Symbolic Trajectory Prediction

Alec Hoyland

October 29, 2019

Abstract

1 What is a trajectory?

For the purposes of this report, a *trajectory* is an ordered set of states which are evolved according to a rule. The purpose of symbolic trajectory prediction, is, if given a few states of the trajectory, can the rule be determined and used to predict future states. Mathematically, a trajectory is a sequence of values $\left[f^k(x)\right]_{k\in\mathbb{N}}$ calculated by the iterated application of a mapping f to an element x of its source. It is analogous to the time-ordered set of states in a dynamical system (*e.g.* Poincaré map) such as the function of position produced by integrating the equations of motion in Newtonian mechanics.

2 What is a feature?

A feature is any important defining characteristic of a snapshot of a trajectory. For example, if the trajectory represents the motion of a ball in flight across a 2-dimensional plane, the crucial features are the coordinates (x, y) parametrized by the time-coordinate t. If the color of the point mattered (and was not time-invariant), then it would also be a feature. For the purposes of this document, the viewer is able to discern which features are important and which are not. This is a result of the miraculous specificity of the visual system in mammals. If, for example, three polygonal shapes were drawn on a blackboard, and the viewer were asked to determine the pattern, it is reasonable to neglect the minute deviations in the lines caused by being drawn by an unsteady hand. Similarly, imperfections in printed symbols could also be ignored.

The best heuristic for determining what is a feature and what is not relies on comparing two adjacent states of the trajectory. Major differences are features; minor differences can be neglected.

3 Representing the trajectory as a vector transformation

One way to imagine a trajectory is to consider a series of points in the time-coordinate t and a function $f(t)$ which produces a vector output $\vec{x}$ where each element in $\vec{x}$ is the numerical value of a feature at time t. In this situation, the trajectory is a vector parametrized by the time-coordinate. To compare two adjacent states, the first derivative can be taken. In this situation, the derivative operator is best represented by the finite difference.

In brief, a finite difference is a mathematical expression of the form $f(x + b) - f(x + a)$. If the finite difference is divded by $b - a$, the difference quotient is defined. The finite difference is frequency applied in numerical analysis, in particular in numerical differential equations and approximations. The first forward finite difference D of x_n is defined by

$$\mathrm{D}x_n = x_n - x_{n-1} \tag{1}$$

Higher-order finite differences can also be defined, such as the second finite difference

$$\mathrm{D}^2x_n = x_n - 2x_{n-1} + x_{n-2} \tag{2}$$

These are first-order correct in accuracy with uniform grid spacing n. More accurate finite differences can be computed using more points. At minimum, approximating a derivative of order n with the fewest points (and therefore worst accuracy) requires $n + 1$ points.

If two states of the trajectory are known, x_n and x_{n-1}, then the next point can be approximated as

$$x_{n+1} \approx \frac{1}{\Delta}\mathrm{D}x_n + x_n = \frac{x_n - x_{n-1}}{\Delta} + x_n \tag{3}$$

where Δ is the difference in the time-coordinate between indices n and $n-1$. If higher-order finite differences are used, the estimate will be more accurate, up to the derivative order equal to the order of the polynomial representation of the function. This requires knowing most past states. Any well-behaved function can be represented by its Taylor series. From Einar Hille, we have a generalization of the Taylor series that converges to the value of the function for any bounded, continuous function on $(0, \infty)$ using the calculus of finite differences. For $t > 0$,

$$f(a + t) = \lim_{\Delta \to 0^+} \sum_{n=0}^{\infty} \frac{t^n}{n!\Delta^n}\mathrm{D}^n f(a) \tag{4}$$

If f is a polynomial of order n then $n + 1$ terms of the series are needed to fully represent it. Once the predictor

$$\mathrm{P}(a) = \left[\frac{1}{n!}\mathrm{D}^n f(a)\right]_{n \in \mathbb{N}} \tag{5}$$

is known, it can be evolved forward by taking the dot product with $\left[\frac{t^n}{\Delta^n}\right]_{n\in\mathbb{N}}$ to find $f(a+t)$.

Example. Here is a simple example. The source code is in

`https://github.com/hasselmonians/hasselmo-tracking/hille.jl`

written in Julia. The goal is to predict the one-dimensional trajectory $f(t) = -40 + 100t - 10t^2$. For $\Delta = 0.01$, the first and second finite differences are computed at $f(2\Delta)$. By the discrete Taylor series (Hille series), the form of the solution can be found.

$$f(2\Delta + t) = f(0) + \frac{t}{\Delta}\mathrm{D}f(2\Delta) + \frac{t^2}{2\Delta^2}\mathrm{D}f(2\Delta) \tag{6}$$

which is equivalent to the Taylor series

$$f(t) = x_0 + v_0 t + \frac{1}{2}a_0 t^2 \tag{7}$$

Example. Here we consider a multi-dimensional situation. The rule is that at time $t \in \mathbb{N}$, we draw a square with the top left vertex at the origin with side length $s = t + 1$. The source code is in

`https://github.com/hasselmonians/hasselmo-tracking/square-tracing.jl`

written in Julia. Using the visual cortex as an oracle, we identify eight features, the four coordinate-pairs which note the vertices of the square. One vertex (top left) never changes. The top right vertex proceeds according to the rule $(0, t+1)$, the bottom left $(-1-t, 0)$, and the bottom right by $(-1-t, -1-t)$. Thus, each coordinate evolves either linearly or doesn't change (constant). Finite difference approximations can be taken for each coordinate independently, and the rule can be determined after 2 steps.

4 Representing the trajectory as function transformations

This formalism was proposed by Marc Howard. It expands the idea of mapping vectors across a time-coordinate to a larger space of problems. Instead, the rule involves defining a series of functions ordered in time: $[f_n(\mathcal{S})]_{n\in\mathbb{N}}$. Each function acts on a surface (S), the specifics of which should not matter for the general formalism. These insights come from the field of functional calculus and arithmetic on random variables.

The "sum" of two functions is the convolution $*$ and the "difference" of two functions is the cross-correlation $\star$. If the rule can be represented by a first-order transformation (relying on only the past two functional states), the rule is

$$f_{n+1} = (f_n \star f_{n-1}) * f_n \tag{8}$$