# Calculation of Iterated-Integral Signatures and Log Signatures

Jeremy Reizenstein, Centre for Complexity Science*
University of Warwick

May 2015, revision of May 2016

**Abstract**

We explain the meaning and calculation of the log signature of a path, and explain some accompanying software tools which demonstrate it.

## 1 Introduction

An iterated integral signature (or Brownian signature) is a sequence of numbers derived from a continuous path. The signature is important in the theory of differential equations driven by rough paths.

When trying to classify types of paths with a machine learning algorithm, if each path is specified as a long list of points then it can be hard for paths which are similar to be seen as similar by the algorithm. If instead the space is discretized and the path is specified in terms of the points hit, then the representation will be very large and sparse, which will make learning slow. The signature may be a good way to represent a path to avoid these problems. Similar smooth curves have similar signatures. The signature is invariant on translations of the path. It is also invariant on reparametrizing the path.

If two paths are different then their signatures will be different, unless one contains a section where the path exactly retraces itself.[5] A signature cannot distinguish between two paths where the only difference is that one path contains an extra section which is backtracked over. In many situations, the relevant paths will not contain precise backtracking. It is possible that approximate backtracking might lead to numerically unstable signatures. If there is a dimension that is monotone along each path (for example time) then backtracking is impossible.

The log signature is a compressed form of the signature, which is more complicated to define. For the same amount (number of levels) of the signature, the log signature contains the same information in fewer numbers. The link between the two is continuous.

This note is a self-contained explanation of the operations required to calculate the log signature of a piecewise-linear path, and an explanation of some python code which assists in so doing. Statements are made without proofs, and there is an emphasis on small examples and some explicit calculations. The main reference on the algebra is [9], and the application to paths is explained in chapter 2 of [7]. The signature and log signature are formally defined in Sections 5 and 6. Section 8 explains the code and how to use it. Section 10 describes a visualisation of the log signature which may be helpful.

## 2 Words

Consider a set $\Sigma = \{\mathbf{1}, \mathbf{2}, \ldots, \mathbf{d}\}$ of $d$ "letters" which has an ordering $<$. The set of words with entries in $\Sigma$ is called the *Kleene Star* of $\Sigma$ and is denoted by $\Sigma^*$. The length of a word $u$ is denoted $|u|$, for example $|\mathbf{3231}| = 4$. The empty word is denoted by $\epsilon$ and the concatenation of words $u$ and $v$ is written $uv$. If a word $w$ is equal to $uv$ for some words $u$ and $v$, then $u$ is said to be a *prefix* of $w$ and $v$ is said to

*Supported by the Engineering and Physical Sciences Research Council

be a *suffix* of $w$. If $u$ and $v$ are both not empty then they are said to be a *proper* prefix and suffix of $w$. For example $\mathbf{1}$ is a proper suffix of $\mathbf{3231}$, and a suffix but not a proper suffix of $\mathbf{1}$. The ordering $<$ on $\Sigma$ can be extended to an ordering $<_L$ on $\Sigma^*$ called *alphabetical order* or *lexicographic order* in the usual way. (Specifically: $\epsilon <_L u$ if $|u| > 0$. For letters $a$ and $b$ and words $u$ and $v$, $au <_L bv$ if $a < b$ or both $a = b$ and $u <_L v$.)

The *free (real) vector space* on the finite set $\Sigma$ is the real vector space with basis given by the elements of $\Sigma$. We will just call it $\mathbb{R}^d$. An element looks like $a_1\mathbf{1} + \cdots + a_d\mathbf{d}$ for real numbers $a_1, \ldots, a_d$.

The *tensor algebra* of the vector space $\mathbb{R}^d$, $T(\mathbb{R}^d)$, is the set of functions from $\Sigma^*$ to $\mathbb{R}$, or equivalently the set of (possibly) infinite formal sums of real multiples of words. The word $u$ in $\Sigma^*$ is identified with the function which takes $u$ to 1 and all other words to 0, or the expression $1u$. We are only ever interested in finite restrictions of this in order to do calculations, in particular we choose an integer $m$ and ignore all words with length longer than $m$. $T^{\underline{m}}(\mathbb{R}^d)$ is the real vector space with basis given by words of length $m$ or less[1]. The function on words which returns their concatenation if it has length $m$ or less and returns the zero element otherwise extends uniquely to a bilinear operation[2] on $T^{\underline{m}}(\mathbb{R}^d)$, which is called the *concatenation product*. For example, in $T^{\underline{4}}(\mathbb{R}^3)$,

$$(7\,\mathbf{132} + 9\epsilon)(2\,\mathbf{1} + 4\,\mathbf{21}) = 14\,\mathbf{1321} + 18\,\mathbf{1} + 36\,\mathbf{21}.$$

The exponential series defines a function from those elements of $T^{\underline{m}}(\mathbb{R}^d)$ which have no term in $\epsilon$ to $T^{\underline{m}}(\mathbb{R}^d)$:

$$\exp(x) := 1\epsilon + x + \frac{1}{2}x^2 + \cdots + \frac{1}{m!}x^m = \sum_{i=0}^{m} \frac{1}{i!}x^i \tag{1}$$

where powers denote the concatenation product of $x$ with itself. Stopping the sum at $m$ is the same as stopping later than $m$, because all higher powers are 0. $\exp(x)$ always has 1 as its $\epsilon$ component.

The series for $\log(1 + x)$ defines a function log from those elements of $T^{\underline{m}}(\mathbb{R}^d)$ which have $\epsilon$ component 1 to $T^{\underline{m}}(\mathbb{R}^d)$:

$$\log(1\epsilon + x) := x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \cdots + (-1)^{m+1}\frac{1}{m}x^m \tag{2}$$

There is never an $\epsilon$ component in the value of $\log(x)$. In fact, log and exp are inverses. For example, up to level 2,

$$\exp(\mathbf{1} + 3\mathbf{2}) = 1\epsilon + \mathbf{1} + 3\mathbf{2} + \frac{1}{2}\mathbf{11} + \frac{3}{2}(\mathbf{12} + \mathbf{21}) + \frac{9}{2}\mathbf{22}$$
$$\log(\exp(\mathbf{1} + 3\mathbf{2})) = \mathbf{1} + 3\mathbf{2}.$$

## 2.1 Lyndon words

Consider ordered repeating patterns with beads in $\Sigma$ like those in Figure 1. They are known as necklaces. We represent the pattern with the word of the letters forming the repeating cycle which follows the order of the arrows and comes first in alphabetical order. For the three necklaces shown, the repeating pattern is $\mathbf{1}$, $\mathbf{1223}$ and $\mathbf{132}$ respectively.

Not all words represent a repeating pattern in this setup. Words which do are called *Lyndon words*. For example, $\mathbf{123}$ is a Lyndon word but $\mathbf{21}$ is not. The empty word is not a Lyndon word. Every single-letter word is a Lyndon word. A word which is comprised of multiple copies of a smaller word is not a Lyndon word. A multi-letter word is a Lyndon word if and only if it comes before all its proper suffixes in alphabetical order. If $w = uv$ is a multi-letter Lyndon word and $v$ is the longest proper suffix of $w$ which is also a Lyndon word, then $u$ is a Lyndon word.

[1] this is known as $T^{(m)}(\mathbb{R}^d)$ in the notation of [7]
[2] i.e. linear in each argument

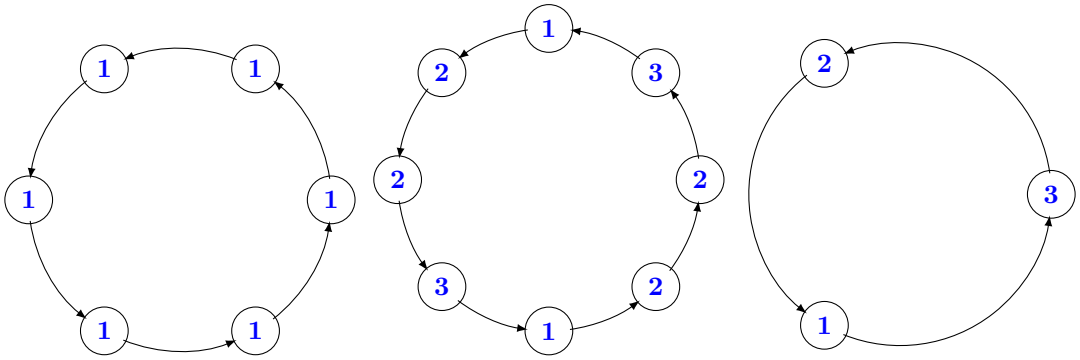



Figure 1: Three necklaces

## 3 Bracketed expressions

A bracketed expression on $\Sigma$ is an expression consisting of one or more letters combined with a binary $[\cdot\,,\cdot]$ operator called the Lie bracket. For example $[\mathbf{1}\,,[[\mathbf{2}\,,\mathbf{1}]\,,\mathbf{4}]]$ or $\mathbf{3}$ or $[\mathbf{2}\,,\mathbf{1}]$. The *foliage* of a bracketed expression is the word formed from its letters, ignoring its brackets and commas. The *depth* of a bracketed expression is the length of its foliage, which is equivalent to one plus the number of sets of brackets it contains. For instance, the foliage of $[\mathbf{1}\,,[[\mathbf{2}\,,\mathbf{1}]\,,\mathbf{4}]]$ is $\mathbf{1214}$ and its depth is 4. Note that the depth is not in general equal to one plus the maximum number of brackets surrounding a letter, for example consider $[[\mathbf{2}\,,\mathbf{1}]\,,[\mathbf{1}\,,\mathbf{4}]]$.

On the real vector space of linear combinations of bracketed expressions, we can extend the bracket operation to a bilinear operation. For example

$$\Big[\,(3\mathbf{1}+9\,[\mathbf{3}\,,\mathbf{2}]\,)\,,4\mathbf{2}\Big] = 12\,[\mathbf{1}\,,\mathbf{2}]+36\,[\,[\mathbf{3}\,,\mathbf{2}]\,,\mathbf{2}].$$

The *free Lie algebra* is this space but where we consider an element unchanged if it undergoes either of the following transformations.

$$[u\,,v]\mapsto -1[v\,,u] \qquad\qquad [u\,,[v\,,w]]\mapsto [[u\,,v]\,,w]+[[w\,,u]\,,v]$$

This means that for any elements $u, v, w$ of the free Lie algebra, denoting the zero element by 0, $[u,u]=0$, and $u$, $v$ and $w$ obey the *Jacobi identity*:

$$[[u\,,v]\,,w]+[[w\,,u]\,,v]+[[v\,,w]\,,u]=0. \tag{3}$$

For example, as elements of the free Lie algebra, the following two expressions are equal

$$\begin{aligned} &-13\,[\mathbf{1}\,,\mathbf{2}]+19\,[\mathbf{3}\,,[[\mathbf{1}\,,\mathbf{2}]\,,\mathbf{4}]\\ &\;\;13\,[\mathbf{2}\,,\mathbf{1}]+19\,[[\mathbf{3}\,,[\mathbf{1}\,,\mathbf{2}]]\,,\mathbf{4}]+19\,[[\mathbf{4}\,,\mathbf{3}]\,,[\mathbf{1}\,,\mathbf{2}.]] \end{aligned}$$

The addition and the multiplication by a scalar are well-defined on the equivalence classes. The free Lie algebra is a real vector space and the Lie bracket is still bilinear.

Level $m$ of the free Lie algebra is the subspace spanned by bracketed expressions of depth $m$. (This is well-defined as the transformations do not change the depth of an expression.) The free Lie algebra is the direct sum of its levels. We define the free Lie algebra up to level $m$, $F^{\underline{m}}(\Sigma)$, as the direct sum of the first $m$ levels. On $F^{\underline{m}}(\Sigma)$, we define the Lie bracket operation as the projection of the free Lie algebra's Lie bracket onto $F^{\underline{m}}(\Sigma)$.

We can define a linear map $\rho$ from the free Lie algebra to $T(\mathbb{R}^d)$ which takes letters to letters and $[u,v]$ to $\rho(u)\rho(v)-\rho(v)\rho(u)$. This definition is consistent, i.e. well defined on equivalence classes. For example $\rho([\mathbf{1}\,,[\mathbf{1}\,,\mathbf{2}]])=\mathbf{112}+\mathbf{211}-2\mathbf{121}$. An element of the image of $\rho$ is called a *lie element* and it never has a component in $\epsilon$. For each $m$, $\rho$ is an injection from $F^{\underline{m}}(\Sigma)$ to $T^{\underline{m}}(\mathbb{R}^d)$. We identify an element $u$ of the free Lie algebra with $\rho(u)$.

### 3.1 Lyndon basis

Having a basis of each level of the free Lie algebra is useful, for instance because it gives a canonical representation of each element. Bracketed expressions of depth $m$ span level $m$ of the free Lie algebra, so it is reasonable to expect a basis in terms of bracketed expressions. Famously, Lyndon words can be used to do this.[3]

Define a function $\sigma$ from Lyndon words to bracketed expressions recursively as follows:

- If $a \in \Sigma$, $\sigma(a) = a$
- If $w = uv$ is a multi-letter Lyndon word and $v$ is the longest proper suffix of $w$ which is a Lyndon word, then $\sigma(w) = [\sigma(u)\,, \sigma(v)]$

Then in fact the image under $\sigma$ of the set of Lyndon words of length $m$ forms a basis of level $m$ of the free Lie algebra. For example, Table 1 shows the elements of the Lyndon basis of $F^{\underline{4}}(\{\mathbf{1}, \mathbf{2}\})$.

| level | $u$ | $\sigma(u)$ |
|---|---|---|
| 1 | **1** | **1** |
| 1 | **2** | **2** |
| 2 | **12** | [**1** , **2**] |
| 3 | **112** | [**1** , [**1** , **2**]] |
| 3 | **122** | [[**1** , **2**] , **2**] |
| 4 | **1112** | [**1** , [**1** , [**1** , **2**]]] |
| 4 | **1122** | [[**1** , [**1** , **2**]] , **2**] |
| 4 | **1222** | [[[**1** , **2**] , **2**] , **2**] |

Table 1: The Lyndon words on two letters up to level 4, and their image under $\sigma$

If $u$ and $v$ are Lyndon words, then $[\,\sigma(u)\,, \sigma(v)\,]$ can be calculated in the Lyndon basis according to the following recursive procedure. (Use only the first relevant step at each stage.)

- If $u = v$, then $[\,\sigma(u)\,, \sigma(v)\,] := 0$
- If $v <_L u$, then $[\,\sigma(u)\,, \sigma(v)\,] := -\,[\,\sigma(u)\,, \sigma(v)\,]$
- If $|u| = 1$, then $[\,\sigma(u)\,, \sigma(v)\,] := \sigma(uv)$
- Let $u = xy$, with $y$ being the longest proper suffix of $u$ which is a Lyndon word. If $v <_L y$, then $[\,\sigma(u)\,, \sigma(v)\,] := [\,\sigma(y)\,, [\,\sigma(v)\,, \sigma(x)\,]\,] + [\,\sigma(x)\,, [\,\sigma(y)\,, \sigma(v)\,]\,]$
- $[\,\sigma(u)\,, \sigma(v)\,] := \sigma(uv)$

For example, $[\,\mathbf{2}\,, \sigma(\mathbf{13})\,]$ is $-\sigma(\mathbf{132})$, and $[\,\mathbf{3}\,, \sigma(\mathbf{12})\,]$ is $\sigma(\mathbf{132}) - \sigma(\mathbf{123})$. Having this procedure, we can do arithmetic between elements of the free Lie algebra cleanly storing all elements in terms of the Lyndon basis. Knowing the longest Lyndon proper suffix of a Lyndon word is seen to be important, and any computer program doing these calculations will be designed with that in mind.

## 4 The Baker-Campbell-Hausdorff formula

With log and exp defined according to the series above, we consider the expression $\log(\exp(\mathbf{1})\exp(\mathbf{2}))$ expanded up to a fixed level $m$. It is in fact a value in $F^{\underline{m}}(\{\mathbf{1}, \mathbf{2}\})$, in other words it is in the image of

[3] I found [6] to be the easiest introduction to this.

$\rho$. For example, up to level 2,

$$\begin{aligned}
\log(\exp(\mathbf{1})\exp(\mathbf{2})) &= \log\left((1\epsilon + \mathbf{1} + \tfrac{1}{2}\mathbf{11})(1\epsilon + \mathbf{2} + \tfrac{1}{2}\mathbf{22})\right) \\
&= \log(1\epsilon + \mathbf{1} + \mathbf{2} + \tfrac{1}{2}\mathbf{11} + \mathbf{12} + \tfrac{1}{2}\mathbf{22}) \\
&= (\mathbf{1} + \mathbf{2} + \tfrac{1}{2}\mathbf{11} + \mathbf{12} + \tfrac{1}{2}\mathbf{22}) - \tfrac{1}{2}(\mathbf{11} + \mathbf{22} + \mathbf{12} + \mathbf{21}) \\
&= \mathbf{1} + \mathbf{2} + \tfrac{1}{2}\mathbf{12} - \tfrac{1}{2}\mathbf{21} \\
&= \mathbf{1} + \mathbf{2} + \tfrac{1}{2}(\mathbf{12} - \mathbf{21}) \\
&= \mathbf{1} + \mathbf{2} + \tfrac{1}{2}[\mathbf{1}\,,\mathbf{2}]
\end{aligned}$$

Similarly up to level 3

$$\begin{aligned}
\log(\exp(\mathbf{1})\exp(\mathbf{2})) &= \log\left((1\epsilon + \mathbf{1} + \tfrac{1}{2}\mathbf{11} + \tfrac{1}{6}\mathbf{111})(1\epsilon + \mathbf{2} + \tfrac{1}{2}\mathbf{22} + \tfrac{1}{6}\mathbf{222})\right) \\
&= \log(1\epsilon + \mathbf{1} + \mathbf{2} + \tfrac{1}{2}\mathbf{11} + \mathbf{12} + \tfrac{1}{2}\mathbf{22} + \tfrac{1}{6}\mathbf{111} + \tfrac{1}{2}\mathbf{122} + \tfrac{1}{2}\mathbf{112} + \tfrac{1}{6}\mathbf{222}) \\
&= (\mathbf{1} + \mathbf{2} + \tfrac{1}{2}\mathbf{11} + \mathbf{12} + \tfrac{1}{2}\mathbf{22} + \tfrac{1}{6}\mathbf{111} + \tfrac{1}{2}\mathbf{122} + \tfrac{1}{2}\mathbf{112} + \tfrac{1}{6}\mathbf{222}) \\
&\quad - \tfrac{1}{2}(\mathbf{11} + \mathbf{22} + \mathbf{12} + \mathbf{21} + \mathbf{111} + \tfrac{3}{2}\mathbf{112} \\
&\qquad + \mathbf{121} + \tfrac{3}{2}\mathbf{122} + \tfrac{1}{2}\mathbf{211} + \mathbf{212} + \tfrac{1}{2}\mathbf{221} + \mathbf{222}) \\
&\quad + \tfrac{1}{3}(\mathbf{111} + \mathbf{112} + \mathbf{121} + \mathbf{122} + \mathbf{211} + \mathbf{212} + \mathbf{221} + \mathbf{222}) \\
&= \mathbf{1} + \mathbf{2} + \tfrac{1}{2}\mathbf{12} - \tfrac{1}{2}\mathbf{21} + \tfrac{1}{12}\mathbf{112} - \tfrac{1}{6}\mathbf{121} + \tfrac{1}{12}\mathbf{122} + \tfrac{1}{12}\mathbf{221} - \tfrac{1}{6}\mathbf{212} + \tfrac{1}{12}\mathbf{221} \\
&= \mathbf{1} + \mathbf{2} + \tfrac{1}{2}(\mathbf{12} - \mathbf{21}) \\
&\quad + \tfrac{1}{12}(\mathbf{1}(\mathbf{12} - \mathbf{21}) - (\mathbf{12} - \mathbf{21})\mathbf{1}) - \tfrac{1}{12}((\mathbf{21} - \mathbf{12})\mathbf{2} - \mathbf{2}(\mathbf{21} - \mathbf{12})) \\
&= \mathbf{1} + \mathbf{2} + \tfrac{1}{2}[\mathbf{1}\,,\mathbf{2}] + \tfrac{1}{12}[\mathbf{1}\,,[\mathbf{1}\,,\mathbf{2}]] - \tfrac{1}{12}[[\mathbf{2}\,,\mathbf{1}]\,,\mathbf{2}]
\end{aligned}$$

This calculation always works, although the algebraic manipulation gets much more tedious as $m$ increases. The expression up to an arbitrary number of terms is called the Baker-Campbell-Hausdorff (BCH) formula. It can be written in any basis for $F^{\underline{m}}(\{\mathbf{1},\mathbf{2}\})$. The coefficients can be calculated in the Lyndon basis using the method of [1]. The authors have saved the results up to level 20 a file called `bchLyndon20.dat` in [2], and in practice we just use their values. If $x$ and $y$ are arbitrary in $F^{\underline{m}}(\Sigma)$, then they can be substituted in as $\mathbf{1}$ and $\mathbf{2}$ to give an expression for $\log(\exp(x)\exp(y))$ as a member of $F^{\underline{m}}(\Sigma)$. For example, in $F^{\underline{4}}(\{\mathbf{1},\mathbf{2},\mathbf{3}\})$,

$$\log(\exp([\mathbf{1},\mathbf{2}])\exp(\mathbf{3})) = [\mathbf{1},\mathbf{2}] + \mathbf{3} + \tfrac{1}{2}[[\mathbf{1},\mathbf{2}],\mathbf{3}] - \tfrac{1}{12}[[\mathbf{3},[\mathbf{1},\mathbf{2}]],\mathbf{3}].$$

Here I could ignore the fourth level of the BCH formula because, seeing as all its terms will have a $\mathbf{1}$, I will get terms of depth 5 after the substitution.

# 5 Signature

A path in $\mathbb{R}^d$ can be described by a continuous map $\gamma : [a, b] \to \mathbb{R}^d$ with $\gamma(t) = (\gamma_{\mathbf{1}}(t), \gamma_{\mathbf{2}}(t), \ldots, \gamma_{\mathbf{d}}(t))$. Its *signature* is an element of $T(\mathbb{R}^d)$, denoted by $X^{\gamma}_{a,b}$. The component the signature in each word (its image on each word) is defined inductively on the length of the word. The signature of the empty word is defined as 1. If $w$ is a word and $i \in \{\mathbf{1}, \mathbf{2}, \ldots, \mathbf{d}\}$ then $X^{\gamma}_{a,b}(wi)$ is defined as $\int_a^t X^{\gamma}_{a,t}\gamma_i'(t)\,dt$. The restriction of the signature to words of length $m$ is called the $m$th **level** of the signature. It contains $d^m$ values. Written in Riemann-Stieltjes form, they are the values of integrals of the form

$$\int_a^b \int_a^{t_1} \cdots \int_a^{t_{m-2}} \int_a^{t_{m-1}} d\gamma_{i_1}(t_m)\, d\gamma_{i_2}(t_{m-1}) \ldots d\gamma_{i_{m-1}}(t_2)\, d\gamma_{i_m}(t_1), \tag{4}$$

where each $i_j$ is allowed to range over values in $\{\mathbf{1}, \mathbf{2}, \ldots, \mathbf{d}\}$. For instance, level 1 of the signature is the total displacement of the path. If $d \geq 3$ then the $\mathbf{231}$ component of the signature of is the value of the integral

$$\int_a^b \int_a^t \int_a^s \gamma_{\mathbf{2}}'(r)\,dr\,\gamma_{\mathbf{3}}'(s)\,ds\,\gamma_{\mathbf{1}}'(t)\,dt = \iiint_{a<r<s<t<b} \gamma_{\mathbf{2}}'(r)\gamma_{\mathbf{3}}'(s)\gamma_{\mathbf{1}}'(t)\,dV \tag{5}$$

$$= \int_a^b \int_a^t \int_a^s d\gamma_{\mathbf{2}}(r)\, d\gamma_{\mathbf{3}}(s)\, d\gamma_{\mathbf{1}}(t) \tag{6}$$

$$= \int_a^b \int_r^b \int_s^b d\gamma_{\mathbf{1}}(t)\, d\gamma_{\mathbf{3}}(s)\, d\gamma_{\mathbf{2}}(r). \tag{7}$$

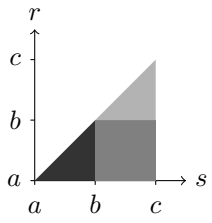
We are interested in the signature up to level $m$, $X^{\gamma,\underline{m}}_{a,b}$, which is an element of $T^{\underline{m}}(\mathbb{R}^d)$.

If $a < b < c$ then the result (from [3]) known as **Chen's identity** states that

$$X^{\gamma}_{a,c}(i_1 i_2 \ldots i_n) = \sum_{j=0}^{n} X^{\gamma}_{a,b}(i_1 i_2 \ldots i_j) X^{\gamma}_{b,c}(i_{j+1} i_{j+2} \ldots i_n). \tag{8}$$

(The products indicated with ellipses can be empty, indicating the empty word, on which any signature takes the value 1.) For example

$$\begin{aligned} X^{\gamma}_{a,c}(\mathbf{12}) &= \int_a^c \int_a^s \gamma_{\mathbf{1}}'(r)\,dr\,\gamma_{\mathbf{2}}'(s)\,ds \\ &= \left[\int_b^c \int_b^s + \int_b^c \int_a^b + \int_a^b \int_a^s\right] \gamma_{\mathbf{1}}'(r)\,dr\,\gamma_{\mathbf{2}}'(s)\,ds \\ &= X^{\gamma}_{b,c}(\mathbf{12}) + X^{\gamma}_{a,b}(\mathbf{1}) X^{\gamma}_{b,c}(\mathbf{2}) + X^{\gamma}_{a,b}(\mathbf{12}). \end{aligned}$$




Restricting up to level $m$, this means that

$$X^{\gamma,\underline{m}}_{a,c} = X^{\gamma,\underline{m}}_{a,b} X^{\gamma,\underline{m}}_{b,c}, \tag{9}$$

using the concatenation product in $T^{\underline{m}}(\mathbb{R}^d)$.

Also, if $\gamma$ is a straight line then

$$X^{\gamma}_{a,b}(i_1 i_2 \ldots i_n) = \frac{1}{n!} \prod_{j=1}^{n} (\gamma_{i_j}(b) - \gamma_{i_j}(a)). \tag{10}$$

Restricting up to level $m$, this means that, for the straight path,

$$X^{\gamma,\underline{m}}_{a,b} = \exp\left((\gamma_{\mathbf{1}}(b) - \gamma_{\mathbf{1}}(a))\mathbf{1} + \cdots + (\gamma_{\mathbf{d}}(b) - \gamma_{\mathbf{d}}(a))\mathbf{d}\right) = \exp\left(\gamma(b) - \gamma(a)\right) \tag{11}$$

Equations 9 and 11 between them make it easy to calculate elements of the signature of a piecewise linear path up to any given level. The code required to do this is simple, although it requires more operations than direct log signature calculations.

# 6 Log signature

Let $S$ be the set which consists of the signature up to level $m$ of every path in $\mathbb{R}^d$. $S$ is not the whole of the vector space $T^{\underline{m}}(\mathbb{R}^d)$, but rather a lower dimensional manifold. $S$ is known as the *group-like elements* of $T^{\underline{m}}(\mathbb{R}^d)$. In fact, $S$ is the image under exp of the image of $\rho$. The *log signature* of $\gamma$ up to level $m$ is $Y^{\gamma,\underline{m}}_{a,b} = \rho^{-1}(\log(X^{\gamma,\underline{m}}_{a,b}))$, an element of $F^{\underline{m}}(\{\mathbf{1},\dots,\mathbf{d}\})$. When the term 'log signature' is used to describe a fixed array of numbers, of course we mean its components in some basis, for example the Lyndon basis.

For speed, it may be best to calculate the log signature explicitly without leaving the free Lie algebra. Here are the two rules which enable this to happen. Equation 11 tells us that if $\gamma$ is a straight line then its log signature is independent of $m$:

$$Y^{\gamma,\underline{m}}_{a,b} = \gamma(b) - \gamma(a). \tag{12}$$

From Equation 9 it follows that for $a < b < c$

$$Y^{\gamma,\underline{m}}_{a,c} = \log\Big(\exp\big(Y^{\gamma,\underline{m}}_{a,b}\big)\exp\big(Y^{\gamma,\underline{m}}_{b,c}\big)\Big). \tag{13}$$

Thus the log signature of the concatenation of two paths can be calculated from the paths using the Baker-Campbell-Hausdorff formula, with the **1** and **2** there above replaced with the log signatures of the two paths.

# 7 Sizes

The size of the signature and log signature up to level $m$ for various values of $d$ and $m$ is as given in Table 2. The fixed 1 coefficient of $\epsilon$ in the signature is omitted as it does not carry information.

| | $d=1$ | | $d=2$ | | $d=3$ | | $d=4$ | | $d=5$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| $m=1$ | 1 | **1** | 2 | **2** | 3 | **3** | 4 | **4** | 5 | **5** |
| $m=2$ | 2 | **1** | 6 | **3** | 12 | **6** | 20 | **10** | 30 | **15** |
| $m=3$ | 3 | **1** | 14 | **5** | 39 | **14** | 84 | **30** | 155 | **55** |
| $m=4$ | 4 | **1** | 30 | **8** | 120 | **32** | 340 | **90** | 780 | **205** |
| $m=5$ | 5 | **1** | 62 | **14** | 363 | **80** | 1,364 | **294** | 3905 | **829** |
| $m=6$ | 6 | **1** | 126 | **23** | 1,092 | **196** | 5,460 | **964** | 19,530 | **3,409** |
| $m=7$ | 7 | **1** | 254 | **41** | 3,279 | **508** | 21,844 | **3,304** | 97,655 | **14,569** |
| $m=8$ | 8 | **1** | 510 | **71** | 9,840 | **1318** | 87,380 | **11,464** | 488,280 | **63,319** |
| $m=9$ | 9 | **1** | 1022 | **127** | 29,523 | **3,502** | 349,524 | **40,584** | 244,1405 | **280,319** |
| $m=10$ | 10 | **1** | 2046 | **226** | 88,572 | **9,382** | 1,398,100 | **145,338** | 12,207,030 | **1,256,567** |
| $m=11$ | 11 | **1** | 4094 | **412** | 265,719 | **25,486** | 5,592,404 | **526,638** | 61,035,155 | **5,695,487** |
| $m=12$ | 12 | **1** | 8190 | **747** | 797,160 | **69,706** | 22,369,620 | **1,924,378** | 305,175,780 | **26,039,187** |

Table 2: The sizes of signatures (plain) and log signatures (bold)

The size of the signature is $\frac{d(d^m-1)}{d-1}$. A formula for the size of each level of the log signature is given in [10].

# 8 The code

If the whole log signature up to level $m$ is being used as a representation in a machine learning algorithm, then often the choice of basis should not matter. Typically the aim is to have fast code to calculate the log signature up to a fixed level in the same basis for many different paths of a fixed dimension. The log

signature of a single segment of a path is easy to calculate. The python script is used to write the code which constructs signatures of multi-segment paths.

The python script represents basis elements of the free Lie algebra by objects of class `LyndonWord`, and elements of the free Lie algebra by objects of type `Polynomial`, which holds a dictionary from `LyndonWord` to *numbers*. By using an algebra of polynomials in arbitrary strings as the *numbers* (classes `N_constantNum` and `N_polynomial`), it can calculate expressions for the components of the log signature of the concatenation of two paths given their log signatures. The script writes functions to do this in the code which it outputs. When the second path is a line segment, the expressions are considerably simpler, and a function for this can also be written. The user needs to pay attention to areas in the code where the comment `USER CONTROL` occurs, to determine such things as which $m$ and $d$ to consider and where the data from [2] has been downloaded. The code produced is not affected by which basis the BCH formula is expressed in, as all the BCH's brackets get expanded.

### Using the generated code

The generated code is used in `C++` to calculate the log signature of a given path. The generated file `bch.h` will need to be `#included`, and the `bch.cpp` will need to be built in the client project. Note that `bch.cpp` has no other dependencies and if high levels are included will take a while to compile, so it is recommended to design the client's build procedure around not recompiling it every build. Here is a quick example of how to use it: If `p` is a `std::vector<std::array<float,3>>` containing the coordinates visited by a 3d path, then the following code will make `logsig` be a `std::array` containing its log signature up to level 6.

```
LogSignature<3,6> logsig{};
if(p.size()>1)
  for(int j=0; j<3; ++j)
    logsig[j]=p[1][j]-p[0][j];
for(size_t i=2; i<p.size(); ++i){
  Segment<3> displacement;
  for(int j=0; j<3; ++j)
    displacement[j]=p[i][j]-p[i-1][j];
  joinSegmentToSignatureInPlace(logsig,displacement);
}
```

## 9 Other bases

This open source `C++` library [8] is able to work with elements of the free Lie algebra, which are represented as variables of type `alg::lie`. Like we do, the library stores them in terms of a basis. The calculation of the basis is in the file `libalgebra/lie_basis.h`, and examples of the basis which it calculates can be generated using the form on the website. The basis used is not the Lyndon basis described earlier.

In [4] is given a general method of constructing a basis of the free Lie algebra once an order on bracketed expressions with a given depth has been specified. (This is extended to an order on all bracketed expressions by saying that expressions with higher depth are greater than those with lower depth). This is called a (generalized) Hall basis. If the order is alphabetical on the foliage, then the basis obtained is the Lyndon basis.

Coropa uses a different order. For Coropa, if two different bracketed expressions $[u\,,v]$ and $[x\,,y]$ have the same length then $[u\,,v]<[x\,,y]$ if $u<x$ or both $u=x$ and $v<y$.

If the variable `match_coropa` is set to `True` in the python code, then it will use this basis. Note that in this case the class `LyndonWord` represents a basis element, which is not actually a Lyndon word. The generated code is a bit longer when this is done; it is not entirely clear why this is.

In [1] an algorithm is given for expressing the BCH formula using any generalized Hall basis with $d = 2$. One of the example bases which is given is called *the (classical) Hall basis*. This is the same as Coropa's basis on 2 words, except that all the brackets are reversed. This means that basis elements of even depth are the same, and those of odd depth are negated.

## 10 A visualisation

It is natural to want to visualise the log signature, and here is a small idea in that direction. In the specific case $d = 2$, $m = 4$, the log signature has eight components, and a two dimensional path with four segments has eight degrees of freedom. The file `view.m` provides an interactive Mathematica 10 visualisation of the relationship between a path made of four segments and its log signature. It depends on another file `bch.m`, which has been generated by additional functionality in the python script, which defines a single function returning the log signature of a path defined by four displacements. The visualisation should appear when `view.m` is run. The log signature appears as widgets on the right of the graph of the path. Because level 1 and level 3 of the log signature are two dimensional, they are represented by 2d controls. The other log signature components are controlled separately. When 'solve' is ticked, you can gently move the widgets to change the components of the log signature and see the path move. Note that the 12 component controls the signed area enclosed by the path. When 'solve' is unticked, you can drag the locators to change the path, and see the corresponding log signature elements move.

## References


[1] Fernando Casas and Ander Murua. "An efficient algorithm for computing the Baker-Campbell-Hausdorff series and some of its applications". In: *Journal of Mathematical Physics* 50.3 (Mar. 2009), p. 033513. eprint: `http://arxiv.org/abs/0810.2656`.

[2] Fernando Casas and Ander Murua. *The BCH formula and the symmetric BCH formula up to terms of degree 20*. URL: `http://www.ehu.eus/ccwmuura/bch.html`.

[3] Kuo-Tsai Chen. "Integration of Paths – A Faithful Representation of Paths by Noncommutative Formal Power Series". In: *Transactions of the American Mathematical Society* 89.2 (1958), pp. 395–407.

[4] Marshall Hall. "A basis for free Lie rings and higher commutators in free groups". In: (1950). URL: `http://www.ams.org/journals/proc/1950-001-05/S0002-9939-1950-0038336-7/home.html`.

[5] Ben Hambly and Terry Lyons. "Uniqueness for the signature of a path of bounded variation and the reduced path group". 2005. URL: `http://arxiv.org/abs/math/0507536`.

[6] Pierre Lalonde and Arun Ram. "Standard Lyndon Bases of Lie Algebras and Enveloping Algebras". In: *Transactions of the American Mathematical Society* 347.5 (1995), pp. 1821–1830.

[7] Terry Lyons, Michael Caruana, and Thierry Lévy. *Differential Equations Driven by Rough Paths*. 2007.

[8] Terry Lyons et al. *CoRoPa Computational Rough Paths (software library)*. 2010. URL: `http://coropa.sourceforge.net/`.

[9] Christophe Reutenauer. *Free Lie Algebras*. 1994.

[10] Wikipedia. *Necklace polynomial*. 2015. URL: `http://en.wikipedia.org/wiki/Necklace_polynomial`.