Iterative Polynomial Sampling for Resource-Efficient Real-Time Trajectory Generation in Hardware

(Draft)

November 19, 2025

Abstract

We describe a simple representation of univariate polynomials on a uniform grid that enables real-time evaluation using only additions. The method is equivalent to the classical forward-difference representation of polynomials, but we formulate it in a way that is directly applicable to FPGA or ASIC implementations for motion control and similar real-time interpolation tasks. In particular, we emphasize a split between (i) high-precision, off-line coefficient processing and (ii) low-precision, on-line difference accumulation.

1 Introduction

Many real-time control applications—such as CNC machines, robot joint controllers, and stepper/servo drives—require evaluation of smooth reference trajectories at a fixed sampling period. These trajectories are often given as low-degree polynomials of time (e.g. cubic or quintic splines, jerk-limited S-curves, etc.).

A straightforward implementation evaluates a polynomial

$$p(t) = a_0 + a_1 t + a_2 t^2 + \cdots + a_N t^N$$

at each sampling instant using Horner's rule, which requires multipliers and a chain of multiply–accumulate operations in the real-time loop.

In hardware with limited resources (e.g. small FPGAs) it is often desirable to avoid multipliers altogether in the hot path. This note summarizes an elementary, but practically useful, fact:

> If p is a polynomial of degree N and we only need its values on a uniform time grid, then all samples can be generated by a fixed linear recurrence using only additions, once a small set of "difference registers" has been initialized.

Mathematically, this is classical forward-difference theory. The goal of this document is to present the construction in a compact and implementation-oriented way, with an eye towards motion-control applications.

2 Polynomials on a Uniform Grid and Forward Differences

Let $p : \mathbb{R} \to \mathbb{R}$ be a polynomial of degree at most N. Fix a sampling period $h > 0$ and an initial time $t_0 \in \mathbb{R}$. We define the sequence

$$y_k := p(t_0 + kh), \qquad k = 0, 1, 2, \ldots$$

of sampled values.

The forward-difference operator Δ acts on such sequences by

$$(\Delta y)_k := y_{k+1} - y_k.$$

Repeated application gives higher-order forward differences $\Delta^m y$ for $m \geq 0$.

2.1 Key property

A standard result is:

Theorem 1. *Let p be a polynomial of degree at most N, and define $y_k = p(t_0 + kh)$ as above. Then*

$$\Delta^{N+1} y \equiv 0,$$

i.e. the $(N+1)$-st forward difference is identically zero. Equivalently, the N-th forward difference $\Delta^N y$ is constant.

In practice, this means that the sequence (y_k) is completely determined by:

- the initial value y_0, and
- the first N forward differences at $k = 0$.

We can therefore encode the entire infinite sequence (y_k) in a finite vector of length $N+1$, and evolve it by a simple recurrence.

3 Iterative Polynomial Sampler

3.1 Difference-register representation

We define a vector of *difference registers*

$$\boldsymbol{r} = (r_0, r_1, \ldots, r_N) \in \mathbb{R}^{N+1}$$

as follows:

$$\begin{aligned}
r_0 &:= \Delta^0 y_0 = y_0, \\
r_1 &:= \Delta^1 y_0 = y_1 - y_0, \\
r_2 &:= \Delta^2 y_0 = (y_2 - y_1) - (y_1 - y_0), \\
&\vdots \\
r_N &:= \Delta^N y_0.
\end{aligned}$$

The registers thus store the forward differences evaluated at $k = 0$.

Given this initialization, the sequence (y_k) can be generated purely by iteratively updating $\boldsymbol{r}$ using additions.

3.2 Update rule

Let $\boldsymbol{r}^{(k)}$ denote the register vector after k updates, with $\boldsymbol{r}^{(0)}$ the initial one defined above. We define the update *per time step* as:

$$r_0^{(k+1)} = r_0^{(k)}, \tag{1}$$
$$r_1^{(k+1)} = r_1^{(k)} + r_0^{(k)}, \tag{2}$$
$$r_2^{(k+1)} = r_2^{(k)} + r_1^{(k)}, \tag{3}$$
$$\vdots \tag{4}$$
$$r_N^{(k+1)} = r_N^{(k)} + r_{N-1}^{(k)}. \tag{5}$$

Proposition 1. *With initialization $\boldsymbol{r}^{(0)}$ as above and the update (1)–(5), one has*

$$r_N^{(k)} = y_k = p(t_0 + kh)$$

for all integers $k \geq 0$.

Sketch. For $N = 0$, the sequence is constant, and the statement is trivial. One can then proceed by induction on N using the identity $\Delta^{m+1} y = \Delta(\Delta^m y)$ and the linearity of Δ.

Operationally, note that summing r_{i-1} into r_i at each step mimics the discrete integration that recovers $\Delta^i y$ from $\Delta^{i-1} y$. Since the initial registers match the forward differences at $k = 0$, each update of $\boldsymbol{r}$ reproduces the correct forward differences at subsequent indices k, and in particular $r_N^{(k)} = \Delta^N y_k = y_k$ for a degree-N polynomial. □

In other words, the highest-order register always contains the current sample y_k, and the lower registers contain the lower-order forward differences at that same index.

3.3 Algorithmic form

For implementation, the update can be written as:

State: registers $r[0], r[1], \ldots, r[N]$.
One sampling step:

1. Output $y = r[N]$.
2. For $i = 0, 1, \ldots, N-1$ in ascending order:

$$r[i+1] \leftarrow r[i+1] + r[i].$$

The crucial point is that this loop uses only additions and registers; there are no multipliers, no explicit powers of t, and no explicit polynomial coefficients in the real-time path.

4 High-Precision Initialization vs Low-Precision Runtime

In many applications it is desirable to separate:

- an *off-line* or *host-side* phase where polynomial coefficients are computed and converted into an internal representation, and

- an *on-line*, fixed-point, real-time phase with a very small hardware footprint.

Let p be given in coefficient form

$$p(t) = \sum_{i=0}^{N} a_i t^i.$$

A typical pipeline is:

1. Given a time interval $[t_0, t_0 + T]$ and a sampling period h, compute $K = \lfloor T/h \rfloor$ and the desired samples $y_k = p(t_0 + kh)$ for $k = 0, \ldots, N$ using a high-precision type (e.g. double-precision floating point).
2. From these, compute the forward differences at $k = 0$ to obtain $r_0, \ldots, r_N$ using the definition of Δ.
3. Quantize each r_i to a fixed-point or integer format suitable for hardware implementation.
4. Upload these quantized registers to the FPGA or ASIC before the segment starts.
5. During runtime, the streaming update rule is applied at each sampling period using purely fixed-point additions.

In motion control, different trajectory segments (e.g. with different velocities, accelerations, or boundary conditions) correspond to different polynomials p, and therefore to different initial register vectors. At segment boundaries, the hardware loads a new set of registers and proceeds.

5 Application to Real-Time Motion Control

Consider one degree of freedom (one axis) of a CNC machine or a robot joint. A planner on a host processor computes a piecewise polynomial trajectory $q(t)$ for position, where each segment is defined on an interval $[t_s, t_s + T_s]$ and is, say, of degree N.

For a given segment s:

- The host selects a sampling period h compatible with the servo loop (e.g. $h = 20\,\mu$s).
- It computes the polynomial coefficients for $q_s(t)$ on $[0, T_s]$ or directly samples $q_s(kh)$, $k = 0, \ldots, N$.
- It constructs the difference-register vector $\boldsymbol{r}_s^{(0)}$ associated with q_s as above.
- It sends $(\boldsymbol{r}_s^{(0)}, K_s)$ to the FPGA, where $K_s = \lfloor T_s/h \rfloor$ is the number of ticks in this segment.

On the FPGA, a trajectory generator block for that axis:

- stores $\boldsymbol{r}_s$ in fixed-point registers,
- at each tick, outputs $q_s(kh) \approx r_N^{(k)}$ and updates the registers using the additive recurrence,
- decrements a segment counter until K_s ticks have elapsed, then requests the next segment from the host.

The streamed samples $q_s(kh)$ can be used directly as position setpoints, or differentiated numerically, or interpreted as velocity or phase increments for a stepper phase accumulator, depending on the control architecture.

This approach yields:

- smooth, high-order trajectories (e.g. jerk-limited motion),
- a small and predictable hardware footprint (adders and registers only),

- the possibility of using higher-precision arithmetic off-line than what is available on the FPGA at run-time.

6 Discussion

The underlying mathematics—the representation of polynomials via finite forward differences and the vanishing of Δ^{N+1}—is classical. However, repackaging this as an "iterative polynomial sampler" with an explicit split between high-precision initialization and low-resource, fixed-point, real-time evaluation is practically appealing for embedded motion-control hardware.

In particular, for applications such as:

- CNC machines and 3D printers,
- multi-axis robot joints,
- stepper and servo drives requiring smooth S-curve trajectories,

this technique offers a compact alternative to Horner-based polynomial evaluation or large look-up tables with interpolation.