

The proof is concluded by repeating the reasoning for each point in $D^2 \setminus D^1$ and taking the overall sums, possible since these points are of finite quantity. □

Proposition 1.2.4.1. *Let X be a piecewise linear curve on N nodes, defined on the mesh D. Then:*

$$[X] = \sum_{t_i \in D} \|X(t_{i+1}) - X(t_i)\| \tag{1.41}$$

Proof. We have to prove that the mesh D corresponding to the nodes is the one capable of maximizing the sums in the bounded variation definition. We use in this proof the temporarily notation shortcut:

$$\sum_D \doteq \sum_{t_i \in D} \|X(t_{i+1}) - X(t_i)\| \tag{1.42}$$

Let D^1 be a generic mesh. We want to prove that $\sum_{D^1} \leq \sum_D$.

First of all, define D^2 as the mesh obtained by adding to D^1 all the nodal points of the piecewise function. In other words, $D^2 = D^1 \cup D$. Because of the lemma above, $\sum_{D^1} \leq \sum_{D^2}$. We conclude the proof by showing $\sum_{D^2} = \sum_D$.

Let $\hat{t}$ be a point in D^2 not included in D. By construction, there must exists two nodes t_i, t_{i+1} in D such that $t_i < \hat{t} < t_{i+1}$. On that point the function has value:

$$X(\hat{t}) = X(t_i) + \frac{\hat{t} - t_i}{t_{i+1} - t_i}(X(t_{i+1}) - X(t_i)) \tag{1.43}$$

by definition of piecewise linear map. Therefore by taking the norm:

$$\|X(\hat{t}) - X(t_i)\| = \|X(t_{i+1}) - X(t_i)\|(\frac{\hat{t} - t_i}{t_{i+1} - t_i}) \tag{1.44}$$

and subtracting to the next point:

$$\|X(t_{i+1}) - X(\hat{t})\| = \|X(t_{i+1}) - X(t_i)\|(1 - \frac{\hat{t} - t_i}{t_{i+1} - t_i}) \tag{1.45}$$

leading to:

$$\|X(t_{i+1}) - X(\hat{t})\| + \|X(\hat{t}) - X(t_i)\| = \|X(t_{i+1}) - X(t_i)\| \tag{1.46}$$

It means that $\sum_{D^2} = \sum_{D^2 \setminus \{\hat{t}\}}$. In other words, the point t can be omitted when computing the sum. Repeating the same procedure for all non-nodes point (they are finite), we have $\sum_{D^2} = \sum_D$ as desired. □

Remark. Piecewise linear maps with a very high number of nodes and oscillating values will tend to have in practice big bounded-variation norms. To make computations more reliable, we will sometimes *rescale* the path with a coefficient $0 < \lambda < 1$ so to reduce the bounded variation ($\|\lambda X\|_{bv} = \lambda \|X\|_{bv}$) still preserving the path's "geometry".

Let's go back the the general approximation theory. The following concise notation will simplify future ideas:

Definition 1.2.6 (piecewise approximating sequence)**.** A sequence $\{X_k\}_{k \in \mathbb{N}}$ of piecewise linear approximations for X, such that the meshsize goes to zero for $k \to \infty$ is called a *piecewise approximating sequence* for X.

The piecewise approximations of a differentiable function with continue first and second derivatives, converge uniformly to the original function when the meshsize goes to zero:

Theorem 1.2.5 (Piecewise convergence: C^2 case)**.** *Let X be a two-times differentiable path $[c, d] \to \mathbb{R}^d$ and $\{X_k\}_{k \in \mathbb{N}}$ a piecewise approximating sequence of X. Then:*

$$\|X - X_k\|_{\sup} \to 0 \quad \textit{if} \quad k \to \infty \tag{1.47}$$

Proof. See paragraph 8.3 from [Kre98], page 169. □

When, additionally, the original curve is smooth, that convergence happens also in the bounded variation norm:

Theorem 1.2.6 (Piecewise convergence: C^∞ case)**.** *Let X be a **smooth** path $[c,d] \to \mathbb{R}^d$ and $\{X_k\}$ a piecewise approximating sequence of X. Then:*

$$\|X - X_k\|_{bv} \to 0 \quad \textit{if} \quad k \to \infty \tag{1.48}$$

Proof. Theorem 1.34, [FV10], Page 35. □

This result is very, very important, essentially because (as we will see) the signature transform is "well approximated" for paths that converge in bounded-variation.

1.2.5 Riemann-Stieltjes integrals

The goal of this section is to extend integration theory to paths of finite variations and clarify convergence properties in relation to smooth or piecewise linear paths.

We work with paths on $\mathbb{R}$, one dimensional. When integrating a more general path in $\mathbb{R}^d$, we interpret all the operations to be component-wise.

Definition 1.2.7 (Riemann-Stieltjes integral)**.** Let X and Y be two paths from $[c,d]$ to $\mathbb{R}$. Let $D_\delta = (t_0^\delta < \cdots < t_i^\delta < \cdots < t_d^\delta)$ be a sequence of dissections of $[c,d]$ with $|D_\delta| = \delta \to 0$. Let ξ_i^δ be some points in $[t_i^\delta, t_{i+1}^\delta]$. Assume that the sums:

$$\sum_{i=0}^{\sharp D_\delta - 1} Y(\xi_i^\delta)(X(t_{i+1}^\delta) - X(t_i^\delta)) \tag{1.49}$$

converge for $\delta \to 0$ to a limit $I \in \mathbb{R}$ *independent* of the choice of ξ_i^δ and the choice of dissections D_δ. Then we define:

$$\int_c^d Y dX \doteq \int_c^d Y(t) dX(t) \doteq I \in \mathbb{R} \tag{1.50}$$

This the Riemann-Stieltjes integral of Y against X.

Theorem 1.2.7 (Integrating against a BV function)**.** *Let $X \in BV([c,d])$ and $Y \in C([c,d])$. Then the Riemann-Stieltjes integral $\int_c^d Y dX$ exists (and is unique).*

Proof. See proposition 2.2, page 45 of [FV10]. □

Example 1.2.7.1 (Integrating against the identity)**.** *When integrating against the identity curve $I : [c,d] \to \mathbb{R}$, $I(t) = t$, we have $\int_c^d Y dI = \int_c^d Y dt$ in the classical Riemannian sense.*

The compatibility does not end here, and we also have a good interplay when working with the smooth case.

Proposition 1.2.7.1 (Integrating against a differentiable function)**.** *If Y, X are two path on $[c,d]$, the former continuous, the latter differentiable. Then $\int_c^d Y dX = \int_c^d Y(t)\dot{X}(t)dt$. The first integral is to be meant in the sense of Riemann-Stieltjes. Therefore, there is no ambiguity when dealing with smooth function integration and the Riemann-Stieltjes path integration can be safely seen as an extension of the previous case.*

Proof. Consult the end of page 47 from [FV10]. □

Another important remark is about the change of variable formula.

Proposition 1.2.7.2 (Change of variable for Riemann-Stieltjes)**.** *Let $Y \in C([c,d])$, $X \in BV([c,d])$, and $\phi : [a,b] \to [c,d]$ a non-decreasing diffeomorphism. Defining $\hat{X} = X \circ \phi$, $\hat{Y} = Y \circ \phi$, we have:*

$$\int_a^b \hat{Y} d\hat{X} = \int_c^d Y dX \tag{1.51}$$

Proof. This is under proposition 2.4 from [FV10], page 48. □

The chain rule above essentially extends the situation for the non-smooth case, having an important consequence as already pointed out for smooth maps: path integrals will be invariant when performing the classic change of coordinate $[0,1] \to [c,d]$, $\phi : t \mapsto c + t(d-c)$ introduced before.

We conclude this section with a continuity property that we will exploit a lot.

Proposition 1.2.7.3 (Convergence in BV implies convergence of the integrals)**.** *For any time interval $[c,d] \subseteq \mathbb{R}$, the map:*

$$\int_c^{\cdot} : C([c,d]) \times BV([c,d]) \to BV([c,d]) \tag{1.52}$$

given by $(Y,X) \mapsto \int_c^{\cdot} Y dX$ is well-defined, of bounded variation, bilinear and continuous. With the open integral we mean the path $\int_c^{\cdot} Y dX : [c,d] \to \mathbb{R}$ mapping $u \mapsto \int_c^u Y dX$.

Proof. See the beginning of section 2.2 from [FV10], page 49. □

Corollary 1.2.7.1 (Convergence of self-integrated integrands)**.** *Let X be a smooth curve $X : [c,d] \to \mathbb{R}$, and $\{X_k\}_{k\in\mathbb{N}}$ a piecewise approximating sequence of X. Then for $k \to \infty$ we have:*

$$\int_c^{\cdot} X_k dX_k \to \int_c^{\cdot} X dX \tag{1.53}$$

in bounded variation (all these open integrals are in $BV([c,d])$).

Proof. Since the original path X is smooth, the piecewise approximation converges in the uniform norm as well as in bounded variation. Apply then the proposition above. □

For further properties of the Riemann-Stieltjes integral we suggest reading chapter 2 of [FV10].

1.2.6 Recap of the section

The goal of this section was to introduce paths integrals for the case of piecewise linear maps, showing their theoretical convergence and invariances as in the case of time reparametrizations (important when "normalizing" data). Of particular relevance will be the case when we integrate the components of a multidimensional discrete path X against each others $\int X^i dX^j$, as better explained in the next chapter.

1.2.7 An additional remark on generic p variation

We defined the notion of $1-$variation without commenting more about the meaning of the number '1'. It is indeed possible to extend the construction and integration theory on paths of bounded $p-$variations by looking at sums $\|X(t_{i+1}) - X(t_i)\|^p$ instead. This connects well with a more abstract mathematical setting called *rough path* theory. This goes beyond the purpose of this work, where we stay in more regular cases and explore multiple applications in modern machine learning.

For more information, we suggest for instance the paper [CK16], section 1.4 or the book [Lyo07].

Chapter 2

The signature transform: definition

2.1 Defining the signature transform

2.1.1 Motivation

At the end of the previous chapter we briefly mentioned the existence of more general integration theories suitable for less regular paths. This leads to the definition of more abstract differential equations, as well as the development of corresponding solving strategies.

Let's consider for a moment a more classic ODE setting, where the Picard algorithm can be exploited to compute the unknown solution. Recall that the strategy is to rewrite the ODE as an integral equation, whose fixed point is the searched solution. Using an iterative algorithm directly derived from the general fixed-point theory, such a solution is finally numerically approximated.

We remark how during the process multiple iterated integrals are computed and finally used to characterize the true solution.

By extending this idea to differential equations *driven by a path* X, as for instance quickly shown in section 1.2.3, page 7 of [CK16], one can show that a specific collection of iterated integrals called the *signature* of X is able to completely characterize the solution for equations of the form "$dY = F(Y)dX$".

Because of this *characterization*, one can intuitively guess that the signature of a path contains "enough information" about it, and might therefore be a good *feature* from a machine learning viewpoint when working with time series.

In other words, if it is true that the signature of a curve X emerges when solving more abstract ODEs, is it on the other hand an object defined only on the path X itself and in principle independent on its original ODE environment.

This is the approach we follow. Instead of using the signature to solve and study *rough differential equations*, we focus more on its intrinsic properties targeting machine learning and numerical applications.

2.1.2 The signature coefficients

Let X be a smooth path $X : [a, b] \to \mathbb{R}^d$. We recall the notion of simplex, which constitutes a fundamental integration domain.

Definition 2.1.1 (simplex)**.** The d-dimensional simplex on $[a, b]$ is the set:

$$\Delta_{ab}^d = \{(u_1, \dots, u_d), u_i \in [a, b], a < u_1 < \cdots < u_d < b\} \tag{2.1}$$

When the base interval is $[0, 1]$, we write Δ^d, omitting the subscript.

To define the signature transform we first need to get the signature coefficients, which can be derived from a combinatorial approach using iterated path integrals.

Let n be an integer and d the codomain dimension in our path $X : [a, b] \to \mathbb{R}^d$. Let $P(d, n)$ be as usual the set of all possible multi-indices $I_n = (i_1, \dots, i_n)$, where $1 \leq i_k \leq d$. Note that $P(d, n)$ contains a total of d^n elements.

Definition 2.1.2 (Signature coefficient)**.** The signature coefficient corresponding to the multi-index $I_n = (i_1, \dots, i_n) \in P(d, n)$ is the real number s_{I_n} defined as:

$$s_{I_n} = \int_{\Delta_{ab}^n} \dot{X}^{i_1}(u_1) \cdots \dot{X}^{i_n}(u_n) du_1 \cdots du_n \tag{2.2}$$

where $\dot{X}^k$ is the derivative of the k-th component of the path X.

Remark. More explicitly, the signature coefficient can be equivalently computed as the concatenation of n integrals:

$$s_{I_N} = \int_a^b \int_a^{u_n} \int_a^{u_{n-1}} \cdots \int_a^{u_2} dX^{i_1}(u_1) \dots dX^{i_n}(u_n) \tag{2.3}$$

This follows directly from the definition of the simplex given above, and is more common in the literature as for instance in the recommended introduction [CK16]. If on the one hand the former formulation is more compact to write, the latter brings to us a great advantage. Being defined as an iteration of *path integrals*, it already shows how the signature can directly be extended on curves of bounded variation!

Furthermore we directly have many properties inherited by path integral theory, like the invariance under translations. Let $\phi : [0, 1] \to [a, b]$ the canonical transform $t \mapsto a + t(b - a)$. Let $\hat{X}$ be the reparametrized curve $\hat{X}(t) = (X \circ \phi)(t) - X(a)$. Then we have:

Proposition 2.1.0.1 (Signature invariance)**.** *The signature coefficients for the curve $\hat{X}$ are exactly the same as for the curve X.*

Proof. This is a corollary from the path integral invariance explained before, a result of the change of variable formula valid for the smooth as well as for the bounded variation case. □

The curve $\hat{X}$ has the computational advantage of starting from 0 and evolving into the interval $[0,1]$, which can be interpreted as a form of data normalization.

Summing up, in this section we defined the notion of signature coefficient for a bounded variation curve, and seen how without loss of generality we can assume to work on $[0,1]$ and to start from the origin.

2.1.3 The signature transform

The complete signature transform is essentially an algebraic arrangement of the previously defined coefficients, so that they mirror multiple analytic properties.

Let $X : [0,1] \to \mathbb{R}^d$ be a bounded variation curve, and $P(d,n)$ as usual the set of all possible multi-indices $I_n = (i_1, \ldots, i_n)$, where $1 \leq i_k \leq d$. As a notation shortcut, let $E = \mathbb{R}^d$. Recall that for each choice of I_n we have the real coefficient s_{I_n} defined in the section before. Furthermore, recall the notation e_{I_n} as a shortcut for $e_{i_1} \otimes \ldots \otimes e_{i_n} \in E^{\otimes n}$.

Definition 2.1.3 (n-th level of the signature)**.** Let n be a positive integer, $X : [0,1] \to E$ be a curve of bounded variation. The corresponding level n of the signature transform is the n-tensor S_n defined as:

$$S_n \doteq \sum_{I_n \in P(d,n)} s_{I_n} e_{I_n} \in E^{\otimes n} \tag{2.4}$$

For $n = 0$, S_0 is always defined as the constant number $1 \in \mathbb{R} = E^0$.

In other words, the level n of a signature transform stores the values of crossed integrations between n components of the speed (derivative) of a path, arranged so to keep track of their integration order.

The signature levels are bounded in norm.

Proposition 2.1.0.2 (Upper bound for level n)**.** *If $X : [0,1] \to E$ is a path of bounded variation, then, for each n:*

$$\|S_n\|_{E^{\otimes n}} \leq \frac{[X]^n}{n!} \tag{2.5}$$

where, as usual, $[X]$ is the previously defined 1-variation of X. We adopt the convention $0! = 1$.

Proof. This is the content of proposition 2.2 in [Lyo07], page 27. The proof uses a reparametrization to simplify integration over the simplex, from which the factorials come from. □

We can finally proceed to define the signature transform, the central object in this work.

Definition 2.1.4 (Signature transform)**.** The signature transform of the bounded variation curve $X : [0,1] \to E$ is the element $S(X) \in T(E) \subset T^\infty(E)$ defined as:

$$S(X) \doteq \sum_{n=0}^{\infty} S_n \in T(E) \tag{2.6}$$

where each S_n has been embedded into $T(E)$ so to make the sum mathematically clear (or, equivalently, it is possible to use direct sums $\oplus$ instead).

The signature is in principle an object belonging to $T^{\infty}(E)$, but the following corollary ensures the correctness of the definition above.

Corollary 2.1.0.1 (Upper bound for the signature transform)**.** *Let $X : [0,1] \to E$ be a path of bounded variation. Then $\|S(X)\|_{T^{\infty}(E)} < \infty$, i.e. $S(X) \in T(E)$.*

Proof. Direct consequence of the proposition above. Indeed:

$$\|S(X)\|_{T^{\infty}(E)} = \sum_{n=0}^{\infty} \|S_n\|_{E^{\otimes n}} \leq \sum_{n=0}^{\infty} \frac{[X]^n}{n!} = \exp([X]) < \infty \tag{2.7}$$

□

2.1.4 Notation in other papers

Some papers in the literature (for instance [NSW+20]) use an even more different notation when referring to the signature transform. The purpose of this section is to clarify this potential ambiguity.

Let $X : [a,b] \to \mathbb{R}^d$ be a curve of bounded variation. The signature is sometimes defined as a sequence of k-tensors $\{X^k\}_{k\in\mathbb{N}}$ each defined as:

$$X^k = \int_{t_1<t_2<\cdots<t_k, t_1,\ldots,t_k\in[a,b]} dX(t_1) \otimes \cdots \otimes dX(t_k) \tag{2.8}$$

The first element X^0 set to $1 \in \mathbb{R}$.

A first look at this writing could confuse readers coming from a different background, since the symbol "$\otimes$" usually refers to product measures when in the context of integration. Here, this is not the case. The notation above should be interpreted essentially as a combinatorial operation, similarly to what we did in the previous section. In other words, for each fixed k, we define a k-tensor called X^k, element of $(\mathbb{R}^d)^{\otimes k}$, such that:

$$X^k = \sum_{I_k \in P(d,k)} x_{I_k} e_{I_k} \tag{2.9}$$

and for each multi-index $I_k = (i_1, \ldots, i_k)$:

$$\begin{aligned} x_{I_k} = \int_{t_1<t_2<\cdots<t_k, t_1,\ldots,t_k\in[a,b]} dX^{i_1}(t_1)\cdots dX^{i_k}(t_k) = \\ \int_{\Delta^k_{ab}} dX^{i_1}(t_1)\cdots dX^{i_k}(t_k) = s_{I_k} \end{aligned} \tag{2.10}$$

This coincides with our definition, therefore there is no conflict between our exposition and other variants in the literature.

2.1.5 The signature truncation

The definition in the section before points out how the signature is essentially an algebraically organized set of an *infinite* number of coefficients. This can be quite impractical for concrete applications. Therefore we mainly work with its truncated version.

Definition 2.1.5 (Truncated signature)**.** For each integer D, the truncated signature at *depth* D is defined as:

$$S^D(X) = \sum_{n=0}^{D} S_n \in T^D(E) \subset T(E) \tag{2.11}$$

The truncated signature can be bounded as a direct corollary of the corresponding proposition above.

Corollary 2.1.0.2 (Bounds for the truncated signature)**.** *Let $X \in BV^d$. Then:*

$$\|S^D(X)\|_{T^D(E)} \leq \sum_{n=0}^{D} \frac{[X]^n}{n!} \tag{2.12}$$

We find useful to clarify an approximation result:

Proposition 2.1.0.3 (Convergence of the truncated signature)**.** *Let X be a path of bounded variation. Then as $D \to \infty$, we have:*

$$S^D(X) \xrightarrow{T(E)} S(X) \tag{2.13}$$

Proof. By direct computation using the bounds above:

$$\|S(X) - S^D(X)\|_{T(E)} = \sum_{n=D+1}^{\infty} \|S_n\|_{E^{\otimes n}} \leq \sum_{n=D+1}^{\infty} \frac{[X]^n}{n!} = \exp([X]^n) - \sum_{n=0}^{D} \frac{[X]^n}{n!} \to 0 \tag{2.14}$$

as $D \to \infty$. □

We conclude this section by pointing out a convergence property with respect to path discretization.

Proposition 2.1.0.4 (Convergence of path approximations)**.** *Let X_k be a sequence of approximations of X, such that as $k \to \infty$:*

$$X_k \xrightarrow{bv} X \tag{2.15}$$

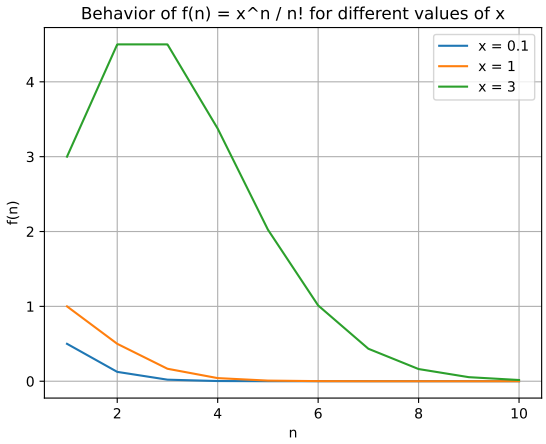
(for instance, this holds with piecewise approximating sequences)

Then, for each $D \in \mathbb{N}$:

$$S^D(X_k) \xrightarrow{T(E)} S^D(X) \tag{2.16}$$

when $k \to \infty$.

Proof. See corollary 2.11 of [Lyo07], page 32. The proof uses the interpretation of the truncated signature as a solution to a particular algebraic differential equation, exploiting then its continuity properties. □



2.1.6 The signature decay

In the previous section we explained how the signature transform is bounded, and in particular that each level n satisfies $\|S_n\|_{E^{\otimes n}} \leq \frac{[X]^n}{n!}$.

We would like to analyze in more details this behavior, so to better understand the effects of truncating the signature.

If we had a strict monotone decrease like $\|S_n\|_{E^{\otimes n}} > \|S_{n+1}\|_{E^{\otimes (n+1)}}$, we would be sure that the information left after stopping the signature is somehow small and controlled.

To reach this goal, we have a look at the function $\frac{x^n}{n!}$. Given a fixed value of x, we know it must decay from a certain n on, since their sums converge to the exponential function $\exp(x)$. Here we remark *when* this decay starts.

The inequality to set is $\frac{x^{n+1}}{(n+1)!} < \frac{x^n}{n!}$ which can be rewritten as $n > x - 1$ since all the involved quantities are positive.

Therefore, the value of x determines the starting point of the decay of the function! In order to graphically convey this message, a plot displaying examples with $x = 0.5, 2, 5$ is attached.

When translating into the signature context, the value of x is the total variation norm of the path. The higher is this value, the later we observe a monotone decay of the signature coefficient.

As a natural consequence of these remarks, we will sometimes modify our paths to reduce their total variation. In principle there is no need to use sophisticated methods: a simple scalar multiplication by $\lambda \in \mathbb{R}$ can be very helpful (indeed we have $[\lambda X] = \lambda[X]$). Choosing $\lambda = \frac{1}{[X]}$ would surely ensure the signature decay from the very beginning on ($x = 1$, as also readable on the plot). On the other hand, too small values can also represent a problem from a numerical viewpoint.

In conclusion, controlling the signature size represents a remark that we should always keep in mind when performing experiments, despite the lack of a

'universal' solution. Each case will be appropriately commented if required.

2.1.7 Practical conclusion

The signature convergence theorems are of crucial importance even from a numerical perspective, as here explained in details.

Given a smooth path $Y : [a, b] \to \mathbb{R}^d$, we are interested in working with its signature, $S(Y)$.

First, we need to accept accessing only a limited amount of its infinite expansion. We choose a target level $D \in \mathbb{N}$ and try to approximate the value of $S^D(Y)$. Thanks to the theorem above, we know the approximation to be exact for $D \to \infty$, but we usually stop around $D = 5$ for computational reasons (exponential growth of the number of coefficients to compute).

Let $\phi : [0, 1] \to [a, b]$ be the canonical translation. Since it is easier to work (and compute) with maps on the interval, we consider the time translation $X = Y \circ \phi$. It does not introduce any change since $S^D(X) = S^D(Y)$. At this point we can also translate the curve so to start from zero.

Finally, we cannot take into account the full smooth path X, but we rather consider a piecewise approximating sequence $\{X_k\}$ and stop at a sufficient small meshsize (i.e. high value of k). This is again in accordance with the theory since $S^D(X_k) \to S^D(X)$ as $k \to \infty$.

The last step is the computation of $S^D(X_k)$, which is precisely the purpose of the next section.

In conclusion, the original infinite tensor $S(Y)$ is approximated with a (generally long) array $S^D(X_k)$ according the rules above. This strategy will be the default method in this work.

2.2 Computing the signature transform

The goal of this section is to explain aspects concerning the practical computation of the signature. In the previous section we clarified how (and why) given a smooth path, we approximate it with linear piecewise interpolations. The signature is evaluated always and only on these approximations, since asymptotic convergence is guaranteed.

In other words, we are only interested in computing signatures on piecewise linear paths $X : [a, b] \to \mathbb{R}^d$. We start with simple cases and proceed progressively.

2.2.1 Working with the segment on $[0, 1]$

The simplest path is provided by the standard unitary segment. When we work under this condition, with a linear path $X : [0, 1] \to \mathbb{R}^d$, we can always rewrite it in the form $X(t) = X(0) + t(X(1) - X(0))$ pointing out its constant derivative. As usual, we use upper indices for the components, for instance $X^1 : [0, 1] \to \mathbb{R}$ for the first, and similarly for the others.

The constant derivatives come into play when computing the signature transform. Since the integral of constants over the standard simplex is exact, we can perform a precise signature computation.

Let $I = \{i_1, \ldots, i_n\}$ be a multi-index of length n, where each value is in $\{1, \ldots, d\}$. In other words, $I \in P(d, n)$. The corresponding signature coefficient on the interval $[0, 1]$ is then:

$$\begin{aligned} s_I(X) = \int_{\Delta^n} dX^{i_1}(t_1) \cdots dX^{i_n}(t_n) = \\ \int_{\Delta^n} (X^{i_1}(1) - X^{i_1}(0)) \cdots (X^{i_n}(1) - X^{i_n}(0)) dt_1 \cdots dt_n = \\ = \prod_{k=1}^{n} (X^{i_k}(1) - X^{i_k}(0)) \int_{\Delta^n} dt_1 \cdots dt_n = \\ = \frac{1}{n!} \prod_{k=1}^{n} (X^{i_k}(1) - X^{i_k}(0)) \end{aligned} \tag{2.17}$$

Where, we recall, Δ^n is the unitary simplex in dimension n,

$$\Delta^n = \{(x_1, \ldots, x_n) \in \mathbb{R}^n, 0 < x_1 < \cdots < x_n < 1\} \tag{2.18}$$

responsible for the factor $\frac{1}{n!}$.

Note that if $X(0) = 0$, the formula becomes even quicker to evaluate. In other words, for segments starting at the origin, the signature can be computed with a fast closed formula.

2.2.2 The factorization property

In this section we explain a trick that will allow to spare a lot of computational resources. Given two paths $X : [a, b] \to \mathbb{R}^d$ and $Y : [b, c] \to \mathbb{R}^d$, their *concatenation* is the map $X * Y : [a, c] \to \mathbb{R}^d$ defined as:

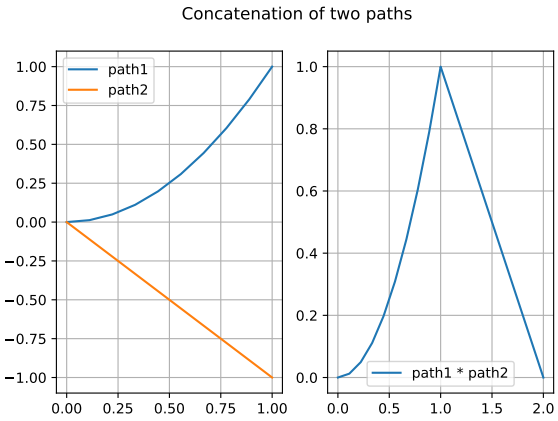
$$(X * Y)(t) \doteq \begin{cases} X(t) & \text{if } t \in [a, b] \\ X(b) + Y(t) - Y(b) & \text{if } t \in [b, c] \end{cases} \tag{2.19}$$

This notion interacts extremely well with the signature, because it commutes with algebraic tensor products:

Theorem 2.2.1 (Chen's identity)**.** *Let $X : [a, b] \to \mathbb{R}^d$ and $Y : [b, c] \to \mathbb{R}^d$ be two paths of bounded variation. Then:*

$$S(X * Y) = S(X) \otimes S(Y) \tag{2.20}$$

Proof. This is theorem 2, page 14, from [CK16]. □



This formula is extremely important and at the core of the algorithm used for computing signatures in practice. The message is the following. To compute the signature of a generic curve, it is enough to split it into simpler pieces, compute the signature of them, and finally take their tensor product. This last step can be compared to the classic product between polynomials, therefore implemented without particular obstacles. The only missing ingredient is to understand a good choice for these "simpler pieces" in which to split the curve.

Since we are working with piecewise linear paths, the answer is already there: they can be written as concatenations of *segments* between every two nodes. And as shown in the section before, the signature computation for segments is straightforward, and even easier when we are on $[0, 1]$ starting on the origin. Therefore, in the next sections we discuss the conversion of paths into this setting despite not strictly required.

2.2.3 Working on the canonical interval

The signature displays an interesting property concerning the time domain, due to the change-of-variable-formula when performing integration.

Remark (Invariance for time reparametrization). Let $X : [a, b] \to \mathbb{R}^d$ be a map of bounded variation. Let $\Psi : [0, 1] \to [a, b]$ be the time rescaling:

$$\Psi(t) = a + t(b - a) \tag{2.21}$$

and $\hat{X} = X \circ \Psi : [0, 1] \to \mathbb{R}^d$, then we have:

$$S(X) = S(\hat{X}) \tag{2.22}$$

We have already proved this result in the sections before. This equation has a fundamental deep consequence for our applications, since it allows to convert every setting to the $[0, 1]$ case, easier to deal with from a computational and notational viewpoint.

We study now in details how to use this transformation.

Let $X : [a, b] \to \mathbb{R}^d$ be a piecewise linear path, and assume it can be characterized by its values at N equidistant nodes (it will be the case in any simulation here provided). In other words, let $h = \frac{b-a}{N}$ and define all *nodes* as the values $\{a, \dots, a + nh, \dots, b\}$ for $n \in \{0, \dots, N\}$. On these $N + 1$ points, the values of X are assumed to be given and stored into a dataset array that we call $x \doteq (x_0, \dots, x_N)$, where $x_n = X(a + nh)$.

On the remaining time points t, the path is understood to be linearly constructed: assuming without loss of generality that $t \in [a + nh, a + (n + 1)h]$ for a fixed n, then:

$$X(t) \doteq X(a + nh) + \frac{t - a - nh}{h}[X(a + (n + 1)h) - X(a + nh)] \tag{2.23}$$

We are interested in computing $S_{[a,b]}(X)$ using the equivalence on $[0, 1]$ with the path $\hat{X} = X \circ \Phi$; therefore the question: how does X change after the composition with Ψ? Which modifications must be done on the dataset array x?

To understand the structure of $\hat{X}$, let's start by dividing $[0, 1]$ in N nodes, $\{0, \dots, nk, \dots, 1\}$ with $k = \frac{1}{N}$ and $n \in \{0, \dots, 1\}$. On these new nodes we have:

$$\hat{X}(nk) = X(\Psi(nk)) = X(a + (b - a)nk) = X(a + nh) \tag{2.24}$$

because $h = \frac{b-a}{N}$. Consequently if $\hat{x} = (\hat{X}(0), \dots, \hat{X}(nk), \dots, \hat{X}(1))$ is the array of the node evaluations for $\hat{X}$, we simply have: $\hat{x} = x$.

Let's investigate the time values in between. For a generic $s \in [nk, (n+1)k]$, note that $\Psi(s) \in [a + nh, a + (n + 1)h]$. Using the interpolation formula for the original X, we have:

$$\begin{aligned} \hat{X}(s) &= X(\Psi(s)) \\ &= X(a + nh) + \frac{\Psi(s) - a - nh}{h}[X(a + h(n + 1)) - X(a + nh)] \\ &= \hat{X}(nk) + \frac{s - nk}{k}[(\hat{X}((n + 1)k) - \hat{X}(nk)] \end{aligned} \tag{2.25}$$

after substituting $\Psi(s) = a + (b - a)s$. But this is precisely the linear interpolation formula on $\hat{X}$. That's very nice and practical! In other words, $\hat{X}$ is a piecewise linear function too, characterized by the N equidistant nodes in $[0, 1]$, on which it takes the same values the original X had on its nodes. Let's sum up the results.

Proposition 2.2.1.1 (Rescaling on the unit interval)**.** *Let $X : [a, b] \to \mathbb{R}^d$ be a piecewise linear map characterized by N equidistant nodes on which it takes values stored in the array $x = (x_0, \dots, x_N)$. Let Ψ be the time rescaling $\Psi : [0, 1] \to [a, b]$, $s \mapsto a + (b - a)s$.*

Then, the composition $\hat{X} = X \circ \Psi$ is a piecewise linear map, characterized by the N equidistant nodes on the unit interval $[0, 1]$, and on which it takes exactly the same values stored in x.

This proposition directly implies the following:

Corollary 2.2.1.1. *With the notation above, both the path X and $\hat{X}$ have the same 1-variation.*

Proof. Both paths are piecewise linear maps, therefore their 1-variation is given by evaluating the differences at the nodes, where their values, as seen, coincide. □

This is important to keep in mind and has a very intuitive interpretation: changing the speed at which we traverse the curve, as well as translating it, do not change its length.

2.2.4 All paths can start from the origin

If $X(0) \in \mathbb{R}^d$ is the starting value a the path under analysis, assumed to run on $[0, 1]$, then $Y : [0, 1] \to \mathbb{R}^d$ defined as $Y(t) = X(t) - X(0)$ is the translated path starting from 0. A direct consequence of derivatives into the signature definition is that:

$$S(X) = S(Y) \tag{2.26}$$

In other words we can always assume without loss of generality to have paths staring from the origin. This condition will facilitate some mathematical remarks and data normalization, especially when combined with the time reparametrization showed above.

2.2.5 Recipe for the signature computation

We can now state a general strategy to compute the signature of a piecewise linear path:

1. split it into its segments;
2. transpose every segment on $[0, 1]$ and starting in the origin;
3. precisely compute the signature of every segment;
4. perform the tensor product between all these results.

The next section is about the particular case of one-dimensional paths, followed then by computational remarks in the Python programming language.

2.2.6 Studying a one dimensional case

Let $X : [0, 1] \to \mathbb{R}$ be a one dimensional curve starting from zero. Call $\lambda = X(1)$ its final point. Since $d = 1$, for each level $n \in \mathbb{N}$ the set $P(1, n)$ contains multi-indexes of "1" repeated n times, $I_n = (1, \dots, 1)$. Since the curve X has only

a single component, in the signature we iteratively integrate such a component against itself multiple times.

An explicit computation similar for what done with the linear segment gives:

$$s_{I_n} = \frac{\lambda^n}{n!} \tag{2.27}$$

meaning that:

$$S(X) = 1 + \sum_{n=1}^{\infty} \frac{\lambda^n}{n!} e_{I_n} = 1 + \sum_{n=1}^{\infty} \frac{\lambda^n}{n!} e_1^{\otimes n} \tag{2.28}$$

In other words, for a one dimensional paths the signature only depends on the final point, giving us not so much information. We will later explain that it is essential to work at least in dimension 2 in order to have a set of useful properties.

If the original path $X(t)$ has values in $\mathbb{R}$, we can instead consider the *augmented* path $t \mapsto (t, X(t))$ taking values in $\mathbb{R}^2$. From a machine learning viewpoint, the extension $t \mapsto (t, X(t))$ can be interpreted as storing data values *and* their timestamps. From now on we will work with augmented paths if not stated otherwise.

2.2.7 The *Signatory* library

When using the Python programming language, it is possible to compute the signature for a curve on $[0, 1]$ in a pretty straightforward way, since (an algorithm similar to) the recipe explained before is essentially already implemented. This is thanks to the library *Signatory*, supported by Python3 and using PyTorch-type of data. Its core is written in C++ and offers overall quite good CPU and GPU performances (at least, in the cases we studied).

The main Signatory routine takes in input an array, say of length N, representing the values of the curve at its N equidistant nodes on $[0, 1]$ (of course, for a curve in $\mathbb{R}^d$, that array contains $d-$dimensional values). This choice is by construction, and at a first sight might seem restrictive. On the other hand, as explained in the paragraph above, the assumption of working on the unitary interval can be done without loss of generality.

Let's write a concrete example. Suppose to have a piecewise linear curve $X : [5, 7] \to \mathbb{R}^2$ composed by $N = 5$ nodes.

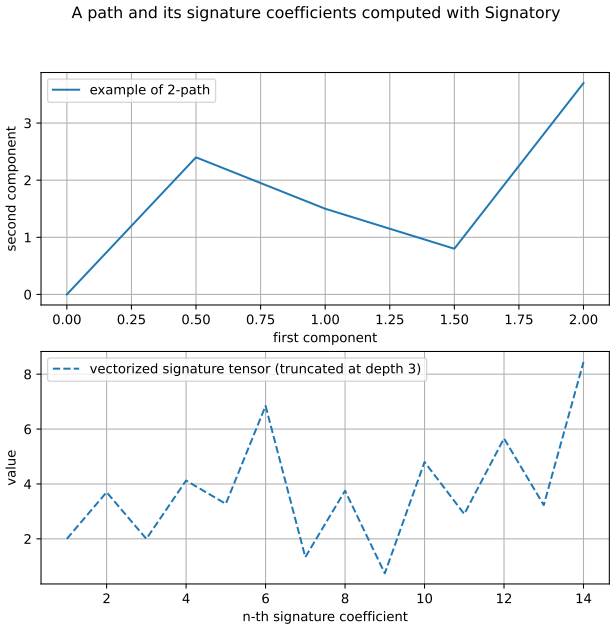
Its first component, being simply the time t as explained in the paragraph above, mirrors the discretization of $[5, 7]$ with N points:

$$X^1 = (5.0, 5.5, 6.0, 6.5, 7.0) \tag{2.29}$$

Regarding the second component, for this example we freely choose:

$$X^2 = (-1.9, 0.5, -0.4, -1.1, 1.8) \tag{2.30}$$

Let's start now the signature computation.



As a first operation, we center the curve on the origin by subtracting the value $(5.0, -1.9)$; call the new curve $Y = (Y^1, Y^2)$. Then, since Signatory only supports curves on $[0, 1]$, we need to rescale Y by using the trick explained in the section before. According to the previous section, these same arrays (Y^1, Y^2) can be interpreted as the node evaluations of the rescaled curve on the time interval $[0, 1]$. Therefore, Y has component:

$$Y^1 = (0, 0.2, 0.4, 0.6, 0.8, 1.0) \tag{2.31}$$

and

$$Y^2 = (0, 2.4, 1.5, 0.8, 3.7) \tag{2.32}$$

This data from Y are ready to be sent to the Signatory main routine.

To conclude the operation, we only need to specify the *depth* at which we want to truncate the (infinite) signature tensor. We choose e.g. to stop at level 3, which should produce an array of length $1 + 2^1 + 2^2 + 2^3 = 15$ since the curve is $2-$dimensional. On the other hand, Signatory omits the first signature coefficient (being always 1 by construction), and therefore the actual output length is 14.

We attach a simple plot illustrating either the curve Y, and the values of its 14 signature coefficients. There is nothing in particular to comment, rather it is an exercise to acquire some degree of familiarity.

2.2.8 A small and complete numerical example

In this section we take a simple smooth curve on which we can compute by hands the first terms of the signature. We compare this exact values with numerical approximations so to validate the whole approach described until now. Comments on the signature bounds are included.

The curve $X : [0, 5] \to \mathbb{R}^2$ is taken from [CK16], and defined as:

$$X(t) \mapsto (3 + t, (3 + t)^2) \tag{2.33}$$

For the sake of computing everything explicitly, we also consider the bounded variation norm of X since it will directly provide bounds for the signature coefficients. We know that $[X] \leq [X^1] + [X^2]$ where X^1 and X^2 are two components. Being differentiable, we directly have $[X^1] = \int_0^5 |\dot{X}^1(t)|dt = 5$ and $[X^2] = 55$, leading to $[X] \leq 60$.

If we only look at the first two signature levels, we expect then $\|S_1\| \leq [X] \leq 60$ and $\|S_2\| \leq \frac{[X]^2}{2} = 1800$. These two levels are actually computable by hands by solving the corresponding integrals. We report the results directly:

$$\begin{aligned} S(X)_1 &= 5 \\ S(X)_2 &= 55 \\ S(X)_{1,1} &= \frac{25}{2} \\ S(X)_{1,2} &= \frac{475}{3} \\ S(X)_{2,1} &= \frac{350}{3} \\ S(X)_{2,2} &= \frac{3025}{2} \end{aligned} \tag{2.34}$$

Note how $\|S_1\| = \sqrt{5^2 + 55^2} \approx 56 \leq 60$ and $\|S_2\| \approx 1500 \leq 1800$, as predicted.

Proceeding now towards numerical implementations, we reparametrize the curve into $[0, 1]$ by the composition with $t \mapsto 5t$, subtract the pair $(3, 9)$ so to have a curve starting from zero, $(0, 0)$. On that resulting path (whose signature, we remark again, is theoretically proven to be the same as the original), we numerically compute the truncated signature (now up to level 2) as follows.

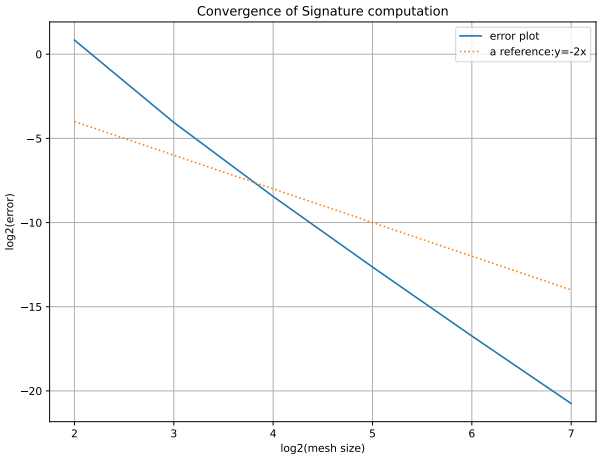
We construct a sequence of piecewise approximating functions $\{X_n\}$, with meshsize halved every time that $n \to n + 1$.

On these piecewise paths, the Signatory library is used to compute the signature until level 2. These values are finally compared with the known true one.

We know that the convergence must hold, since for $n \to \infty$ we have:

$$S^2(X_n) \to S^2(X) \tag{2.35}$$

Figure 2.1: Numerical convergence of the signature for an exact known case.



as proved before, despite we cannot predict the rate.

The logarithmic plot displaying the meshsize against the resulting absolute error is attached here, suggesting, for reference, a convergence rate better than $\frac{1}{\sqrt{n}}$ (n being here the number of points chosen when discretizing the original smooth curve).

We conclude this section with a further remark on controlling the signature in this specific example. If we consider the original curve and numerically compute the coefficients until a certain level, for instance 5, we would notice very big values that will (eventually) start decaying if we compute even higher levels (here not plotted).

On the other hand, if we rescale the curve by dividing it with the scalar $\lambda = 60$ (which, as seen, is an upper bound for its 1-variation), we obtain $Y \doteq \frac{X}{60}$ a curve with 1-variation less than 1. The signature coefficients of Y will start decaying already from the first level.

These considerations are readable in the second attached plot. The numerical norm of the computed signature of Y results to be the number 1.1, which is lower than the theoretical upper bound $\|S\| \leq \exp([Y]) \leq \exp(1) \approx 2.7$. Note that for the original X, the computed signature norm reaches the value of $4.30e + 06$, but is also inferior than the (huge) theoretical bound of e^{60}. On the other hand, dividing by a "too big" scalar can easily produce very small coefficients that can be very hard to handle from a numerical viewpoint. We hope that these comments would help in understanding the relevance of even simple operations like the multiplication of a path by a scalar value.

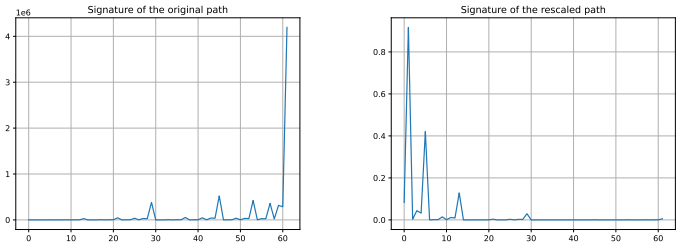


Figure 2.2: The signature coefficients of the original (left) and rescaled (right) curves, up to level 5. Observe the difference in the coefficients decay.

2.2.9 Monte Carlo and other integration strategies

The previous sections explained the use of *algebraic strategies* to compute the signature of a piecewise curve by using tensor products and concatenations. This is essentially (one of) the idea beyond the Python *Signatory* library and other tools available online.

On the other hand, from a conceptual viewpoint, the problem of estimating the signature is essentially a high dimensional integration over the simplex. Therefore numerical quadrature algorithms can be used, but they are usually subjected by the curse of dimensionality especially when computing higher signature terms.

That said, for the sake of providing a form of double verification, we tried to compute signatures using a simple Monte Carlo integration, without any kind of particular optimization. It worked well for the lower coefficients and allowed to confirm the results given by signatory. We remark again how this was done as a form of benchmark and to check our theoretical understanding.

On the other hand, there was no comparison between our naive Monte Carlo method and the speed provided by Signatory, therefore we preferred to remain with Signatory and PyTorch for the rest of the work.

For more information on this library and a general overview about computing the signature, we suggest the paper [KL20], where computational costs are also explained as well as remarks and comparisons between various past strategies.

Chapter 3

The signature transform: key properties

3.1 Four properties of relevance

In this section we study four properties of the signature transform, selected in a way to be used for our later applications. Resources as [Lyo07] offer a more complete overview.

Let BV^d be the Banach space of continuous paths $X : [0, 1] \to \mathbb{R}^d$ of bounded variation and BV_0^d its subspace of paths starting from zero.

Let $E \doteq \mathbb{R}^d$, and $T(E)$ be the Hilbert space of elements in $T^\infty(E) = \bigoplus_{n\in\mathbb{N}} E^{\otimes n}$ of finite norm. Let's consider the map:

$$S : BV^d \to T(E) \tag{3.1}$$

associating to every path of bounded variation its signature transform.

We have already carefully explained how it works and why it is well defined. We now investigate its range, injectivity and some additional properties concerning dimensionality reduction and functional approximation.

3.1.1 Comments about surjectivity

The signature transform is **not** surjective, since the coefficients are not independent to each other. They are subjected to mutual relationships:

Proposition 3.1.0.1 (Shuffle product identity)**.** *Let $X \in BV^d$. For any couple of multi-indexes $I = (i_1, \dots, i_k)$ and $J = (j_1, \dots, j_m)$, with each index $i_1, \dots, i_k, j_1, \dots, j_m \in \{1, \dots, d\}$, there exists a set of multi-indexes $\mathcal{K}$ such that:*

$$s_I(X)s_J(X) = \sum_{K\in\mathcal{K}} s_K(X) \tag{3.2}$$

Each multi-index in $\mathcal{K}$ has length $k + m$.

Proof. Please refer to [CK16], page 12, theorem 1, where a more complete formulation of the theorem is also available. □

The set $\mathcal{K}$ is actually computable and very precise, but here omitted since our goal is to point out some implications of this theorem.

First, we learn that we can transform any *product* between signature coefficients, into *sum* of coefficients of higher order (in particular it holds for any polynomial evaluation). This interpretation will be important later.

Second, the signature coefficients are revealed to be mutually connected. Therefore if the values in a tensor $g \in T(E)$ does not respect the identity above, there is no path whose signature corresponds to g, disproving surjectivity. Nevertheless, one can try to answer the following question: given a tensor in $h \in T(E)$ with *compatible* numbers that satisfy the identity given in the theorem, can we find a path whose signature corresponds to h? This is called the "signature inversion" problem and is, for instance, discussed in [Gen15]. Experiments in this direction are also provided in the last chapter of this thesis.

3.1.2 Injectivity and uniqueness

In general, two different paths *can* have the same signature. We are aware of this property. For any path X, the addition of a constant c gives $S(X) = S(X + c)$. To ensure uniqueness, a first step is then to establish a common starting point, say zero, to eliminate the possibility of translations. But can two paths starting in 0 still have the same signature? The answer is positive again, using for instance a time reparametrization of the interval. We also carefully discussed in details this property, but in case the reference [CK16], page 11, section 1.3.1 can offer further remarks. In case of necessity, it's fairly easy to circumvent these issues by *augmenting* the path:

Definition 3.1.1 (Path augmentation)**.** Let $X : [0, 1] \to \mathbb{R}^d$ such that $X \in BV_0^d$. It's *augmentation* is the path $\hat{X} : [0, 1] \to \mathbb{R}^{d+1}$ defined as:

$$\hat{X}(t) \doteq (t, X(t)) \doteq (t, X^1(t), \dots, X^d(t)) \tag{3.3}$$

We have $\hat{X} \in BV_0^{d+1}$.

Augmenting a path simply means to add timestamps as first coordinate. This is the same operation introduced before when discussing an example for the signature computation in dimension one. Every augmented path has dimension at least 2, still starts from zero and is of bounded variation ($[\hat{X}] \leq 1 + [X]$).

Proposition 3.1.0.2 (Augmentation embedding)**.** *Let $\iota : BV_0^d \to BV_0^{d+1}$ be the map $X(t) \mapsto (t, X(t))$. Then it is injective and continuous.*

Proof. Suppose $X \to Y$ in BV_0^d. Then:

$$\begin{aligned} \|\iota(X) - \iota(Y)\|_{bv} = \|(t, X(t)) - (t, Y(t))\|_{bv} = \\ \|(0, X(t) - Y(t))\|_{bv} = \|(X(t) - Y(t))\|_{bv} \to 0 \end{aligned} \tag{3.4}$$

□

We will usually indicate with $\hat{X}$ the image $\iota(X)$. Augmenting a path is a way to ensure uniqueness:

Theorem 3.1.1 (Signature uniqueness)**.** *Let X, $Y \in BV_0^d$. Let $\hat{X}$ and $\hat{Y}$ be their augmentations in BV_0^{d+1}. Then:*

$$\hat{X} \neq \hat{Y} \implies S(\hat{X}) \neq S(\hat{Y}) \tag{3.5}$$

Proof. We refer to [Fer20], proposition 1, page 5. From a intuitive viewpoint, adding the first coordinate t disables the invariance under time reparametrization since changes in time are now "tracked". □

Augmenting a path ensures the injectivity of the signature transform, very useful especially when there is interest in discriminating paths belonging to different "classes" (typical in the machine learning context). It is also acceptable from a practical viewpoint, since adding the time coordinate can be interpreted as writing "timestamps", something mandatory anyway when processing time series datasets. Finally, rescaling them in $[0, 1]$ and translating to the origin not only give us the mathematical foundation for uniqueness, but also encourage a better processing of data.

3.1.3 Dimensionality reduction

Suppose to have two different augmented paths, the first linearly approximated with N points, the second with $M > N$. This is something that in practice can happen, for instance when measurements are done in two different situations.

Normally, it would be hard to compare them due to the different number of nodes. On the other hand, when computing their signatures the resulting arrays (obtained by linearizing the tensors) always have the same length!

The computed signature for the second path will be more accurate, but this might of secondary importance in settings where the priority is to have a quick rough comparison of multiple data.

This reasoning does not stop here. Let's suppose to work with high-frequency data, for instance 2-dimensional augmented paths with millions of nodes. Managing them can be challenging or even not suitable for concrete algorithm implementations, due to computational constraints.

On the other hand, computing the first few levels of the signature would convert each time series into arrays of length around sixty, offering a huge gain with respect to manageability.

In conclusion, the signature transform presents itself also as a tool to encourage comparison of data and reduce dimensionality.

3.1.4 Functional linearization

This section represents a great occasion to introduce the problem of *functional regression*. We start with the classic setting of ordinary regression.

Definition 3.1.2 (Dataset)**.** A $d-$dimensional *regression dataset* is a finite collection of ordered couples $\{(\hat{x}_1, y_1), \dots, (\hat{x}_K, y_K)\}$ where $\hat{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$.

The classic regression problem consists in:

- fixing a class of functions, $\mathcal{F}$, from $\mathbb{R}^d$ to $\mathbb{R}$;
- finding the "best" $f \in \mathcal{F}$ such that $f(\hat{x}_i) \approx y_i$

The function f is then used on *new* data to make predictions.

Our approach is a little different. We interpret each row $\hat{x}_i \in \mathbb{R}^d$ as a collection of the d node values of an hypothetical piecewise linear function X_i (uniformly discretized on $[0, 1]$ with d equidistant nodes). Our dataset is therefore seen as a finite set of pairs "(function, value)" rather than "(arrays, value)".

We also need to set the first value of $\hat{x}_i$ to be 0. We can do it for instance by translation, but also to artificially add a zero is legit. If not differently specified, translations will be used in our applications.

Note how having a finite number of K points (functions X_i) the dataset D can be interpreted as a compact set of continuous functions of bounded variation starting from zero.

We are therefore looking for some optimal (continuous) functional $F : D \to \mathbb{R}$ such that $F(X_i) \approx y_i$ for every i.

The following theorem manages to strongly simplify the research:

Theorem 3.1.2 (The linearization theorem)**.** *Let $D \subset BV_0^d$ be a compact set of paths of bounded variation starting in zero. For each $X \in D$, denote with $\hat{X}$ the augmented path $t \mapsto (t, X(t))$. Let $F : D \to \mathbb{R}$ be a continuous functional. Then, for every $\epsilon > 0$, there exists a natural $N \in \mathbb{N}$ and $l \in T^N(\mathbb{R}^d)$ such that:*

$$|F(X) - (l, S^N(\hat{X}))_{T^N(\mathbb{R}^d)}| < \epsilon \tag{3.6}$$

Sketch of the proof. We refer to [Fer20], page 5 and [LLN13], theorem 3.1, for the actual proof and sketch here the key ideas.

Let $F : D \to \mathbb{R}$ be a continuous functional. Since the augment embedding $\iota : BV_0^d \to BV_0^{d+1}$ is continuous, a continuous map $\hat{F}$ is induced on $\iota(D)$ by defining $\hat{F}(\iota(X)) \doteq F(X)$. The set $\iota(D)$ is still compact since image of a continuous function. Similarly, since the map $S : BV_0^{d+1} \to T(E)$ is continuous, we obtain a continuous functional: $S_F : S(\iota(D)) \to \mathbb{R}$ simply by $S_F(S(\iota(X))) \doteq \hat{F}(\iota(X)) = F(X)$. The domain on this functional is still a compact set.

At this point it is possible to show that there are algebraic conditions that make possible to use the Weierstrass approximation theorem, so that S_F can be approximated with a *polynomial* on the signature coefficients.

But since every polynomial on the signature coefficients can be rewritten as a linear sum (Proposition 3.1.0.1), the reasoning is concluded. □

Note that since the first element of the signature is always the real number 1 (because of the time augmentation t), the scalar product $(l, S^N(\hat{X}))$ can be interpreted as a "linear model with bias" or simply "linear model" in the machine learning usual terminology.

In conclusion, the main theorem in this section is explaining that machine learning *nonlinear* regression problems can be approximated with *linear* one, provided working on the *signature of data* instead of their original form.

This is definitely quite useful and very promising, despite some words of cautions are mandatory, since there is neither information concerning the rate of convergence nor about the needed truncation level N.

3.2 A connection to probability theory

Until now we worked on a deterministic, non random setting. In this section we extend our vision by adding observations from probability theory. More specifically, we analyze how the signature transform connects naturally to the moment problem in the case of real random variables.

3.2.1 The moments of a real random variable

Let X be a one dimensional real random variable. Integrating powers of X originates the sequence of its moments.

Definition 3.2.1 (Moment)**.** For each $n \in \mathbb{N}$, its moment of order n is the integral

$$M_n \doteq \mathbb{E}[X^n] \in \mathbb{R} \tag{3.7}$$

where by convention $M_0 = 1$.

It is possible to prove that for any random variable X, either *all* moments exist, or they exist for every order less that a number $k \geq 0$. For instance, we have $k = 0$ for the Cauchy distribution, since all its moments diverge. Sometimes we have explicit values for every n, as for instance in the case of Gaussian distributions.

The moments are connected to the characteristic function of a random variable, arising from its Taylor expansion around the origin. Since characteristic functions uniquely determine random variables, one might ask how strong moments are able to characterize probability laws.

Definition 3.2.2 (The moment problem)**.** The *moment problem* is the problem of deducing the complete probability distribution of a random variable X, knowing only its sequence of moments.

Generally speaking, the answer to the moment problem is negative. It is possible to construct two real random variables with the same moments despite having different distributions. We refer [Var01], page 22, chapter 2.2 for the construction of such an example. The informal idea is relatively easy to grasp, since it would be similar to the problem of completely characterize a complex function by only looking at its Taylor expansion around the origin.

On the other hand, this interpretation allows some positive results by using ideas coming from complex analysis. If X and Y are two random variables with

same moments of every order, then they have the same distributions providing satisfying additional conditions as (for instance) described in theorem 2.2 from [Var01]. In this case, we say that the two random variables are *compatible* with the moment problem.

Definition 3.2.3 (Compatibility)**.** Two real random variables X and Y are *compatible with the moment problem* if and only if having the same moments implies being equidistributed, in symbols $X \sim Y$.

The current topic is also very relevant in practical applications. Suppose that we have two sets of samples coming from two sources X and Y. The goal is to understand if they are likely coming from the same probability distribution. If it is true that there exists more sophisticated statistical tools (always welcome to complement our approach), to compare their moments is a good first benchmark. If the moments are "different enough", X and Y are likely not equidistributed.

3.2.2 The signature moment equivalence for the real case

Assuming a basic knowledge or real random variables, we very quickly recall the notion of stochastic process as its "dynamic" counterpart.

Definition 3.2.4 (stochastic process)**.** A collection of random variables $\{X_t\}_{t\in[0,1]}$ is called a stochastic process. If Y_t and X_t are two stochastic processes, they are equidistributed if and only if for every finite choice of $t_1 < \cdots < t_n$, the vectors $(X_{t_1}, \ldots, X_{t_n})$ and $(Y_{t_1}, \ldots, Y_{t_n})$ are equidistributed.

Let now X be a real ("static") random variable. The signature transform is an interesting tool when working with time series, therefore we bring X into this setting by defining the stochastic process $X_t \doteq tX$.

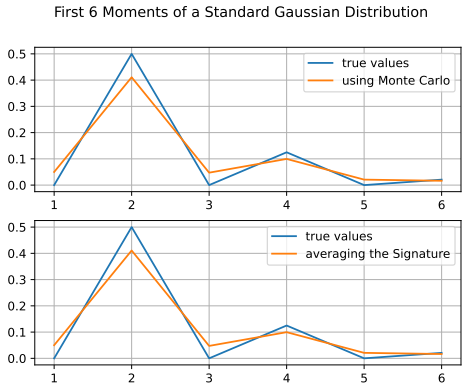
Note that for each realization $X(\omega)$, we have a path $[0,1] \to \mathbb{R}$ defined as $t \mapsto tX(\omega)$. Therefore we interpret the stochastic process X_t as a random path and call it $\tilde{X}$ with an abuse of notation.

Since each realization of $\tilde{X}$ is simply a linear segment, we can explicitly compute the signature as pointed out previously. The only multi-indices involved are of the form $(1), (1,1), \ldots$, and so on, as also commented in the past sections. Let then $n \in \mathbb{N}$ and $I_n = (1, \ldots, 1)$ a multi-index of length n composed by only 1s. The signature coefficients are:

$$S_{I_n}(\tilde{X}) = \frac{1}{n!} X^n \tag{3.8}$$

leading to:

- $\mathbb{E}[S_{I_0}(\tilde{X})] = 1 = M_0$
- $\mathbb{E}[S_{I_1}(\tilde{X})] = \frac{M_1}{1}$
- $\mathbb{E}[S_{I_2}(\tilde{X})] = \frac{M_2}{2}$
- ...



- $\mathbb{E}[S_{I_n}(\tilde{X})] = \frac{M_n}{n!}$

In other words, we verified that computing the expected signature of the random path $\tilde{X}$ is *equivalent* to computing all the moments of X.

Recalling that with the symbol $S^N(X)$ we mean the signature truncated at level N, patching all the pieces together we obtain:

Corollary 3.2.0.1 (Signature characterization for the real case)**.** *Let X and Y be two real random variables compatible with the moment problem. Let $\tilde{X}$ and $\tilde{Y}$ be the two linear paths defined as above. Then we have:*

$$X \sim Y \iff \mathbb{E}[S^N(\tilde{X})] = \mathbb{E}[S^N(\tilde{Y})] \quad \forall N \in \mathbb{N} \tag{3.9}$$

In the next section we perform a numerical validation of this principle.

3.2.3 A numerical example for the signature as moments

If X is a standard normal variable, $X \sim N(0,1)$, its moments are zero for odd parity and $\frac{2^{(-n/2)}}{(n/2)!}$ otherwise. We compute them by using two methods:

- averaging X^n samples via Monte Carlo;
- using the expected signature of the paths for $\tilde{X}$.

By comparing the numerical results with the true values, we estimate the errors and validate the theoretical understanding. We used 100 standard normal samples and the Signatory library to compute the required path signatures. The (small) confidence intervals are not represented for simplicity.

The plots highlight how both methods work flawlessly. Differently from the naive Monte Carlo averages, the signature method has the great advantage of enjoying a greater level of generalization as we explain in the upcoming sections.

3.2.4 Extending on path augmentation

In the previous chapters we insisted on the importance of "augmenting" paths by adding the time coordinate t, for reasons concerning data normalization and signature uniqueness. Therefore in order to keep a coherent style we briefly discuss its adaptation for this section.

Definition 3.2.5 (Time augmentation for a stochastic process)**.** Let $\{X_t\}_{t\in[0,1]}$ be a stochastic process. Its time augmentation is the stochastic process $\{\hat{X}_t\}_{t\in[0,1]}$ defined as $\hat{X}_t = (t, X_t)$ for each $t \in [0, 1]$.

We remark how:

Proposition 3.2.0.1 (equidistribution)**.** *Two stochastic processes $\{X_t\}$ and $\{Y_t\}$ are equidistributed if and only if their time augmentations $\{\hat{X}_t\}$ and $\{\hat{Y}_t\}$ are equidistributed.*

If X is a real random variable, call $\hat{X}$ the time augmentation of the process $\tilde{X}$ (again, we omit indices with an abuse of notation). We interpret the process as a random path whose first coordinate is deterministic and simply t, therefore as a random variable in the space of continuous functions.

The signature characterization still holds:

Corollary 3.2.0.2 (Signature characterization for the augmented real case)**.** *Let X and Y be two real random variables compatible with the moment problem. Let $\hat{X}$ and $\hat{Y}$ be their time augmentation. Then we have:*

$$X \sim Y \iff \mathbb{E}[S^N(\hat{X})] = \mathbb{E}[S^N(\hat{Y})] \quad \forall N \in \mathbb{N} \tag{3.10}$$

Proof. Suppose $X \sim Y$. Then we have $\hat{X} \sim \hat{Y}$. For each integer N, the random variables $S^N(\hat{X})$, $S^N(\hat{Y})$ are results of the same measurable transform (finite collections of integrals) and therefore equidistributed. In particular, they have the same expectation.

Conversely, if $\mathbb{E}[S^N(\hat{X})] = \mathbb{E}[S^N(\hat{Y})]$ for each truncation level N, then in particular the signature coefficients corresponding to the moments of the original X and Y are the same for each N and $X \sim Y$ since they are assumed to be compatible. □

There is a reason for which we rephrased this theorem using the time augmentation notation, despite being a mathematically straightforward technique. Indeed, the formulation here presented still works with the same notation in a much more general case, as explained in the next paragraph.

3.2.5 Expected signature for stochastic processes

We discussed the signature characterization for random variables X with values in $\mathbb{R}$, therefore for point random data.

If, more generally, we have a random variable X with values in $\mathbb{R}^d$, we can interpret these d numbers as the d node values for a corresponding piecewise

linear path. In other words, we read the multidimensional random variable X as a random path in BV^1, which can then be translated and augmented in BV_0^2.

We pursue this direction because in this work we focus on time-series data (rather than general statistic theory for multidimensional variables), and because in this setting an important theorem can be stated.

Theorem 3.2.1 (Signature characterization of stochastic processes)**.** *Let X, Y random variables in BV_0^{d-1}. Consider their augmentation $\hat{X}$, $\hat{Y}$ with realizations in BV_0^d. Assume that the complex series $\sum_{n=0}^{\infty} z^n \mathbb{E}[\|S^n(\hat{X})\|_{E^{\otimes n}}]$ and $\sum_{n=0}^{\infty} z^n \mathbb{E}[\|S^n(\hat{Y})\|_{E^{\otimes n}}]$ have an infinite radius of convergence (the tensor $S^n(\hat{X})$ being the n-th level of the signature). Then:*

$$X \sim Y \iff \mathbb{E}[S(\hat{X})] = \mathbb{E}[S(\hat{Y})] \tag{3.11}$$

Sketch of the proof. The intuitive idea is to transform the stochastic processes X_t and Y_t into *random tensors* by looking at their composition with the signature. Then, in this appropriate algebraic tensor space, one develops a generalization of probability theory and "solve" the moment problem there, proving this theorem as a corollary.

We suggest consulting theorem 2.2, page 4 of [NSW+20]. Note that for each random path $X \in BV_0^{d-1}$, its augmentation is in BV_0^d which is precisely the space "$\Omega_0^1(J, \mathbb{R}^d)$" in the referred paper. Further comments are also available at the end of page 23 of [LM22] and in [CL13]. □

This result is absolutely crucial for all our application, and can be heavily used with a numerical perspective in mind. Before moving into applications, we remark some theoretical consequences.

3.2.6 Building statistical tests

Let's go back to the case of X and Y one dimensional random variables. The theorem above gives us a criteria under which X and Y are compatible. Indeed, if the radius of convergence of $\sum_{n=0}^{\infty} \mathbb{E}[\|S_n(\hat{X})\|]$ (and similarly for Y) is finite, they can be characterized by the expected signatures.

The same idea still works if X, Y are two multidimensional random variables, when considering again the random segments (t, tX) and (t, tY).

In other words, the signature can be used to build statistical test for more classical settings and generalize the notion of moments to multidimensional random variables and even time series.

We do not explore more in this direction for reasons of time, but we suspect that connections to some already known statistical methods can be discovered.

3.2.7 The log-signature

At this point it is worth mentioning another attempt when developing the signature theory keeping numerical and machine learning applications in mind. We know that the signature transform:

- is capable of **linearizing** nonlinear path functionals;
- can **characterize** many path classes, deterministic and random;
- can become very **high dimensional** for longer truncations.

In an attempt to contrast the curse of dimensionality, another transform is available through algebraic manipulation and is called the *log-signature*.

The name comes from the logarithm, whose real Taylor expansion is generalized in a formal algebraic sense. The following holds:

- given a signature, its algebraic logarithm produces the log-signature;
- given a log-signature, its *algebraic exponential map* gives the signature back.

Therefore the signature and the log-signature conceptually contain the same amount of "information". On the one hand, the log-signature presents a first advantage since it has a much smaller dimension. The arrays produced after vectorizing the tensors are far shorter than the other obtained by the classic signature, and consequently easier to manipulate.

On the other hand, this advantage has a cost: the linearization property fails and there is no more characterization.

The interpretation is also harder to state. For instance, we know that for a one dimension random variable X, the expected signature contains the momenta. On the other hand, the expected log-signature of X would simply be the expectation of X, of very limited utility.

One can suspect a possible connection between the usual logarithm and the logarithm of the signature coefficient. This is not the case, since the logarithm involved is an abstract generalization which only loosely resembles the common operation from a formal viewpoint.

Furthermore, if for reals we have $\log[ab] = log[a] + \log[b]$, on tensor algebras is it *not* true that $\log[a \otimes b] = \log[a] + \log[b]$, making some attempts we did to connect this topic to others, not working (for instance, we tried to involve the cumulants as logarithm expansions in the characteristic functions).

For more information on the topic we suggest the paper [CK16], where in particular equations (1.60), (1.54) and (1.58) can be used as examples to prove the just mentioned non-distributivity of the algebraic logarithm.

Part II

Applications

Chapter 4

Clustering and visualization

4.1 Stochastic macro-clustering

We start the second part of this work with a familiar task typical in machine learning: dataset clustering. We develop a technique suitable when data are in form of time series, interpreted as samples coming from stochastic processes.

We introduce the notion of "macro-clustering", briefly defined as the idea of grouping and visualizing the members of a *family* of processes. This is in contrast to our next section ("micro-clustering"), where with similar techniques we study paths coming from a *single* fixed process.

We study numerical strategies that finally allow a visualization of data on an easy-to-read two dimensional plane. The key idea is to use the expected signature as a way to measure how different two processes are. Since we need to estimate averages, a quick recap on confidence intervals is included. Pairwise differences originate a *distance matrix*, which can be visualized on a two dimensional plane thanks to the "Multidimensional Scaling Algorithm" available in the machine learning literature. To measure how these expectations are "spread" helps in understanding how rich the family of processes is. This leads to the notion of *macro-variance*, here measured in a percentage form, so to support quick interpretations of the results.

Many numerical experiments follow and we take the opportunity to recap the notion of *Geometric Brownian Motion*, pillar example in the context of mathematical finance.

4.1.1 A recap on confidence intervals

Since we often use estimations of expected quantities (averages), we include a recap on confidence intervals. The reference is [KPS94], section 1.5.

4.1.2 Confidence intervals for Gaussian samples

Let $\{X_i\}_{i=1,\dots,n}$ be a collection of n samples coming from a *Gaussian* one dimensional random variable X, with mean μ and variance σ^2. Let $\hat{\mu} \doteq \frac{1}{n}\sum_{i=1}^{n} X_i$ be the empirical mean, and $\hat{\sigma}^2 \doteq \frac{1}{n-1}\sum_{j=1}^{n}(X_j - \hat{\mu})^2$ the (unbiased) empirical variance. By using the Student's t-distribution it is possible to construct 99% *confidence intervals*, i.e. values $a = a(n)$ such that:

$$P[\mu \in (\hat{\mu} - a, \hat{\mu} + a)] > 0.99 \tag{4.1}$$

making possible to guess where the true value of μ lies only on the base of observed data. For the case of $n = 100$, the variable a turns out to be computed as $a = 0.262\hat{\sigma}$, leading to the following quick and practical result:

Proposition 4.1.0.1. *Let X_i be a set of* 100 *Gaussian samples from $N(\mu, \sigma)$. Then the true mean μ belongs to the interval $(\hat{\mu} - 0.262\hat{\sigma}, \hat{\mu} + 0.262\hat{\sigma})$ with a probability higher that* 99%.

It is of course possible to use different number of samples. The quantity a is generally proportional to $\frac{\hat{\sigma}}{\sqrt{n}}$ and commonly referred as the Monte Carlo error. We refer to the table at page 33 in [KPS94] for further information.

4.1.3 Confidence intervals for general samples

Let $\{Y_j\}_{j=1,\dots,N}$ a set of identically distributed samples from the one dimensional random variable Y with mean μ and variance σ^2. We are again interested in estimating the value of μ, but are unable to use the theorem above since the samples are not assumed to follow a Gaussian distribution.

We then rely on the Central Limit Theorem. Assume to have N big enough so to be written as $N = 100b$, with $b > 20$. Divide then the dataset into 100 *batches*, each containing b samples. Consider then the new dataset of 100 samples $\{Z_i\}_{i=1,\dots,100}$, where $Z_i \doteq \frac{1}{b}\sum_{k=(i-1)b+1}^{ib} Y_k$ for each $i = 1, \dots, 100$.

In other words, we are splitting the original dataset into subsets and collecting the averages of them. Thanks to the Central Limit Theorem, the new dataset $\{Z_i\}$ of 100 samples is now (approximately) distributed as a Gaussian variable with mean μ (same as the original data!) and (new) variance $\eta^2 = \frac{\sigma^2}{b}$.

If $\hat{\mu}$ and $\hat{\eta}^2$ are the empirical mean and variance of the dataset Z_i, thanks to the theorem in the previous section we have:

$$P[\mu \in (\hat{\mu} - 0.262\hat{\eta}, \hat{\mu} + 0.262\hat{\eta})] > 0.99 \tag{4.2}$$

We performed a simple numerical experiment by using 2000 samples uniformly distributed in $(0, 1)$. They have been divided into 100 batches of length 20 and the overall mean estimated as 0.50 ± 0.02. Attached there are two histogram plots. The first refers to the original dataset Y_i, while the second to the transformed Z_i. Despite being basic results, is it good to clarify and visualize them as well.

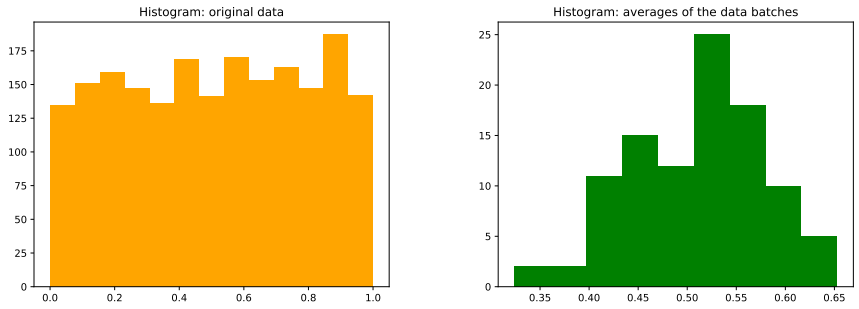


Figure 4.1: The initial set of 2000 samples follows an uniform distribution. We divided it into 100 batches containing each 20 samples, considering then the averages values of each batch. The resulting dataset of 100 points follows approximately a Normal distribution, on which we can apply the confidence interval estimation.

4.1.4 Confidence intervals on higher dimensions

We extend the confidence interval estimation to the multidimensional case. Many methods are available in the literature. We rather develop a naive and simple one, with focus on immediate applicability and interpretability.

Let X be a random variable in $\mathbb{R}^d$ and $\mu \in \mathbb{R}^d$ its mean. Assume to have n samples of X. By acting independently on every (one-dimensional) coordinate, we obtain confidence intervals for each single coordinate by using the algorithm previously described.

In other words, if $\hat{\mu} = (\hat{\mu}_1, \dots, \hat{\mu}_d)$ and $\hat{\sigma} = (\hat{\sigma}_1, \dots, \hat{\sigma}_d)$ are the empirical means and standard deviations of each coordinate, then the confidence interval rule holds on every component:

$$P(\mu_i \in [\hat{\mu}_i - c_i, \hat{\mu}_i + c_i]) \geq 0.99 \quad \forall i \in \{1, \dots, d\} \tag{4.3}$$

where c_i depends on $\hat{\sigma}_i^2$ as described before. For instance, in the Gaussian case with $n = 100$, we have $c_i = 0.262\hat{\sigma}_i^2$. For the non-Gaussian case, we need to divide the dataset in batches and operate ad mentioned in the section above. Independent on the case under analysis, we always end up with appropriate values c_i. Let now $c \doteq \max_i c_i$.

Proposition 4.1.0.2 (The confidence ball)**.** *Given the following sets:*

1. $A_1 \doteq \{y \in \mathbb{R}^d | y_i \in [\hat{\mu}_i - c_i, \hat{\mu}_i + c_i], \forall i \in \{1, \dots, d\}\}$
2. $A_2 \doteq \{y \in \mathbb{R}^d | y_i \in [\hat{\mu}_i - c, \hat{\mu}_i + c], \forall i \in \{1, \dots, d\}\}$
3. $A_3 \doteq \{y \in \mathbb{R}^d | \|\hat{\mu} - y\| \leq c\sqrt{d}\}$

the following inclusions hold:

$$A_1 \subseteq A_2 \subseteq A_3 \tag{4.4}$$

Furthermore:

$$P[\mu \in A_3] \geq 0.99 \tag{4.5}$$

Proof. By definition c is the biggest of the (positive) values c_i, therefore the inclusion $A_1 \subseteq A_2$ is immediate.

Let now $y \in A_2$.

If $y_i - \hat{\mu}_i$ is positive, from $y_i \leq \hat{\mu}_i + c$ we obtain $(y_i - \hat{\mu}_i)^2 \leq c^2$ (since c is always positive).

If $y_i - \hat{\mu}_i$ is negative, from $y_i \geq \hat{\mu}_i - c$ we get $y_i - \hat{\mu}_i \geq -c$ and therefore $\hat{\mu}_i - y_i \leq c$ (both positive quantities now), giving $(y_i - \hat{\mu}_i)^2 \leq c^2$ again.

The fact that $y \in A_3$ is proved by inserting that inequality into the following computation:

$$\|\hat{\mu} - y\| = \sqrt{\sum_{i=1}^{d} (\hat{\mu}_i - y_i)^2} \leq \sqrt{\sum_{i=1}^{d} c^2} \leq c\sqrt{d} \tag{4.6}$$

Finally, the set-theoretic inclusions $A_3 \supseteq A_2 \supseteq A_1$ implies:

$$P[\mu \in A_3] \geq P[\mu \in A_2] \geq P[\mu \in A_1] \geq 0.99 \tag{4.7}$$

where the last step comes directly by construction of A_1 and c, concluding the proof. □

The set A_3 is usually indicated with $B_{\hat{\mu}}(c\sqrt{d})$, the ball in $\mathbb{R}^d$ centered in $\hat{\mu}$ with radius $c\sqrt{d}$. In other words we proved the following:

Proposition 4.1.0.3 (The confidence ball)**.** *Let X be a random variable in $\mathbb{R}^d$ with mean μ. Let X_i be n samples from X used to estimate the average using the classic mean $\hat{\mu} = \frac{1}{n}\sum_i^n X_i$. Let c_i be the componentwise confidence intervals as previously discussed. If $c \doteq \max_i c_i$, then:*

$$P[\mu \in B_{\hat{\mu}}(c\sqrt{d})] \geq 0.99 \tag{4.8}$$

Note how we started from a multidimensional random variable in $\mathbb{R}^d$ and obtained a confidence radius $c\sqrt{d}$ which is a single real number that scales with the variable dimension. These properties will be very useful in the upcoming applications.

4.1.5 The signature for stochastic processes

In the previous chapters we described multiple properties of the signature transform. We briefly recap some of them, according to what relevant in this chapter.

A wide range of processes X_t on $t \in [0, 1]$ (abbreviated X) can be *characterized* by their expected signatures $\mathbb{E}[S(X)]$: if two share the same average signatures, they are equidistributed. This allows to interpret the signature of a time series as an exhaustive, ordered collection of statistics. This is similar to what happens for sequences of *moments* for the case of one dimensional random variables.

When moving to implementations, multiple limitations come into play. The infinite dimensional signature tensor must be truncated, expectations are replaced with averages on finite samples, and continuous processes are simulated on discrete times. On the other hand, during the previous chapters we learned some tricks that help in preparing the ground. For instance, to work with the augmented process $(t, X(t))$ instead of the original one, to normalize the paths so to start by the origin, and even divide X by some constant so to better control the signature coefficients' decay.

Let's now consider a family of k stochastic processes $X_1, X_2, \dots, X_k$, denote by $\hat{X}_1, \dots, \hat{X}_k$ their time augmentations, $\hat{X}_i(t) = (t, X_i(t))$ for $t \in [0, 1]$. The $k \times k$ matrix of pairwise differences:

$$D_{ij} \doteq \|\mathbb{E}[S(\hat{X}_i) - S(\hat{X}_j)]\|_{T(\mathbb{R}^d)} \tag{4.9}$$

has entries such that $D_{ij} = 0$ if and only if the processes X_i and X_j are equidistributed. We numerically approximate this matrix. For each $i \in \{1, \dots, k\}$:

1. fix a time mesh h for the interval $[0, 1]$;
2. sample N (discretized) path distributed as $\hat{X}_i$, with time mesh h;
3. using the Signatory library, compute the signature of each of the N sampled path. Each computation gives an array of a certain length q. For instance, by stopping at depth 5 for a process $\hat{X}_i \in \mathbb{R}^2$, we get N arrays of length $q = 62 = 2^1 + \dots 2^5$;
4. estimate their average $E_i \in \mathbb{R}^d$ and corresponding interval ball $C_i \in \mathbb{R}$ (as explained in the sections above).

In conclusion, to each process X_i we associate a couple (E_i, C_i) where $E_i \in \mathbb{R}^q$ approximates its expected signature, whose true value has 99% of probability of being in the q-dimensional ball of radius $C_i\sqrt{q} \in \mathbb{R}$ centered in E_i. The distance matrix is finally approximated with:

$$D_{ij} \approx d_{ij} \doteq \|E_i - E_j\|_{\mathbb{R}^q} \tag{4.10}$$

Entries with "big" values indicate dissimilarity between the processes, and conversely. To increase the safety in practice, we are more likely to trust detected differences, rather than similarities.

4.1.6 The macrovariance of a family of processes

For two real numbers x and y we define their *symmetric relative error* as $r(x, y) \doteq \frac{|x-y|}{\max(|x|,|y|)}$. For two arrays $X, Y \in \mathbb{R}^n$, the mean symmetric relative error is the average over the coordinates, $msre(X, Y) \doteq \frac{1}{n}\sum_{i=1}^{n} r(X_i, Y_i)$. The obtained number can be multiplied by 100 for a percentage interpretation. The definition is motivated by the fact that absolute errors, expressed as norm differences as in the distance matrix above, produce sometimes number which

are hard to interpret. Our numerical experiments are enhanced with a percentage number which helps in understanding how "spread" the family of processes $X_1, \dots, X_k$ is:

1. compute the *centrum* $\chi = \frac{1}{k}\sum_{i=1}^{k} E_i$, average between the approximated expected signatures, then $v_i = msre(\chi, E_i)$ be the relative distance between the reference point χ and the E_i;

2. the *macrovariance* is the average between all the v_i.

Since every expected signature E_i represents (in theory uniquely) the corresponding process X_i, the macrovariance tells us how far they are located with respect to their center average. Intuitively speaking, a family of stochastic processes with a small macrovariance should have similar members, since they are all very close to a common central point. The next section finally explains how we can visualize the point E_i so to have overall informative plots.

4.1.7 The multidimensional scaling algorithm

Given a generic $k \times k$ distance matrix between points in some high dimensional space $\mathbb{R}^d$, we are interesting in visualizing these k points on the plane, in dimension two, in order to provide a visual interpretation. For instance, it is possible that they form clusters which can be subsequently analyzed by using appropriate algorithms (for instance, k-means). Our favorite solution comes from a known algorithm in machine learning called *multidimensional scaling*.

In few words, it simply searches for k points in $\mathbb{R}^2$ whose distance matrix is approximately the original one. After the execution, the algorithm correctness can be quickly checked by comparing the error between the distance matrices. If it is true that the solutions are generally not unique (translations or rotations will still produce valid points), it does not constitute a problem for our experiments. Our goal is a visualization to gain insight, to clarify the processes differences. Possible further steps are always run on the original data.

We conclude this section with an important remark. Since the points to plot will often be the expected signatures E_i, it is important to understand how to manage the confidence balls C_i. We propose an heuristic approach: when E_i are projected into dimension 2, the value of $C_i\sqrt{d}$ is multiplied by $\frac{\sqrt{2}}{\sqrt{d}}$ so to rescale it with the new dimension 2 instead of the original d.

4.1.8 A simple experiment: Gaussian segments

In this section we compute the similarities between various centered Gaussians. We choose $k = 11$ variables differing only for their standard deviations σ_i, ranging in $\{0.3, 0.6, 0.9, \dots, 3.0\}$.

A total of $N = 10000$ random segments $[0,1] \ni t \mapsto (t, tX)$ for each variable are simulated, the signatures truncated at depth 5 before taking their expectations. After having built the (approximated) distance matrix, the processes

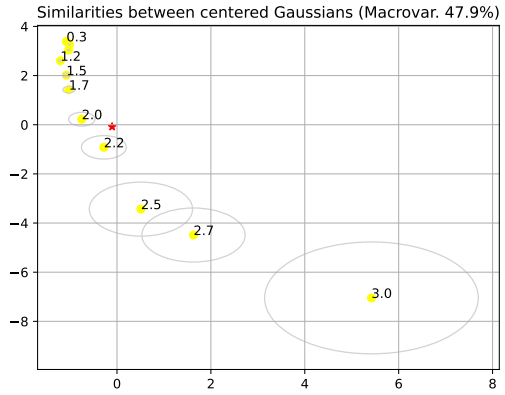


Figure 4.2: Simulations with simple Gaussian segments. Each dots represents a process whose label is the standard deviation. The circles are the rescaled confidence radius. Their overall center χ is represented by the red star. The macrovariance, i.e. how far the points are from the center, is around 48% (high value). The processes here are governed by a one-dimensional parameter (they only differ because of the σ_i), and they place themselves among a single curve (one-dimensional manifold).

are visualized on the plane by using the multidimensional scaling algorithm. Comments on this experiment are reported directly in the plot's caption.

4.1.9 A complete example using the Brownian motion

In this section we perform a second experiments by using instances of geometric Brownian motions. All the needed definitions follow in form of a brief recap.

Definition 4.1.1 (Brownian motion)**.** A stochastic process B_t for $t \in [0, 1]$ is a *standard Brownian motion* if the following conditions hold:

- B_t has almost surely continuous realizations;
- $B_0 = 0$;
- for every $0 \leq r < t \leq 1$, $B_t - B_r \sim N(0, t - r)$;
- for every possible choice $0 \leq r < t < a < s \leq 1$, the random variables $B_t - B_r$ and $B_s - B_a$ are independent.

We use the following simple rule to numerically simulate a Brownian.

Definition 4.1.2 (Brownian motion simulation)**.** Choose a time division N, set $W_0 = 0$ and $h = \frac{1}{N-1}$. Then for every $n \in 1, \ldots, N-1$ define:

$$W_n = W_{n-1} + \sqrt{h} N(0, 1) \tag{4.11}$$

This process $\{W_n\}_{n=0,\dots N-1}$ converges in distribution to a Brownian motion for $h \to 0$.

We refer again to [KPS94] for a more complete description, as well as the general methods for stochastic differential equations.

The Brownian motion can be used as a "trick" to randomly perturb an exponential growth, with various direct applications for instance in the field of mathematical finance (Black-Scholes model, to cite one). This originates the so called *geometric* Brownian motion.

Definition 4.1.3 (Geometric Brownian motion)**.** Let W_t be a Brownian motion and $S_0 > 0$ a starting value. For a couple of parameters $\mu \in \mathbb{R}$ and $\sigma > 0$, the corresponding geometric Brownian motion is the stochastic process:

$$S_t = S_0 \exp((\mu - \frac{\sigma^2}{2})t + \sigma W_t) \tag{4.12}$$

Dividing by S_0 and taking the logarithm we obtain the *log-return* process:

$$L_t = (\mu - \frac{\sigma^2}{2})t + \sigma W_t \tag{4.13}$$

We generally work with log-returns since they always start from zero and are therefore easier to include in the context of the signature.

The simulation of a geometric Brownian motion (sometimes abbreviated with GBM) can be done by running a Brownian motion as above and then applying the formula in the definition. More accurate and interesting methods are available and described in the book previously mentioned ([KPS94]), but they are not needed for our experiments.

We measure the distances in a family of geometric Brownian motions described by changing their parameters μ and σ as follows.

The values of μ (the "trend") are chosen in the range $(-1, 1)$, while the values of σ in the interval $(0, 0.8)$ (i.e. up to a "volatility of 80%"). A total of 20 combinations are selected and reported as labels in the attached plot.

We consider $N = 20$ timesteps and a total of 2000 path samples per parameter.

The results are as usual plotted in two dimensions producing a quite interesting picture. We apologize for some aesthetic overlapping, but this compromise was necessary to keep the content readable.

Observe how there is a clear splitting between the area with a positive trend (bottom part, gray dots), and the one where μ is negative (upper part, orange dots). A interesting radial structure is also quite evident. Once a value of μ is fixed, variations of σ are located on seemingly regular arcs, and the other way around.

The processes are seemingly ordered on a *two*-dimensional grid, coherently with the fact the they are originally described by *pairs* of parameters. This phenomenon happened also before when we studied Gaussian segments. They were described by a *single* parameter, and the dots aligned themselves among a *one* dimensional submanifold (curve).

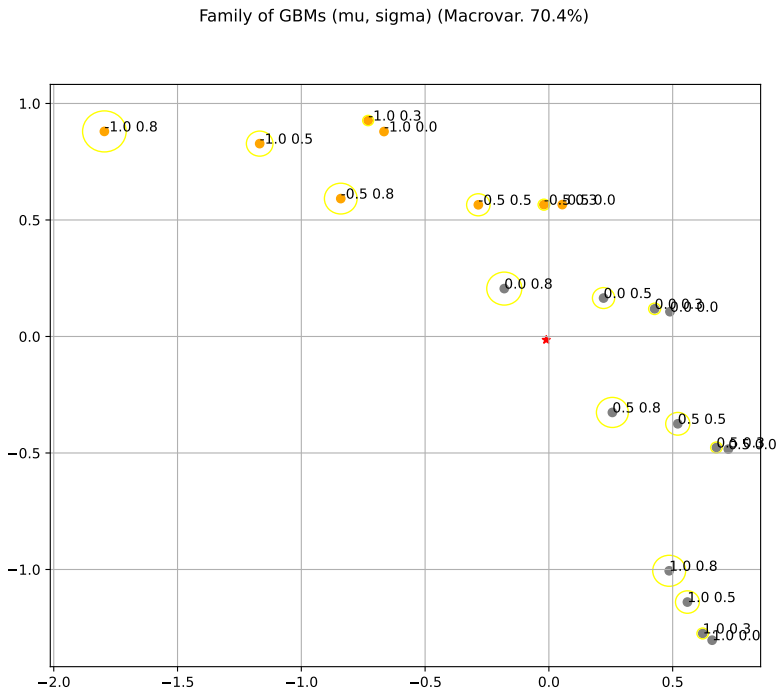


Figure 4.3: Each dot represents a geometric Brownian motion whose parameters are described by the label in the form (μ, σ). The macrovariance is above 70% suggesting remarkable differences between the processes. They ultimately seem to be ordered on a very regular two-dimensional "radial grid".

4.1.10 Recap of the experiments

In the two previous experiments we always started with a precise family of processes, generally described by varying certain parameters in a set $\mathcal{P}$. For instance we had $\mathcal{P} = \{0.3, \dots, 3\}$, in the case of Gaussian segments, we had $\mathcal{P}$ done by 20 pairs in the case of geometric Brownian motions.

In all cases, we used the macrovariance as a quantity to measure how spread, rich, the family of the corresponding processes is.

We validate even more this interpretation in the next section.

4.1.11 Increasing the family of processes

In this section we always work with centered Gaussian random walks.

The set $\mathcal{P}$ is made explicit later, but the reader can assume to be always given by 10 positive real numbers. For each choice of $p \in \mathcal{P}$, we simulate 10000 Gaussian random walks with increments of mean zero, standard deviation p and mesh composed by 30 time points in $[0, 1]$. The average of their 10000 signatures is stored and truncated at level 5. The macrovariance between them is measured and considered as the result of a single simulation.

Differently from before, this time this whole experiment is repeated 9 times, every time choosing a *larger* set of $\mathcal{P}$.

Initially we set $\mathcal{P} = \{1, \dots, 1\}$, expecting therefore a zero macrovariance since all the 10 simulated paths were theoretically equidistributed. Despite this, the numerical estimation had a value of around 3%.

We then progressively increase $\mathcal{P}$ to the interval $[1, 1.5]$, then $[1, 2]$, $[1, 2.5]$, and larger and larger until $[1, 5]$. Every interval was always divided in 10 equal parts so to keep the family of processes of the same cardinality. The overall results are attached in form of plots. It is possible to clearly read an increase in the macrovariance (y-axis) every time the parameter set is made larger (x-axis).

As a way to support even more the interpretations of the results, we plotted the expected signatures of the 10 processes for the case $\mathcal{P} = [1, 5]$. It is possible to "read" the macrovariance by looking at how different these values are.

4.1.12 Conclusions

In this chapter we studied the problem of measuring differences between stochastic processes by using the signature transform. We started with the case of a fixed family of processes, and measured their difference using the empirical notion of *macrovariance*. We also introduced the geometric Brownian motion which is of importance especially in contexts like financial applications and will be used again in later chapters.

Visualization also covered an important role, and we proposed a way to plot the family of processes in two dimensions with the inclusion of confidence radius so to keep track of uncertainties.

In the next section we perform a very similar analysis, but focusing on the sample paths produced from a *single* process instead.

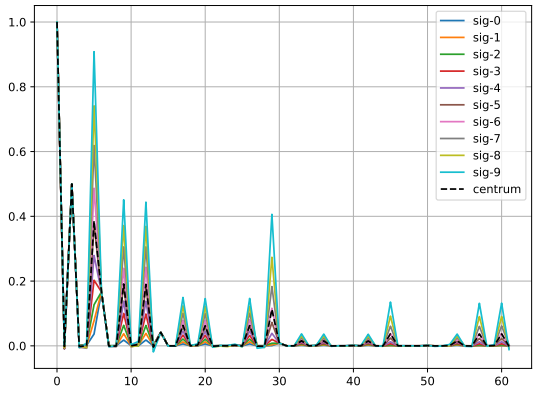


Figure 4.4: The average signatures of the 10 (dataset of) different Gaussian random walks. The macrovariance is here pictorially understandable by looking at how the spikes progressively change and cover wide range of values, when compared to their overall mean dashed in black (label "centrum").

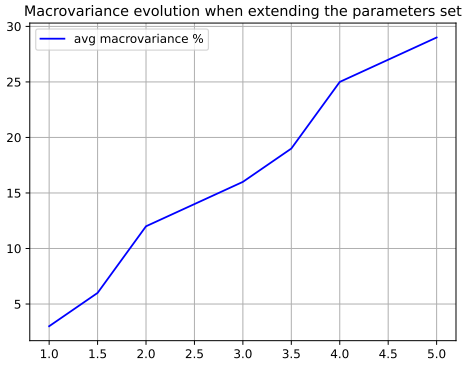


Figure 4.5: Values x on the x-axis represents the range defining $\mathcal{P} = [1, x]$ in which we choose 10 parameters and run the macrovariance analysis. The first value corresponds to the singleton $\{1\}$ which means that all the simulated paths are from the same stochastic process, therefore the macrovariance is theoretically zero. It does not numerically happen ($\sim 3\%$) and it points out well the effects of discrete truncation. Despite these limitations, it's easy to see that increasing the parameter ranges implies increasing the macrovariance of the resulting family as intuitively expected.

4.2 Stochastic micro-clustering

4.2.1 Introduction

In the previous section we introduced the idea of *macrovariance* as a tool to quantify the "diversity" in a family of stochastic processes. We follow a similar approach, this time in another setting: a *single* process is fixed, then we try to investigate the diversity in its random samples. The suggested methodology is based on the use of the signature transform. We introduce a percentage quantity here conventionally called the *microvariance*. Numerical experiments are performed in that regard, including applications in the field of reinforcement learning.

4.2.2 Definition of microvariance

Assume to have n sample paths $X_1, \dots, X_n$ coming from a stochastic process X satisfying our usual hypothesis. In order to quantify how different the samples are, we act as follows:

1. compute the (truncated) signature S_i of every path X_i;
2. estimate the average $E_X = \frac{1}{n}\sum_{i=1}^{n} S_i$, approximation of the expected signature of the process X;
3. for each sample X_i, compute the symmetric relative error r_i between X_i and E_X;
4. store the *microvariance*, defined as the average between the n values of r_i.

For a given process X with multiple samples, we estimate its microvariance as a way to understand how different and sparse the samples are with respect to their center. The difference with respect to classic techniques lies in the *space* in which we perform the measurement. The signatures of the data is in principle stronger than naive euclidean methods since the average signature is able to *characterize* a process, while the pointwise average path is not. It is possible to have two processes having the same mean path, but if two processes have the same mean signature, they are equidistributed. It is from this idea that comes the intuition of measuring sparsity using the average signature as central point (instead of the average path).

We test this method on a Gaussian random walks before moving to more interesting contexts.

4.2.3 Experimenting with Gaussian walks

The data used in this experiment are precisely the same as in the last section about the macrovariance. We work with Gaussian random walks with a variable parameter σ representing the standard deviations of the progressive increments.

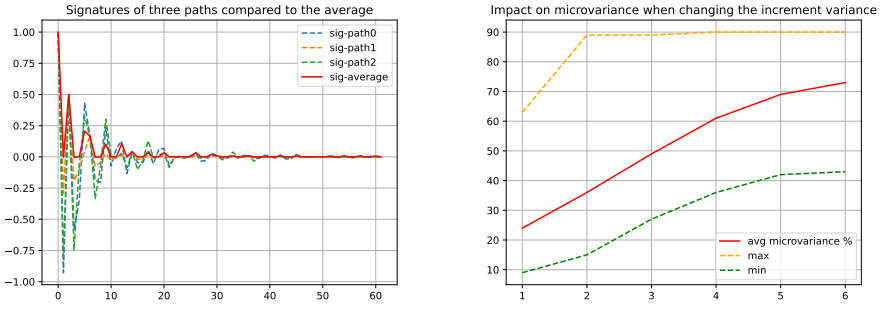


Figure 4.6: In the left plot we simply compare the average signature (red) with the signature of three sample paths for the case $\sigma = 6$. Their mean (relative) differences give us the microvariance measurement. On the left, we observe how an increase in the standard deviation of the simulated Gaussian walks (x-axis) leads to an increase in the microvariance (y-axis), as intuitively expected since the produced paths are more spread.

For each value of σ we have a single process and therefore a microvariance measurement on their samples. We observe here how much the microvariance increases when we vary the values of σ.

Intuitively speaking (from an Euclidean viewpoint), the higher the standard deviation, the higher the microvariance should be, since the sample paths are farther from their mean.

The results are actually in line with this intuition and are here reported in the attached plots.

4.2.4 Applications in reinforcement learning

In this section we cover an application in the field of *reinforcement learning*. We directly refer to resources like [RSS18], freely available on the official webpage `http://incompleteideas.net/book/the-book.html`, as well as [Ber23] from the website `http://www.mit.edu/~dimitrib/RLCOURSECOMPLETE.pdf` for a more mathematical description.

In the reinforcement learning setting (abbreviated "RL"), the programmer has to tune an "agent" capable at each time step to decide which "action" to perform. Once an action is taken, the "state" of the system changes and the agent is given a "reward". Given a time horizon, the programmer's goal is to write an agent capable of maximizing the total reward.

The mathematical description changes depending on the nature of the elements into play. One can have a finite and small set of possible actions, or a continuous one where approximation techniques are then required. The time horizon does not need to be finite, and some solutions are possible thanks to the *dynamic programming principle* which builds a strong connection between

Figure 4.7: A representation of the cart-pole problem. Applying the action 1 pushes the cart to right. The agent must then react properly so to preserve the pole equilibrium.

reinforcement learning and classic optimal control theory.

Examples of applications are vast and from multiple fields. Some are more fun and recreational, like the game of chess, go, or classic Atari games, others more serious and oriented towards new technologies like self-driving cars, robotics and automated trading.

In this section we focus on the cart-pole problem, very famous in the field, easy to to introduce and generally considered to be a good toy benchmark when testing reinforcement learning algorithms. We suggest the following link to get more information on that regard: `https://www.gymlibrary.dev/environments/classic_control/cart_pole/`. In our applications we stay focus on this specific case.

In the cart-pole problem, at every time step the agent can perform only two actions: "push right", encoded with the integer 1, or "push left", encoded as 0.

The state of the system at time t, call it $X(t) = (X^1(t), X^2(t), X^3(t), X^4(t))$ is always described by using four real numbers. The first is simply the cart position, the second its velocity, the third the pole angle and the last the pole angular velocity.

On the base of the system state, it is possible to deduce if the pole is still in equilibrium or not. If the equilibrium is broken, the system stops and the reward corresponds to the current time step. Otherwise, the game proceeds.

A reward of over 475 points is considered a "win". In other words, the goal of the programmer is to write an agent capable of pushing the cart right or left at every time step so to preserve the pole equilibrium for longer than 475 time units.

The entire interface is already packaged and available in Python thanks to the stable-baseline3 library (`https://stable-baselines3.readthedocs.io/en/master/`).

We call the entire 5-dimensional collection of the states $\{t, X(t)\}_{t=1,2,\dots,T}$ a *rollout*, where the time horizon T depends on how well the agent performs. The stable-baseline3 library allows to quickly train agents by using standard available algorithms, for instance PPO, DQN and A2C.

In our experiment, for each algorithm we train an agent to keep the equilibrium for more than 500 time units.

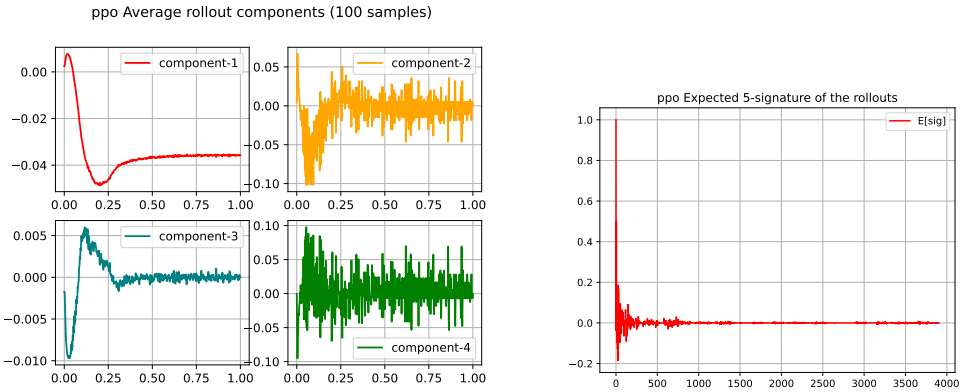


Figure 4.8: Left: the average rollouts in the trained PPO agent. Note how the velocities oscillates a lot, sign that they are trying to preserve the equilibrium. Right: the average signature. It is interesting to notice how coefficients in positions higher that 1000 are approximately irrelevant.

Then, once the training is done, we store 100 rollouts from the agent until time $T = 500$. Due to intrinsic probabilistic strategies, despite having being trained one time, the agent will *not* always perform *precisely the same* rollout.

In other words, to every algorithm (PPO, DQN or A2C) we associate a fixed trained agent, which can be interpreted as a $5-$dimensional stochastic process X_{PPO}, X_{DQN}, X_{A2C}. The samples of the processes are given by the agent rollouts, dataset of 100 random paths of length 500 (that we rescale on $[0, 1]$).

In the pure reinforcement learning setting, an important question is: given a training method, how much "variability" does it produce in the resulting trained agent? The word "variability" is on purpose not precise and not well defined, being an open topic in the reinforcement learning field with synonym like "population variation" and others.

In our approach we offer a clear interpretation. The "variability" will be nothing but the measure of the microvariance as described in the previous sections.

Other than measuring this quantity, we perform a further analysis of the data. For each of the three agents, we plot an average of the rollouts so to gain an insight of their performances. We then compute the signature transform on the samples, so to estimate the expected signatures and use the multidimensional scaling visualize them in dimension two.

In all the three cases two clusters are identified. For each of them, we plot the average paths of their members so to give an interpretation of the results.

All the plots are attached and independently well commented.

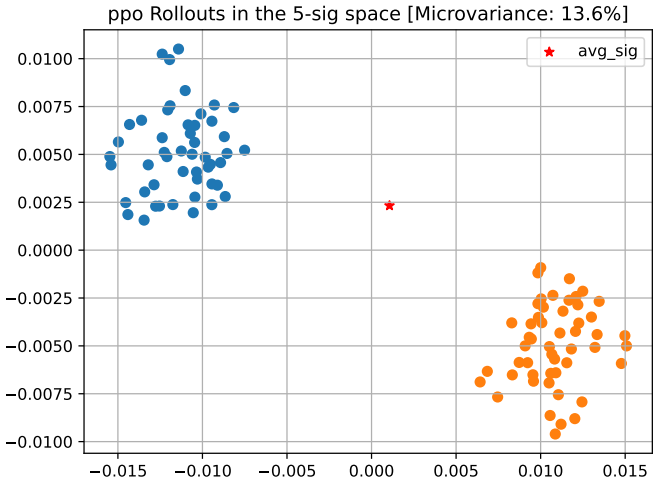


Figure 4.9: The rollout samples can be easily clustered into two groups. Their microvariance is around 14%, meaning that there are few differences between the various rollouts.

4.2.5 Comparing the three agents

In the previous section we studied the sampled rollouts from every (fixed) agent (PPO, DQN or A2C), aiming at understanding which of them produces more "variability". Clustering in the signature space revealed how they always displayed two main groups, whose interpretation was easy for the PPO algorithm but less straightforward for the remaining two.

In retrospective, we can further apply the techniques from the macro-clustering chapter, in order to *compare* the three agents directly! We proceed exactly as previously done with the experiment in that chapter. We use the expected signatures already computed and plot them in two-dimensions by using the multidimensional rescaling algorithm. The results are attached and very easy to interpret. It seems that the three agents are essentially equidistant, but sensitively different with respect to each others.

In order to test the algorithm even further, we also repeated the same procedure this time training three agents under the *same* algorithm (PPO). The plot highlights how they are numerically detected as (very likely...) equidistributed, as theoretically expected.

In conclusion, between this and the previous section we explained in details how the signature transform can be used to perform some data analysis when working with time series. We added an hopefully interesting example coming from reinforcement learning to point out the flexibility of our ideas, not limited to any specific field of application.

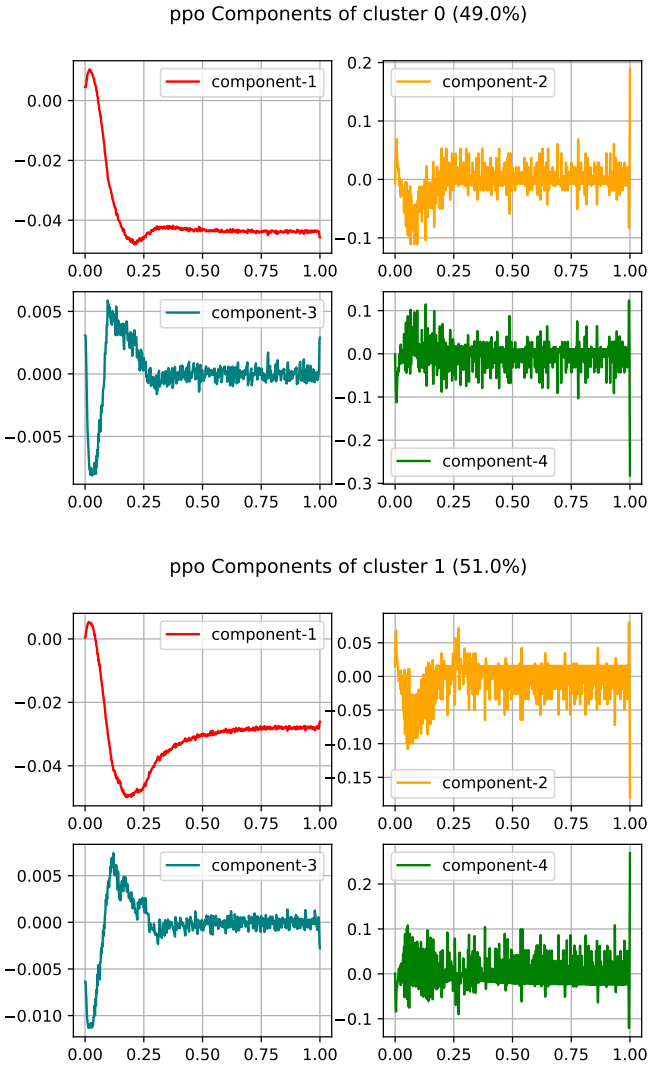


Figure 4.10: In the plot above, the average rollouts of samples belonging to the first and then second cluster. Trajectories in the first cluster have a cart position below -0.04, while the samples in the second go beyond and stays below -0.02 (component-1).

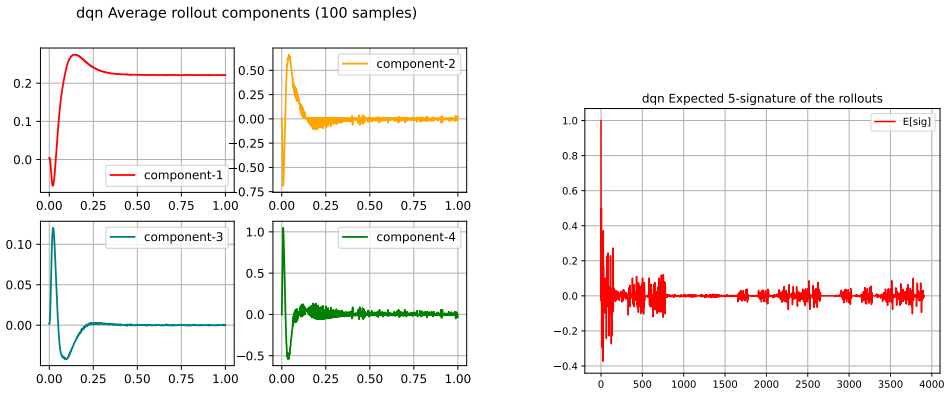


Figure 4.11: Left: the average rollouts in the trained DQN agent. Note how the velocities oscillates less than in the PPO algorithm, meaning that the agent is more stable. Right: the average signature. Note how the higher order coefficients still have relevant values, possibly meaning that the paths have a more regular periodicity (for more about this interpretation, compare to the sinusoidal case in the later section on "shape analysis").

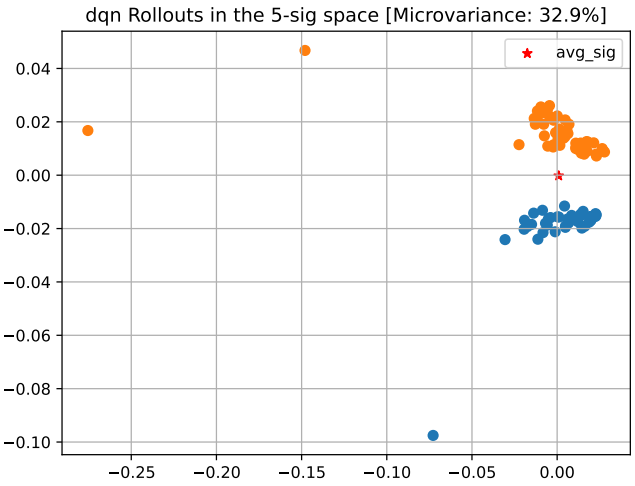


Figure 4.12: For the DQN algorithm again, the rollouts can be clustered into two groups. This time, their microvariance is higher and around 33%. According to our measurements, the DQN algorithm produces agents with higher "variation" than PPO.

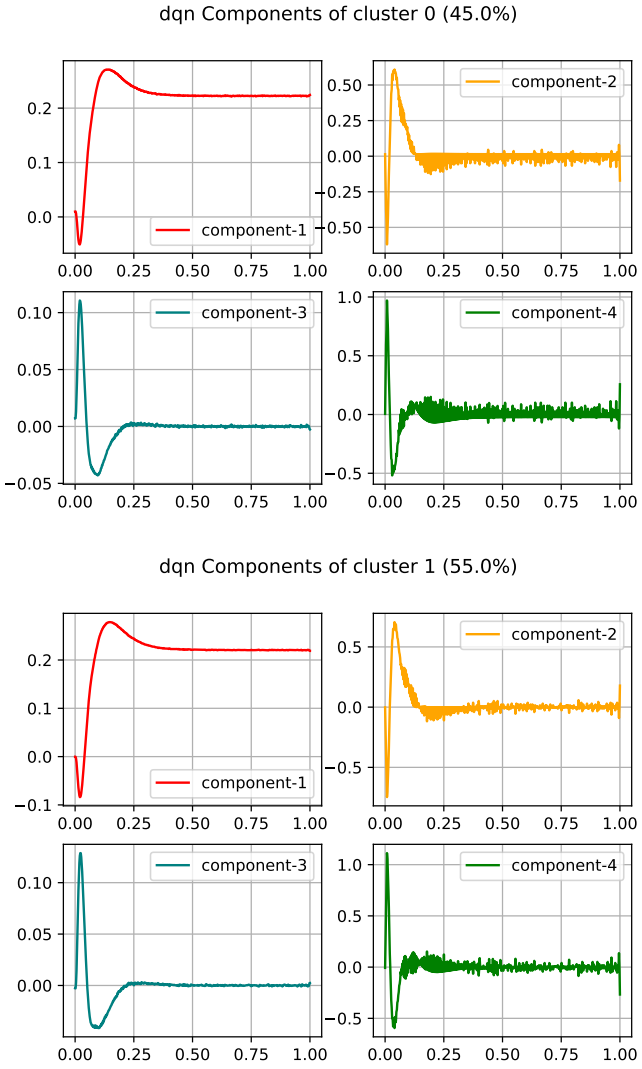


Figure 4.13: The mean rollouts for the first and second clusters are here plotted. This time is harder to give an Euclidean interpretation of the clusters. It's possible that the only criterion are the ending points of the velocities (components 2 and 4).

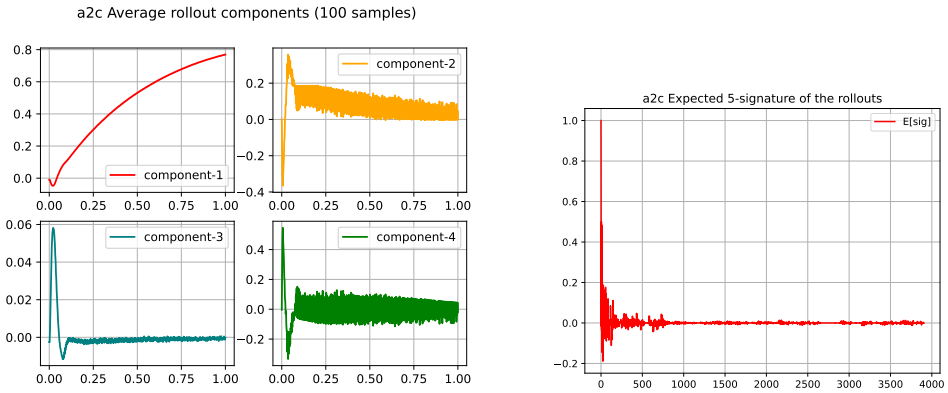


Figure 4.14: Left: the average rollout when using the a2c algorithm. It is possible to read how the position slowly increases to 0.8, which is still considered a "winning" value according to the algorithm documentation (bounds not reached). On the right the expected signature is showed, here again higher order coefficients seem to have approximately zero relevance.

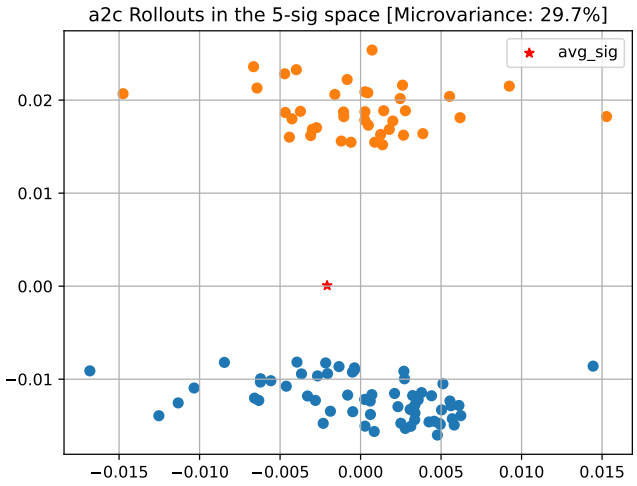


Figure 4.15: As in the previous processes, two clusters are available. The microvariance is comparable to the DQN case and higher than the PPO agent.

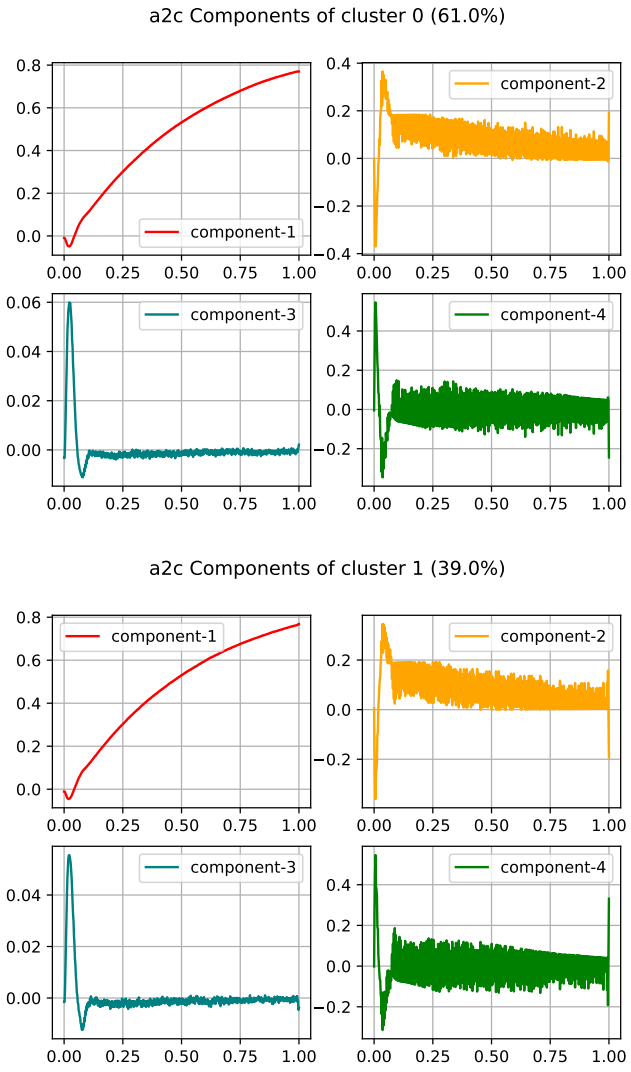


Figure 4.16: The average rollouts of the two clusters. There is again some difficulty in giving a straight interpretation, but it is possible that the discriminant is again just the final velocity (component-2, positive or negative).

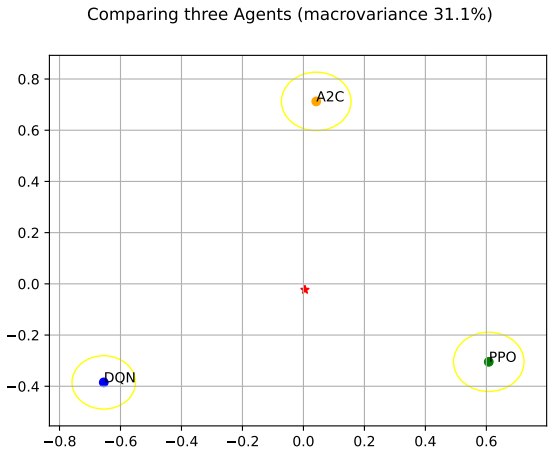


Figure 4.17: The three RL agents plotted in two dimensions, such that each point represents their expected signature. Confidence intervals are given by the circles, while the average distance with respect to the center (star point) has a relative value of around 30%, suggesting a moderate difference between the agents. Finally, it is interesting to note how they place themselves so that they can be considered equidistant.

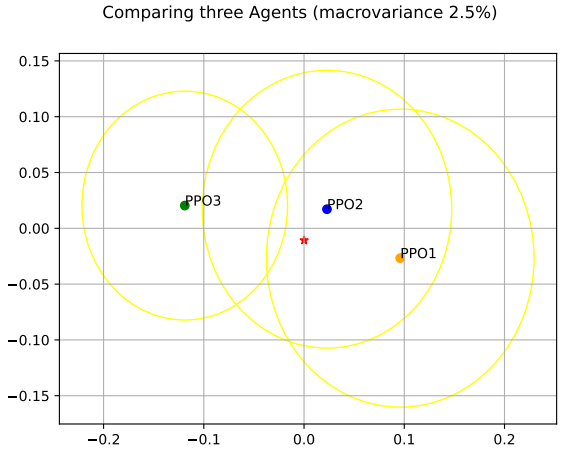


Figure 4.18: The same experiment repeated again after training three PPO agents. Since the algorithm is the same, the three agents should theoretically be equidistributed processes. The plot validates this idea, thanks to the largeness of confidence intervals and a very small macrovariance of 2.5%.

Chapter 5

Approximating nonlinear functionals

5.1 The max operator

5.1.1 The deterministic case

In this chapter we discuss numerical experiments to support the idea of approximating *nonlinear* functionals on time series by using *linear* functionals on their signatures. Recall that this is one of the four properties mentioned in the chapter before, based on the section "Function linearization" (3.1.4). We start by commenting the mathematical max operator either in the deterministic and probabilistic case, moving then on further applications where we build a tool to detect correlation between time series data.

For the first experiment, we build a dataset of 500 samples of the form $\{(x_n, y_n)\}$, where each x_n is a sequence of 20 numbers. Each x_n starts from 0, while the 19 remaining values are random and sampled from independent Gaussians of mean 0 and variance 3. For each time series x_n, the corresponding value is $y_n = \max x_n$. The max operator is nonlinear, very common and simple to implement, therefore it constitutes a good example to start our investigation.

The dataset is $50 : 50$ split into two subsets, train and validation. We use the PyTorch scientific framework to test two different models for our data. Both are *linear models*, i.e. equations of the form $f(x) = ax + b$ where the vectors a, b are chosen so to minimize the loss function (mean square error) between $f(x_i)$ and y_i.

The first model takes in input the time series x_i itself, therefore "raw" data with no particular modification. Since the max operator is nonlinear, we do not expect good results. The second linear model acts instead on the *signatures* of the augmented paths, truncated at depth 5.

The loss minimization is done by using PyTorch's Adam gradient descent algorithm with 10000 epochs, and in both the experiments the running time

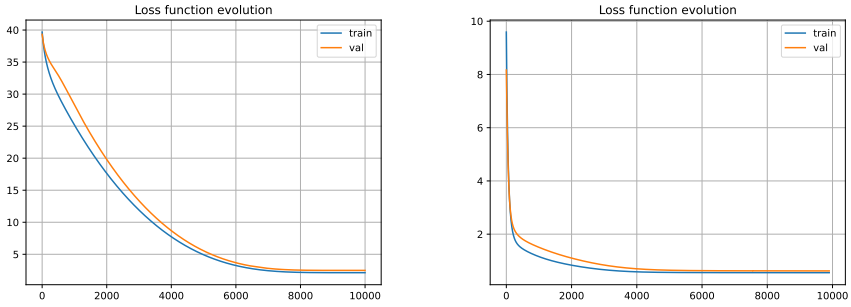


Figure 5.1: The loss function decay when training the linear model on time series (left). When instead we use their signature transform, this decay happens faster (right picture).

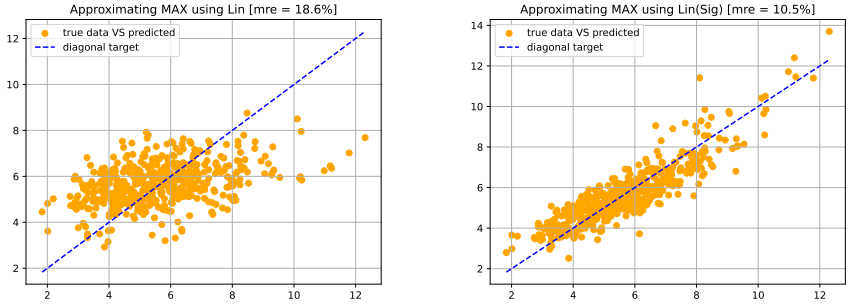


Figure 5.2: Comparing the approximation of the max operator. A simple linear model (left picture, mean relative error of 18%), and a linear model on the *signature* of data (right picture, mean relative error of 10%).

was a matter of seconds.

Recall that we actively minimize the loss function only using training data, and any loss decay happening also on validation points ensures the absence of overfitting.

After the training procedure, the models are evaluated on the full dataset and the mean relative errors are displayed. The signature linear model produces a far smaller error (10% against 18%) and had overall a better performance during the loss function reduction.

We also offer the results in a readable way thanks to a plot in the form '(true value, predicted value)', so that the more the picture gets close to the identity function, the better it is. Coherently with its lower error, the signature model follows the diagonal better than the simple linear one.

The results on this section add a touch of concreteness to the approximation

theorem explained in the theoretical chapter, suggesting that the signature can be an useful tool already when combined with simple linear regressions.

We proceed with cautious optimisms in that regard, pointing out that hyperparameters can definitely influence the results, like for instance the learning rate, the number of training epochs and the level of the signature truncation. We repeated the experiment under different circumstances and we observed some output changes, but overall the signature model always showed a faster loss decay (allowing therefore a reduced training time) and never performed worse than the linear one, supporting its use in further contexts.

5.1.2 The expected payoff

For a general process X_t and a functional F the question of approximating the value of $\mathbb{E}[F(X_t)]$ is commonly of great interest. We prefer the writing $F(X)$ to highlight that the functional might depend on the whole path instead of a precise fixed time t. In the context of finance mathematics this is strong connected to the problem of *pricing*, where F depends on the option under analysis (payoffs) and X is for instance the behavior of the stock asset.

Many theories and strategies are available depending on the case under analysis, but we focus on a simple idea. If by using the signature we can approximate $F(X) \approx (k, S(X))$, for some tensor $k \in T(E)$, then *informally*:

$$\mathbb{E}[F(X)] \approx (k, \mathbb{E}[S(X)]) \tag{5.1}$$

because of the linearity of the expectation, which in the case of a truncated signature at level N translates into:

$$\mathbb{E}[F(X)] \approx (k, \mathbb{E}[S^N(X)])_{\mathbb{R}^p} \tag{5.2}$$

for the Euclidean scalar product in an suitable dimension $p = p(N)$.

We perform a numerical experiment in support of this idea. In order to vaguely connect with finance, we choose paths X_t given by (log-returns of) geometric Brownian motions and F as the max operator, often used in many payoffs (usually combined with other operations).

Our dataset $\{z_i, y_i\}$ is composed by a total of 200 records, $i = 1, \dots, 200$ now explained in details.

We know that any (log-) geometric Brownian motion depends on two parameters μ and σ. Therefore, we choose μ from $[-2, 2]$ (interval divided in 10 steps), σ from $[0.4, 3]$ (interval divided in 20 steps), for a total of 200 combinations of parameters. Each combination corresponds to an index i of our records.

For each i (so, a fixed pair (μ, σ)), a total of 500 paths of length 30 are simulated. For each path the max is computed: the label $y_i \in \mathbb{R}$ is defined as their average.

Each of the 500 path is then augmented (i.e. a first coordinate equals to the time t is added), and its signature (truncated at level 5) computed. The average of all (500) of them is stored as value z_i. Therefore for each i, z_i is in

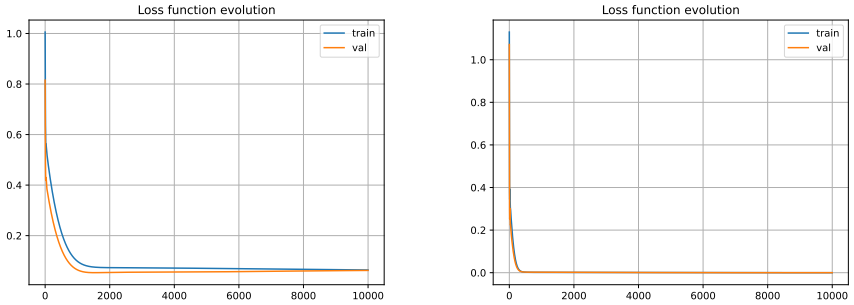


Figure 5.3: Loss functions during the training process of 10000 epochs. The right figure represents the model without the use of the signature, the left figure when using this transform to preprocess data. Observe the remarkable performance boost. Furthermore, the right model seems to likely to overfit.

our case an array of length 62 (consult chapter 2 for more information about this number).

Summing up, the z-entries in our dataset are done by 200 arrays of length 62, representing signature expectations of (log)geometric Brownian motions ruled by different parameters, and the y-label by 200 scalars given by the average max values reached by each parameter's choice. The data are mixed and split $50:50$ (train and validation). Finally a linear model approximation by using PyTorch is performed and the final prediction error computed.

In order to better understand the effect the using the signature, the experiment is repeated again on the original data without passing the time series under the signature transform.

The results are directly readable in the plot and display a clear advantage when using the signature method.

In conclusion we showed how the signature transform can be effectively used to linearize functionals even in the case of probabilistic settings. We would like to also point out how these simulations required only few minutes on the author's laptop and a standard Python installation, which is a strong point in terms of portability and usability.

5.2 A correlation classifier

Given a dataset of time series it is often of great importance to understand if some paths are correlated. In such a case, observing few of them could be enough to qualitatively predict the behavior of the others. For instance, if two stocks A and B are negatively correlated and we track an increase in the price of A, a decrease in the value of B should be expected.

In this section we try to understand if the signature transform can be suitable for this kind of tasks. Since we previously interpreted the signature as a

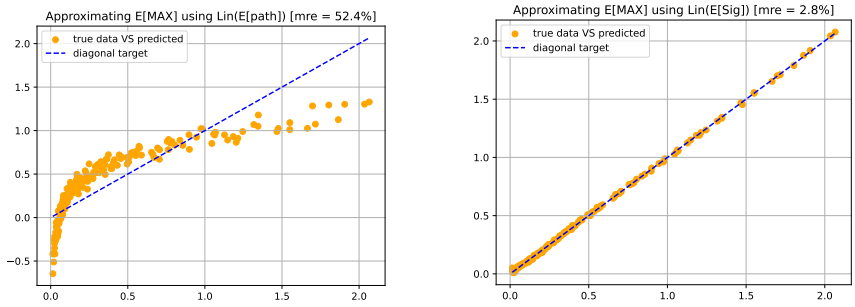


Figure 5.4: Approximating the expected max value of geometric Brownian motions ruled by different parameters. The plot on the top represents a naive linear model, with a mean average error of 50%. The plot on the bottom refers to the same data, but after having been preprocessed by the signature. The error hugely decreases to 3%, showing with no doubt one of the strongest effect of using the signature transform.

generalization of moments for time series, our attempt has at least an intuitive justification. We expect the information about the correlation to be encoded into the signature transform, possibly in the first few levels being it a quantity related to the variance (second moment) in the one-dimensional case.

We choose two cases of studies to test experimental ideas.

5.2.1 Experiment 1: classification of two Gaussian walks

In our first experiment the dataset is composed by 4000 paths of dimension 3 and labels 0 or 1, constructed as follows.

Each path X (we omit its index to improve readability) is as a collection of 50 node values on $[0, 1]$, always with starting point $0 \in \mathbb{R}^3$.

The first component X^1 equals the standard time discretization, $X^1_n = \frac{n}{49}$ for $n = 0, \dots, 49$.

The remaining two components require a bit of explanation. We set a positive correlation parameter $\rho \in [0, 1]$, a mean vector $\mu = 0 \in \mathbb{R}^2$ and covariance matrices $C_0 = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}$ and $C_1 = \begin{pmatrix} 1 & -\rho \\ -\rho & 1 \end{pmatrix}$.

We then define two Gaussian random variables in $\mathbb{R}^2$, $G_0 \sim N(\mu, C_0)$ and $G_1 \sim N(\mu, C_1)$. Note that the elements in the pairs generated by G_0 have correlation ρ, while the one coming from G_1 have negative correlation $-\rho$.

For each subsequent time index $n \in [1, \dots, 49]$ define:

$$(X^2_n, X^3_n) = (X^2_{n-1}, X^3_{n-1}) + G_0 \in \mathbb{R}^2 \tag{5.3}$$

if a path is labeled with 0. Replace G_0 by G_1 for paths labeled as 1.

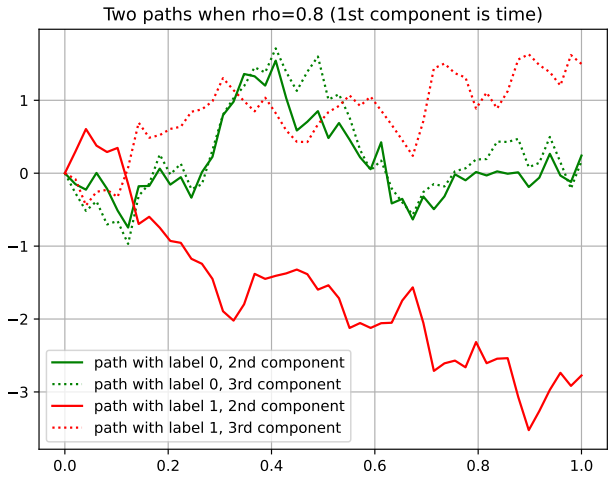


Figure 5.5: Two 3-dimensional paths from the dataset. Their first component is always the time $[0, 1]$ used as x-axis, while the other two components are plotted. The increments for the walk 0 (green) are positive correlated, while the increments with label 1 (red) are negatively correlated.

In other words, each path is an augmented Gaussian random walk starting in zero and with increments obeying a specific known correlation.

Half of the paths are labeled with 0 and follow a fixed positive correlation, while half with 1 and enjoy a negative one. The paths are transformed by using the signature at depth 4, obtaining a labeled dataset of 4000 arrays of length $3 + 3^2 + 3^3 + 3^4 = 120$ (the "3" coming from the dimension, the "4" from the truncation level, as previously explained).

The data are randomly shuffled and split into train and validation sets, each containing 2000 samples. As usual the task is to tune a model on the base of training data, but capable of correctly predict the labels for both training and validation sets. We explore two methods to reach this task.

Recall from previous sections that stochastic processes like the random walks here defined can be theoretically characterized by their expected signatures. Therefore the following strategy seems legit:

- compute the average signature of the paths with label 1, call is S_1;
- repeat the same for paths with label 0, call the average S_0;
- for each element in the dataset X, if its signature is closer to S_0 classify it as 0, otherwise as 1;

We call this strategy the "closest expectation classifier". The potential problem with this strategy is soon explained. If two stochastic processes have similar

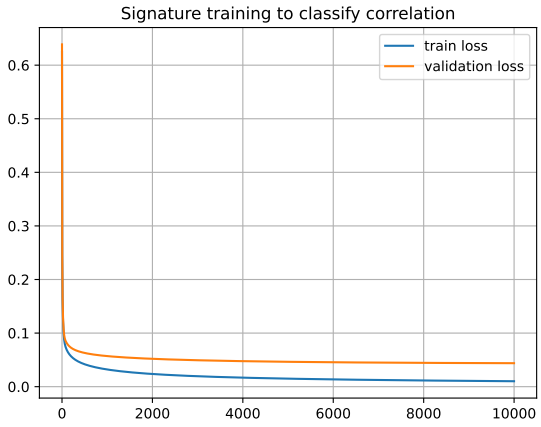


Figure 5.6: The decay of loss (y-axis) when training the linear classifier for 10000 epochs (x-axis). The loss behaves very regularly for both the training and the validation data. This plot refers to $\rho = 0.8$, but it is similar for $\rho = 0.4$. On the other hand, the training "fails" for $\rho = 0$, as it should be. The final stationarity reflects that we are approaching the modeling limit.

empirical expected signatures and enough big "variance" when generating paths, it's likely that miss-classification will happen.

Another problem can be given by having a number of paths not enough big to create confidence intervals small enough. Finally the signature must be truncated at some level, possibly compromising the theorem's application.

The second strategy to solve the classification problem is based on the linear approximation theorem. We know that every functional on paths (in particular, the "classifier functional" associating each path to its class), can be approximated with a *linear function* over the signature. We therefore set the simplest linear classifier in PyTorch, trained with stochastic gradient descent over 10000 epochs with learning rate 0.1. A plot displaying the loss decaying is attached, suggesting success and lack of overfitting.

The whole experiment is repeated overall three times, with different values of ρ chosen as 0, 0.4 and 0.8. For $\rho = 0$, since we then have $G_0 = G_1$ we expect the algorithm to be unable to learn (since there is no difference in the paths), producing an accuracy around 50%. Despite not being particularly interesting, it is important to test against basic cases like that.

Higher correlation are intuitively "easier" to spot, therefore we expect the algorithm to work better with $\rho = 0.8$, and worse for $\rho = 0.4$.

The results follow and are summarized in compact tables. The title "LINEAR CLASSIFIER" refers to the method of using the linear model on the signature of our data, while "CLOSEST EXD" to classify on the base of the closest expected signature.

LINEAR CLASSIFIER	$\rho = 0$	$\rho = 0.4$	$\rho = 0.8$
Training accuracy	57%	86%	99%
Validation accuracy	50%	83%	98%

CLOSEST EXD	$\rho = 0$	$\rho = 0.4$	$\rho = 0.8$
Training accuracy	49%	51%	51%
Validation accuracy	50%	49%	50%

The experiments offer crystal clear results, suggesting how the closest expectation algorithm is *not* suitable at all for this situation. The explanation (confirmed by a quick numerical check here not reported) is that the two processes are "too close" in the signature space (i.e. the norm of the difference between the averages of their signature is relatively small) and therefore the method is highly sensitive to oscillation in the path's signatures.

On the other hand, the linear method perform greatly and perfectly matches our expectations. It is then repeated on more complex situations in the upcoming experiments.

We would like to finally point out some brief remarks. All the experiments in this section have been repeated multiple times with consistent results, and finally the random seed has been fixed so to allow reproducibility. The role of hyperparameters like the learning rate or the depth for the signature truncation should not be underestimated, since their influence of the results can be huge.

5.2.2 A closer look to the expected signature

If on the one hand we are satisfied with the results above, we think it is worth spending even more time to better understand and *interpret* them.

The expected signatures (able to characterize the processes) are here plotted (computed with classic Monte Carlo averages, with confidence intervals pictorially omitted since small enough to be now ignored).

The plots are in our opinion interesting, since we can clearly see a sort of "symmetric" structure between the two mean signatures. Linear separation is therefore intuitively expected to be possible. Furthermore the main changes start with the coefficient in position number 8, i.e. at position 9 since it counts from 0. And since $9 = 3^2$, this is precisely where the second level of the signature begins (the number 3 comes from the fact that we worked with three-dimensional paths). This is compatible with the idea that if for real variables the correlation relates to second moments, then for time series it influences the second level of the expected signature.

5.2.3 Experiment 2: classification of many Gaussian walks

In this section we perform a slightly more general experiment, where we aim at detecting a possible positive or negative correlation, without being restricted on precise values of ρ as before.

The dataset is similarly constructed, but this time we choose 6000 paths and *three* labels, 0, 1 and 2, equally distributed.

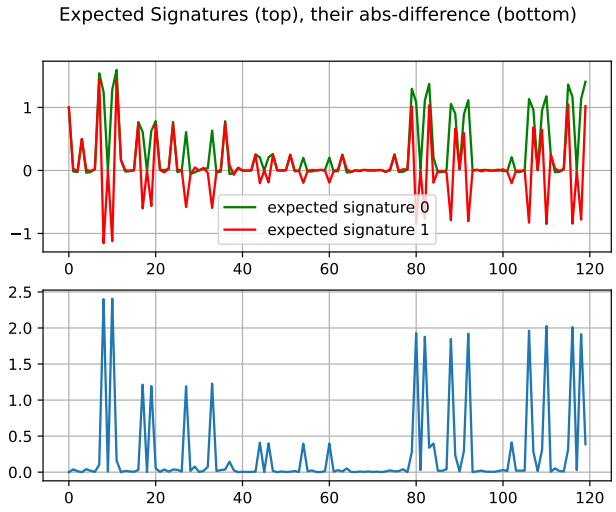


Figure 5.7: Top: the expected signatures of the paths with positive correlation, in green, and with negative correlation, in red. Bottom: their difference in absolute values. Note how some components seem to follow a symmetric behavior.

For every path with label 0, a different random constant is sampled as $\rho \in (-1, -0.5)$, then its Gaussian increments set to follow that negative correlation. Similarly, when the label is the value 1, the increments follow a "weak" correlation with a random $\rho \in (-0.4, 0, 4)$, while a label of 2 determines a strong positive correlation with a random $\rho \in (0.5, 1)$. We remark again that the correlation values are sampled every time for each path, therefore data with the same label are now no more coming from the same distribution and, differently from the previous experiment, there is no mathematical meaning in taking their "expected signature".

We confirm the use of the linear signature method with the same parameters and description as in the experiment before. Since this time we have three labels, the results are given in form of a classification matrix. We expect good performance when classifying samples with strong positive and negative correlations, and probably more errors when dealing with weak or absent correlations. The training phase seems to work nicely and the loss function behaves well on either training and validation data.

TOTAL training accuracy: $\sim 81\%$

TRAINING	predicted label 0	predicted label 1	predicted label 2
true label 0	30	3.6	0.17
true label 1	5.6	21	5.8
true label 2	0.13	3.8	30

TOTAL validation accuracy: $\sim 78\%$

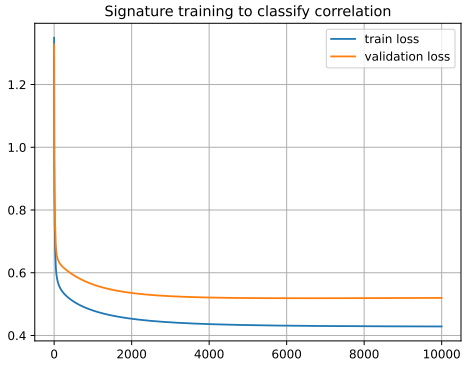


Figure 5.8: As in the experiment before, we observe overall a decay of loss (y-axis) when training the linear classifier for 10000 epochs (x-axis). Differently from before, the loss is higher possibly reflecting the more sophisticated data. Since stationarity is still reached, the limit capacity of our model is probably hit.

VALIDATION	predicted label 0	predicted label 1	predicted label 2
true label 0	29	4.6	0.13
true label 1	6.6	21	6.1
true label 2	0.20	4.1	28

The results are very consistent with our expectations. The trained linear classifiers perform overall pretty good, being in general capable to predict the right label on unseen data with around 78% of accuracy. The row with more errors (both in the training and validation cases) is always the second one corresponding to label 1, i.e. paths with weak correlation, on which we actually expected worse performances.

We point out how our results come from a *linear* model, particularly fast and easy to implement. If simple algorithms involving the signature already give reasonably results, maybe this transform can be included in more complex methods to improve their performance even more.

5.2.4 Working with geometric Brownian motions

In the case of Gaussian walks is it relatively simple to set alternative strategies, because by progressively storing the differences it is possible to evaluate their averages and directly estimate the correlations, classifying the paths efficiently without using the signature at all. On the other hand, not only the signature can provide a further support to already existing tools ("double-check"), but the linear classifier can be easily generalized.

As explained in the chapter before, geometric Brownian motions are classes of stochastic processes very useful for (and not limited to) financial modeling. If

pairs of paths are available but there is a lack of knowledge about their governing parameters σ and μ, it is in general not possible to estimate their correlation (as far as the author knows).

Motivated by these remarks, we repeat our experiments using now log-returns of geometric Brownian motions instead of simple Gaussian walks.

5.2.5 Experiment 3: correlation with two fixed classes of GBM

This experiment is precisely the same as experiment number 1, but instead of using Gaussian random walks we simulate instances of (log-returns) of geometric Brownian motions.

Each path is then a three dimensional one, with first component equals to the time t while the remaining two given by a couple of (log-returns) of geometric Brownian motions with a fixed correlation ρ.

We have already explained how to numerically approximate a GBM, but for specific remarks covering the correlated case we refer to [Shr04], example 4.6.6 at page 171.

The geometric Brownian motion parameters are set as $\mu = 0.4$ and $\sigma = 0.5$, and the number of time steps is 50 as before. The signature is truncated again at depth 4, and the learning rate is $1e-1$.

The dataset is split into two classes with labels 0 (given by paths of correlation ρ), and 1 for paths with correlation $-\rho$.

In order to keep a sense of coherence with the experiments before, we initially chosen $\rho = 0.8$, then 0.4 and finally 0 as a base case test.

Examples of paths are plotted, as well the loss functions showing again a sign of success. The final results are reported in the table.

Concerning the expected signature, note how the difference happens again at the second level of its coefficients.

LINEAR CLASSIFIER	$\rho = 0$	$\rho = 0.4$	$\rho = 0.8$
Training accuracy	54%	82%	98%
Validation accuracy	48%	80%	98%

5.2.6 Experiment 4: correlation with three classes of GBM

In this fourth section we build a scenario very similar to what already done with Gaussian walks in our second experiment.

A total of 6000 three dimensional paths are simulated, divided into three homogeneous classes with labels 0, 1 or 2. Each path has always the first coordinate set as the time t, and the remaining two as (log-returns) of geometric Brownian motions ($\mu = 0.4$, $\sigma = 0.5$) following a specific correlation. If a paths has label 0, the second and third coordinates have a random correlation $\rho \in (-1, -0.5)$. If the label is 1, then this value is taken from $(-0.4, 0.4)$. Finally for paths classified as 2 the random positive correlation lies in $\rho \in (0.5, 1)$. The full dataset is then split into two random parts of training and validation data.

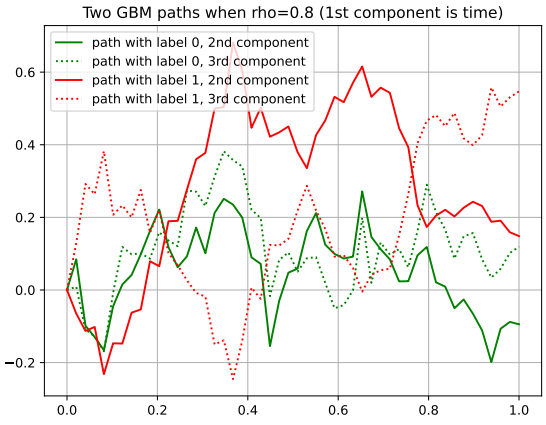


Figure 5.9: Log-returns of geometric Brownian motions with different correlations. Green components share a positive correlation of $\rho = 0.8$, while red coordinates a negative one of $-\rho$.

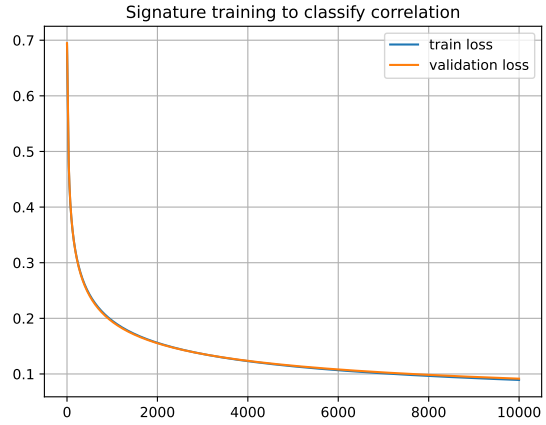


Figure 5.10: Training the linear classifier for the $\rho = 0.8$ case. The loss function behaves perfectly well and suggests that a longer training could improve the results even more.

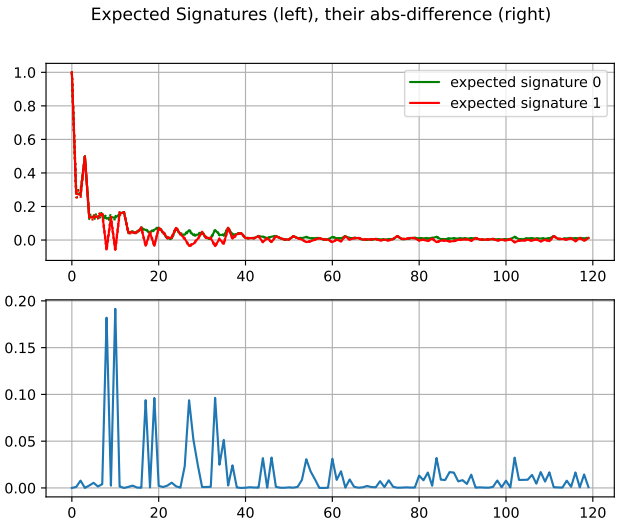


Figure 5.11: Expected signature for the case $\rho = 0.8$. There is an important information to point out: the first remarkable difference happens again at the coefficient in position 8, which corresponds to the second level of the signature.

The paths are transformed using the signature truncated at level 4, and a linear classifier is used to learn from training data and classify points from the unseen validation set. A total of 20000 epochs are run with learning rate $2e-1$.

We refer to the previous experiment number 2 in case further details are desired. The results follow and are overall optimistic. Note how the most difficult case is still given by the paths with label 1, most often misclassified, as in the case of simple Gaussian walks.

TOTAL Training accuracy: $\sim 76\%$

TRAINING SET	predicted label 0	predicted label 1	predicted label 2
true label 0	29	3.8	0.20
true label 1	7.5	17	7.8
true label 2	0.37	4.1	30

TOTAL Validation accuracy: $\sim 73\%$

VALIDATION SET	predicted label 0	predicted label 1	predicted label 2
true label 0	27	5.1	0.70
true label 1	8.1	17	8.9
true label 2	0.50	3.8	28

5.2.7 Conclusion

In the past section we showed how the signature can be used to detect correlation between time general series, assuming multiple samples are available. For the

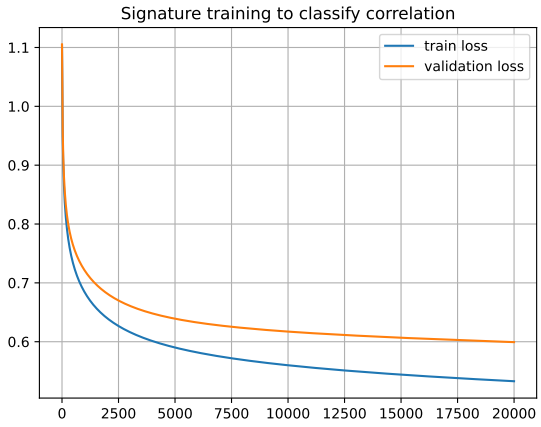


Figure 5.12: The training phase in this more complex experiment is harder than the previous cases, but still good and promising.

one dimensional case the correlation relates to the variance and therefore the second moment of a random variable. Similarly, it seems that the correlation has a connection to the second level of the expected signature of a time series. This remark could be a starting point for the developing of further detection systems, as well as improvements for already existing standard statistical methods.

Chapter 6

Signature shape analysis

In this chapter we try to develop a deeper intuition about the signature "geometrical" interpretation. We start by investigating the problem of reconstructing a curve from a given signature tensor, focusing on simple cases of common interest like straight lines, impulses and sinusoidal signals. Finally we suggest a method to possibly generate artificial synthetic data based on observed time series.

6.1 The problem of inverting the signature

Given a time series we can apply the signature transform and get a tensor of a certain depth. The question comes naturally: is it possible to "reverse" the process? Can we reconstruct a path by only observing its signature transform? We briefly described this idea before, and recall that the answer is negative.

For the sake of avoiding any ambiguity we spend a couple of lines commenting the use of the words "signature inversion". Given a signature tensor, sometimes in the literature one calls its "inverse" the tensor such that, multiplied with the original, gives the unity element. This is legit because the tensor algebra where each signature image lives can be endowed with a group structure. This problem is solved. Indeed, given a path X in $[0,1]$ with signature $S(X)$, its time-reversed path $Y(t) = X(1-t)$ satisfies:

$$S(Y) \otimes S(X) = 1 \tag{6.1}$$

and therefore is its "inverse". This is explained in [CK16] (theorem 3, page 15). Now that this algebraic definition has been clarified, we highlight how our problem is completely independent on that.

With the statement "signature inversion" we refer exclusively to what initially stated: given a signature tensor S, find a path X such that $S(X) = S$. Because the signature mapping is not surjective, there is usually no solution to this problem. We need to add some further conditions.

First of all, we only take into account two-dimensional paths on $[0,1]$ starting from 0 and having the time t as first coordinate. The monotonic behavior

would guarantee the signature injectivity, while keeping the dimension as low as possible will be helpful from a computational viewpoint. Restricting finally on tensors in the map's range, we get a bijection and can proceed with our investigation.

6.2 Numerical remarks

There are multiple problems when using the signature in practice, for instance we lose the precious injectivity property when truncating the tensor to a finite level. It is generally possible to have two paths such that their signatures are the same up to level N, differing then only afterwards. There is no actual "solution" for this, but a good level of truncation would anyway preserve enough geometrical properties.

In all our experiments, with letters like X or Y we refer to one-dimensional paths, while with $\hat{X}$ to the corresponding time-augmentation $\hat{X}(t) = (t, X(t))$. We work with discretized piecewise linear paths as consistently done in all the chapters before. The cardinality of the mesh is here denoted with n.

Let N be a positive integer referring to the truncation level, and S a finite dimensional array with length $p = 2^{(N+1)} - 2$. We are then looking for a one dimensional path X such that $S(\hat{X}(t)) = S$ (the arithmetic formula before comes from the way in which the signature is computed under the Signatory library). We approach this problem simply as a minimization of:

$$\|S(t, Z(t)) - S\|_{\mathbb{R}^p} \tag{6.2}$$

among all possible one-dimensional arrays Z of length $n-1$ (first value is fixed to be zero) representing the node values of the path.

Since the truncated signature is a combination of differentiable operations (see for instance [KL20] for a more detailed discussion), a gradient descent approach is possible and when using PyTorch we can take advantage of its automatic differentiation.

To see the algorithm in action, we perform multiple tests. Each of them is constituted by the following steps:

- generate a discrete 1-dimensional path, X, starting from 0 and having random $n-1$ values for each successive node point, each value distributed as $N(0, 1)$;
- compute the signature $S(\hat{X})$ until level 5;
- use a gradient descent (10000 epochs, learning rate of $1e-2$) to minimize the norm described above. Call Y the obtained 1-dimensional curve;
- finally compute the errors $err(S(\hat{X}), S(\hat{Y}))$ and $err(X, Y)$.

We run the tests with differently mesh sizes n starting from 4 and running until 32, doubling at every step. For each fixed meshsize, the tests are repeated

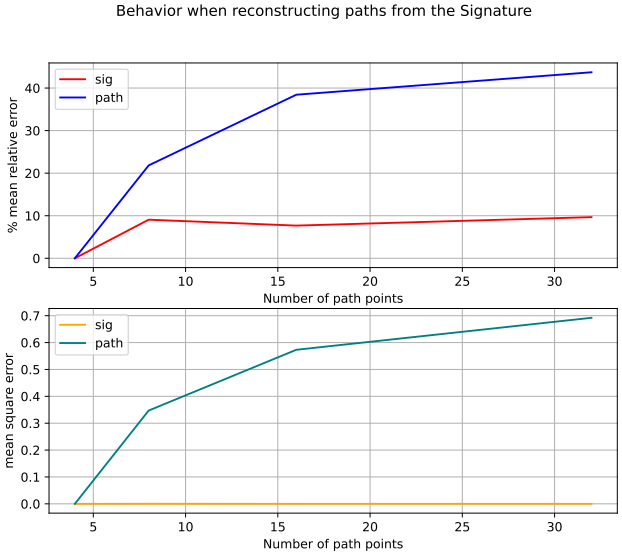


Figure 6.1: The typical errors (absolute and relative, on the signatures and on paths) obtained on average when reconstructing a path from its signature. On the x-axis the number of time discretization points, from 4 to 32. The signature absolute error is always lower than $1e-3$, and it is the quantity directly minimized by the gradient descent algorithm.

100 times so that we can have a hint about the *typical*, averaged errors coming from the use of this algorithm.

Since a simple gradient descent approach did not perform so well, the learning rate has been programmed so to detect oscillations or slowdown and automatically decrease or increase its value by 10%.

Two plots are finally attached. The first shows the results of our repeated tests, as a form of benchmark before moving to applications.

The second plot shows the effect of using different truncation levels. The same curve is reconstructed on the base of its signatures of depth 3, 5, 7 and finally 9. It is interesting not only to have a confirm of the quality improvements, but also to see how the curve is qualitatively reconstructed.

In general is is possible to increase the number of epochs and the signature truncation so to ultimately obtain better errors, but at the expenses of the running time. With our typical choice of parameters (30 time points, signature depth 5 and 10.000 epochs starting with a learning rate of $1e-2$) we are able to perform a single reconstruction in matter of seconds on a modern Desktop PC even without any particular further optimization.

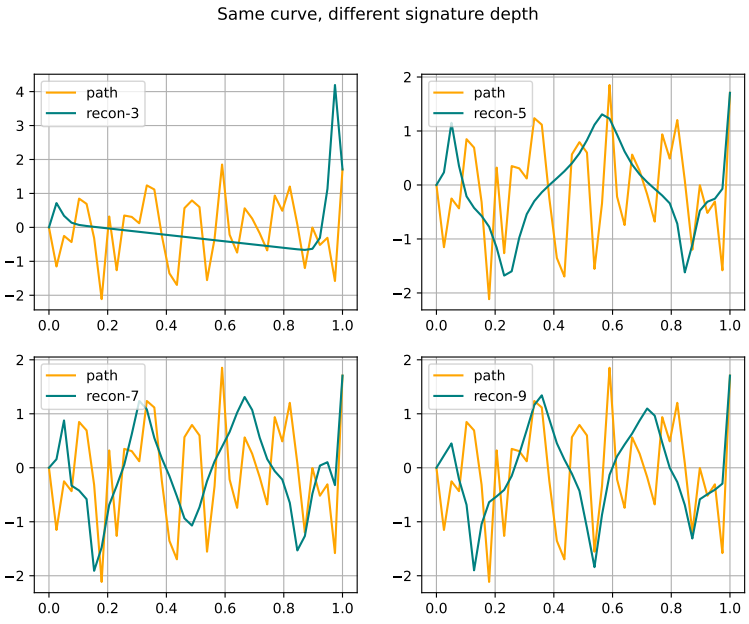


Figure 6.2: Reconstructing the same curve using signatures of different levels, 3, 5, 7 and 9. The path errors decrease from 52% to 45%, with improvements especially from a qualitative viewpoint. On the other hand, the errors between the *signatures* of the reconstructed curves and the original are always less than 7% (not plotted).

6.3 Shape analysis: overview

The goal of the following sections will be to study how the signature transform behaves when applied on some intuitive "typical" shapes, like sinusoidal, straight lines and Gaussian impulses. We are interested in checking how the signature measure their differences, as well as and in observing their reconstructions.

In all cases (if not otherwise specified) the paths are discretized with a total of 40 points, the reconstruction trained for 5000 epochs with a final mean square error less than 0.01. The signature is truncated at level 5 producing arrays of length $p = 62$. The curves are also normalized so to have values between -1 and 1 to improve readability.

Similarly to what done for stochastic processes, we measure the similarity between two curves $\hat{X}$ and $\hat{Y}$ by looking and the norm $\|S(\hat{X}) - S(\hat{Y})\|_{\mathbb{R}^{62}}$ (there is no "expectation" since the curves are deterministic). To intuitively interpret the produced number, we use the following heuristic idea.

We sampled 100.000 couples (X, Y) of paths starting from zero, with all the remaining values uniformly sampled from $[-1, 1]$. We looked at the mean value of the quantity $\|S(\hat{X}) - S(\hat{Y})\|$, resulting to be $\tau = 0.594$ +- 0.003 (very stable upon repeated simulations).

Therefore when the signature between two curves is higher than this value, we interpret as they would differ "more than the average", and the other way around. Alternatively, one can use again a relative percentage distance. We preferred the use of the absolute norm so to point out an alternative approach. This is of course *not* rigorous, but can be surely helpful in view of concrete data and applications.

Each example always shows three curves respectively numbered as 1, 2, 3. We report their signature distances (with interpretations with respect to the threshold τ) and observe whether they can be reconstructed from their signatures.

6.4 Shape analysis: straight lines

In this example we observe the signatures of a line of the form $X(t) = (t, \alpha t)$ with $\alpha = 0, 0.5, 2$. The first case where the curve is constant zero is of easy interpretation. The effects in the signature will be only given by the time t which will follow the predicted factorial decay.

The cases with $\alpha = 0.5$ and $\alpha = 2$ are related since one can be obtained from the other by a simple scalar multiplication in the second coordinate. Therefore the mixed signature terms between the two curves must be the same, up to a coefficient starting with value $4 = \frac{2}{0.5}$ and factorially increasing with the signature's depth. The effect is that the behavior observed for the case $\alpha = 0.5$ is now "amplified".

If we only look at their signatures, the three curves differ reciprocally as described in the table. The curves 1 and 2 differ "a bit less that the average", while all the other cases much more. This might be interpreted as the fact

that the signature is very sensitive to scalar multiplication, as we know from its definition directly.

Curves	Signature distance	Compared to τ	Similar?
1-2	0.648	1.1 τ	likely not
1-3	3.840	6.5 τ	not
2-3	3.350	5.6 τ	not

6.5 Shape analysis: sinusoidal curves

In this example we take a sinusoid with increasing frequencies $(2, 4$ and $8)$. Attached there are the plot for the reconstruction, as well as the signature distances listed in the usual table.

Curves	Signature distance	Compared to τ	Similar?
1-2	0.290	0.49 τ	likely
1-3	0.432	0.73 τ	likely
2-3	0.148	0.25 τ	likely

This example requires a further remark. For the second and third curve, we needed to increase the training time, and set the adaptive gradient descent with 20.000 epochs. This was actually not enough, since the loss function for the curve reconstruction was finally around 0.03 and not lower our desired benchmark of 0.01. We have an interpretation for this behavior. Let's focus on the second component of the augmented curve, therefore the *sin()* itself. The first sinusoid (here called "simple") has a certain signature $S(X)$. The curve Y corresponding to the sinusoid with *doubled* frequency can be interpreted as a time-concatenation of two simple sinusoids! Thanks to the Chen's formula, we know that:

$$S(Y) = S(X) \otimes S(X) \tag{6.3}$$

If we think of signatures as polynomials, the tensor product can be seen as polynomial products. When polynomials of lower degree are multiplied, the results is a polynomial with a higher one. Similarly, when the tensor product between two signatures is performed, it is possible that coefficients corresponding to a level previously very low have now some higher value, making the previously truncation choice less effective.

In other words, a deeper truncation level could be required to preserve the information about periodicity. This idea is partially validated in the attached plot, where the signature level is now 6 (instead of 5), the final loss function goes lower than 0.01. On the other hand, further tests should be performed if there is an actual interest in linking the periodicity of a curve to the depth of its signature truncation from a more general viewpoint.

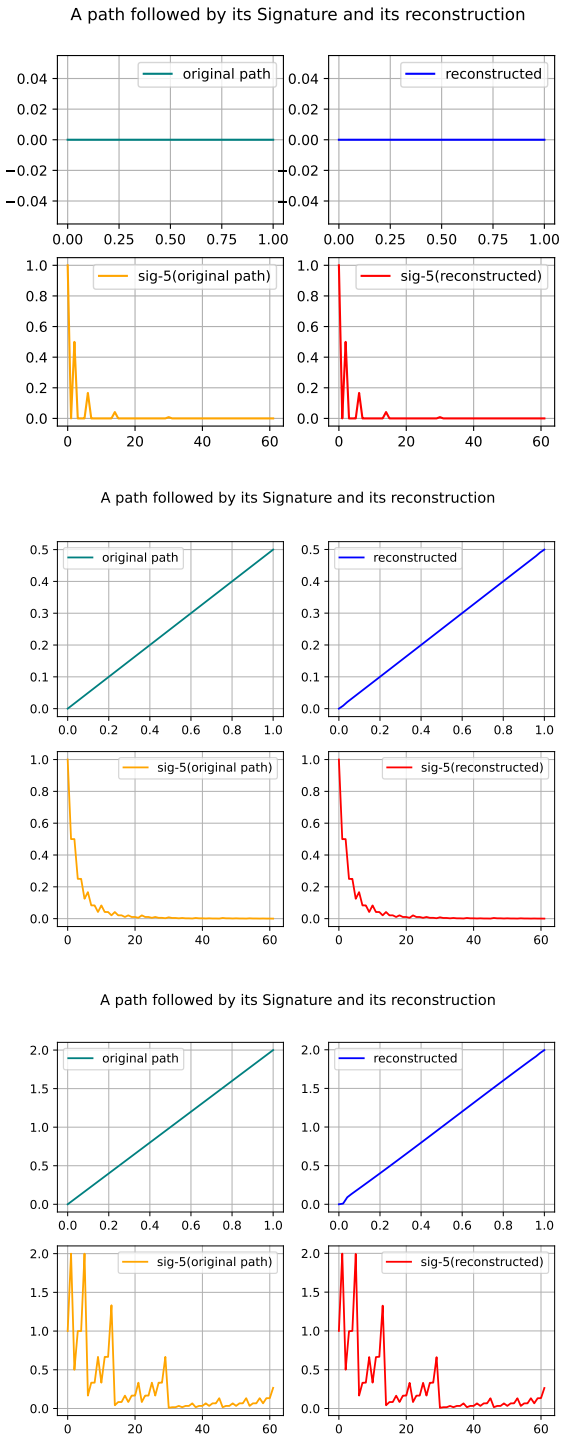


Figure 6.3: Signature analysis for straight lines. The three plots refer to the cases $\alpha = 0, 0.5$ and 2 respectively.

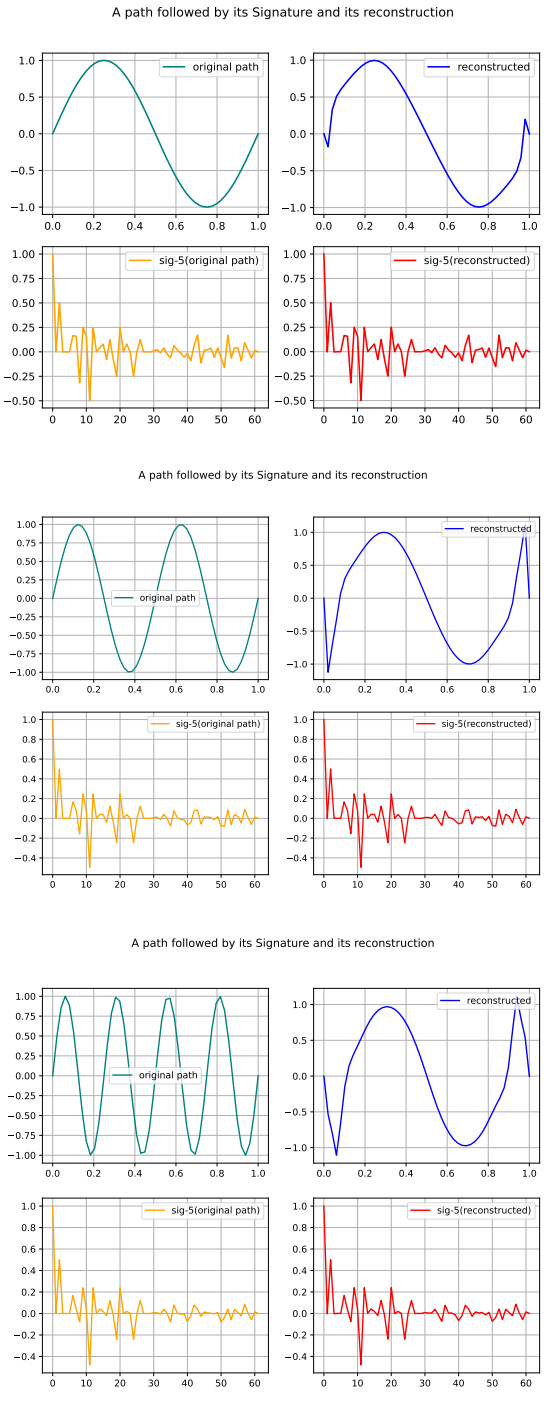


Figure 6.4: Signature reconstruction for three sinusoidal curves with increasing frequencies.

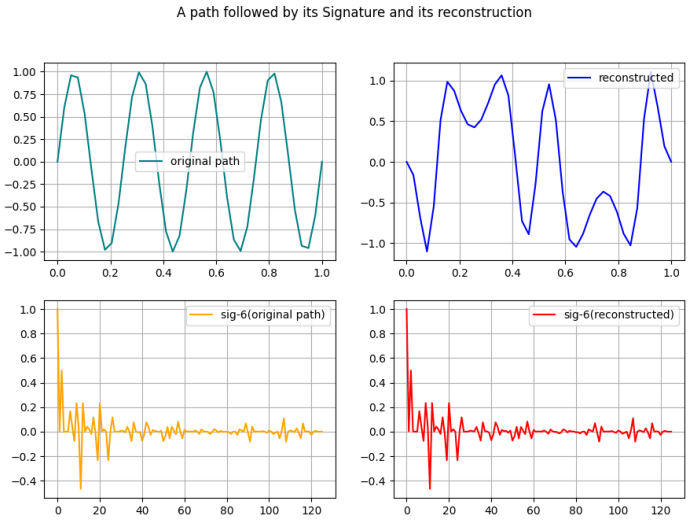


Figure 6.5: Choosing the signature at the deeper level 6 allows to reconstruct the periodicity far better than before. This can be a consequence of the Chen's formula transforming paths concatenations in algebraic tensor products.

6.6 Shape analysis: impulses

In this section we analyze the typical Gaussian impulse. Let's ignore for a moment the time component and focus on the second coordinate only. Note that each impulse is composed by three phases. A flat zero phase, the spike, and then the flat phase again. Therefore we always have a concatenation of these three steps. Their difference is only in how much they last, but since the signature is invariant under time reparametrization, these three phases will always have the same signature and so their composition!

In this example we can particularly appreciate the introduction of the first coordinate t, capable of "keeping track" of possible reparametrizations and therefore ensuring injectivity.

In conclusion, it is of no surprise that all three curves have similar signatures, as described in the following table.

Curves	Signature distance	Compared to τ	Similar?
1-2	0.027	0.045 τ	yes
1-3	0.047	0.079 τ	yes
2-3	0.027	0.045 τ	yes

It is certainly interesting to see how the signatures are yes extremely close, but still manage to convey information about where the "peak" is located by looking the reconstruction plots.

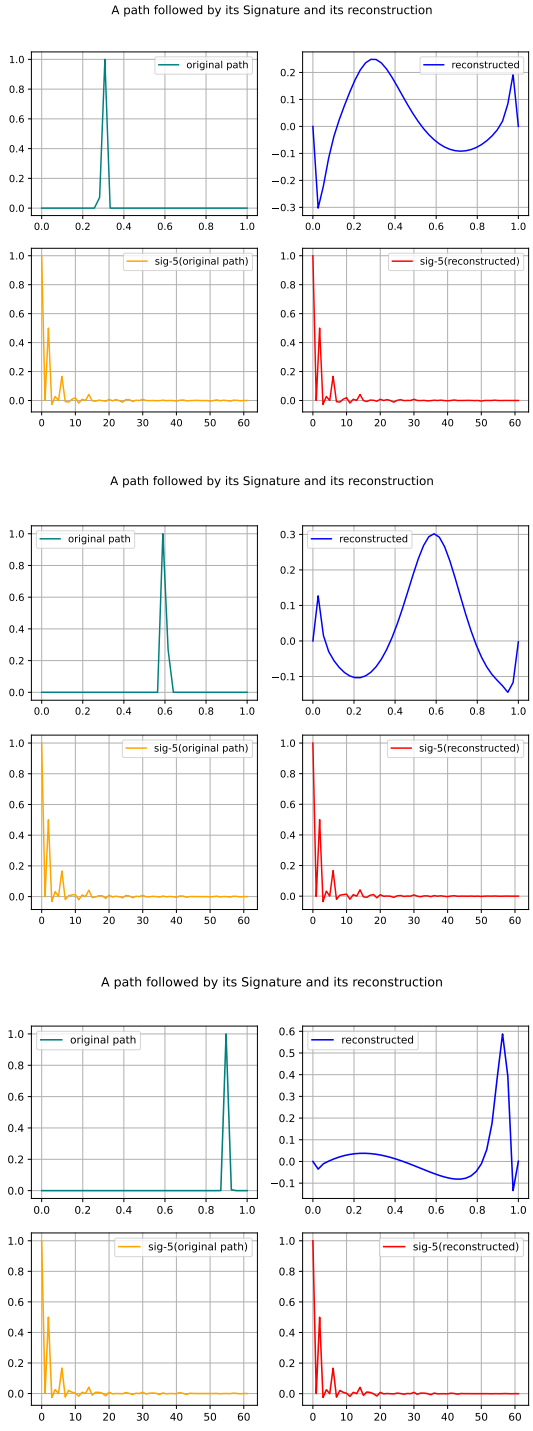


Figure 6.6: Despite being the signatures always very close to each other, each time it is possible to approximately understand "where" the impulse peak is originally located.

It is also interesting to observe how the coefficients after position 20 seem to be essentially irrelevant. Impulses are not periodic at all, and this fact is potentially compatible with the previous observation that the need of a deeper truncation might be connected to the presence of massive periodicity.

With this example we conclude our shape analysis. The next section will be devoted to the generation of artificial time series with a simple algorithm merging together the past remarks on stochastic processes and the technique for the signature inversion explained in this chapter.

6.7 Generation of artificial data

The generation of artificial synthetic data is nowadays a common challenge in multiple fields. We propose a highly experimental technique based on the use of the signature transform. We assume that a collection of equidistributed time series $\{X_i\}$ is available. The goal it to generate a new set of data $\{Y_j\}$ such that it is "reasonably to believe" it originated from the same source as the X_i.

There is on purpose some ambiguity in this description, since the precise requirements on Y_j usually vary depending on the field of application. For instance one can provide a specific list of statistical tests to be passed, as done in the paper [WKKK19] ("stylized facts" at page 3) where some examples in quantitative finance are studied. On that regard, we also recommend the reading [NSW+20]. The two papers together provide nice examples of using the signature transform for the problem of data generation combined with deep neural networks.

In this section we follow another approach, but in retrospective there are some similarities that we point out later.

Recall that for two suitable stochastic processes X and Y the expected signature is able to characterize their distributions (see our previous chapters for a more precise statement). Our algorithm directly follows from that property:

1. compute the expected signature of the samples $\{X_i\}$, call it S_X;
2. generate a set of tensors, $\{T_j\}$ such that their average coincides with S_X;
3. for each T_j, construct a path Z_j by using the inversion algorithm described at the beginning of this chapter;
4. the set $\{Z_j\}$ constitutes the resulting artificial data.

Theoretically speaking, the random paths $\{Z_j\}$ will have by construction the same expected signature of $\{X_i\}$, following therefore the same random distribution thanks to the characterization theorem.

On the other hand, there are multiple limitations when running this idea into practice.

First of all, we of course have only a finite number of samples and therefore the average estimations are subjected to an approximation error.

Then, the signature computation must be truncated at some level. In general it is possible for two difference processes to share the same expected signature up to a certain finite level, differing only later. This fact constitutes an obstacle, but on the other hand truncating the signature to a "reasonable" level is enough to preserve statistical similarities, which is at the end the final goal in this context.

The next inconvenience is about how to generate the random tensors T_j. In our experiment, we perturb componentwise the tensor S_X with a centered Gaussian of a fixed standard variation: the resulting tensors are likely *not* to be signatures of any path (the signature is not surjective), but nevertheless when the inversion is performed the produced paths will have a signature not so different from the starting tensors values. Ultimately the signature average is preserved up to some error that is well monitored.

Finally, the experience with inversion in the previous sections told us how paths constructed in this way are likely to be "smoother" and less oscillating, which is also a factor to consider.

It's now time to describe in details our experiment.

As in many previous chapters, we choose to work with log-returns of geometric Brownian motions, in a dataset composed by 100 time series of lengths 30, with $\mu = 1.$ and $\sigma = 0.4$.

The signatures are truncated at level 6, and once their average S_X is computed, is then perturbed 50 times componentwise with a centered Gaussian with standard deviation 0.02.

For each perturbation a path is constructed and stored, for a total of 50 artificial time series. The average signature of them is computed again and compared with the reference S_X. The closer they are, the better, since we can expect a good preservation of statistical similarities.

Finally, in order to estimate the errors on the paths themselves, the following test is performed. For (log-returns of) geometric Brownian motions, the expected path follows the simple evolution $t \mapsto (\mu - \frac{\sigma^2}{2})t$. Therefore, both the averages of $\{X_j\}$ and the artificial $\{Z_j\}$ are compared to that reference. The simulated have an mean relative error of 6%, while the generated of 12%.

In order to provide also a visual feedback, six paths (three original, three generated) are also plotted so to support an intuitive understanding.

The results of the experiment have some interesting consequences. If it is true that the errors and the shapes are possibly too far from something realistic, on the other hand the proposed algorithm is simple and has a lot of room for improvements. For instance it is known that the signature coefficients are *not* independent to each other (as we also remarked before), and more refined ways to perturb the tensor S_X can be consequently developed.

In retrospective, the paper mentioned before can (essentially) be interpreted as a way to use deep neural networks to match the expected signatures similarly as we did here. In conclusion, the results are overall very encouraging and we genuinely believe there is a potential in this direction of research.

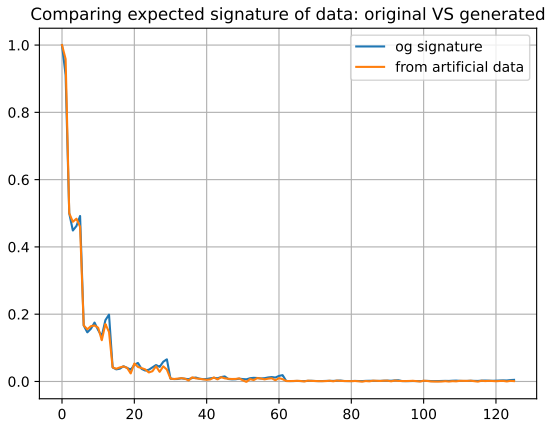


Figure 6.7: The difference between two expected signatures. The first computed using the original dataset, the second using the artificial data. The closer they are, the better. Their mean relative error is around 30%. Confidence intervals (small enough) are omitted.

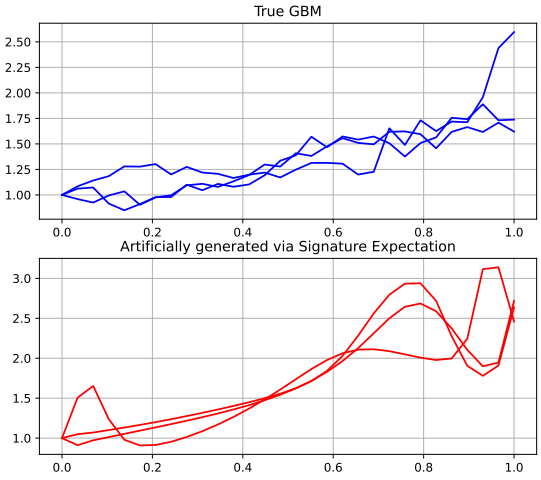


Figure 6.8: Three examples of original paths, and three artificial. When compared to the theoretical expected path, the original differ by 6% while the generated by 12%.

Chapter 7

Conclusion

We conclude this work by adding a list of resources that the reader can find helpful to extend the knowledge of the signature transform in connection to other areas of mathematics (different from the already mentioned *rough path theory*):

- we used the signature in the continuous case, always using piecewise linear interpolation. The paper [CPSF22] extends the study to **jump processes**, aiming at more realistic financial applications;
- in the paper [AFS18] the authors focus on paths described by polynomials. By connecting their zeroes with the signature transform, the notion of *signature varietes* is introduced building a potential bridge with **algebraic geometry**;
- the problem of reconstructing a curve given a signature is the main topic in [Gen15], where the focus is on solid theoretical developments;
- we used more classic machine learning approaches (multidimensional scaling, k-means clustering,...), but a lot can be done also in the setting of **neural networks**. Recurrent networks are one of the modern ways to model time series. They can be seen ad a Euler discretization of continuous *neural ODEs*, on which the signature transform can help in numerical stability. See for instance the papers [MSK$^+$20] and [BFT22];
- the paper [KO19] focuses on building **kernels** using the signature transform. In particular it establishes a methodical way such that, starting with some data endowed with a "static" kernel, such a kernel can be then extended on *sequences* of these data by using the signature transform. The continuous case, as well as the discrete one are carefully analyzed. According to this viewpoint, the difference of the expected signatures between two processes (that we extensively used) connects very well with the notion of *maximum mean discrepancy*. The signature kernel can also be connected to **partial differential equations** as explained in [SCF$^+$20].

Bibliography

[AFS18] Carlos Améndola, Peter Friz, and Bernd Sturmfels. Varieties of signature tensors. *Forum of Mathematics, Sigma 7 (2019) e10*, 2018.

[Ber23] Dimitri P. Bertsekas. *A Course in Reinforcement Learning*. Athena Scientific, 2023.

[BFT22] Christian Bayer, Peter K. Friz, and Nikolas Tapia. Stability of deep neural networks via discrete rough paths. *SIAM Journal on Mathematics of Data Science 5(1), 2023, pp 50-76*, 2022.

[CK16] Ilya Chevyrev and Andrey Kormilitzin. A primer on the signature method in machine learning. *ArXiv e-prints*, 2016.

[CL13] Ilya Chevyrev and Terry Lyons. Characteristic functions of measures on geometric rough paths. *Annals of Probability, Volume 44, Number 6 (2016), 4049-4082*, 2013.

[CPSF22] Christa Cuchiero, Francesca Primavera, and Sara Svaluto-Ferro. Universal approximation theorems for continuous functions of càdlàg paths and lévy-type signature models. *ArXiv e-prints*, 2022.

[ES20] Thomas Viehmann Eli Stevens, Luca Antiga. *Deep Learning with PyTorch*. Manning Publications Co., 2020.

[Fer20] Adeline Fermanian. Functional linear regression with truncated signatures. *ArXiv e-prints*, 2020.

[FV10] Peter K. Friz and Nicolas B. Victoir. *Multidimensional Stochastic Processes as Rough Paths*. Cambridge University Press, feb 2010.

[Gen15] Xi Geng. Reconstruction for the signature of a rough path. *ArXiv e-prints*, 2015.

[Hac19] Wolfgang Hackbusch. *Tensor Spaces and Numerical Tensor Calculus*. Springer International Publishing, 2019.

[KL20] Patrick Kidger and Terry Lyons. Signatory: differentiable computations of the signature and logsignature transforms, on both cpu and gpu. *ArXiv e-prints, published at ICLR 2021*, 2020.

[KO19] Franz J. Király and Harald Oberhauser. Kernels for sequentially ordered data. *Journal of Machine Learning Research 20 (2019) 1-45*, 2019.

[KPS94] Peter E. Kloeden, Eckhard Platen, and Henri Schurz. *Numerical Solution of SDE Through Computer Experiments*. Springer Berlin Heidelberg, 1994.

[Kre98] Rainer Kress. *Numerical Analysis*. Springer, 1998.

[LLN13] Daniel Levin, Terry Lyons, and Hao Ni. Learning from the past, predicting the statistics for the future, learning an evolving system. *ArXiv e-prints*, 2013.

[LM22] Terry Lyons and Andrew D. McLeod. Signature methods in machine learning. *ArXiv e-prints*, 2022.

[Lyo07] Lévy Lyons, Caruana. *Differential Equations Driven by Rough Paths*. Springer, 2007.

[MSK+20] James Morrill, Cristopher Salvi, Patrick Kidger, James Foster, and Terry Lyons. Neural rough differential equations for long time series. *Proceedings of the 38 th International Conference on Machine Learning, PMLR 139, 2021.*, 2020.

[NSW+20] Hao Ni, Lukasz Szpruch, Magnus Wiese, Shujian Liao, and Baoren Xiao. Conditional sig-wasserstein gans for time series generation. *ArXiv e-prints*, 2020.

[RSS18] Andrew G. Barto Richard S. Sutton. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[SCF+20] Cristopher Salvi, Thomas Cass, James Foster, Terry Lyons, and Weixin Yang. The signature kernel is the solution of a goursat pde. *SIAM Journal on Mathematics of Data Science*, 2020.

[Shr04] Steven Shreve. *Stochastic Calculus for Finance II: Continuous-Time Models*. Springer Finance, 2004.

[Var01] Varadhan. *Probability Theory*. Courant lecture notes, 2001.

[WKKK19] Magnus Wiese, Robert Knobloch, Ralf Korn, and Peter Kretschmer. Quant gans: Deep generation of financial time series. *Quantitative Finance, 2020*, 2019.