Open Access | https://doi.org/10.25080/DXWY3560

# RoughPy

Streaming data is rarely smooth

**Sam Morley**[1], and **Terry Lyons**[1]

[1]University of Oxford

SciPy 2024

July 8 - July 14, 2024

Proceedings of the 23rd Python in Science Conference
ISSN: 2575-9752

**Abstract**

Rough path theory is a branch of mathematics arising out of stochastic analysis. One of the main tools of rough path analysis is the signature, which captures the evolution of an unparametrised path including the order in which events occur. This turns out to be a useful tool in data science applications involving sequential data. RoughPy is our new Python package that aims change the way we think about sequential streamed data, by viewing it through the lens of rough paths. In RoughPy, data is wrapped in a stream object which can be composed and queried to obtain signatures that can be used in analysis. It also provides a platform for further exploration of the connections between rough path theory and data science.

**Keywords** sequential data, unparametrised paths, time series, rough paths, signatures, data science, machine learning, signature kernels, Log-ODE method

## 1. Introduction

Sequential data appears everywhere in the modern world: text, finance, health records, radio (and other electromagnetic spectra), sound (and speech), etc. Traditionally, these data are tricky to work with because of the exponential complexity and different scales of the underlying process. Until recently, with the development of transformers and large language models, it has been difficult to capture the long-term pattern whilst also capturing the short-term fine detail. Rough path theory gives us tools to work with sequential, ordered data in a mathematically rigorous way, which should provide a means to overcome some of the inherent complexity of the data. In this paper, we introduce a new package *RoughPy* for working with sequential data through the lens of rough path theory, where we can perform rigourous analyses and explore different ways to understand sequential data.

Rough paths arise in the study of *controlled differential equations* (CDEs), which generalise ordinary differential equations (ODEs) and stochastic differential equations [1], [2]. These are equations of the form $\mathrm{d}Y_t = f(Y_t, \mathrm{d}X_t)$, subject to an initial condition $Y_0 = y_0$, that model a non-linear system driven by a input path $X$. One simple CDE turns out to be critical to the theory:

$$\mathrm{d}S_t = S_t \otimes \mathrm{d}X_t \qquad S_0 = \mathbf{1}. \tag{1}$$

The solution of this equation is called the *signature* of $X$. It is analogous to the exponential function for ODEs, in that the solution of any CDE can be expressed in terms of the signature of the driving path. When the path $X$ is sufficiently regular, the signature can be computed directly as a sequence of iterated integrals. In other cases, we can still solve CDEs if we are given higher order data that can be used in place of the iterated integrals. A path equipped with this higher order data is called a *rough path*.

The signature turns out to be a useful summary of sequential data. It captures the order of events but not the necessarily the rate at which these events occur. The signature is robust to irregular sampling and provides a fixed-size view of the data, regardless of how many

**Published** Jul 10, 2024

**Correspondence to**
Sam Morley
sam.morley@maths.ox.ac.uk

observations are used to compute it. This means the signature can be a useful feature map to be used in machine learning for sequential data. There are numerous examples of using signatures of analysing sequential data outlined in Section 2.

Besides signatures, there are two other rough path-based methods that have found their way into data science in recent years. These are the signature kernel and neural CDEs. Both of these enjoy the same robustness of the signature, and expand the range of applications of rough path-based methods. We give a short overview of these methods in Section 1.2.

There are several Python packages for computing signatures of sequential data, including `esig` [3], `iisignature` [4], and `signatory` [5]. These packages provide functions for computing signatures from raw, structured data presented in an $n \times d$ array, where $d$ is the dimension of the stream and $n$ is the number of samples. This means the user is responsible for interpreting the data as a path and arranging the computations that need to be done.

RoughPy is a new package for working with sequential data and rough paths. The design philosophy for this package is to shift the emphasis from simply computing signatures on data to instead work with streams. A *stream* is a view of some data as if it were a rough path, that can be queried over intervals to obtain a signature. The actual form of the data is abstracted away in favour of stream objects that closely resemble the mathematics. The aim is to change the way that users think about sequential data and advance the understanding of path-like data analysis.

On top of the streams, RoughPy also provides concrete implementations for elements of the various algebras associated with rough path analysis. These include free tensor algebras, shuffle tensor algebras, and Lie algebras (Section 1.1). This allows the user to easily manipulate signatures, and other objects, in a more natural manner. This allows us to quickly develop methods by following the mathematics.

The paper is organised as follows. In the remainder of this section, we give a brief overview of the mathematics associated with rough path theory, and provide some additional detail for the signature kernel and neural CDEs. In Section 2 we list several recent applications of signatures and rough path-based methods in data science applications. These applications should serve to motivate the development of RoughPy. Finally, in Section 3 we give a more detailed overview of the RoughPy library, the types and functions it contains, and give an example of how it can be used.

RoughPy is open source (BSD 3-Clause) and available on GitHub https://github.com/datasig-ac-uk/roughpy.

## 1.1. Mathematical background

In this section we give a very short introduction to signatures and rough path theory that should be sufficient to inform the discussion in the sequel. For a far more comprehensive and rigorous treatment, we refer the reader to the recent survey [6]. For the remainder of the paper, we write $V$ for the vector space $\mathbb{R}^d$, where $d \geq 1$.

A *path* in $V$ is a continuous function $X : [a, b] \to V$, where $a < b$ are real numbers. For the purposes of this discussion, we shall further impose the condition that all paths are of bounded variation. The value of a path $X$ at some parameter $t \in [a, b]$ is denoted $X_t$.

The signature of $X$ is an element of the *(free) tensor algebra*. For $n \geq 0$, the $n$th tensor power of $V$ is defined recursively by $V^{\otimes 0} = \mathbb{R}$, $V^{\otimes 1} = V$, and $V^{\otimes n+1} = V \otimes V^{\otimes n}$ for $n > 1$. For example, $V^{\otimes 2}$ is the space of $d \times d$ matrices, and $V^{\otimes 3}$ is the space of $d \times d \times d$ tensors. The *tensor algebra over* $V$ is the space

$$\mathrm{T}((V)) = \left\{ \boldsymbol{x} = (x_0, x_1, \ldots) : x_j \in V^{\otimes j} \, \forall j \geq 0 \right\} \tag{2}$$

equipped with the tensor product $\otimes$ as multiplication. The tensor algebra is a *Hopf algebra*, and comes equipped with an antipode operation $\alpha_V : \mathrm{T}((V)) \to \mathrm{T}((V))$. It contains a group $\mathrm{G}(V)$ of elements under tensor multiplication and the antipode. The members of $\mathrm{G}(V)$ are called *group-like* elements. For each $n \geq 0$, we write $\mathrm{T}^n(V)$ for the *truncated tensor algebra* of degree $n$, which is the space of all $\boldsymbol{x} = (x_0, x_1, \ldots)$ such that $x_j = 0$ whenever $j > n$. Similarly, we write $\mathrm{T}^{>n}((V))$ for the subspace of elements $\boldsymbol{x} = (x_0, x_1, \ldots)$ where $x_j = 0$ whenever $j \leq n$, which is an ideal in $\mathrm{T}((V))$ and $\mathrm{T}^n(V) = \mathrm{T}((V))/\mathrm{T}^{>n}((V))$. The truncated tensor algebra is an algebra, when given the *truncated tensor product*, obtained by truncating the full tensor product.

The signature $\mathrm{S}(X)_{s,t}$ of a path $X : [a,b] \to V$ over a subinterval $[s,t) \subseteq [a,b]$ is $\mathrm{S}(X)_{s,t} = \left(1, \mathrm{S}_1(X)_{s,t}, \ldots\right) \in \mathrm{G}(V)$ where for each $m \geq 1$, $\mathrm{S}_m(X)_{s,t}$ is given by the iterated (Riemann-Stieltjes) integral

$$\mathrm{S}_m(X)_{s,t} = \underset{s<u_1<u_2<\ldots<u_m<t}{\int \ldots \int} \mathrm{d}X_{u_1} \otimes \mathrm{d}X_{u_2} \otimes \ldots \otimes \mathrm{d}X_{u_m}. \tag{3}$$

The signature respects concatenation of paths, meaning $\mathrm{S}(X)_{s,t} = \mathrm{S}(X)_{s,u} \otimes \mathrm{S}(X)_{u,t}$ for any $s < u < t$. This property is usually called *Chen's relation*. Two paths have the same signature if and only if they differ by a *tree-like path* [7]. The signature is translation invariant, and it is invariant under reparametrisation.

The *dual* of $\mathrm{T}((V))$ is the *shuffle algebra* $\mathrm{Sh}(V)$. This is the space of linear functionals $\mathrm{T}((V)) \to \mathbb{R}$ and consists of sequences $(\lambda_0, \lambda_1, \ldots)$ with $\lambda_k \in (V^*)^{\otimes k}$ and where $\lambda_k = 0$ for all $k$ larger than some $N$. (Here $V^*$ denotes the dual space of $V$. In our notation $V^* \cong V$.) The multiplication on $\mathrm{Sh}(V)$ is the *shuffle product*, which corresponds to point-wise multiplication of functions on the path. Continuous functions on the path can be approximated (uniformly) by shuffle tensors acting on $\mathrm{G}(V)$ on the signature. This is a consequence of the Stone-Weierstrass theorem. This property is sometimes referred to as *universal non-lineararity*.

There are several *Lie algebras* associated to $\mathrm{T}((V))$. Define a *Lie bracket* on $\mathrm{T}((V))$ by the formula $[\boldsymbol{x}, \boldsymbol{y}] = \boldsymbol{x} \otimes \boldsymbol{y} - \boldsymbol{y} \otimes \boldsymbol{x}$, for $\boldsymbol{x}, \boldsymbol{y} \in \mathrm{T}((V))$. We define subspaces $L_m$ of $\mathrm{T}((V))$ for each $m \geq 0$ inductively as follows: $L_0 = \{\mathbf{0}\}$, $L_1 = V$, and, for $m \geq 1$,

$$L_{m+1} = \mathrm{span}\{[\boldsymbol{x}, \boldsymbol{y}] : \boldsymbol{x} \in V, \boldsymbol{y} \in L_m\}. \tag{4}$$

The space of formal Lie series $\mathcal{L}(V)$ over $V$ is the subspace of $\mathrm{T}((V))$ containing sequences of the form $(\ell_0, \ell_1, \cdots)$, where $\ell_j \in L_j$ for each $j \geq 0$. Note that $\mathcal{L}(V) \subseteq \mathrm{T}^{>0}(V)$. For any $\boldsymbol{x} \in \mathrm{T}(V)$ we define

$$\exp(\boldsymbol{x}) = \sum_{n=0}^{\infty} \frac{\boldsymbol{x}^{\otimes n}}{n!} \quad \text{and} \quad \log(\mathbf{1} + \boldsymbol{x}) = \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{n} \boldsymbol{x}^{\otimes n}. \tag{5}$$

For any path $X$, we have $\mathrm{LogSig}(X)_{s,t} := \log\left(\mathrm{S}(X)_{s,t}\right) \in \mathcal{L}(V)$, and we call this the *log-signature* of $X$ over $[s,t)$. This is an alternative representation of the path, but doesn't enjoy the same unievrsal non-linearity of the signature.

## 1.2. Rough paths in data science

Now we turn to the applications of rough path theory to data science. Our first task is to form a bridge between sequential data and paths. Consider a finite, ordered sequence $\{(t_1, \boldsymbol{x}_1, \ldots, t_N, \boldsymbol{x}_N)\}$ of observations, where $t_j \in \mathbb{R}$, and $\boldsymbol{x}_j \in V$. (More generally, we might consider $\boldsymbol{x}_j \in \mathcal{L}(V)$ instead. That is, data that already contains higher-order information. In our language, it is a genuine rough path.) We can find numerous paths that interpolate these observations; a path $X : [t_0, t_N] \to V$ such that, for each $j$, $X_{t_j} = \boldsymbol{x}_j$. The simplest interpolation is to take the path that is linear between adjacent observations.

Once we have a path, we need to be able to compute signatures. For practical purposes, we truncate all signatures (and log-signatures) to a particular degree $M$, which we typically call the *depth*. The dimension of the ambient space $d$ is usually called the *width*. Using linear interpolation, we can compute the iterated integrals explicitly using a free tensor exponential of the difference of successive terms:

$$\mathrm{Sig}^M\big([t_j, t_{j+1})\big) = \exp_M\big(\boldsymbol{x}_{j+1} - \boldsymbol{x}_j\big) := \sum_{j=0}^{M} \frac{1}{j!}\big(\boldsymbol{x}_{j+1} - \boldsymbol{x}_j\big)^{\otimes j}. \tag{6}$$

Here, and in the remainder of the paper, we shall denote the empirical signature over an interval $I$ by $\mathrm{Sig}(I)$ and the log-signature as $\mathrm{LogSig}(I)$. We can compute the signature over arbitrary intervals by taking the product of the these terms, using the multiplicative property of the signature.

### 1.2.1. *The signature transform:*

Most of the early applications of rough paths in data science, the (truncated) signature was used as a feature map [8]. This provides a summary of the path that is independent of the parameterisation and the number of observations. Unfortunately, the signature grows geometrically with truncation depth. If $d > 1$, then the dimension of $\mathrm{T}^M(V)$ is

$$\sum_{m=0}^{M} d^m = \frac{d^{M+1} - 1}{d - 1} \tag{7}$$

The size of the signature is a reflection of the complexity of the data, where data with a higher complexity generally needs a higher truncation level and thus a larger signature. It is worth noting that this still represents a significant compression of stream information in many cases.

For some applications, it might be possible to replace the signature with the log-signature. The log-signature is smaller than the signature, but we lose the universal non-linearity property of the signature. Alternatively, we might turn to other techniques that don't require a full calculation of the signature (such as the signature kernel below). As the connection between rough paths and data science becomes more mathematically mature, we will likely find new ways to use the signature without requiring its full size.

### 1.2.2. *Signature kernels:*

Kernel methods are useful tools for learning with sequential data. Mathematically, a *kernel* on a set $W$ is a positive-definite function $k : W \times W \to \mathbb{R}$. Kernels are often quite easy to evaluate because of the kernel trick, which involves embedding the data in a inner product space, with a feature map, in which the kernel can be evaluated by simply taking an inner product. Informally, kernels measure the similarity between two points. They are used in a variety of machine learning tasks such as classification.

The *signature kernel* is a kernel induced on the space of paths by combining the signature with an inner product defined on the tensor algebra [9]. The theory surrounding the signature kernel has been expanded several times since their introduction [10], [11]. Typically, the inner product on $\mathrm{T}((V))$ will itself by derived from an inner product on $V$, extended to the tensor algebra.

Signatures are infinite objects, so we can't simply evaluate inner products on the tensor algebra. Fortunately, we can approximate the signature kernel by taking inner products of truncated signatures. Even better, it turns out that, in certain cases, the signature kernel can be realised as the solution to a partial differential equation (PDE) of Goursat type. This means the full signature kernel can be computed from raw data without needing to compute full signatures [12].

In fact, in recent preprint, it has been shown that there are higher order solvers for signature kernels by rewriting the kernel solution of a system of PDEs of Goursat type [13]. A critical part of their method involves the adjoint of both left and right free tensor multiplication, which are not available in any current package for computing signatures. These functions are provided by RoughPy.

#### 1.2.3. *Neural controlled differential equations:*

Neural CDEs are a method for modelling irregular time series. We consider CDEs of the form

$$\mathrm{d}Y_t = f_\theta(Y_t)\,\mathrm{d}X_t \tag{8}$$

where $f_\theta$ is a neural network. We can treat the path $Y$ as "hidden state" that we can tune using data to understand the relationship between the driving path $X_t$ and some response. Neural CDEs can be regarded as a continuous-time analogue of a recurrent neural network [14].

Neural CDEs initially showed some promising results on several benchmarks but now lag behind current state-of-the-art approaches to time series modelling. The latest iteration of neural CDEs are the recently introduced Log-neural controlled differential equations [15], which make use of the *Log-ODE* method for solving rough differential equations in order to boost the performance of neural CDEs.

## 2. Current applications of rough paths

In this section we enumerate several applications where rough paths have been used to develop or improve methods. This list presented here is certainly not exhaustive. In addition to the literature cited below, there are numerous additional references and worked examples, in the form of Jupyter notebooks, available on the DataSig website (https://datasig.ac.uk/examples).

### *2.1. Detecting interference in radio astronomy data*

Radio frequency interference (RFI) is a substantial problem in the field of radio astronomy. Even small amounts of RFI can obscure the faint signals generated by distant stellar objects and events. The problem of identifying RFI in a signal falls into a class of semi-supervised learning tasks called *novelty* (or *anomaly*) *detection*. Rough path methods have been applied to develop a novelty detection framework based on rough path methods to detect RFI in radio astronomy data from several radio telescopes [16]. Their result show that their framework is effective at detecting even faint RFI within the test data. This work is based on a general novelty detection framework [17].

Signatures kernels have also been used for a similar problem of detecting malware by inspecting the streaming tree of processes on a computer system T. Cochrane, P. Foster, V. Chhabra, M. Lemercier, T. Lyons, and C. Salvi [18]. Their method uses a support vector machine classifier to identify processes that are malicious compared to "normal" behaviour learned via training on a corpus of normality.

### *2.2. Tracking mood via natural language processing*

One application of rough paths in natural language processing has been in the domain of mental health [19], [20]. In this work, the authors present a model for identifying changes in a person's mood based on their online textual content. Many mental health conditions have symptoms that manifest in the (textual) expression, so this could be a powerful tool for mental health professionals to identify changes in patients and intervene before the state develops. Their model achieves state-of-the-art performance vs existing models on two datasets.

### 2.3. *Predicting battery cell degradation*

Another recent application of signatures is to predict the degradation of lithium-ion cells [21]. They use signature features to train a model that can accurately predict the end of life of a cell using relatively low-frequency sampling compared to existing models. They also observed that the performance at higher frequency was comparable to other models.

### 2.4. *Prediction of sepsis in intensive care data*

One of the first effective demonstrations of the utility of signatures and rough paths based methods in healthcare was in the 2019 PhysioNet challenge [22]. In this contest, teams were invited to develop models to predict sepsis in patients from intensive care unit data. In this challenge, a team utilising signatures to enhance predictive power placed first in the official phase of the challenge. Since then, signatures and other rough path based approaches have been used in several other clinical contexts [20], [23], [24]. Clinical data is often irregularly sampled and often exhibits a high degree of missingness, but it can also be very high-frequency and dense. Rough path based methods can handle these data in an elegant way, and retain the structure of long and short term dependencies within the data.

### 2.5. *Human action recognition*

The task of identifying a specific action performed by a person from a short video clip is very challenging. Signatures derived from landmark data extracted from the video has been used to train classification models that achieved state-of-the-art performance compared with contemporary models [25], [26], [27]. (See also preprint papers [28], [29].) Also in the domain of computer vision, signatures have been used to produce lightweight models for image classification [30] and in handwriting recognition tasks [31].

## 3. RoughPy

RoughPy is a new library that aims to support the development of connections between rough path theory and data science. It represents a shift in philosophy from simple computations of signatures for sequential data, to a representation of these data as a rough path. The design objectives for RoughPy are as follows:

1. provide a class that presents a rough path view of some source of data as a rough path, exposing methods for querying the data over intervals to get a signature or log-signature;
2. provide classes and functions that allow the users to interact with the signatures and other algebraic objects in a natural, mathematical manner;
3. all operations should be differentiable and objects should be interoperable with objects from machine learning, such as TensorFlow (JAX) and PyTorch.

The first two objectives are simple design and implementation problems. The final objective presents the most difficulty, especially interoperability between RoughPy and common machine learning libraries. There are array interchange formats for NumPy-like arrays, such as the Python Array API standard [32] and the DLPack protocol [33]. These provide part of the picture, but in order for them to be fully supported, RoughPy must support a variety of compute backends such as CUDA (NVidia), ROCm/HIP (AMD), and Metal (Apple).

RoughPy is a substantial library with numerous components, mostly written in C++ with a Python interface defined using Pybind11 [34]. The original design of the library closely followed the C++ template libraries libRDE and libalgebra [35], although it has seen many iterations since.

In the remainder of this section, we discuss some of the core components of RoughPy, give an example of using RoughPy, and discuss the future of RoughPy.

## *3.1. Free tensors, shuffle tensors, and Lie objects*

In order to properly support rough path based methods and allow users to write code based on mathematical concepts, we provide realisations of several algebra types. The algebras provided in RoughPy are `FreeTensor`, `ShuffleTensor`, and `Lie`, which define elements of a particular free tensor algebra, shuffle tensor algebra, and Lie algebra respectively. Each of these algebras is initialized with a width, depth, and scalar coefficient type, encapsulated in a `Context` object.

In addition to the algebra classes, RoughPy provides a number of supporting functions, including antipodes and half-shuffle products for `FreeTensor`/`ShuffleTensor` objects, and adjoint operators for left free tensor multiplication. These are operations that are frequently used in the theory of rough paths, and will likely be necessary in developing new applications later (as in the signature kernels).

RoughPy algebras are designed around a flexible scalar ring system that allows users to perform calculations with different accuracy, or derive expressions by using polynomial coefficients. For most applications, single or double precision floating point numbers will provide a good balance between performance and accuracy. (Double precision floats are the default.) When more precision is required, rational coefficients can be used instead. These are backed by GMP rationals for fast, arbitrary precision rational arithmetic [36]. Polynomial coefficients can be used to derive formulae by performing calculations. This is a powerful technique for understanding the terms that appear in the result, particularly whilst testing and debugging.

## *3.2. Intervals*

RoughPy is very careful in the way it handles intervals. All intervals in RoughPy are half-open, meaning that they include one end point but not the other; they are either *clopen* $[a, b) := \{t : a \le t < b\}$ or *opencl* $(a, b] := \{t : a < t \le b\}$. Besides the type (clopen or opencl), all intervals must provide methods for retrieving the infimum ($a$ in the above notation) and the supremum ($b$ above) of the interval as double precision floats. This is enforced by means of an abstract base class `Interval`. The main concrete interval types are `RealInterval`, an interval with arbitrary real endpoints, and `DyadicInterval`, as described below. For brevity, we shall only consider clopen intervals.

A *dyadic interval* is an interval $D_k^n := [k/2^n, (k+1)/2^n)$, where $k, n$ are integers. The number $n$ is often described as the *resolution* of the interval. The family of dyadic intervals of a fixed resolution $n$ partition the real line so that every real number $t$ belongs to a unique dyadic interval $D_n^k$. Moreover, the family of all dyadic intervals have the property that two dyadic intervals are either disjoint or one contains the other (including the possibility that they are equal).

In many cases, RoughPy will granularise an interval into a dyadic intervals. The *dyadic granularisation* of $[a, b)$ with resolution $n$ is $[k_1/2^n, k_2/2^n)$ where $k_1 = \max\{k : k/2^n \le a\}$ and $k_2 = \max\{k : k/2^n \le b\}$. In effect, the dyadic granularisation is the result of “rounding” each end point to the included end of the unique dyadic interval that contain it.

### *3.3. Streams*

Streams are central to RoughPy. A RoughPy `Stream` is a rough path view of some underlying data. It provides two key methods to query the object over intervals to retrieve either a signature or log-signature. Importantly, once constructed, the underlying data is inaccessible except by querying via these methods. `Stream`s are designed to be composed in various ways, such as by concatenation, in order to build up more complex streams. A `Stream` is actually a (type-erasing) wrapper around a more minimal `StreamInterface` abstract class.

We construct streams by a factory function associated with each different `StreamInterface`, which might perform some compression of the underlying data. For example, a basic `StreamInterface` is the `LieIncrementStream`, which can be constructed using the associated `from_increments` factory function (a static method of the class), which accepts an $n \times d$ array of *increment data*. These data will typically be the differences between successive values of the data (but could also include higher-order Lie terms). This is similar to the way that libraries such as `esig`, `iisignature`, and `signatory` consume data.

RoughPy streams cache the result of log-signature queries over dyadic intervals so they can be reused in later calculations. To compute the log-signature over any interval $I$, we granularise at a fixed stream resolution $n$ to obtain the interval $\tilde{I} = [k_1/2^n, k_2/2^n)$, and then compute

$$\mathrm{LogSig}(\tilde{I}) = \log\left(\prod_{k=k_1}^{k_2-1} \exp(\mathrm{LogSig}(D_k^n))\right). \tag{9}$$

The $\mathrm{LogSig}(D_k^n)$ terms on the right-hand-side are either retrieved from the cache, or computed from the underlying source. This is essentially the Campbell-Baker-Hausdorff formula applied to the log-signatures at the finest level. In practice, we can actually reduce the number of terms in the product, by merging complementary dyadic intervals that appear in the granularisation. We further optimise by using a fused multiply-exponential ($A\exp(B)$) operation.

Signatures are always computed by first computing the log-signature and then exponentiating. Directly computing the signature as a product of exponentials of (cached) log-signatures might accumulate enough numerical errors to drift slightly from a group-like tensor. That is, the result might not actually be a true signature. Taking the logarithm and then exponentiating back to obtain the signature has the effect of correcting this numerical drift from a true signature.

Aside from the basic `LieIncrementStream`, there are several other implementations of the `StreamInterface` currently available in RoughPy. The `BrownianStream` approximates Brownian motion by generating normal distributed increments over dyadic intervals of arbitrary resolution on demand, forming a reasonable approximation of true Brownian motion. The `ExternalDataStream` is an interface for loading data from various external sources, such as from a database or specialised data format. Currently, only sound files are supported but we plan to extend support for other sources as the need arises. This will certainly include "online" data sources such as computer peripheral devices (e.g. microphones).

The other main `StreamInterface` implementation is the `PiecewiseAbelianStream`, which is an important construction from CDE. A piecewise Abelian path, or log-linear path, is an example of a *smooth rough path*, which generalises piecewise linear approximations of an arbitrary stream. Formally, an *Abelian path* $Y$ is a pair $([a,b), \boldsymbol{y})$ where $a < b$ and $\boldsymbol{y} \in \mathcal{L}(V)$. The log-signature over an arbitrary interval $[u,v) \subseteq [a,b)$ is given by

$$\mathrm{LogSig}(Y)_{u,v} = \frac{v-u}{b-a}\boldsymbol{y}. \tag{10}$$

A *piecewise Abelian path* is the concatenation of finitely many Abelian paths with adjacent intervals. For any rough path $X$ and partition $\{a = t_0 < t_1 < \ldots < t_N = b\}$ there is a piecewise Abelian approximation for this path given by

$$\left\{\left(\left[t_{j-1}, t_j\right), \mathrm{LogSig}(X)_{t_{j-1},t_j}\right) : j = 1, \ldots, N\right\}. \tag{11}$$

This construction turns out to be vital for computing signature kernels [32] and for solving CDEs [2], [15]. In particular, this construction can be used to compress data at some degree, which can the be used in computations at a higher degree.

### 3.4. *Example*

In this section we show a very simple example of how to use RoughPy to construct a stream and compute a signature. This example is similar to the first few steps of the tutorial found in the RoughPy documentation. RoughPy can be installed using `pip`, where prebuilt wheels are available for Windows, Linux, and MacOs:

```
pip install roughpy
```

We refer the reader to this documentation for much more detail. We will construct a stream in $\mathbb{R}^{26}$ by taking each letter in a word, “scipy” in this example, as the increments of a path:

```
import numpy as np

text = "scipy"
increments = np.zeros((5, 26), dtype="int8")
for i, c in enumerate(text):
    increments[i, ord(c) - 97] = 1
```

Now we import RoughPy and construct a `Stream` using the factory mentioned above. One other critical ingredient is the algebra `Context`, which is used to set up a consistent set of algebra objects with the desired width (26), truncation level (2), and coefficient type (`Rational`).

```
import roughpy as rp

ctx = rp.get_context(width=26, depth=2,
      coeffs=rp.Rational)
stream = rp.LieIncrementStream.from_increments(
         increments, ctx=ctx)
```

Now we can compute the signature of the stream over the whole domain of the stream $[0, 4]$ by omitting the interval argument:

```
sig = stream.signature()
print(sig)
# { 1() 1(3) 1(9) 1(16) 1(19) 1(25) 1/2(3,3)
#   1(3,9) 1(3,16) 1(3,25) 1/2(9,9) 1(9,16)
#   1(9,25) 1/2(16,16) 1(16,25) 1(19,3) 1(19,9)
#   1(19,16) 1/2(19,19) 1(19,25) 1/2(25,25) }
```

The first term of the signature is always 1, and the empty parentheses indicate the empty tensor word. The next five terms correspond to the counts of each unique letter that appears, the number in parentheses indicates the letter (with `a` being 1). The final terms indicate the order in which each pair of letters appear in the word. For instance, the term `1(3,9)` indicates that a `c` appears before an `i`.

**Remark 1**.

It turns out that most words in the English language can be distinguished using only their level 2 signatures. The first level signatures groups words into anagrams. The second level signature counts the occurrences of each ordered pair of letters. There are relatively few words that require level 3 data. From the standard Linux dictionary, containing around 80,000 words, there are two pairs of words that require the level 3 terms: “toot” and “otto”, and “naan” and “anna”. This is shown in the RoughPy documentation. Similar patterns can be observed in other languages too, including French, Spanish, German, Russian, and Lithuanian.

This is only the beginning of the story. From here, we can use the signatures to compute the similarity between streams, via the signature kernel for instance, or used as features in a variety of machine learning problems. More detailed examples of how to use signatures in data science are given on the DataSig website https://datasig.ac.uk/examples.

### *3.5. The future of RoughPy*

RoughPy is continuously evolving. At time of writing, the current version uses libalgebra and libalgebra-lite (libalgebra with fewer templates) for computations. Unfortunately, this made it difficult to achieve the differentiability and computation device support that we want. We are currently changing the way we implement vectors and algebras to provide the support for on-device computation that we want. Making the operations differentiable is crucial for machine learning, and will be the biggest challenge.

Long term, we need to expand support for signature kernels and CDEs. As applications of these tools grow in data science, we will need to devise new methods for computing kernels, or solving CDEs. We will also build a framework for constructing and working with linear maps, and homomorphisms. For example, one very useful linear map is the extension of the $\log$ function to the whole tensor algebra.

## 4. Conclusions

The use of rough path theory in data science is rapidly expanding and provides a different way to view sequential data. Signatures, and other methods arising from rough path theory, are already used in a wide variety of applications, with great effect. The next steps in overcoming the difficulty in modeling sequential data will require a change of perspective. Viewing these data through the lens of rough path theory might provide this change.

RoughPy is a new Python library for working with streamed data using rough path methods. It is designed to abstract away the form and source of data so that analysis can be performed by querying path-like objects. This approach is much closer to the mathematics. It also allows users to interact with the various algebras associated with rough paths (free tensor algebra, shuffle tensor algebra, Lie algebra) in a natural way. RoughPy is under active development, and a long list of improvements and extensions are planned.

## References

[1] T. J. Lyons, “Differential equations driven by rough signals.,” *Revista Matemática Iberoamericana*, vol. 14, no. 2, pp. 215–310, 1998, [Online]. Available: http://eudml.org/doc/39555

[2] T. J. Lyons, M. Caruana, and T. Lévy, *Differential Equations Driven by Rough Paths:´Ecole d'Eté de Probabilités de Saint-Flour XXXIV - 2004*. Springer Berlin Heidelberg, 2007. doi: 10.1007/978-3-540-71285-5.

[3] T. Lyons and D. Maxwell, “esig.” 2017.

[4] J. F. Reizenstein and B. Graham, “Algorithm 1004: The Iisignature Library: Efficient Calculation of Iterated-Integral Signatures and Log Signatures,” *ACM Transactions on Mathematical Software*, vol. 46, no. 1, pp. 1–21, 2020, doi: 10.1145/3371237.

[5] P. Kidger and T. Lyons, “Signatory: differentiable computations of the signature and logsignature transforms, on both CPU and GPU.” [Online]. Available: https://arxiv.org/abs/2001.00706

[6] T. Lyons and A. D. McLeod, “Signature Methods in Machine Learning.” [Online]. Available: https://arxiv.org/abs/2206.14674

[7] B. Hambly and T. Lyons, “Uniqueness for the signature of a path of bounded variation and the reduced path group,” *Annals of Mathematics*, vol. 171, no. 1, pp. 109–167, 2010, doi: 10.4007/annals.2010.171.109.

[8] P. Kidger, P. Bonnier, I. Perez Arribas, C. Salvi, and T. Lyons, “Deep Signature Transforms,” in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, Eds., 2019. doi: 10.48550/arXiv.1905.08494.

[9] F. J. Kiraly and H. Oberhauser, “Kernels for Sequentially Ordered Data,” *Journal of Machine Learning Research*, vol. 20, no. 31, pp. 1–45, 2019, doi: 10.48550/arXiv.2102.03657.

[10] A. Fermanian, P. Marion, J.-P. Vert, and G. Biau, “Framing RNN as a kernel method: A neural ODE approach,” in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021, pp. 3121–3134. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2021/file/18a9042b3fc5b02fe3d57fea87d6992f-Paper.pdf

[11] T. Cass, T. Lyons, and X. Xu, “Weighted signature kernels,” *The Annals of Applied Probability*, vol. 34, no. 1A, 2024, doi: 10.1214/23-aap1973.

[12] C. Salvi, T. Cass, J. Foster, T. Lyons, and W. Yang, “The Signature Kernel Is the Solution of a Goursat PDE,” *SIAM Journal on Mathematics of Data Science*, vol. 3, no. 3, pp. 873–899, 2021, doi: 10.1137/20m1366794.

[13] M. Lemercier and T. Lyons, “A High Order Solver for Signature Kernels.” [Online]. Available: https://arxiv.org/abs/2404.02926

[14] P. Kidger, J. Morrill, J. Foster, and T. Lyons, “Neural Controlled Differential Equations for Irregular Time Series,” in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020, pp. 6696–6707. doi: 10.48550/arXiv.1906.08215.

[15] B. Walker, A. D. McLeod, T. Qin, Y. Cheng, H. Li, and T. Lyons, “Log Neural Controlled Differential Equations: The Lie Brackets Make a Difference,” 2024, doi: 10.48550/ARXIV.2402.18512.

[16] P. Arrubarrena, M. Lemercier, B. Nikolic, T. Lyons, and T. Cass, “Novelty Detection on Radio Astronomy Data using Signatures.” [Online]. Available: https://arxiv.org/abs/2402.14892

[17] Z. Shao, R. S.-Y. Chan, T. Cochrane, P. Foster, and T. Lyons, “Dimensionless Anomaly Detection on Multivariate Streams with Variance Norm and Path Signature.” [Online]. Available: https://arxiv.org/abs/2006.03487

[18] T. Cochrane, P. Foster, V. Chhabra, M. Lemercier, T. Lyons, and C. Salvi, “SK-Tree: a systematic malware detection algorithm on streaming trees via the signature kernel,” in *2021 IEEE International Conference on Cyber Security and Resilience (CSR)*, 2021, pp. 35–40. doi: 10.1109/csr51186.2021.9527933.

[19] T. Tseriotou, A. Tsakalidis, P. Foster, T. Lyons, and M. Liakata, “Sequential Path Signature Networks for Personalised Longitudinal Language Modeling,” in *Findings of the Association for Computational Linguistics: ACL 2023*, 2023, pp. 5016–5031. doi: 10.18653/v1/2023.findings-acl.310.

[20] T. Tseriotou *et al.*, “Sig-Networks Toolkit: Signature Networks for Longitudinal Language Modelling,” in *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, N. Aletras and O. De Clercq, Eds., St. Julians, Malta, 2024, pp. 223–237. doi: 10.48550/arXiv.2312.03523.

[21] R. Ibraheem, Y. Wu, T. Lyons, and G. dos Reis, “Early prediction of Lithium-ion cell degradation trajectories using signatures of voltage curves up to 4-minute sub-sampling rates,” *Applied Energy*, vol. 352, p. 121974, 2023, doi: 10.1016/j.apenergy.2023.121974.

[22] J. H. Morrill, A. Kormilitzin, A. J. Nevado-Holgado, S. Swaminathan, S. D. Howison, and T. J. Lyons, “Utilization of the Signature Method to Identify the Early Onset of Sepsis From Multivariate Physiological Time Series in Critical Care Monitoring,” *Critical Care Medicine*, vol. 48, no. 10, pp. e976–e981, 2020, doi: 10.1097/ccm.0000000000004510.

[23] S. N. Cohen *et al.*, “Subtle variation in sepsis-III definitions markedly influences predictive performance within and across methods,” *Scientific Reports*, vol. 14, no. 1, 2024, doi: 10.1038/s41598-024-51989-6.

[24] G. Falcioni, A. Georgescu, E. Molimpakis, L. Gottlieb, T. Kuhn, and S. Goria, “Path Signature Representation of Patient-Clinician Interactions as a Predictor for Neuropsychological Tests Outcomes in Children: A Proof of Concept,” in *2023 IEEE International Conference on Medical Artificial Intelligence (MedAI)*, 2023, pp. 1–9. doi: 10.1109/medai59581.2023.00008.

[25] W. Yang, T. Lyons, H. Ni, C. Schmid, and L. Jin, “Developing the Path Signature Methodology and Its Application to Landmark- Based Human Action Recognition,” in *Stochastic Analysis, Filtering, and Stochastic Optimization*, Springer International Publishing, 2022, pp. 431–464. doi: 10.1007/978-3-030-98519-6_18.

[26] J. Cheng *et al.*, "Skeleton-Based Gesture Recognition With Learnable Paths and Signature Features," *IEEE Transactions on Multimedia*, vol. 26, pp. 3951–3961, 2024, doi: 10.1109/tmm.2023.3318242.

[27] S. Liao, T. Lyons, W. Yang, K. Schlegel, and H. Ni, "Logsig-RNN: a novel network for robust and efficient skeleton-based action recognition," 2021. doi: 10.48550/arXiv.2110.13008.

[28] M. R. Ibrahim and T. Lyons, "FaceTouch: Detecting hand-to-face touch with supervised contrastive learning to assist in tracing infectious disease," 2023, doi: 10.48550/ARXIV.2308.12840.

[29] L. Jiang, W. Yang, X. Zhang, and H. Ni, "GCN-DevLSTM: Path Development for Skeleton-Based Action Recognition." [Online]. Available: https://arxiv.org/abs/2403.15212

[30] M. R. Ibrahim and T. Lyons, "ImageSig: A signature transform for ultra-lightweight image recognition," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2022, pp. 3648–3658. doi: 10.1109/cvprw56347.2022.00409.

[31] Z. Xie, Z. Sun, L. Jin, H. Ni, and T. Lyons, "Learning Spatial-Semantic Context with Fully Convolutional Recurrent Network for Online Handwritten Chinese Text Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 8, pp. 1903–1917, 2018, doi: 10.1109/tpami.2017.2732978.

[32] A. Meurer *et al.*, "Python Array API Standard: Toward Array Interoperability in the Scientific Python Ecosystem," in *Proceedings of the 22nd Python in Science Conference*, in SciPy. 2023, pp. 8–17. doi: 10.25080/gerudo-f2bc6f59-001.

[33] DLPack, "Open In Memory Tensor structure." 2023.

[34] W. Jakob, J. Rhinelander, and D. Moldovan, "pybind11 – Seamless operability between C++11 and Python." 2017.

[35] S. Buckley *et al.*, "CoRoPa – Computational Rough Paths project." 2006.

[36] T. Granlund and the GMP development team, "GNU MP: The GNU Multiple Precision Arithmetic Library," 2012.