Title:
Rootfinding in Bernstein Basis

Authors:
Gábor Valasek, valasek@inf.elte.hu, Eötvös Loránd University, Budapest

Keywords:
Polynomial rootfinding, Bernstein basis, Newton-Raphson, Bracketed Newton method

DOI: 10.14733/cadconfP.2024.334-338

Introduction:

A wide variety of operations have well-known formulae in terms of Bézier control data. For example, succinct and convenient formulations are available for subdivision, degree elevation, and integration. However, some basic identities, such as the quadratic formula, are less frequently found in the literature. The purpose of this paper is to formulate solutions to root finding in the Bernstein basis. We show that the quadratic formula and the Vieta identities between the roots have straightforward expressions in the Bernstein basis. Beyond that, we adapt Yuksel's [1] algorithm of robust root finding to the Bernstein basis. We show that a recursion-free formulation akin to that of Peters et al. [2] is viable which have similar numerical characteristics as the original but is more suited for limited computational environments, such as GPUs.

Quadratic formula and root identities in Bernstein basis:

Let us assume that we are given polynomials in the Bernstein basis. One may find its roots by converting the Bézier control data to monomial coefficients and apply the classic quadratic formula. However, basis conversions incur extra computations and can amplify the error in coefficients. To circumvent this, the following theorem provides a direct quadratic formula for polynomials in the Bernstein basis.

Theorem 1. *Let $b(t) = (1-t)^2 b_0 + 2t(1-t)b_1 + t^2 b_2$, $t \in [0,1]$, denote a quadratic Bézier curve with $b_0, b_1, b_2 \in \mathbb{R}$ control scalars. Then the roots of the polynomial are*

$$t_{1,2} = \frac{-\Delta b_0 \pm \sqrt{b_1^2 - b_0 b_2}}{\Delta^2 b_0} \ , \tag{2.1}$$

where $\Delta^{k+1} b_i = \Delta^k b_{i+1} - \Delta^k b_i$, $i \in \{0, \dots, n-k-1\}$ are the order k forward differences with $\Delta^0 b_i = b_i$. The Vieta formulae in Bernstein coordinates are expressed as

$$t_1 + t_2 = -2\frac{\Delta b_0}{\Delta^2 b_0} \tag{2.2}$$

$$t_1 \cdot t_2 = \frac{b_0}{\Delta^2 b_0} \ . \tag{2.3}$$

Proof. The algebraic solution to $b(t) = 0$ is derived by first converting the Bézier control data to monomial basis coordinates symbolically, then applying the standard quadratic formula. The conversion yields

$$\begin{aligned}(1-t)^2 b_0 + 2t(1-t)b_1 + t^2 b_2 &= (b_0 - 2b_1 + b_2)t^2 + (2t - 2t^2)b_1 + t^2 b_2 \\ &= \Delta^2 b_0 \cdot t^2 + 2\Delta b_0 \cdot t + b_0 \ .\end{aligned}$$

Thus, the quadratic formula in terms of Bézier controla data is written as

$$\begin{aligned}t_{1,2} &= \frac{-2\Delta b_0 \pm \sqrt{(2\Delta b_0)^2 - 4 \cdot \Delta^2 b_0 \cdot b_0}}{2 \cdot \Delta^2 b_0} \\ &= \frac{-\Delta b_0 \pm \sqrt{(b_1 - b_0)^2 - (b_2 - 2b_1 + b_0) \cdot b_0}}{\Delta^2 b_0} \\ &= \frac{-\Delta b_0 \pm \sqrt{b_1^2 - 2b_0 b_1 + b_0^2 - (b_2 - 2b_1 + b_0)b_0}}{\Delta^2 b_0} \\ &= \frac{-\Delta b_0 \pm \sqrt{b_1^2 - b_0 b_2}}{\Delta^2 b_0} \ .\end{aligned}$$

Similarly, the Vieta formulae follow from direct computation as

$$t_1 + t_2 = \frac{-\Delta b_0 + \sqrt{b_1^2 - b_0 b_2}}{\Delta^2 b_0} + \frac{-\Delta b_0 - \sqrt{b_1^2 - b_0 b_2}}{\Delta^2 b_0} = -2\frac{\Delta b_0}{\Delta^2 b_0}$$

and

$$\begin{aligned}t_1 \cdot t_2 &= \frac{-\Delta b_0 + \sqrt{b_1^2 - b_0 b_2}}{\Delta^2 b_0} \cdot \frac{-\Delta b_0 - \sqrt{b_1^2 - b_0 b_2}}{\Delta^2 b_0} \\ &= \frac{b_0^2 - 2b_0 b_1 + b_0 b_2}{(\Delta^2 b_0)^2} \\ &= \frac{b_0 \cdot \Delta^2 b_0}{(\Delta^2 b_0)^2} \\ &= \frac{b_0}{\Delta^2 b_0} \ .\end{aligned}$$

□

Compared to the monomial quadratic formula, the above does not have constant multipliers in the discriminant and the denominator. Depending on the execution environment, this may translate into fewer operations.

Algorithm 1 SolveQuadraticBernstein(b_0, b_1, b_2)

$\Delta b_0, \Delta b_1 \leftarrow b_1 - b_0, b_2 - b_1$
$\Delta^2 b_0 \leftarrow \Delta b_1 - \Delta b_0$
if $|\Delta^2 b_0| \approx 0$ **then** ▷ Input is linear
  return $\left(-\frac{b_0}{2\Delta b_0}, N/A\right)$
end if
$D \leftarrow b_1^2 - b_0 b_2$
if $D \geq 0$ **then**
  $s \leftarrow -\left(\Delta b_0 + \text{sign}\left(\Delta b_0\right) \cdot \sqrt{D}\right)$ ▷ Stable root nominator
  $t_1 \leftarrow \frac{s}{\Delta^2 b_0}$
  $t_2 \leftarrow \frac{b_0}{\Delta^2 b_0 \cdot t_1}$
  return $(\min\{t_1, t_2\}, \max\{t_1, t_2\})$ ▷ Root ordering is optional in general, we rely on it later
else
  return $(N/A, N/A)$
end if

In practice, a numerically stable solution has to address the potential cancellation of significant digits in the nominator when $|\Delta b_0| \approx \sqrt{b_1^2 - b_0 b_2}$. A common approach [3] is to carry out the operation that avoids an effective subtraction, that is, select addition if $-\Delta b_0$ is non-negative and subtraction otherwise. Once a numerically more robust root is identified in such a way, one can apply the Vieta formulae to compute the other root. These are summarized in Algorithm 1. N/A corresponds to non-existing or non-real roots. The algorithm also handles the cases when the input is linear and when $t = 0$ is a root.

For a degree n polynomial, one has the generalized Vieta formulae

$$\sum_{i=0}^{n} r_i = -\frac{a_{n-1}}{a_n} \ , \ \prod_{i=0}^{n} r_i = (-1)^n \frac{a_0}{a_n} \tag{2.4}$$

which can be adapted to the Bernstein basis over $[0, 1]$ by using the identities [4]

$$a_i = \binom{n}{i} \Delta^i b_i \tag{2.5}$$

to obtain

$$\sum_{i=0}^{n} r_i = -\frac{n\Delta^{n-1} b_0}{\Delta^n b_0} \ , \ \prod_{i=0}^{n} r_i = (-1)^n \frac{b_0}{\Delta^n b_0} \tag{2.6}$$

Higher degree polynomials:

Let us consider the general degree case. The i-th degree n Bernstein polynomial is denoted by $B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}$. Let us express the solutions to $b(t) = \sum_{i=0}^{b} B_i^n(t) b_i = 0$ in terms of the $b_i \in \mathbb{R}$ control data scalars. Closed-form expressions are not available above degree 4. However, even degrees 3 and 4 are often easier to handle numerically with iterative methods.

For degree n polynomials in the monomial basis, Yuksel proposed a bracketed Newton method in [1] that falls back to bisection if an unsafe step is inferred during the Newton phase. The brackets, i.e. subintervals, are determined such that the function is monotone over each. The endpoints of these subintervals are the zeros of the derivative of the input polynomial. These, in turn, are identified by

finding the roots of the derivative of the input, i.e. a degree $n-1$ polynomial. This yields a recursive algorithm until a degree is reached where an explicit root formula is known, for example a quadratic. Despite the simplicity of this approach, our goal is to implement the solution on the GPU that does not support recursion natively. As such, we followed a similar approach as shown by Peters et al. in [2] that transforms the recursion into an iteration.

First, we compute the roots of the $(n-2)$-th derivative of the input, which is a quadratic polynomial. The roots are the endpoints of the brackets for the $(n-3)$-th derivative, yielding up to 2 intervals. The subtlety of implementation lies in avoiding storage requirements that would be $\Theta(n^2)$. This can be done by noting that by successive integration and proper handling of integration constants, we can reconstruct the the original polynomial.

Our Bernstein-based modification is summarized in Algorithm 2. It discards constant multipliers of the derivatives, as we only need to identify roots that are invariant under scaling. This also keeps computations within the bounds of the initial control data and does not inflate. The bracketed NewtonBisection method follows [1, 2] adapted to the Bernstein basis. This means that if the Newton step leaves the input $[r_i, r_{i+1}]$ bracket, it falls back to bisection.

Algorithm 2 FindRootsBernstein($\{b_i\}_{i=0}^n$, $[a,b] \subset \mathbb{R}$, $\epsilon > 0$)

▷ Compute the scaled Bézier control date of the derivatives
$\{d_i\}_{i=0}^n \leftarrow \{b_i\}_{i=0}^n$ ▷ Coefficients of the derivatives
$\{r_i\}_{i=0}^n \leftarrow a$ ▷ Roots that also correspond to bracket endpoints for derivatives
for $k = 1, \ldots, n-2$ **do** ▷ Compute the control data of the scaled derivative that is a quadratic
  for $i = 0, \ldots, n-k$ **do**
    $d_i \leftarrow d_{i+1} - d_i$ ▷ Discard the degree multiplier in the derivative for numerical robustness
  end for
end for
▷ Find the roots of the quadratic derivative and initialize the first brackets
$t_1, t_2 \leftarrow$ SolveQuadraticBernstein(d_0, d_1, d_2)
$r_{n-2} \leftarrow t_1$ if $t_1 \neq N/A$ else a
$r_{n-1} \leftarrow t_2$ if $t_2 \neq N/A$ else a
$r_n \leftarrow b$
▷ Iterate from the $n-3$-th derivative up to the input polynomial
for $k = 3, \ldots, n$ **do** ▷ The degree of the polynomial we compute in the loop body is k
  for $i = k-1, \ldots, 0$ **do** ▷ Decreasing loop
    $d_i \leftarrow d_{i+1} - d_i$ ▷ Integrate coefficients, discard scale
  end for
  $beginValue \leftarrow d_0$ ▷ Evaluate the derivative at a
  ▷ Process brackets $[r_i, r_{i+1}]$ and compute the root in each
  for $i = 0, \ldots, n-k-1$ **do**
    $r \leftarrow$ NewtonBisection$(\{r_i\}_{i=0}^n, \{d_i\}_{i=0}^n, [r_i, r_{i+1}], beginValue, b-a, \epsilon)$ if converged else r_{i-1}
  end for
end for
$r_n \leftarrow N/A$

Conclusions:

We derived a stable quadratic formula for polynomials supplied in the Bernstein basis. For higher degree polynomials, we adapted Yuksel's bracketed Newton method for the computation of the real roots

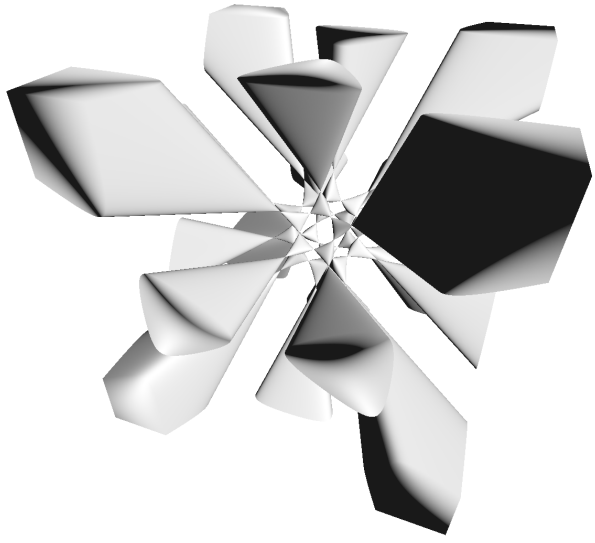

Fig. 1: Rendering a modified Barth Sextic. The composition of the parametric ray and the implicit function was written in the Bernstein basis as a degree six polynomial. Difference between render times was statistically insignificant, despite the fact that the Bernstein polynomials were evaluated with the quadratic-time de Casteljau algorithm while the monomial polynomial relied on linear-time Horner.

of polynomials, following Peters et al.'s GPU implementation considerations. The monomial and our proposed Bernstein methods provide similar performance characteristics but we can rely on a numerically more stable evaluation for function and derivative values via the de Casteljau algorithm, as well as employ trivial empty space skipping by considering whether the bounding interval of the control points contains 0. The method was tested on scenes such as shown in Fig. 1. The main application of our formulation is when the input polynomials are already given in the Bernstein basis, which allows user to bypass conversion from Bernstein to monomial basis. Future research is required to investigate if linear time Bézier evaluation algorithms can provide improved performance.

Gábor Valasek, https://orcid.org/0000-0002-0007-8647

References:

[1] Yuksel, C.: High-Performance Polynomial Root Finding for Graphics. *Proc. ACM Comput. Graph. Interact. Tech.*. **5** (2022,7), https://doi.org/10.1145/3543865

[2] Peters, C.; Patel, T.; Usher, W.; Johnson, C.: Ray Tracing Spherical Harmonics Glyphs. *Vision, Modeling, And Visualization.* (2023) https://doi.org/10.2312/vmv.20231223

[3] Press, W.; Teukolsky, S.; Vetterling, W.; Flannery, B.: Numerical Recipes 3rd Edition: The Art of Scientific Computing. (Cambridge University Press,2007) https://dl.acm.org/doi/10.5555/1403886

[4] Barrio, R.; Peña, J.: Basis conversions among univariate polynomial representations. *Comptes Rendus. Mathematique.* **339**, 293-298 (2004) https://doi.org/10.1016/j.crma.2004.06.017