Downsampling Discrete Trajectories

Pascal Leroy (phl)

2026-01-26

This document describes the computations that are performed by the method `Append` of class `DiscreteTrajectorySegment` to downsample the points produced by an integrator and build a compact yet accurate representation of discrete trajectories.

Overview

An integrator produces a stream of tuples (t_i, q_i, p_i) giving the degrees of freedom of a massless body at discrete times t_i. The purpose of downsampling is twofold:

— Construct cubic Hermite splines that interpolate between consecutive t_i to make it possible to evaluate the degrees of freedom at any time with sufficient accuracy.

— Make the representation more compact by dropping the tuples (t_i, q_i, p_i) for $i \in]1, n[$ if the Hermite spline based on (t_1, q_1, p_1) and (t_n, q_n, p_n) approximates these tuples with sufficient accuracy.

A Brute Force Algorithm

The best way to describe the problem we want to solve is to present a brute-force algorithm that provides an exact solution:

Algorithm 1: BruteForceDownsampling.

Input: A tolerance η and a stream of tuples (t_i, q_i, p_i).
Output: A stream of intervals $[j_1, j_2]$ such that the Hermite spline based on the tuples $(t_{j_1}, q_{j_1}, p_{j_1})$ and $(t_{j_2}, q_{j_2}, p_{j_2})$ approximates the input tuples (t_i, q_i, p_i) for $i \in [j_1, j_2]$ with a tolerance better that η.

```
1. Let j ← 1.
2. while not at end of input stream do
3.     Construct the Hermite spline h based on the tuples (t_j, q_j, p_j) and
       (t_i, q_i, p_i).
4.     if max_{k=j…i} ‖h(t_k) − q_k‖_2 > η then
5.         Emit the interval [j, i − 1].
6.         Let j ← i − 1.
7.     end
8. end
9. Emit the interval [j, n] where n is the index of the last element in the stream.
```

A few things are worth noting here:

— Only the tuples $(t_{j_1}, q_{j_1}, p_{j_1})$ and $(t_{j_2}, q_{j_2}, p_{j_2})$ corresponding to the bounds of the intervals $[j_1, j_2]$ need to be stored permanently. The other tuples can be interpolated with an accuracy better than η.

— The Hermite splines do not need to be stored permanently; they can be reconstructed based on the tuples $(t_{j_1}, q_{j_1}, p_{j_1})$ and $(t_{j_2}, q_{j_2}, p_{j_2})$.

— The algorithm is optimal in the sense that it produces the longest possible intervals $[j_1, j_2]$ that satisfy the tolerance η.

— For each input tuple, the algorithm needs to scan all past tuples since the upper bound j of the last emitted interval. Therefore, the algorithm is quadratic in the number of tuples in the input stream.

A Binary Search Algorithm

In order to avoid the quadratic complexity, we used to use a binary search algorithm as follows:

Algorithm 2: BinarySearchDownsampling.

Input: A tolerance η, an integer N, and a stream of tuples (t_i, q_i, p_i).
Output: A stream of intervals $[j_1, j_2]$ such that the Hermite spline based on the tuples $(t_{j_1}, q_{j_1}, p_{j_1})$ and $(t_{j_2}, q_{j_2}, p_{j_2})$ approximates the input tuples (t_i, q_i, p_i) for $i \in [j_1, j_2]$ with a tolerance better that η.

```
Let A be an array of size N used to store tuples.
while not at end of input stream do
    if A is full then
        Emit SearchInterval(A, 1, N).
        Clear A.
    end
    Append (t_i, q_i, p_i) to A.
end
Function SearchInterval(A, i_1, i_2) is
    Construct the Hermite spline h based on the tuples (t_{i_1}, q_{i_1}, p_{i_1}) and
      (t_{i_2}, q_{i_2}, p_{i_2}).
    if max_{k=i_1...i_2} ||h(t_k) − q_k||_2 > η then
        Let k ← ⌊i_2/2⌋.
        Emit SearchInterval(A, i_1, k).
        Emit SearchInterval(A, k + 1, i_2).
    else
        Emit [i_1, i_2].
    end
end
```

This algorithm has the following properties:

— The array A must be stored permanently since it contains tuples that have not been downsampled yet and that will be needed to make a future decision about the intervals to emit.
— The choice of N involves a trade-off: N must be large enough that the downsampling is effective and produces long intervals $[j_1, j_2]$; but it must not be so large that the size of A affects performance.
— The algorithm has a complexity of $\mathcal{O}(N \log N)$ on average and $\mathcal{O}(N^2)$ in the worst case.
— The algorithm does not produce the longest possible intervals: in the worst case, all the emitted intervals may be too small by a factor of 2.

An Efficient Streaming Algorithm

In this section we present an algorithm that, like the brute-force algorithm, is streaming (so a decision is made on each input tuple without having to resort to intermediate storage), whose complexity is linear in the size of the stream, and which generally produces a much more compact output than the binary search algorithm. Without loss of generality we assume that, when the tuple of index i is received, no interval was ever emitted (so the first interval to emit will necessarily have the lower bound 1).

The algorithm is incremental, so let's assume that we have already constructed a Hermite spline h_{i-1} with an error bound ε_{i-1} such that:

$$\max_{k=1\cdots i-1} \left\| h_{i-1}(t_k) - q_k \right\|_2 \leq \varepsilon_{i-i} \leq \eta$$

It is obviously possible to build such a spline, because the error on a spline build based on two consecutive points in the stream is 0. In other words, $\varepsilon_2 = 0$ (ε_1 is not defined because a spline cannot be constructed using a single point).

When the point with index i is received, we construct the Hermite spline h_i based on the tuples (t_1, q_1, p_1) and (t_i, q_i, p_i). Noting that $h_i(t_i) = q_i$ we have:

$$\max_{k=1\cdots i} \left\| h_i(t_k) - q_k \right\|_2 = \max_{k=1\cdots i-1} \left\| h_i(t_k) - q_k \right\|_2$$

And, by the triangular inequality:

$$\max_{k=1\cdots i-1} \left\| h_i(t_k) - q_k \right\|_2 \leq \max_{k=1\cdots i-1} \| h_i(t_k) - h_{i-1}(t_k) \|_2 + \max_{k=1\cdots i-1} \left\| h_{i-1}(t_k) - q_k \right\|_2$$

The first term is bounded by the L^∞ norm of $h_i - h_{i-1}$ and the second term is bounded, because of our recurrence hypothesis, by ε_{i-1}. Thus:

$$\max_{k=1\cdots i} \left\| h_i(t_k) - q_k \right\|_2 \leq \| h_i - h_{i-1} \|_\infty + \varepsilon_{i-1}$$

Therefore, if we can compute $\| h_i - h_{i-1} \|_\infty$ we can find an upper bound ε_i on the left-hand of this inequality and use it to determine if h_i is still within the tolerance η or if the interval must be split.

Unfortunately $h_{i-1}(t)$ and $h_i(t)$ are 3rd degree polynomials, so $\| h_i(t) - h_{i-1}(t) \|_2^2$ is a 6th degree polynomial, and finding its extrema requires the resolution of a 5th degree equation, which would probably be overly expensive.

We can note however that, by construction, the h_i are all exact for $t = t_1$. Therefore, $h_{i-1}(t_1) = h_i(t_1)$ and $h'_{i-1}(t_1) = h'_i(t_1)$. This implies that $(t - t_1)^2$ divides the polynomial $h_i(t) - h_{i-1}(t)$: there exist a 1st degree polynomial $q_i(t)$ such that:

$$h_i(t) - h_{i-1}(t) = (t - t_1)^2 q_i(t)$$

Taking the norm we obtain:

$$\| h_i(t) - h_{i-1}(t) \|_2^2 = (t - t_1)^4 \left\| q_i(t) \right\|_2^2$$

To find the extrema we need to compute the roots of the derivative:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}\,t} \| h_i(t) - h_{i-1}(t) \|_2^2 &= 4(t - t_1)^3 \left\| q_i(t) \right\|_2^2 + 2(t - t_1)^4 \left(q_i(t) \cdot q'_i(t) \right) \\ &= 4(t - t_1)^3 \left(q_i(t) \cdot \left(q_i(t) + \frac{1}{2}(t - t_1) q'_i(t) \right) \right) \end{aligned}$$

Now the polynomial $q_i(t) \cdot \left(q_i(t) + \frac{1}{2}(t - t_1) q'_i(t) \right)$ is a 2nd degree polynomial and its roots may be computed efficiently. In order to compute the extrema of $h_i(t) - h_{i-1}(t)$, we must evaluate it at the roots of its derivative, and at t_{i-1}, since it could have an extremum there that does not correspond to a zero of the derivative.

Having computed the extrema of $h_i(t) - h_{i-1}(t)$, we set:

$$\varepsilon_i = \| h_i - h_{i-1} \|_\infty + \varepsilon_{i-1}$$

If $\varepsilon_i < \eta$, we are sure that the polynomial $h_i(t)$ is within the tolerance. If on the other hand $\varepsilon_i \geq \eta$ we have to assume that it is not, and we must emit an interval. Note that this decision is somewhat pessimistic because of the use of the triangular inequality above, but in practice the polynomials $h_i(t)$ change slowly with i, so that $h_{i-1}(t_k) - q_k$ and $h_i(t_k) - q_k$ are nearly colinear and the triangular inequality is close to an equality. This means that this technique tends to yield a solution relatively close to the optimal obtained by the brute-force algorithm, and significantly better than the one obtained by binary search.

This analysis leads to the following algorithm:

Algorithm 3: EfficientDownsampling.

Input: A tolerance η and a stream of tuples (t_i, q_i, p_i).

Output: A stream of intervals $[j_1, j_2]$ such that the Hermite spline based on the tuples $(t_{j_1}, q_{j_1}, p_{j_1})$ and $(t_{j_2}, q_{j_2}, p_{j_2})$ approximates the input tuples (t_i, q_i, p_i) for $i \in [j_1, j_2]$ with a tolerance better that η.

```
1.  Let j ← 1.
2.  Let ε ← 0.
3.  while not at end of input stream do
4.      Construct the Hermite spline h_{j,i} based on the tuples (t_j, q_j, p_j) and
        (t_i, q_i, p_i).
5.      Let ε ← ε + ‖h_{j,i} − h_{j,i−1}‖_∞.
6.      if ε > η then
7.          Emit the interval [j, i − 1].
8.          Let j ← i − 1.
9.          Let ε ← 0.
10.     end
11. end
12. Emit the interval [j, n] where n is the index of the last element in the stream.
```

It is worth noting that the state of the algorithm now includes ε, a quantity that cannot be reconstructed once tuples have been dropped. In order for the downsampling to be restartable (*e.g.*, after a call to `ForgetAfter`) the ε_i must be stored permanently.