Signature

The signature summarizes important information about a path. Dr. Benjamin Graham's notation is followed in this section.

Let $[S, T] \subset R$; $d, k, m \in N$; $V = R^d$.

Consider a continuous function $X : [S, T] \rightarrow V$.

For $[s, t] \subset [S, T]$, the k-th iterated integral of X is the d^k-dimensional vector $X_{s,t}^k$.

The signature truncated at level m is $S(X)_{s,t} = (1, X_{s,t}^1, X_{s,t}^2, \dots, X_{s,t}^m)$ ← Eq. 2.1

Let $\Delta_{s,t} = X_t - X_s$ be the path displacement.

If X is a straight line,

$$X_{s,t}^1 = \Delta_{s,t}\ , \quad X_{s,t}^2 = \frac{\Delta_{s,t} \otimes \Delta_{s,t}}{2!}\ , \quad X_{s,t}^3 = \frac{\Delta_{s,t} \otimes \Delta_{s,t} \otimes \Delta_{s,t}}{3!}\ , \dots \qquad \leftarrow \text{Eq 2.2}$$

The symbol $\otimes$ corresponds to the Kronecker matrix product, also known as matrix direct product.

$$A_{m \times n} \otimes B_{p \times q} = C_{mp \times nq}$$

Where,

$$c_{\alpha\beta} = a_{ij} b_{kl}$$

$$\alpha = p(i-1) + k$$

$$\beta = q(j-1) + l$$

An Algorithm to Calculate Iterated Integrals

Assume a piecewise linear path between N points, in a d-dimensional space, specified as a dxN matrix M.

- Let the iterated integral of the straight path from the i^{th} point to $(i+1)^{th}$ point be a_i.
- For any k and any $(j_1, \dots, j_k)$
  - $a_i^{(j_1,\dots,j_k)} = \frac{1}{k!}\prod_{h=1}^{k}(M_{j_h,i+1} - M_{j_h,i})$ ← Eq 2.3
- Let the iterated integral of the whole path from the 1st point to $(i+1)^{th}$ point be b_i.
  - $b_1 = a_1$
  - $b^{()} = a^{()} = 1$
  - $b_{i+1}^{(j_1,\dots,j_k)} = \sum_{h=0}^{k} b_i^{(j_1,\dots,j_h)} a_{i+1}^{(j_{h+1},\dots,j_k)}$ ← Eq 2.4

I wrote a python code using this algorithm to calculate iterated integrals. Then, I tested it against the already available python libraries.

- iisignature – by Dr. Benjamin Graham and Jeremy Reizenstein.
- esig - by CoRoPa (https://coropa.sourceforge.io/)

All 3 gave the same results, which meant that the algorithm is correct. I also found a handwriting English character dataset named UJIpenchars (https://archive.ics.uci.edu/ml/datasets/UJI+Pen+Characters+(Version+2)).

Formula to Calculate 2nd Iterated Integral of a 2-dimensional Path

Let the 2-dimensional path be $X_t = \left(X_t^{(1)}, X_t^{(2)}\right) \in R^2$.

Let us use the notation $X_{a,b} = X_a - X_b$

For fixed times $t_0 < t_1 < \cdots < t_n$, the $(n+1)$ data points are $X_{t_0}, X_{t_1}, \dots, X_{t_n}$.

Let $S(X)_{t_i,t_j}^{k,l}$ be a 2nd iterated integral for $t_i < t_j$ and $k, l \in \{1,2\}$.

$$S(X)_{t_i,t_j}^{k,l} = \int_{t_i}^{t_j} X_{t_i,r}^{(k)} dX_r^{(l)} = \sum_{m=i}^{j-1} \int_{t_m}^{t_{m+1}} X_{t_i,r}^{(k)} dX_r^{(l)} \qquad \leftarrow \text{Eq 2.5}$$

Where, $\int_{t_m}^{t_{m+1}} X_{t_i,r}^{(k)} dX_r^{(l)} = (X_{t_{m+1}}^{(l)} - X_{t_m}^{(l)}) \left[\left(X_{t_m}^{(k)} - X_{t_i}^{(k)}\right) + \frac{1}{2}(X_{t_{m+1}}^{(k)} - X_{t_m}^{(k)})\right]$

Formula to Calculate 3rd Iterated Integral of a 2-dimensional Path

I derived this formula by myself using rough paths theory.

Let the 2-dimensional path be,

$$X_t = \left(X_t^{(1)}, X_t^{(2)}\right) \in R^2.$$

Let us use the notation,

$$X_{a,b} = X_a - X_b$$

For fixed times $t_0 < t_1 < \cdots < t_n$, the *(n+1)* data points are $X_{t_0}, X_{t_1}, \ldots, X_{t_n}$.

Let $S(X)_{t_i,t_j}^{k,l,p}$ be a 3rd iterated integral for $t_i < t_j$ and $k, l, p \in \{1,2\}$.

$$S(X)_{t_i,t_j}^{k,l,p} = \int_{t_i}^{t_j} S(X)_{t_i,s}^{k,l} dX_s^{(p)} = \sum_{m=i}^{j-1} \int_{t_m}^{t_{m+1}} S(X)_{t_i,s}^{k,l} dX_s^{(p)} \qquad \leftarrow \text{Eq 2.6}$$

Where,

$$\int_{t_m}^{t_{m+1}} S(X)_{t_i,s}^{k,l} dX_s^{(p)}$$

$$= \left(X_{t_{m+1}}^{(p)} - X_{t_m}^{(p)}\right)\left[S(X)_{t_i,t_m}^{k,l} + \frac{1}{2}\left(X_{t_{m+1}}^{(l)} - X_{t_m}^{(l)}\right)\left(X_{t_m}^{(k)} - X_{t_i}^{(k)}\right) + \frac{1}{6}\left(X_{t_{m+1}}^{(l)} - X_{t_m}^{(l)}\right)\left(X_{t_{m+1}}^{(k)} - X_{t_m}^{(k)}\right)\right]$$

$S(X)_{t_i,t_m}^{k,l}$ is given by Eq. 2.5.

I wrote a python code using this formula and checked it against the libraries and the algorithm. All of them gave the same results, which meant that the formula I derived was correct. I also wrote a C++ code to do the same.