A comprehensive comparison of algorithms for evaluating rational Bézier curves

Chiara Fuda · Andriamahenina Ramanantoanina · Kai Hormann

Abstract

Bézier curves are very important tools in various fields and applications, such as computer graphics and computer-aided design. The de Casteljau algorithm is the first method introduced for evaluating polynomial Bézier curves, later also generalized to the rational case and surfaces. Although it presents an elegant definition through convex combinations and generally yields stable results, it has quadratic time complexity, which means that its computational cost can increase significantly with the number of control points. This represents a significant limitation, especially when dealing with high-degree curves and real-time applications. For this reason, numerous studies have been conducted in order to provide alternative approaches and more efficient algorithms. In this paper, we present a collection of the most commonly used algorithm in the state-of-the-art, also providing a comparison of their efficiency and their numerical stability.

Citation Info

Journal
Dolomites Research Notes on Approximation

Volume
17(3), September 2024

Pages
56–79

DOI
10.14658/PUPJ-DRNA-2024-3-9

1 Introduction

Bézier curves were originally introduced in the context of car modelling for major automotive manufacturers [4, 5, 7]. Nowadays, their utility extends across numerous fields, like computer-aided design, simulation, approximation, robotics, artificial intelligence, etc. Many applications in these domains require real-time interactions or live updates, thus necessitating fast evaluation times. For this reason, over the years, there have been numerous studies dedicated to developing efficient evaluation algorithms for Bézier curves. In this paper, we aim to present a comparison of the most commonly used algorithms, focusing not only on their efficiency, but also on their numerical stability.

Given a set of $n+1$ control points $P_0,\dots,P_n \in \mathbb{R}^2$ with associated positive weights $w_0,\dots,w_n \in \mathbb{R}$, we define a *rational Bézier curve* $P\colon [0,1]\to\mathbb{R}^2$ as

$$P(t)=\frac{\sum_{i=0}^{n} B_i^n(t) w_i P_i}{\sum_{i=0}^{n} B_i^n(t) w_i}, \tag{1}$$

where

$$B_i^n(t)=\binom{n}{i} t^i (1-t)^{n-i}, \qquad i=0,\dots,n, \tag{2}$$

represents the Bernstein basis composed by polynomials of degree n and $t\in[0,1]$ is the parameter along the curve. There exist numerous methods for computing rational Bézier curves, such as adaptations of the classic de Casteljau algorithm for polynomials in the rational case, or more efficient approaches employing Horner-like schemes or basis conversions. We now present the most commonly used algorithms and, for each of them, we describe how it is implemented and provide the pseudocode in the Appendix. Afterwards, we investigate the numerical stability of each algorithm and derive an upper bound on the relative error for most of them (Section 2). Finally, we conduct an efficiency analysis (Section 3) and present some numerical experiments to support our results (Section 4).

1.1 Rational de Casteljau algorithms

The most straightforward approach to compute $P(t)$ is by using the classic quadratic time de Casteljau algorithm for polynomials [6]. In the case of a rational Bézier curve of the type in (1), we recall that it can be considered as the central projection of the spatial polynomial curve

$$\hat{P}(t)=\sum_{i=0}^{n} B_i^n(t)\hat{P}_i, \qquad \hat{P}_i=\begin{pmatrix} w_i P_i \\ w_i \end{pmatrix}, \tag{3}$$

under the projection

$$\operatorname{proj}(x,y,z)=\left(\frac{x}{z},\frac{y}{z}\right). \tag{4}$$

This implies that we can apply the classical de Casteljau algorithm to $\hat{P}(t)$ and then project the final result according to (4) (Algorithm 1 and 2). This process is equivalent to first computing the values of the numerator and the denominator with the recursive formulas

$$\begin{cases} N_i^0 = w_i P_i, \\ N_i^r = N_i^{r-1}(1-t)+N_{i+1}^{r-1}\,t \end{cases} \quad \text{and} \quad \begin{cases} D_i^0 = w_i, \\ D_i^r = D_i^{r-1}(1-t)+D_{i+1}^{r-1}\,t \end{cases} \tag{5}$$

for $i=0,\dots,n$ and $r=1,\dots,n$, respectively, and then the final result as $P(t)=N_0^n\big/D_0^n$. We also note that this method exhibits quadratic complexity.

Alternatively, Farin [13] adapts this approach into a more robust quadratic time algorithm (Algorithm 3) with additional geometric meaning, given by

$$\begin{cases} w_i^0 = w_i, \\ P_i^0 = P_i, \\ w_i^r = w_i^{r-1}(1-t)+w_{i+1}^{r-1}\,t, \\ P_i^r = \frac{P_i^{r-1}w_i^{r-1}}{w_i^r}(1-t)+\frac{P_{i+1}^{r-1}w_{i+1}^{r-1}}{w_i^r}\,t \end{cases} \tag{6}$$

for $i=0,\dots,n$ and $r=1,\dots,n$, and $P(t)=P_0^n$.

1.2 Horner-like algorithms

Volk and Schumaker [23] are the first to achieve an algorithm for computing polynomial Bézier curves with linear time complexity. Their idea is to use nested multiplications for the computation, which results in a significant gain in terms of efficiency. We present a straightforward extension of the VS algorithm by first applying it on the numerator and the denominator of $P(t)$, and then simplifying some common factors. In particular, we express the rational Bézier curve in (1) equivalently as

$$P(t)=\frac{\sum_{i=0}^n x^{n-i}\binom{n}{i}w_i P_i}{\sum_{i=0}^n x^{n-i}\binom{n}{i}w_i}, \qquad x=\begin{cases}(1-t)/t, & t>1/2,\\ t/(1-t), & t\le 1/2.\end{cases} \tag{7}$$

There are many methods for evaluating a polynomial; Goldman [17, Chapter 5] highlights two approaches: one using a Horner scheme, and the other employing a ladder pattern. However, Warren [27] shows that these forms are equivalent for the monomial basis, so we consider the former. Therefore, the VS algorithm evaluates the numerator and the denominator using a Horner scheme (Algorithm 4 and 5) as

$$P(t)=\begin{cases} \dfrac{\binom{n}{n}w_nP_n+x\left(\binom{n}{n-1}w_{n-1}P_{n-1}+x\left(\binom{n}{n-2}w_{n-2}P_{n-2}+\cdots+x\left(\binom{n}{1}w_1P_1+x\binom{n}{0}w_0P_0\right)\cdots\right)\right)}{\binom{n}{n}w_n+x\left(\binom{n}{n-1}w_{n-1}+x\left(\binom{n}{n-2}w_{n-2}+\cdots+x\left(\binom{n}{1}w_1+x\binom{n}{0}w_0\right)\cdots\right)\right)}, & t>1/2,\\[2ex] \dfrac{\binom{n}{0}w_0P_0+x\left(\binom{n}{1}w_1P_1+x\left(\binom{n}{2}w_2P_2+\cdots+x\left(\binom{n}{n-1}w_{n-1}P_{n-1}+x\binom{n}{n}w_nP_n\right)\cdots\right)\right)}{\binom{n}{0}w_n+x\left(\binom{n}{1}w_1+x\left(\binom{n}{2}w_2+\cdots+x\left(\binom{n}{n-1}w_{n-1}+x\binom{n}{n}w_n\right)\cdots\right)\right)}, & t\le 1/2. \end{cases}$$

With the same strategy, Farin [14] presents another Horner-like algorithm (Algorithm 4 and 6) by setting $s=1-t$ and computing $P(t)$ in (1) as

$$\begin{aligned} P(t)&=\frac{\sum_{i=0}^n t^i s^{n-i}\binom{n}{i}w_iP_i}{\sum_{i=0}^n t^i s^{n-i}\binom{n}{i}w_i}\\ &=\frac{\left(\cdots\left(\left(\binom{n}{0}w_0P_0s+\binom{n}{1}w_1P_1t\right)s+\binom{n}{2}w_2P_2t^2\right)s+\cdots+\binom{n}{n-1}w_{n-1}P_{n-1}t^{n-1}\right)s+\binom{n}{n}w_nP_nt^n}{\left(\cdots\left(\left(\binom{n}{0}w_0s+\binom{n}{1}w_1t\right)s+\binom{n}{2}w_2t^2\right)s+\cdots+\binom{n}{n-1}w_{n-1}t^{n-1}\right)s+\binom{n}{n}w_nt^n}. \end{aligned} \tag{8}$$

1.3 Geometric approach

On the one hand, while the rational de Casteljau adaptation by Farin [13] has some nice geometric interpretation, it can only be done in quadratic time. On the other hand, the VS algorithm has linear time complexity, but it lacks geometric interpretation and properties. For this reason, Woźny and Chudy [28] introduce a new linear time algorithm that has a nice geometric interpretation. In particular, $P(t)$ can be computed recursively (Algorithm 7) using a Horner-like scheme and convex combinations as

$$\begin{cases} h_0 = 1, & h_i = \dfrac{w_i h_{i-1} t(n-i+1)}{w_{i-1} i(1-t) + w_i h_{i-1} t(n-i+1)}, \\ T_0 = P_0, & T_i = (1-h_i) T_{i-1} + h_i P_i. \end{cases} \tag{9}$$

From these recursive formulas, this algorithm has an elegant geometric interpretation since $T_i \in [T_{i-1}, P_i]$.

1.4 Wang–Ball algorithm

Another approach to achieve an algorithm with linear time complexity is by converting the Bernstein basis into a different basis. There exist several methods in this direction, such as transforming the Bernstein into the Wang–Ball basis [10, 20, 26], the DP basis [8, 11, 12], and other similar types of bases [9]; the former is proven to be the most efficient. The rational Wang–Ball curve, defined by the control points $R_0, \ldots, R_n$ with their respective weights $v_0, \ldots, v_n$, is given by

$$P(t) = \frac{\sum_{i=0}^{n} A_i^n(t) v_i R_i}{\sum_{i=0}^{n} A_i^n(t) v_i}, \tag{10}$$

where the Wang–Ball basis $\{A_i^n\}_{i=0,\ldots,n}$ is defined as

$$A_i^n(t) = \begin{cases} (2t)^i (1-t)^{i+2}, & 0 \le i \le \lfloor n/2 \rfloor - 1, \\ (2t)^{\lfloor n/2 \rfloor} (1-t)^{\lfloor n/2 \rfloor}, & i = \lfloor n/2 \rfloor, \\ (2(1-t))^{\lfloor n/2 \rfloor} t^{\lceil n/2 \rceil}, & i = \lceil n/2 \rceil, \\ A_{n-i}^n(1-t), & \lceil n/2 \rceil + 1 \le i \le n. \end{cases} \tag{11}$$

Actually, in order to achieve a linear time method, its implementation uses a recursive algorithm similar to (6), but for the new set of control points and weights (Algorithm 10). Specifically, it starts by setting

$$n_0 = n, \qquad v_i^0 = v_i, \qquad \text{and} \qquad R_i^0 = R_i, \qquad i = 0, \ldots, n_0, \tag{12}$$

and then, at each step $r = 1, \ldots, n$ of the recursion, it defines $n_r = n - r$ new weights and control points. In particular, if n_r is odd, they are given by

$$\begin{cases} v_i^r = v_i^{r-1}, & i = 0, \ldots, \frac{n_r-3}{2}, \\ v_i^r = v_i^{r-1}(1-t) + v_{i+1}^{r-1} t, & i = \frac{n_r-1}{2}, \\ v_i^r = v_i^{r-1}, & i = \frac{n_r+1}{2}, \ldots, n_r, \end{cases} \quad \text{and} \quad \begin{cases} R_i^r = R_i^{r-1}, & i = 0, \ldots, \frac{n_r-3}{2}, \\ R_i^r = \frac{R_i^{r-1} v_i^{r-1}}{v_i^r}(1-t) + \frac{R_{i+1}^{r-1} v_{i+1}^{r-1}}{v_i^r} t, & i = \frac{n_r-1}{2}, \\ R_i^r = R_i^{r-1}, & i = \frac{n_r+1}{2}, \ldots, n_r, \end{cases} \tag{13}$$

while, if n_r is even, they are

$$\begin{cases} v_i^r = v_i^{r-1}, & i = 0, \ldots, \frac{n_r}{2} - 2, \\ v_i^r = v_i^{r-1}(1-t) + v_{i+1}^{r-1} t, & i = \frac{n_r}{2} - 1, \frac{n_r}{2}, \\ v_i^r = v_i^{r-1}, & i = \frac{n_r}{2} + 1, \ldots, n_r, \end{cases} \quad \text{and} \quad \begin{cases} R_i^r = R_i^{r-1}, & i = 0, \ldots, \frac{n_r}{2} - 2, \\ R_i^r = \frac{R_i^{r-1} v_i^{r-1}}{v_i^r}(1-t) + \frac{R_{i+1}^{r-1} v_{i+1}^{r-1}}{v_i^r} t, & i = \frac{n_r}{2} - 1, \frac{n_r}{2}, \\ R_i^r = R_i^{r-1}, & i = \frac{n_r}{2} + 1, \ldots, n_r, \end{cases} \tag{14}$$

and the result is $P(t) = R_0^n$. Before proceeding with this algorithm, there is a prepossessing step to get the values $v_0, \ldots, v_n$ and $R_0, \ldots, R_n$ (Algorithm 8 and 9). In particular, the weights and control points of the Bézier and Wang–Ball representations can be converted back-and-forth by means of a matrix multiplication [19]. However, for the sake of numerical stability, Dejdumrong et al. [10] present the explicit formulas to obtain

the Wang–Ball control points and weights from the corresponding Bézier ones, that are

$$\begin{cases} v_0 = w_0, \\ v_n = w_n, \\ v_i = \frac{1}{2^i}\Big[\binom{n}{i} w_i - \sum_{k=0}^{i-1} 2^k \binom{n-2-2k}{i-k} v_k - \sum_{k=n-i+1}^{n} 2^{n-k} \binom{2k-2-n}{k-i} v_k\Big], & i < \lfloor n/2 \rfloor, \\ v_i = \frac{1}{2^{n-i}}\Big[\binom{n}{i} w_i - \sum_{k=0}^{n-i} 2^k \binom{n-2-2k}{i-k} v_k - \sum_{k=i+1}^{n} 2^{n-k} \binom{2k-2-n}{k-i} v_k\Big], & i > \lceil n/2 \rceil, \\ v_i = \frac{1}{2^i}\Big[\binom{n}{i} w_i - \sum_{k=0}^{i-1} 2^k \binom{n-2-2k}{i-k} v_k - \sum_{k=i+2}^{n} 2^{n-k} \binom{2k-2-n}{k-i} v_k\Big], & i = \lfloor n/2 \rfloor, \\ v_i = \frac{1}{2^{n-i}}\Big[\binom{n}{i} w_i - \sum_{k=0}^{i-2} 2^k \binom{n-2-2k}{i-k} v_k - \sum_{k=i+1}^{n} 2^{n-k} \binom{2k-2-n}{k-i} v_k\Big], & i = \lceil n/2 \rceil \end{cases} \tag{15}$$

and

$$\begin{cases} R_0 = P_0, \\ R_n = P_n, \\ R_i = \frac{1}{2^i v_i}\Big[\binom{n}{i} w_i P_i - \sum_{k=0}^{i-1} 2^k \binom{n-2-2k}{i-k} v_k R_k - \sum_{k=n-i+1}^{n} 2^{n-k} \binom{2k-2-n}{k-i} v_k R_k\Big], & i < \lfloor n/2 \rfloor, \\ R_i = \frac{1}{2^{n-i} v_i}\Big[\binom{n}{i} w_i P_i - \sum_{k=0}^{n-i} 2^k \binom{n-2-2k}{i-k} v_k R_k - \sum_{k=i+1}^{n} 2^{n-k} \binom{2k-2-n}{k-i} v_k R_k\Big], & i > \lceil n/2 \rceil, \\ R_i = \frac{1}{2^i v_i}\Big[\binom{n}{i} w_i P_i - \sum_{k=0}^{i-1} 2^k \binom{n-2-2k}{i-k} v_k R_k - \sum_{k=i+2}^{n} 2^{n-k} \binom{2k-2-n}{k-i} v_k R_k\Big], & i = \lfloor n/2 \rfloor, \\ R_i = \frac{1}{2^{n-i} v_i}\Big[\binom{n}{i} w_i P_i - \sum_{k=0}^{i-2} 2^k \binom{n-2-2k}{i-k} v_k R_k - \sum_{k=i+1}^{n} 2^{n-k} \binom{2k-2-n}{k-i} v_k R_k\Big], & i = \lceil n/2 \rceil. \end{cases} \tag{16}$$

We note that, before computing v_k and R_k, $k = 0, \dots, n$, the weights v_i and v_{n-i} and the control points R_i and R_{n-i}, $i = 0, \dots, k-1$, must be computed.

1.5 Bernstein–Fourier algorithm

Another series of approaches involving a transformation to another form are explored in [1, 2, 3]; the most efficient amongst them is the Bernstein–Fourier method. It involves applying the Inverse Fast Fourier Transform (IFFT) on the control points, that is computing the points $\hat{S}_i = \text{ifft}(\hat{P}_i)$, $i = 0, \dots, n$, for $\hat{P}_i$ in (3) (Algorithm 12). Then, $P(t)$ is the central projection on the xy-plane under the projection (4) of

$$\hat{P}(t) = \sum_{i=0}^{n} \big(\zeta^i t + (1-t)\big)^n \hat{S}_i, \tag{17}$$

where the ζ_i, $i = 0, \dots, n$, are the roots of unity of order $n+1$. Its implementation (Algorithm 11 and 13) requires $O(n \log n)$ time and involves complex number operations. However, there are some optimizations that can be performed so that this method can compete with the aforementioned methods (Algorithm 11 and 14). First, we note that

$$\hat{S}_{n+1-i} = \overline{\hat{S}_i} \qquad \text{for } i = \begin{cases} 1, \dots, \frac{n}{2}, & \text{if } n \text{ is even}, \\ 1, \dots, \frac{n-1}{2}, & \text{if } n \text{ is odd}. \end{cases}$$

Additionally, by letting $s = 1 - t$, we have

$$\overline{\big(\zeta_i t + (1-t)\big)^n} = \zeta^i \big(\zeta_i s + (1-s)\big)^n.$$

Hence, we can compute $\hat{P}(t)$ and $\hat{P}(1-t)$ simultaneously with $\hat{S}_k$, $k = 0, \dots, N$, for $N = (n+1)/2$. Then, if n is even, we have

$$\hat{P}(t) = \hat{S}_0 + 2 \sum_{i=1}^{n/2} \text{Re}\Big(\big(\zeta_i t + (1-t)\big)^n \hat{S}_i\Big),$$

$$\hat{P}(1-t) = \hat{S}_0 + 2 \sum_{i=1}^{n/2} \text{Re}\left(\big(\zeta_i t + (1-t)\big)^n \frac{\overline{\hat{S}_i}}{\overline{\zeta_i}}\right),$$

while, if n is odd, we have

$$\hat{P}(t) = \hat{S}_0 - (1-2t)^n \hat{S}_N + 2\sum_{i=1}^{N-1} \mathrm{Re}\Big(\big(\zeta_i t + (1-t)\big)^n \hat{S}_i\Big),$$
$$\hat{P}(1-t) = \hat{S}_0 - (1-2t)^n \hat{S}_N + 2\sum_{i=1}^{N-1} \mathrm{Re}\left(\big(\zeta_i t + (1-t)\big)^n \frac{\overline{\hat{S}_i}}{\overline{\zeta_i}}\right).$$

1.6 Barycentric algorithm

Finally, Ramanantoanina and Hormann [21] propose another alternative to convert the rational Bézier representation to a barycentric rational interpolating form (Algorithm 17 or, for a more optimized version, Algorithm 18). In particular, given a set of interpolation points $Q_0, \dots, Q_n$ with their respective weights $u_0, \dots, u_n$ and nodes $t_0, \dots, t_n$, a barycentric rational interpolant is defined as

$$P(t) = \frac{\sum_{i=0}^{n} \frac{u_i}{t-t_i} Q_i}{\sum_{i=0}^{n} \frac{u_i}{t-t_i}}. \tag{18}$$

The barycentric interpolation points and weights are related with the corresponding Bézier ones as

$$Q_i = P(t_i) \qquad \text{and} \qquad u_i = z(t_i) \prod_{k \neq i} \frac{1}{t_i - t_k}, \qquad i = 0, \dots, n,$$

where $z(t)$ is the denominator of $P(t)$ in (1). A common choice for the set of nodes is given by the Chebyshev nodes of the second kind in $[0,1]$, which are defined as $t_{n-i} = (\cos(i\pi/n)+1)/2$, $i = 0, \dots, n$. In this case, the weights turn out to be computed in linear time as [22]

$$u_i = (-1)^i \delta_i z(t_i), \qquad \delta_i = \begin{cases} 1/2, & i = 0 \text{ or } i = n, \\ 1, & i = 1, \dots, n-1. \end{cases}$$

Alternatively, we can also use uniformly distributed nodes $t_i = i/n$, $i = 0, \dots, n$, with weights of the form

$$u_i = (-1)^i \binom{n}{i} z(t_i).$$

For the sake of efficiency, we propose to compute the values $Q_i = P(t_i)$ by evaluating the rational Bézier curve P at t_i through an adapted version of the rational VS algorithm. Doing so, we can also obtain the values $z(t_i)$ within the same algorithm (Algorithm 15 and 16) as

$$z(t_i) = \sum_{i=0}^{n} x^{n-i} \binom{n}{i} w_i \times \begin{cases} t^n, & t > 1/2, \\ (1-t)^n, & t \leq 1/2, \end{cases} \tag{19}$$

for x in (7).

2 Numerical stability

Let us now focus on analysing the numerical stability of the different algorithms that evaluate a rational Bézier curve. We will examine all the methods introduced previously, except for the Bernstein–Fourier algorithm.

To proceed, we consider a computer that uses a set $\mathbb{F}$ of *floating-point numbers* with the corresponding *machine epsilon* ϵ and let $\mathrm{fl}\colon \mathbb{R} \to \mathbb{F}$ be the *rounding function* that maps each $x \in \mathbb{R}$ to the closest floating-point approximation $\mathrm{fl}(x) \in \mathbb{F}$. We then study the relative error $E \in \mathbb{R}^2$ defined as

$$E(t) = \frac{|\mathrm{fl}(P(t)) - P(t)|}{|P(t)|} = \left(\frac{|\mathrm{fl}(P_x(t)) - P_x(t)|}{|P_x(t)|}, \frac{|\mathrm{fl}(P_y(t)) - P_y(t)|}{|P_y(t)|} \right) \tag{20}$$

for each algorithm, where $P(t)$ is the exact result and $\mathrm{fl}(P(t))$ that of its finite-precision implementation. To do so, we assume [25] that for any $x \in \mathbb{R}$, $x \neq 0$, the relative error is bounded from above by the machine epsilon ϵ, or, equivalently, we can always find some $\delta \in \mathbb{R}$ with $|\delta| < \epsilon$, such that

$$\mathrm{fl}(x) = x(1+\delta). \tag{21}$$

The same holds for any arithmetic operation $* \in \{+, -, \times, \div\}$ between two arbitrary floating-point numbers $x, y \in \mathbb{F}$, that is, there exists some $\delta \in \mathbb{R}$ with $|\delta| < \epsilon$, such that

$$\mathrm{fl}(x * y) = (x * y)(1+\delta). \tag{22}$$

This property can also be extended to cases involving multiple operations, such as sums or products, where the upper bound on $|\delta|$ depends on the number of operations performed; for more detailed information, we refer the interested reader to Fuda et al. [16, Section 2]. Finally, we always assume that the input data t, w_i and P_i are floating-point numbers, so they do not introduce any numerical error during the computation.

2.1 Convex combinations

We start by examining the numerical stability of algorithms that evaluate a rational Bézier curve P at t through a recursive method defined by convex combinations. Specifically, we focus on the rational de Casteljau algorithm and the Wang–Ball algorithm. Regarding the former defined in (6), our analysis begins with a study of the error propagation in the weights w_i^r, followed by an investigation into the relative error of the values P_i^r. These results lead to an upper bound on the relative error E in (20) in the case of $P(t) = P_0^n$.

Lemma 2.1. *For any $t, w_0, \ldots, w_n \in \mathbb{F}$ and $r \in \{1, \ldots, n\}$, there exist $\omega_0^r, \ldots, \omega_n^r \in \mathbb{R}$, such that the weights w_i^r in (6) satisfy $\mathrm{fl}(w_i^r) = w_i^r(1+\omega_i^r)$, $i = 0, \ldots, n$, with $|\omega_i^r| \leq U(w_i^r)\epsilon + O(\epsilon^2)$ and*

$$U(w_i^r) = 3r.$$

Proof. First, we notice that

$$\begin{aligned}\mathrm{fl}(w_i^r) &= w_i^{r-1}(1+\omega_i^{r-1})(1-t)(1+\delta_1) + w_{i+1}^{r-1}(1+\omega_{i+1}^{r-1})t(1+\delta_2)\\ &= w_i^{r-1}(1-t)(1+\omega_i^{r-1}+\delta_1+O(\epsilon^2)) + w_{i+1}^{r-1}t(1+\omega_{i+1}^{r-1}+\delta_2+O(\epsilon^2)),\end{aligned}$$

where δ_1 and δ_2 are the errors introduced by the operations in the first and second addends, respectively, that are one product and one sum in both cases, plus one subtraction for the first addend only. Therefore, it follows from (22) that $|\delta_1|, |\delta_2| \leq 3\epsilon + O(\epsilon^2)$. Moreover, the intermediate value theorem further guarantees that

$$\mathrm{fl}(w_i^r) = (w_i^{r-1}(1-t) + w_{i+1}^{r-1}t)(1+\omega_i^r),$$

for some $\omega_i^r \in [\min(\omega_i^{r-1}+\delta_1+O(\epsilon^2), \omega_{i+1}^{r-1}+\delta_2+O(\epsilon^2)), \max(\omega_i^{r-1}+\delta_1+O(\epsilon^2), \omega_{i+1}^{r-1}+\delta_2+O(\epsilon^2))]$. Now, we can prove the statement by induction over r. The base case follows by the fact that $w_i^0 = w_i$, therefore $\omega_i^0 = 0$ for all $i = 0, \ldots, n$. Finally, the inductive step from $r-1$ to r follows from the fact that $|\omega_i^r| \leq \max_{j=i,i+1} |\omega_j^{r-1}| + 3\epsilon + O(\epsilon^2)$, together with the inductive hypothesis, that is $|\omega_i^{r-1}| \leq 3(r-1)\epsilon + O(\epsilon^2)$, $i = 0, \ldots, n$. □

Proposition 2.2. *For any $t, w_0, \ldots, w_n, P_0, \ldots, P_n \in \mathbb{F}$ and $r \in \{1, \ldots, n\}$, the relative errors of the P_i^r in (6) satisfy*

$$\frac{|\mathrm{fl}(P_i^r(t)) - P_i^r(t)|}{|P_i^r|} \leq \frac{\sum_{k=0}^r B_k^r(t)|P_{i+k}w_{i+k}|}{\left|\sum_{k=0}^r B_k^r(t)P_{i+k}w_{i+k}\right|}(3r^2+5r)\epsilon + O(\epsilon^2), \qquad i = 0, \ldots, n.$$

Therefore, the relative error in (20) for $P(t) = P_0^n$ satisfies

$$E(t) \leq \frac{\sum_{k=0}^n B_k^n(t)|P_k w_k|}{\left|\sum_{k=0}^n B_k^n(t)P_k w_k\right|}(3n^2+5n)\epsilon + O(\epsilon^2).$$

Proof. Denoting by φ_i^r the relative errors introduced by the computation of P_i^r, $i = 0, \ldots, n$ and $r = 1, \ldots, n$, we first notice that

$$\mathrm{fl}(P_i^r) = \frac{P_i^{r-1}(1+\varphi_i^{r-1})w_i^{r-1}(1+\omega_i^{r-1})(1-t)(1+\delta_1) + P_{i+1}^{r-1}(1+\varphi_{i+1}^{r-1})w_{i+1}^{r-1}(1+\omega_{i+1}^{r-1})t(1+\delta_2)}{w_i^r(1+\omega_i^r)},$$

where $|\omega_j^m| \le 3m\epsilon + O(\epsilon^2)$, $j = i, i+1$ and $m = r-1, r$, by Lemma 2.1 and δ_1 and δ_2 are the errors introduced by the operations in the first and second addends of the numerator, respectively, that are two products, one sum, and one division each, plus one subtraction for the first addend only. Therefore, it follows from (22) that $|\delta_1|, |\delta_2| \le 5\epsilon + O(\epsilon^2)$. By Taylor expansion, we know that

$$\frac{1}{(1+\omega_i^r)} = 1 - \omega_i^r + O(\epsilon^2),$$

hence

$$\mathrm{fl}(P_i^r) = P_i^r + \frac{P_i^{r-1} w_i^{r-1}(1-t)}{w_i^r}(\varphi_i^{r-1} + \omega_i^{r-1} - \omega_i^r + \delta_1 + O(\epsilon^2)) + \frac{P_{i+1}^{r-1} w_{i+1}^{r-1} t}{w_i^r}(\varphi_{i+1}^{r-1} + \omega_{i+1}^{r-1} - \omega_i^r + \delta_2 + O(\epsilon^2)).$$

Then, using the fact that $\mathrm{fl}(P_i^r) - P_i^r = P_i^r \varphi_i^r$, the triangle inequality, and the upper bounds on the relative errors introduced by the weights and the operations, we obtain

$$\begin{aligned} |P_i^r \varphi_i^r w_i^r| &\le |P_i^{r-1}\varphi_i^{r-1} w_i^{r-1}(1-t) + P_{i+1}^{r-1}\varphi_{i+1}^{r-1} w_{i+1}^{r-1} t| \\ &\quad + |P_i^{r-1} w_i^{r-1}(1-t)(\omega_i^{r-1} - \omega_i^r + \delta_1) + P_{i+1}^{r-1} w_{i+1}^{r-1} t(\omega_{i+1}^{r-1} - \omega_i^r + \delta_2)| + O(\epsilon^2) \\ &\le |P_i^{r-1}\varphi_i^{r-1} w_i^{r-1}|(1-t) + |P_{i+1}^{r-1}\varphi_{i+1}^{r-1} w_{i+1}^{r-1}| t \\ &\quad + (|P_i^{r-1} w_i^{r-1}|(1-t) + |P_{i+1}^{r-1} w_{i+1}^{r-1}| t)(6r+2)\epsilon + O(\epsilon^2). \end{aligned} \tag{23}$$

In general, we know that[1] $P_j^m w_j^m = \sum_{k=0}^m B_k^m P_{j+k} w_{j+k}$, $j = 0, \dots, n$ and $m = 1, \dots, n$, therefore, by also using the relations $B_k^{r-1}(1-t) = (r-k)/r\, B_k^r$ and $B_k^{r-1} t = (k+1)/r\, B_{k+1}^r$, $k = 0, \dots, r-1$, we obtain

$$\begin{aligned} |P_i^{r-1} w_i^{r-1}|(1-t) + |P_{i+1}^{r-1} w_{i+1}^{r-1}| t &= \sum_{k=0}^{r-1} B_k^{r-1}(1-t)|P_{i+k} w_{i+k}| + \sum_{k=0}^{r-1} B_k^{r-1} t |P_{i+1+k} w_{i+1+k}| \\ &= \sum_{k=0}^{r-1} \frac{r-k}{r} B_k^r |P_{i+k} w_{i+k}| + \sum_{k=0}^{r-1} \frac{k+1}{r} B_{k+1}^r |P_{i+1+k} w_{i+1+k}| \\ &= B_0^r |P_i w_i| + \sum_{k=1}^{r-1} \left(\frac{r-k}{r} + \frac{k}{r} \right) B_k^r |P_{i+k} w_{i+k}| + B_r^r |P_{i+r} w_{i+r}| \\ &= \sum_{k=0}^{r} B_k^r |P_{i+k} w_{i+k}| \end{aligned} \tag{24}$$

and, by (23),

$$|P_i^r \varphi_i^r w_i^r| \le |P_i^{r-1}\varphi_i^{r-1} w_i^{r-1}|(1-t) + |P_{i+1}^{r-1}\varphi_{i+1}^{r-1} w_{i+1}^{r-1}| t + \sum_{k=0}^{r} B_k^r |P_{i+k} w_{i+k}|(6r+2)\epsilon + O(\epsilon^2). \tag{25}$$

Now, we can prove the statement by induction over r. The base case follows by the fact that $P_i^0 = P_i$, $i = 0, \dots, n$, hence $\varphi_i^0 = 0$. Finally, the inductive step from $r-1$ to r follows from the inductive hypothesis, that is

$$|P_i^{r-1}\varphi_i^{r-1} w_i^{r-1}| \le \sum_{k=0}^{r-1} B_k^{r-1} |P_{i+k} w_{i+k}| [3(r-1)^2 + 5(r-1)]\epsilon + O(\epsilon^2), \qquad i = 0, \dots, n,$$

together with (25) and the fact that, by (24),

$$\sum_{k=0}^{r-1} B_k^{r-1}(1-t)|P_{i+k} w_{i+k}| + \sum_{k=0}^{r-1} B_k^{r-1} t |P_{i+1+k} w_{i+1+k}| = \sum_{k=0}^{r} B_k^r |P_{i+k} w_{i+k}|.$$

□

We now turn our attention to the definition of the Wang–Ball algorithm in (12)–(14), which is very similar to the rational de Casteljau method in (6), except for two differences. Firstly, only the "central" Wang–Ball weights and control points are updated at each step $r = 1, \dots, n$. Secondly, we cannot assume that the input data v_i and R_i are exact, as they are themselves the result of the conversion formulas in (15)–(16). On the

[1] In the proof, we omit the dependence on the variable t of the basis functions, that is, B_i^n means $B_i^n(t)$.

one hand, although only a few weights change at each iteration, the final error propagation is the same as for the recursive formulas in (6), because some of the v_i^r and R_i^r are modified at each step r. Consequently, we can use the same proof technique of Lemma 2.1 to analyse the error propagation in the weights v_i^r and of Proposition 2.2 to get the upper bounds on the relative errors of the values R_i^r and $P(t)=R_0^n$. On the other hand, in this scenario, we also have to consider the initial errors in the weights v_i and control points R_i, which are introduced in the preprocessing step that converts the Bézier weights and control points into their corresponding Wang–Ball ones. Therefore, we state below the equivalent of Lemma 2.1 and Proposition 2.2 in the case of Wang–Ball algorithm.

Lemma 2.3. *Suppose that there exist $\upsilon_0^0,\dots,\upsilon_n^0\in\mathbb{R}$ with*

$$\mathrm{fl}(v_i)=v_i(1+\upsilon_i^0),\qquad |\upsilon_i^0|\le U(v_i)\epsilon+O(\epsilon^2),\qquad i=0,\dots,n,$$

for some constants $U(v_i)$. Then, for any $r\in\{1,\dots,n\}$, there exist $\upsilon_0^r,\dots,\upsilon_{n_r}^r\in\mathbb{R}$, such that the weights v_i^r in (13)–(14) satisfy $\mathrm{fl}(v_i^r)=v_i^r(1+\upsilon_i^r)$, $i=0,\dots,n_r$, with $|\upsilon_i^r|\le U(v_i^r)\epsilon+O(\epsilon^2)$ and

$$U(v_i^r)=3r+\max_{j=0,\dots,n}U(v_j).$$

Proposition 2.4. *Suppose that there exist $\upsilon_0^0,\dots,\upsilon_n^0\in\mathbb{R}$ with*

$$\mathrm{fl}(v_i)=v_i(1+\upsilon_i^0),\qquad |\upsilon_i^0|\le U(v_i)\epsilon+O(\epsilon^2),\qquad i=0,\dots,n$$

and $\rho_0^0,\dots,\rho_n^0\in\mathbb{R}$ with

$$\mathrm{fl}(R_i)=R_i(1+\rho_i^0),\qquad |\rho_i^0|\le U(R_i)\epsilon+O(\epsilon^2),\qquad i=0,\dots,n,$$

for some constants $U(v_i)$ and $U(R_i)$. Then, for any $r\in\{1,\dots,n\}$, the relative errors of the R_i^r in (13)–(14) satisfy

$$\frac{|\mathrm{fl}(R_i^r(t))-R_i^r(t)|}{|R_i^r|}\le\frac{\sum_{k=0}^r A_k^r(t)|R_{i+k}v_{i+k}|}{\left|\sum_{k=0}^r A_k^r(t)R_{i+k}v_{i+k}\right|}\Big(3r^2+5r+\max_{j=0,\dots,n}U(v_j)+\max_{k=0,\dots,n}U(R_k)\Big)\epsilon+O(\epsilon^2),\qquad i=0,\dots,n.$$

Therefore, the relative error in (20) for $P(t)=R_0^n$ satisfies

$$E(t)\le\frac{\sum_{k=0}^n A_k^n(t)|R_kv_k|}{\left|\sum_{k=0}^n A_k^n(t)R_kv_k\right|}\Big(3n^2+5n+\max_{j=0,\dots,n}U(v_j)+\max_{k=0,\dots,n}U(R_k)\Big)\epsilon+O(\epsilon^2).$$

Finally, to provide a comprehensive understanding of the error propagation within the Wang–Ball algorithm, we also present an analysis of the numerical stability of the conversion formulas in (15)–(16), which provides an initial estimate of the constants $U(v_i)$ and $U(R_i)$, $i=0,\dots,n$, of Lemma 2.3 and Proposition 2.4. Before delving into these details, we introduce some notation to shorten the expressions of the v_i and R_i. Considering $i\in\{0,\dots,n\}$, we define $e\in\mathbb{N}$ as

$$e=\begin{cases} i, & i\le\lfloor n/2\rfloor,\\ n-i, & i\ge\lceil n/2\rceil\end{cases}$$

and the sets of indexes $I_{1,i}$ and $I_{2,i}$ as

$$I_{1,i}=\begin{cases}\{0,1,\dots,i-1\}, & i\le\lfloor n/2\rfloor,\\ \{0,1,\dots,i-2\}, & i=\lceil n/2\rceil,\\ \{0,1,\dots,n-i\}, & i>\lceil n/2\rceil,\end{cases}\qquad I_{2,i}=\begin{cases}\{n-i+1,n-i+2,\dots,n\}, & i<\lfloor n/2\rfloor,\\ \{i+2,i+3,\dots,n\}, & i=\lfloor n/2\rfloor,\\ \{i+1,i+2,\dots,n\}, & i\ge\lceil n/2\rceil.\end{cases}$$

We then set

$$b_i=\binom{n}{i},\qquad a_k=2^k\binom{n-2-2k}{i-k},\qquad\text{and}\qquad c_k=2^{n-k}\binom{2k-2-n}{k-i},$$

so that we can express the weights v_i in (15) as

$$v_i=\frac{1}{2^e}\left(b_iw_i-\sum_{k\in I_{1,i}}a_kv_k-\sum_{k\in I_{2,i}}c_kv_k\right),\qquad i=0,\dots,n\tag{26}$$

and the control points R_i in (16) as

$$R_i = \frac{1}{2^e v_i}\left(b_i w_i P_i - \sum_{k\in I_{1,i}} a_k v_k R_k - \sum_{k\in I_{2,i}} c_k v_k R_k \right), \qquad i = 0,\dots,n. \tag{27}$$

Moreover, we denote by M_i the maximum of the constants $A_i = \max\{a_k : k \in I_{1,i}\}$ and $C_i = \max\{c_k : k \in I_{2,i}\}$, $i = 1,\dots,n-1$.

Lemma 2.5. *For any $t, w_0,\dots,w_n \in \mathbb{F}$, there exist $\upsilon_0,\dots,\upsilon_n \in \mathbb{R}$, such that the Wang–Ball weights v_i in (26) satisfy* $\mathrm{fl}(v_i) = v_i(1+\upsilon_i)$, $i = 0,\dots,n$, *with* $|\upsilon_i| \le U(v_i)\epsilon + O(\epsilon^2)$ *and*

$$\begin{aligned} U(v_i) = {} & \frac{\max_{j=1,n-1,\dots,n-i,i}\left(b_j w_j + \sum_{k\in I_{1,j}} a_k v_k + \sum_{k\in I_{2,j}} c_k v_k\right)}{\left|b_i w_i - \sum_{k\in I_{1,i}} a_k v_k - \sum_{k\in I_{2,i}} c_k v_k\right|} M_1 M_{n-1}\cdots M_{n-i} M_i \\ & \times \begin{cases} (2i+1)!, & i < \lceil n/2\rceil, \\ (2(n-i)+2)!, & i \ge \lceil n/2\rceil. \end{cases} \end{aligned} \tag{28}$$

Proof. First of all, we notice that the weights are computed in the order $v_0, v_n, v_1, v_{n-1}, v_2, v_{n-2},\dots, v_{m-1}, v_m$, for $m = \lceil n/2\rceil$. Therefore, when computing v_i, $i = 1,\dots,n-1$, the number of v_k, $k \in I_{1,i}\cup I_{2,i}$, involved in (26) are exactly $2i$, if $i < \lceil n/2\rceil$, and $2(n-i)+1$, otherwise. At the end, they are at most n, which is the case of the "central" weight v_m. The proof is carried out assuming $i < \lceil n/2\rceil$, but similar arguments can be applied to the case $i \ge \lceil n/2\rceil$.

We first notice that[2]

$$\begin{aligned} \mathrm{fl}(v_i) &= \frac{1}{2^e}\left(b_i w_i(1+\delta_i) - \sum_{k\in I_{1,i}} a_k v_k(1+\upsilon_k)(1+\delta_k) - \sum_{k\in I_{2,i}} c_k v_k(1+\upsilon_k)(1+\delta_k)\right) \\ &= v_i + \frac{1}{2^e}\left(b_i w_i \delta_i - \sum_{k\in I_{1,i}} a_k v_k(\upsilon_k + \delta_k + O(\epsilon^2)) - \sum_{k\in I_{2,i}} c_k v_k(\upsilon_k + \delta_k + O(\epsilon^2))\right), \end{aligned}$$

where δ_j, $j = i$ or $j \in I_{1,i}\cup I_{2,i}$, are the errors introduced by the operations in the addends. In particular, these errors are affected at most[3] by one product and $2i$ sums. Therefore, it follows from (22) that $|\delta_j| \le (2i+1)\epsilon + O(\epsilon^2)$. Then, using the fact that $\mathrm{fl}(v_i) - v_i = v_i\upsilon_i$, the triangle inequality, and the upper bounds on the relative errors introduced by the operations in the addends, we obtain

$$|v_i\upsilon_i| \le \frac{1}{2^e}\left[\left(b_i w_i + \sum_{k\in I_{1,i}} a_k v_k + \sum_{k\in I_{2,i}} c_k v_k\right)(2i+1)\epsilon + \sum_{k\in I_{1,i}} a_k|v_k\upsilon_k| + \sum_{k\in I_{2,i}} c_k|v_k\upsilon_k| + O(\epsilon^2)\right].$$

We know that in $\sum_{k\in I_{1,i}} a_k|v_k\upsilon_k| + \sum_{k\in I_{2,i}} c_k|v_k\upsilon_k|$ are performed $2i-1$ sums, therefore, it follows that

$$|v_i\upsilon_i| \le \frac{1}{2^e}\left[\left(b_i w_i + \sum_{k\in I_{1,i}} a_k v_k + \sum_{k\in I_{2,i}} c_k v_k\right)(2i+1)\epsilon + (2i-1)M_i \max_{k\in I_{1,i}\cup I_{2,i}} |v_k\upsilon_k| + O(\epsilon^2)\right].$$

We can then use this inequality recursively and, recalling that $\upsilon_0 = \upsilon_n = 0$ and each time we go one step back in the recursion the set $I_{1,k}\cup I_{2,k}$ decreases by one, we get

$$|v_i\upsilon_i| \le \frac{1}{2^e}\max_{j=1,n-1,\dots,n-i,i}\left(b_j w_j + \sum_{k\in I_{1,j}} a_k v_k + \sum_{k\in I_{2,j}} c_k v_k\right)(2i+1)M\epsilon + O(\epsilon^2),$$

where

$$M = 1 + (2i-1)M_i + (2i-1)(2i-2)M_i M_{n-i} + \cdots + (2i-1)!M_i M_{n-i}\cdots M_{n-1}M_1 \le (2i-1)!M_i M_{n-i}\cdots M_{n-1}M_1 \times 2i,$$

which gives the statement. □

[2] Any operation with powers of 2 are exact in floating-point arithmetic, so they do not introduce any relative error.

[3] We assume that the computations of all binomial coefficients involve only integer operations, therefore they do not introduce any floating-point relative error.

Lemma 2.6. *For any $t, w_0, \dots, w_n \in \mathbb{F}$, there exist $\rho_0, \dots, \rho_n \in \mathbb{R}$, such that the Wang–Ball control points R_i in (27) satisfy $\mathrm{fl}(R_i) = R_i(1+\rho_i)$, $i = 0, \dots, n$, with $|\rho_i| \le U(R_i)\epsilon + O(\epsilon^2)$ and*

$$U(R_i) = U(R_i v_i) + U(v_i) + 1,$$

for $U(v_i)$ in (28) and

$$\begin{aligned} U(R_i v_i) \le & \frac{\max_{j=1,n-1,\dots,n-i,i}\left(b_j w_j |P_j| + \sum_{k\in I_{1,j}} a_k v_k |R_k| + \sum_{k\in I_{2,j}} c_k v_k |R_k|\right)}{\left|b_i w_i P_i - \sum_{k\in I_{1,i}} a_k v_k R_k - \sum_{k\in I_{2,i}} c_k v_k R_k\right|} M_1 M_{n-1} \cdots M_{n-i} M_i \\ & \times \begin{cases} (2i+2)!, & i < \lceil n/2 \rceil, \\ [2(n-i)+3]!, & i \ge \lceil n/2 \rceil. \end{cases} \end{aligned} \tag{29}$$

Proof. The study of the propagation of the error in $R_i v_i = 1/2^e\left(b_i w_i P_i - \sum_{k\in I_{1,i}} a_k v_k R_k - \sum_{k\in I_{2,i}} c_k v_k R_k\right)$ can be done with the same procedure used in Lemma 2.5, with the differences that every addend is now affected by one more product by P_i or R_k, and we also have to consider the relative errors υ_i introduced by the weights v_i, $i = 0, \dots, n$. Therefore, denoting by ϕ_i and δ the errors introduced by the computation of $R_i v_i$ and the division by v_i, respectively, we obtain by (22) and Lemma 2.5 that

$$\mathrm{fl}(R_i) = \frac{1}{2^e v_i(1+\upsilon_i)}\left(b_i w_i P_i - \sum_{k\in I_{1,i}} a_k v_k R_k - \sum_{k\in I_{2,i}} c_k v_k R_k\right)(1+\phi_i)(1+\delta), \qquad i = 0, \dots, n, \tag{30}$$

for $|\upsilon_i| \le U(v_i)\epsilon + O(\epsilon^2)$ with $U(v_i)$ in (28), $|\phi_i| \le U(R_i v_i)\epsilon + O(\epsilon^2)$ with $U(R_i v_i)$ in (29), and $|\delta| \le \epsilon$. Therefore, we can use Taylor expansion in (30) to get

$$\mathrm{fl}(R_i) = R_i(1+\phi_i)(1+\delta)(1-\upsilon_i + O(\epsilon^2)) = R_i(1+\phi_i - \upsilon_i + \delta + O(\epsilon^2))$$

and the statement follows for $\rho_i = \phi_i - \upsilon_i + \delta$ with $|\rho_i| \le |\phi_i| + |\upsilon_i| + |\delta| \le (U(R_i v_i) + U(v_i) + 1)\epsilon + O(\epsilon^2)$. □

It is worth noting that the upper bounds on the relative errors derived for v_i and R_i appear to be large even for moderate values of n. However, in our experiments, we did not observe instability in their implementations, even when considering $n = 50$. Therefore, we believe that there is room for improvement in these bounds.

2.2 Horner schemes

We continue our analysis by studying the error propagation that occurs in the algorithms that evaluate a rational Bézier curve P at t through a Horner scheme, which is the case of the implementations of the two formulas in (7) and (8). In these specific contexts, we can use an already known theorem by Fuda et al. [16] that gives an upper bound for any function that is expressed in the form

$$r(x) = \frac{\sum_{k=0}^{N} a_k(x) f_k}{\sum_{j=0}^{M} b_j(x)} \tag{31}$$

for some data values f_k and functions a_k and b_j, $k = 0, \dots, N$ and $j = 0, \dots, M$.

Theorem 2.7. *Suppose that there exist $\alpha_0, \dots, \alpha_N \in \mathbb{R}$ with*

$$\mathrm{fl}(a_k(x)) = a_k(x)(1+\alpha_k), \qquad |\alpha_k| \le A\epsilon + O(\epsilon^2), \qquad k = 0, \dots, N \tag{32}$$

and $\beta_0, \dots, \beta_M \in \mathbb{R}$ with

$$\mathrm{fl}(b_j(x)) = b_j(x)(1+\beta_j), \qquad |\beta_j| \le B\epsilon + O(\epsilon^2), \qquad j = 0, \dots, M \tag{33}$$

for some constants A and B. Then, assuming that the data f_i are given as floating-point numbers, the relative forward error of r in (31) satisfies

$$\frac{|\mathrm{fl}(r(x)) - r(x)|}{|r(x)|} \le (N+2+A)\alpha(x)\epsilon + (M+B)\beta(x)\epsilon + O(\epsilon^2),$$

where

$$\alpha(x) = \frac{\sum_{k=0}^{N} |a_k(x) f_k|}{\left|\sum_{k=0}^{N} a_k(x) f_k\right|} \qquad \text{and} \qquad \beta(x) = \frac{\sum_{j=0}^{M} |b_j(x)|}{\left|\sum_{j=0}^{M} b_j(x)\right|},$$

for ϵ small enough.

We can use this result for both formulas in (7) and (8), as they fit the expression in (31) for $N = M = n$, $f_k = P_k$, and $a_k = b_k = x^{n-k}\binom{n}{k} w_k$ or $a_k = b_k = t^k s^{n-k}\binom{n}{k} w_k$, $k = 0, \dots, n$, respectively. Moreover, assuming that the binomial coefficients are implemented without introducing any floating-point relative error via integer arithmetic, the computations of the a_k, $k = 0, \dots, n$, involve two products plus at most n subtractions, n divisions, and $n-1$ products for x^{n-k} in (7) and two products plus at most n subtractions and $n-1$ products in case of $t^k s^{n-k}$ in (8). This implies that the constants in (32) and (33) are $A = B = 3n+1$ in case of formula in (7) and $A = B = 2n+1$ in case of (8). Therefore, it follows from Theorem 2.7 that the relative error E in (20) for $P(t)$ computed with (7) satisfies

$$E(t) \leq \frac{\sum_{k=0}^{n} |B_k^n(t) w_k P_k|}{\left|\sum_{k=0}^{n} B_k^n(t) w_k P_k\right|} (4n+3)\epsilon + (4n+1)\epsilon + O(\epsilon^2), \tag{34}$$

while, if computed with (8),

$$E(t) \leq \frac{\sum_{k=0}^{n} |B_k^n(t) w_k P_k|}{\left|\sum_{k=0}^{n} B_k^n(t) w_k P_k\right|} (3n+3)\epsilon + (3n+1)\epsilon + O(\epsilon^2).$$

Notably, the difference $1-t$ cannot be problematic, because we assume that t is an exact floating-point number. However, if instead t is the floating-point approximation of a real number, then the formula in (8) may become unstable when t approaches 1. Conversely, the formula in (7) represents a stable way to evaluate P thanks to the distinction of the two cases in the definition of x.

2.3 Geometric approach

We proceed to analyse the error propagation of the recursive algorithm given by the formulas in (9). In particular, we first study how the error propagates during the computation of the values h_i, $i = 0, \dots, n$, and then we examine the relative errors of the values T_i, $i = 0, \dots, n$. This analysis finally leads to an upper bound on the relative error E in (20) in the case of $P(t) = T_n$.

Lemma 2.8. *For any $t, w_0, \dots, w_n \in \mathbb{F}$, there exist $\eta_0, \dots, \eta_n \in \mathbb{R}$, such that the h_i in (9) satisfy $\mathrm{fl}(h_i) = h_i(1+\eta_i)$, $i = 0, \dots, n$, with $|\eta_i| \leq U(h_i)\epsilon + O(\epsilon^2)$ and*

$$U(h_i) = 2^3(2^i - 1).$$

Proof. We first notice that

$$\begin{aligned}\mathrm{fl}(h_i) &= \frac{w_i h_{i-1}(1+\eta_{i-1}) t(n-i+1)(1+\delta_1)}{w_{i-1} i(1-t)(1+\delta_2) + w_i h_{i-1}(1+\eta_{i-1}) t(n-i+1)(1+\delta_3)} \\ &= \frac{w_i h_{i-1} t(n-i+1)(1+\delta_1+\eta_{i-1}+O(\epsilon^2))}{w_{i-1} i(1-t)(1+\delta_2) + w_i h_{i-1} t(n-i+1)(1+\delta_3+\eta_{i-1}+O(\epsilon^2))},\end{aligned}$$

where δ_1 is the error introduced by the floating-point operations in the numerator, that are three products and one division, and δ_2 and δ_3 are those related to the first and second addends in the denominator, respectively, that are two products, one subtraction, and one sum for the former and three products and one sum for the latter. Therefore, it follows from (22) that $|\delta_1|, |\delta_2|, |\delta_3| \leq 4\epsilon + O(\epsilon^2)$. Moreover, the intermediate value theorem further guarantees that

$$\mathrm{fl}(h_i) = \frac{w_i h_{i-1} t(n-i+1)(1+\delta_1+\eta_{i-1}+O(\epsilon^2))}{(w_{i-1} i(1-t) + w_i h_{i-1} t(n-i+1))(1+\delta_{i-1})},$$

for some $\delta_{i-1} \in \left[\min(\delta_2, \delta_3+\eta_{i-1}+O(\epsilon^2)), \max(\delta_2, \delta_3+\eta_{i-1}+O(\epsilon^2))\right] = \left[\delta_2, \delta_3+\eta_{i-1}+O(\epsilon^2)\right]$, and the Taylor expansion of $1/(1+\delta_{i-1})$ gives

$$\mathrm{fl}(h_i) = \frac{w_i h_{i-1} t(n-i+1)}{w_{i-1} i(1-t) + w_i h_{i-1} t(n-i+1)} (1+\delta_1+\eta_{i-1}-\delta_{i-1}+O(\epsilon^2)) = h_i(1+\delta_1+\eta_{i-1}-\delta_{i-1}+O(\epsilon^2)).$$

We define $\eta_i = \delta_1 + \eta_{i-1} - \delta_{i-1} + O(\epsilon^2)$, hence, by using the triangle inequality and the upper bounds on the relative errors introduced by the operations, we have

$$|\eta_i| \leq |\delta_1| + |\eta_{i-1}| + |\delta_{i-1}| + O(\epsilon^2) \leq 8\epsilon + 2|\eta_{i-1}| + O(\epsilon^2), \qquad i = 1, \dots, n.$$

Now, we can prove the statement by induction over i. The base case follows by the fact that $h_0 = 1$, therefore $\eta_0 = 0$. Finally, the inductive step from $i-1$ to i follows immediately from the inductive hypothesis, that is $|\eta_{i-1}| \le 2^3(2^{i-1}-1)\epsilon + O(\epsilon^2)$. □

Proposition 2.9. *For any $t, w_0, \dots, w_n, P_0, \dots, P_n \in \mathbb{F}$ and $r \in \{1, \dots, n\}$, the relative errors of the T_i in (9) satisfy*

$$\frac{|\mathrm{fl}(T_i(t)) - T_i(t)|}{|T_i|} \le \frac{\sum_{k=0}^{i} B_k^n(t)|P_k w_k|}{\left|\sum_{k=0}^{i} B_k^n(t) P_k w_k\right|}\left(\max_{k=1,\dots,i} \frac{1}{1-h_k} 2^3 i(2^i-1) + 3i\right)\epsilon + O(\epsilon^2), \qquad i = 1, \dots, n.$$

Therefore, the relative error in (20) for $P(t) = T_n$ satisfies

$$E(t) \le \frac{\sum_{k=0}^{n} B_k^n(t)|P_k w_k|}{\left|\sum_{k=0}^{n} B_k^n(t) P_k w_k\right|}\left(\max_{k=1,\dots,n} \frac{1}{1-h_k} 2^3 n(2^n-1) + 3n\right)\epsilon + O(\epsilon^2).$$

Proof. Denoting by τ_i the relative errors introduced by the computation of T_i, $i = 0, \dots, n$, we first notice that

$$\begin{aligned}
\mathrm{fl}(T_i) &= (1-h_i(1+\eta_i))T_{i-1}(1+\tau_{i-1})(1+\delta_1) + h_i(1+\eta_i)P_i(1+\delta_2) \\
&= (1-h_i)T_{i-1}\left(1 - \frac{h_i\eta_i}{1-h_i} + \tau_{i-1} + \delta_1 + O(\epsilon^2)\right) + h_i P_i(1 + \eta_i + \delta_2 + O(\epsilon^2)) \\
&= T_i + (1-h_i)T_{i-1}\left(-\frac{h_i\eta_i}{1-h_i} + \tau_{i-1} + \delta_1 + O(\epsilon^2)\right) + h_i P_i(\eta_i + \delta_2 + O(\epsilon^2)),
\end{aligned}$$

where $|\eta_i| \le 2^3(2^i-1)\epsilon + O(\epsilon^2)$ by Lemma 2.8 and δ_1 and δ_2 are the errors introduced by the operations in the first and second addends, respectively, that are one product and one sum each, plus one subtraction for the first addend only. Therefore, it follows from (22) that $|\delta_1|, |\delta_2| \le 3\epsilon + O(\epsilon^2)$. Then, using the fact that $\mathrm{fl}(T_i) - T_i = T_i\tau_i$, the triangle inequality, and the upper bounds on the relative errors introduced by the values h_i and the operations, we obtain

$$\begin{aligned}
|T_i\tau_i| &\le (1-h_i)|T_{i-1}|\left(\frac{h_i|\eta_i|}{1-h_i} + |\delta_1|\right) + h_i|P_i|(|\eta_i| + |\delta_2|) + (1-h_i)|T_{i-1}\tau_{i-1}| + O(\epsilon^2) \\
&\le \left((1-h_i)|T_{i-1}| + h_i|P_i|\right)\left(\frac{1}{1-h_i} 2^3(2^i-1) + 3\right)\epsilon + (1-h_i)|T_{i-1}\tau_{i-1}| + O(\epsilon^2).
\end{aligned} \tag{35}$$

Recalling that $h_j = B_j^n w_j \big/ \sum_{k=0}^{j}(B_k^n w_k)$ [28], we can use the relation of T_i in (9) recursively to express

$$\begin{aligned}
(1-h_i)|T_{i-1}| &= \sum_{j=1}^{i}\prod_{k=0}^{i-j}(1-h_{i-k})h_{j-1}|P_{j-1}| = \sum_{j=1}^{i}\prod_{k=0}^{i-j}\left(1 - \frac{B_{i-k}^n w_{i-k}}{\sum_{l=0}^{i-k} B_l^n w_l}\right)\frac{B_{j-1}^n w_{j-1}}{\sum_{l=0}^{j-1} B_l^n w_l}|P_{j-1}| \\
&= \sum_{j=1}^{i}\prod_{k=0}^{i-j}\frac{\sum_{l=0}^{i-k-1} B_l^n w_l}{\sum_{l=0}^{i-k} B_l^n w_l}\frac{B_{j-1}^n w_{j-1}}{\sum_{l=0}^{j-1} B_l^n w_l}|P_{j-1}| = \frac{\sum_{j=1}^{i} B_{j-1}^n |w_{j-1}P_{j-1}|}{\sum_{l=0}^{i} B_l^n w_l},
\end{aligned}$$

and, by (35),

$$|T_i\tau_i| \le \frac{\sum_{k=0}^{i} B_k^n(t)|P_k w_k|}{\sum_{k=0}^{i} B_k^n(t) w_k}\left(\frac{1}{1-h_i} 2^3(2^i-1) + 3\right)\epsilon + (1-h_i)|T_{i-1}\tau_{i-1}| + O(\epsilon^2). \tag{36}$$

Now, we can prove the statement by induction over $i = 1, \dots, n$ and, to this end, we recall from [28] that $T_i = \sum_{k=0}^{i} B_k^n w_k P_k \big/ \sum_{k=0}^{i}(B_k^n w_k)$. The base case follows by the fact that $T_0 = P_0$, therefore $\tau_0 = 0$. Finally, the inductive step from $i-1$ to i follows from the inductive hypothesis, that is,

$$|T_{i-1}\tau_{i-1}| \le \frac{\sum_{k=0}^{i-1} B_k^n(t)|P_k w_k|}{\sum_{k=0}^{i-1} B_k^n(t) w_k}\left(\max_{k=1,\dots,i-1} \frac{1}{1-h_k} 2^3(i-1)(2^{i-1}-1) + 3\right)\epsilon + O(\epsilon^2),$$

together with (36) and

$$(1-h_i)\frac{\sum_{k=0}^{i-1} B_k^n(t)|P_k w_k|}{\sum_{k=0}^{i-1} B_k^n(t) w_k} = \frac{\sum_{k=0}^{i-1} B_k^n(t)|P_k w_k|}{\sum_{k=0}^{i} B_k^n(t) w_k} \le \frac{\sum_{k=0}^{i} B_k^n(t)|P_k w_k|}{\sum_{k=0}^{i} B_k^n(t) w_k}.$$

□

2.4 Barycentric approach

In this case, we observe that the barycentric form of P in (18) can be expressed as in (31) with $N=M=n$, $f_i=Q_i$, and $a_i=b_i=u_i/(t-t_i)$, $i=0,\dots,n$, therefore we can use once again Theorem 2.7. However, the latter assumes that the values f_i are floating-point numbers, while the Q_i are the result of a prepossessing step that leads to a set of perturbed initial data. Consequently, we derive an upper bound on the relative error E in (20) via Theorem 2.7, while also considering this difference.

Corollary 2.10. *Assuming that the values $Q_i=P(t_i)$, $i=0,\dots,n$, are computed by evaluating the rational Bézier curve P at t_i through the implementation of the VS formula in (7) and that the $z(t_i)$ are defined as in (19), then the relative error in (20) for $P(t)$ computed by implementing the barycentric formula in (18) satisfies*

$$E(t)\le\left(10n+5+\max_{j=0,\dots,n}U(Q_j)\right)\frac{\sum_{i=0}^n\left|\frac{u_iQ_i}{t-t_i}\right|}{\left|\sum_{i=0}^n\frac{u_iQ_i}{t-t_i}\right|}\epsilon+(10n+3)\frac{\sum_{i=0}^n\left|\frac{u_i}{t-t_i}\right|}{\left|\sum_{i=0}^n\frac{u_i}{t-t_i}\right|}\epsilon+O(\epsilon^2),$$

where

$$U(Q_i)=\frac{\sum_{i=0}^n|B_i^n(t_i)w_iP_i|}{\left|\sum_{i=0}^nB_i^n(t_i)w_iP_i\right|}(4n+3)+4n+1,\qquad i=0,\dots,n. \tag{37}$$

Proof. Since the values Q_i are computed with (7), we know from (34) that there exist $\theta_0,\dots,\theta_n\in\mathbb{R}$, such that they satisfy $\mathrm{fl}(Q_i)=Q_i(1+\theta_i)$, $i=0,\dots,n$, with $|\theta_i|\le U(Q_i)\epsilon+O(\epsilon^2)$ and $U(Q_i)$ in (37). Moreover, in the computation of the $z(t_i)$, we first introduce at most $4n+1$ floating-point relative errors in the VS algorithm to get the denominators $\sum_{i=0}^n x^{n-i}\binom{n}{i}w_i$, and then we perform at most other $2n-1$ products, which happens in the case of $(1-t)^n$. This means that there exist $\zeta_0,\dots,\zeta_n\in\mathbb{R}$, such that the $z(t_i)$ satisfy $\mathrm{fl}(z(t_i))=z(t_i)(1+\zeta_i)$, $i=0,\dots,n$, with $|\zeta_i|\le U(z(t_i))\epsilon+O(\epsilon^2)$ and

$$U(z(t_i))=6n.$$

Also, we know that the computation of $u_i/z(t_i)=\prod_{k\neq i}\frac{1}{t_i-t_k}$ introduces at most $3n$ floating-point operations [16, Lemma 1], which, together with the previous equation, leads to the existence of $\mu_0,\dots,\mu_n\in\mathbb{R}$, such that the u_i satisfy $\mathrm{fl}(u_i)=u_i(1+\mu_i)$, $i=0,\dots,n$, with $|\mu_i|\le U(u_i)\epsilon+O(\epsilon^2)$ and

$$U(u_i)=U(z(t_i))+3n+1=9n+1.$$

Finally, the statement follows by a corollary of Theorem 2.7 [16, Corollary 1], which, by also considering the initial errors in the data Q_i, gives

$$E(t)\le\left(n+4+\max_{i=0,\dots,n}U(u_i)+\max_{j=0,\dots,n}U(Q_j)\right)\frac{\sum_{i=0}^n\left|\frac{u_iQ_i}{t-t_i}\right|}{\left|\sum_{i=0}^n\frac{u_iQ_i}{t-t_i}\right|}\epsilon+\left(n+2+\max_{i=0,\dots,n}U(u_i)\right)\frac{\sum_{i=0}^n\left|\frac{u_i}{t-t_i}\right|}{\left|\sum_{i=0}^n\frac{u_i}{t-t_i}\right|}\epsilon+O(\epsilon^2).$$

□

2.5 Summary

By defining the *conditioning functions*

$$\kappa_P(t)=\frac{\sum_{k=0}^n B_k^n(t)|P_kw_k|}{\left|\sum_{k=0}^n B_k^n(t)P_kw_k\right|},\qquad \kappa_R(t)=\frac{\sum_{k=0}^n A_k^n(t)|R_kv_k|}{\left|\sum_{k=0}^n A_k^n(t)R_kv_k\right|},\qquad\text{and}\qquad \kappa_Q(t)=\frac{\sum_{i=0}^n\left|\frac{u_iQ_i}{t-t_i}\right|}{\left|\sum_{i=0}^n\frac{u_iQ_i}{t-t_i}\right|}, \tag{38}$$

and recalling that

$$\Lambda_n(t)=\frac{\sum_{i=0}^n\left|\frac{u_i}{t-t_i}\right|}{\left|\sum_{i=0}^n\frac{u_i}{t-t_i}\right|} \tag{39}$$

is already known as the *Lebesgue function*, we proved that the relative error E in (20) can be bounded as

$$E(t)\leq\begin{cases}\kappa_P(t)(3n^2+5n)\epsilon+O(\epsilon^2), & P(t)=P_0^n \text{ in (6)},\\ \kappa_P(t)(4n+3)\epsilon+(4n+1)\epsilon+O(\epsilon^2), & P(t) \text{ in (7)},\\ \kappa_P(t)(3n+3)\epsilon+(3n+1)\epsilon+O(\epsilon^2), & P(t) \text{ in (8)},\\ \kappa_P(t)\left(\max\limits_{k=1,\dots,n}\dfrac{1}{1-h_k}2^3n(2^n-1)+3n\right)\epsilon+O(\epsilon^2), & P(t)=P_0^n \text{ in (9)},\\ \kappa_R(t)\left(3n^2+5n+\max\limits_{j=0,\dots,n}U(v_j)+\max\limits_{k=0,\dots,n}U(R_k)\right)\epsilon+O(\epsilon^2), & P(t)=R_0^n \text{ in (12)–(14)},\\ \kappa_Q(t)\left(10n+5+\max\limits_{j=0,\dots,n}U(Q_j)\right)\epsilon+\Lambda_n(t)(10n+3)\epsilon+O(\epsilon^2), & P(t) \text{ in (18)}.\end{cases}$$

Hence, we expect that all methods that use the Bernstein basis in (2), namely those defined by the formulas in (6)–(8) and (9), behave similarly in terms of numerical stability. The only exception might arise with the latter method if any of the h_k, $k=1,\dots,n$, is very close to 1. However, Woźny and Chudy [28] have already addressed this issue by suggesting to use the relation

$$1-h_k=\frac{h_k}{h_{k-1}}\frac{w_{k-1}k(1-t)}{w_k t(n-k+1)}.$$

Regarding instead the methods that employ a different basis, such as the Wang–Ball and the barycentric algorithms, even under the assumption that all the preprocessing steps are stable, there are scenarios where κ_R or κ_Q are bigger than κ_P, or vice versa. As a consequence, these algorithms may exhibit instability even when the formulas in (6)–(9) are stable. However, for the barycentric form, instability is less likely to occur if Chebyshev nodes are chosen. In fact, multiplying both numerator and denominator of the function κ_Q by $\left|\sum_{i=0}^n u_iQ_i/(t-t_i)\right|$, we can see that

$$\kappa_Q(t)=\frac{\sum_{i=0}^n\left|\frac{u_iQ_i}{t-t_i}\right|}{\left|\sum_{i=0}^n\frac{u_i}{t-t_i}\right|}\frac{1}{|P(t)|}\leq\max_{i=0,\dots,n}|Q_i|\Lambda_n(t)\frac{1}{|P(t)|}\leq\max_{i=0,\dots,n}|P_i|\Lambda_n(t)\kappa_P(t)\frac{\sum_{k=0}^n B_k^n(t)|w_k|}{\sum_{k=0}^n B_k^n(t)|P_k\,w_k|}.$$

In particular, if $\min_{i=0,\dots,n}|P_i|\neq 0$, then

$$\kappa_Q(t)\leq\kappa_P(t)\Lambda_n(t)\frac{\max_{i=0,\dots,n}|P_i|}{\min_{i=0,\dots,n}|P_i|}. \tag{40}$$

Moreover, it is well known [24] that the Lebesgue function grows only logarithmically in n for Chebyshev nodes. Therefore, if κ_P has a good behaviour, then we can expect the method to be always stable when the ratio between the biggest and the smallest control points is small. On the contrary, the Lebesgue function related to equidistant nodes exhibits exponential growth in n [24], hence we can have unstable results even for moderate values of n with uniformly distributed nodes. We will show that these scenarios can indeed occur in Section 4 through numerical experiments.

3 Efficiency analysis

Rational de Casteljau (RDC) / Farin–de Casteljau (FDC). We recall that evaluating the numerator and the denominator of a rational Bézier curve as in (5) and then dividing the results is equivalent to evaluating the spatial curve (3) and applying the central projection on the final result. To that, for the RDC, we need to precompute the points $\hat{P}_i$ as in Algorithm 1, and evaluate $\hat{P}(t)$ as in Algorithm 2. The FDC algorithm is a more robust alternative of the RDC algorithm described in (6) and implemented optimally in Algorithm 3.

Rational VS (RVS) / Rational Horner–Bézier (RHB). In order to optimise the algorithms and to compute them in linear time, we precompute the factors $\binom{n}{k}w_kP_k$ and $\binom{n}{k}w_k$ as in Algorithm 4, and then we evaluate P as in Algorithm 5 and Algorithm 6, respectively.

Linear time geometric (LTG). Although it is not displayed in (9), for numerical reasons, the authors deemed necessary to distinguish the cases $t\in[0,0.5]$ and $t\in(0.5,1]$ as in Algorithm 7.

Table 1: Comparison between the number of floating–point operations for the preprocessing (top) and the main algorithm (bottom) for each method.

method	preprocessing
RDC	$d(n+1)$
FDC	0
RVS	$(d+1)(n-1)+2d$
RHB	$(d+1)(n-1)+2d$
LTG	0
CHE	$(n+1)(2dn+d+2n+8+2\log_2(n)+2h)+(d+1)(n-1)+2d$
UNI	$(n+1)(2dn+d+2n+7+2\log_2(n))+(d+1)(n-1)+2d$
RWB (n even)	$2dn^2+4dn-2d+\frac{1}{2}n$
RWB (n odd)	$2dn^2+4dn-2d+\frac{1}{2}n-\frac{1}{2}$
RBF	$O(dn\log n)$

method	add/sub	mult	div	total
RDC	$\frac{1}{2}dn(n+1)+1$	$dn(n+1)$	d	$\frac{3}{2}dn^2+\frac{3}{2}dn+d+1$
FDC	$\frac{1}{2}(d+2)n(n+1)+1$	$\frac{1}{2}(2d+3)n(n+1)$	0	$\frac{3}{2}dn^2+\frac{3}{2}dn+\frac{5}{2}n^2+\frac{5}{2}n+1$
RVS	$(d+1)n+1$	$(d+1)n$	$d+1$	$2dn+d+2n+2$
RHB	$(d+1)(n-1)+d+2$	$2dn+3n$	d	$3dn+d+4n+1$
LTG	$(d+2)n+1$	$2(d+2)n$	$n+1$	$3dn+7n+2$
CHE	$(d+2)(n+1)$	$d(n+1)$	$n+1+d$	$2dn+3d+3n+3$
UNI	$(d+2)(n+1)$	$d(n+1)$	$n+1+d$	$2dn+3d+3n+3$
RWB (n even)	$\frac{3}{2}n(d+1)+1$	$\frac{3}{2}n(2d+2)$	$\frac{3}{2}n$	$\frac{9}{2}dn+6n+1$
RWB (n odd)	$\frac{1}{2}(3n-1)(d+1)+1$	$\frac{1}{2}(3n-1)(2d+2)$	$\frac{1}{2}(3n-1)$	$\frac{9}{2}dn-\frac{3}{2}d+6n-1$
RBF (n even)	$\frac{n}{2}(d+4\log_2 n+2)+d+1$	$\frac{n}{2}(2d+8\log_2 n+2)+d$	d	$6n\log_2(n)+2dn+3d+2n+1$
RBF (n odd)	$\frac{n-1}{2}(d+4\log_2 n+2)+2d+2$	$\frac{n-1}{2}(2d+6\log_2 n+2)+2d+2\log_2 n+1$	d	$8n\log_2(n)+2dn+\frac{7}{2}d+2n-4\log_2 n+1$

Barycentric algorithm with Chebyshev nodes (CHE) / with uniform nodes (UNI). To get the data of the barycentric form, we can use any of the previously cited algorithms. We choose to adapt the RVS algorithm as in Algorithm 15 to get t_i, Q_i, and $z(t_i)$) simultaneously. We recall that we can get $z(t_i)$ as in (19). The data of the barycentric form, such as the interpolation points, the weights, and the nodes, are precomputed in Algorithm 16. We present two ways of evaluating the barycentric form. Algorithm 17 evaluates the barycentric form in (18) in the classical way. However, since the distributions of the nodes are symmetric, we can compute $P(t)$ and $P(1-t)$ at the same time. For instance, if we want to get the values of $P(t)$ for $t=k/M$, $k=0,\dots,M$, then the number of flops by using Algorithm 18 is $M(n+1)/2$ less than using Algorithm 17.

Rational Wang–Ball (RWB). The weights and the control points of (10) are precomputed in Algorithm 9. Despite its recursive appearance in (13)–(14), the evaluation of a rational Wang–Ball curve is done in linear time in Algorithm 10.

Rational Bernstein–Fourier (RBF). The algorithm for evaluating $\hat{P}$ is presented in Algorithm 13. However, for a large number of evaluations, we can also use Algorithm 14 to compute $\hat{P}(t)$ and $\hat{P}(1-t)$ in parallel in order to have an optimal runtime. In the algorithms, we assume that the computation of x^n, $x\in\mathbb{R}$, in (19) is done with a logarithmic algorithm in the worst case, that is, it involves $2\log_2(n)$ multiplications. The computation of z^n, $z\in\mathbb{C}$, in (17) can be done using de Moivre's formula $z^n=r^n(\cos(n\theta)+i\sin(n\theta)$, where $\theta=\arg z$ and $r=|z|$. However, since a complex multiplication involves 6 real operations, here we assume that it is $12\log_2(n)$.

We compare the number of floating-point operations in the implementation of each method in Table 1. We denote by d the dimension of the space, n the degree of the curve, and let h be the cost of evaluating a trigonometric function $(\cos,\sin,\tan)$.

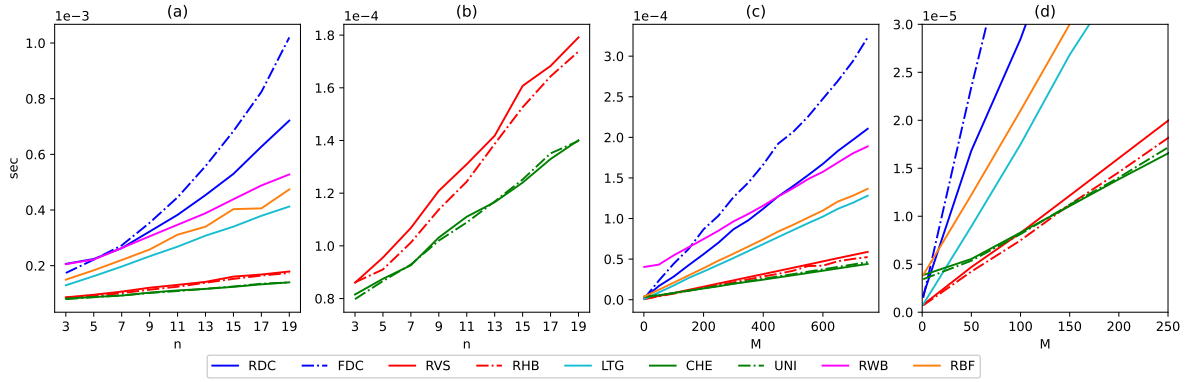

Figure 1: Runtime of all algorithms for computing a rational Bézier curve with $P_i = 100i\binom{1}{1}+\binom{1}{1}$ and $w_i = i \mod 2+1$. We first consider $M = 2500$ evaluation points for $n = 3,5,7,\ldots,19$ (a) and provide a zoom-in view on the fastest methods (b). We then fix the degree at $n = 20$ and vary the number of evaluation points $M = 1,50,100,150,\ldots,750$ (c), with a zoom-in view on the domain $[1,250]$ (d).

4 Numerical experiments

We implemented all the methods in `C++` and computed the exact value $P(t)$ of the Bézier curve in multiple-precision (1024 bit) floating-point arithmetic with the *MPFR* library [15]. Moreover, we used the *Eigen* module [18] to compute the Inverse Fast Fourier Transform in the Bernstein–Fourier algorithm. The results are obtained using a Ubuntu system on a Dell computer with 8 cores i7-10510U CPU 1.80GHz and 16 GiB of RAM. The codes are compiled with *CMake* compiler optimisation flag `-O3`.

4.1 Efficiency comparison

To compare the efficiency of the different algorithms, we run a first experiment with respect to the degree n of a rational Bézier curve. This is important because, although cubic curves are more familiar and commonly used, higher degree curves are particularly interesting for achieving more precision and smoothness, especially for complex shapes. Therefore, we evaluate rational Bézier curves of degree $n = 3,5,7,\ldots,19$ with control points defined as $P_i = 100i\binom{1}{1}+\binom{1}{1}$ and weights $w_i = i \mod 2+1$, $i = 0,\ldots,n$, at $M = 2500$ equidistant evaluation points in $[0,1]$. The results are obtained from averaging the result of 1000 reruns. Figure 1 displays the runtime of all algorithms (a), clearly showing that the RVS, RHB, UNI, and CHE algorithms outperform all the others[4]. It is also evident (b) that the algorithms that use the barycentric form are faster than those employing the Horner-scheme evaluation, with the difference becoming more significant as n increases. This loss of efficiency in the RVS and RHB algorithms is due to the computation of large binomial coefficients with integer arithmetic, which becomes computationally expensive for big values of n.

In the second experiment, we consider the same setup as before, but perform the comparison with respect to the number of evaluation points M. Specifically, we keep the control points and weights consistent with the previous experiment and we set $n = 20$ and $M = 1,50,100,150,\ldots,750$. The results are shown in Figure 1 (c) and reconfirm that RVS, RHB, UNI, and CHE are the fastest. However, looking closer in the zoom-in plot (d), we notice that the RVS and RHB algorithms win over the UNI and CHE algorithms only if $M < 200$, otherwise the situation is reversed. This effect arises from the quadratic time preprocessing step required by the barycentric algorithms. Although the latter is a one-time operation, it is relevant only for few evaluations, while it becomes negligible as M grows.

4.2 Numerical stability comparison

To compare the numerical stability of all the algorithms, we evaluate the relative error E in (20) for 1000 equidistant evaluation points in $[0,1]$ using the various implementations of $\mathrm{fl}(P(t))$ in double precision. If the results are on the order of the machine epsilon, approximately 10^{-16}, then we can conclude that the method is stable, otherwise it suggests instability.

[4] Bezerra [2] shows that the RBF algorithm is faster than the RVS for $n < 8$, but this does not happen in our experiment, possibly due to a different implementation technique.

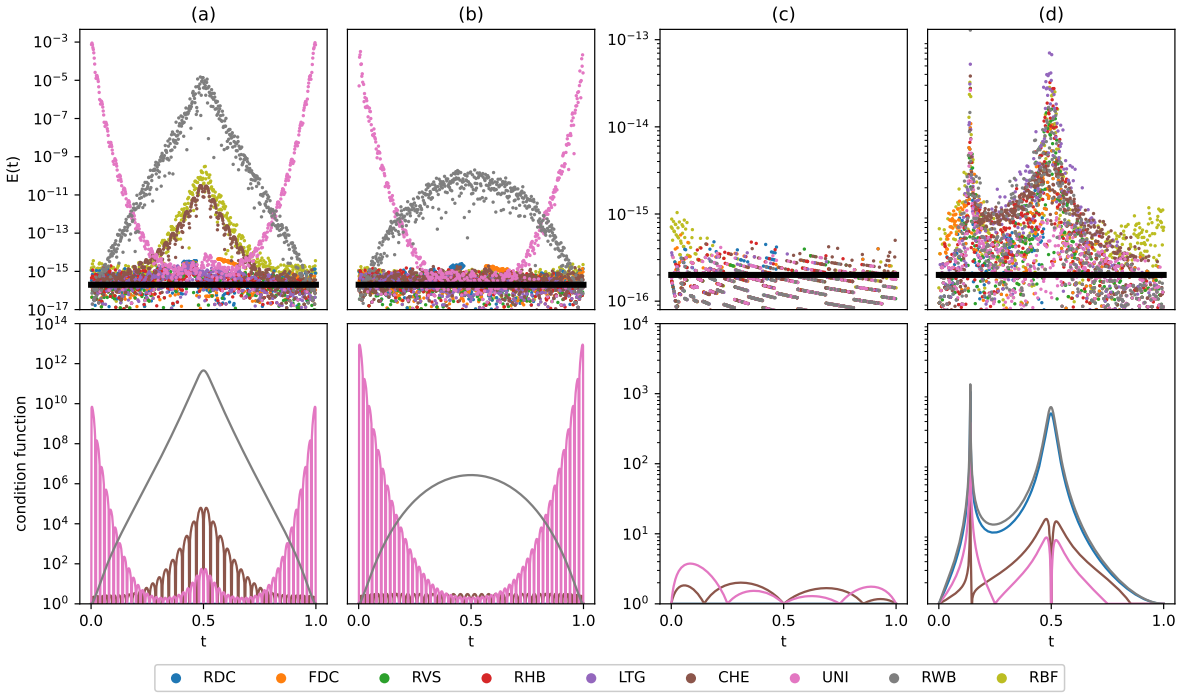

Figure 2: Relative errors of all algorithms (top) for computing a rational Bézier curve and their related conditioning function (bottom) on a logarithmic scale. We first consider $n = 50$, $P_i = \binom{x_i}{y_i}$ for x_i and y_i in (41), and $w_i = i \bmod 2 + 1$, and we observe the results related to the x-coordinate (a) and y-coordinate (b). We then set $n = 4$, $P_0 = \binom{10}{-100}$, $P_1 = \binom{20}{200}$, $P_2 = \binom{30}{-200}$, $P_3 = \binom{40}{101}$, $P_4 = \binom{50}{101}$ and $w_i = 1$, $i = 0, \ldots, n$, and show the results for the x-coordinate (c) and y-coordinate (d). The black line represents the machine epsilon in double precision.

In the first experiment, we consider a rational Bézier curve of degree $n = 50$ with control points $P_i = \binom{x_i}{y_i}$, $i = 0, \ldots, n$, for

$$x_i = \begin{cases} 1, & i = 0, \ldots, 9 \text{ and } i = 41, \ldots, n, \\ 10^6, & i = 10, \ldots, 40 \end{cases} \qquad \text{and} \qquad y_i = \sin \frac{i\pi}{n+1}, \tag{41}$$

and weights $w_i = i \bmod 2 + 1$. In Figure 2 (a,b), we observe that all the methods defined via the Bernstein basis are stable, while the others exhibit numerical problems. In particular, the CHE, UNI, RWB, and RBF algorithms are unstable with respect to the x-coordinate (a), as well as for the y- coordinate (b), except for the CHE algorithm. Although we cannot determine the cause of instability in the RBF algorithm, our theoretical results in Section 2 can explain the other cases. Indeed, we proved that the relative error of the RWB algorithm depends on the conditioning functions κ_R, while those of CHE and UNI on κ_Q. In this case, even though the initial data give a good conditioning function related to the Bernstein basis, that is, $\kappa_P(t) = 1$, the conversion to a different basis leads to unfavorable behaviour for both κ_R and κ_Q, as shown in Figure 2 (bottom). Moreover, it is worth noting that, as expected, the CHE algorithm exhibits instability with respect to the x-coordinate because the ratio between the biggest and the smallest $|x_i|$ is 10^6. However, under circumstances where this ratio is not big, the CHE algorithm is typically stable, even for large values of n.

Finally, we want to examine a more realistic experiment, thus we take a low degree curve by setting $n = 4$. On the one hand, we observe in Figure 2 that the relative error related to the x-coordinate (c) is perfectly stable, with all the conditioning functions small. On the other hand, all the relative errors related to the y-coordinate (d) exhibit spikes in some parts of the domain, reaching an order of 10^{-13}. This behaviour is also reflected in the conditioning functions. However, where these spikes occur, the values of $P_y(t)$ are very small because the curve is crossing the t-axis, therefore this may not be a stability issue, but rather a consequence of dividing by very small values in $E_y(t)$ in (20). However, for our 1000 equidistant evaluation points, the minimum absolute value of $P_y(t)$ is 0.13424 in the domain of the first spike and 0.3116 for the second, thus indicating that we are not so close to the values where $P_y(t) = 0$. Furthermore, plotting the absolute errors leads to a similar behaviour without these spikes, but still with magnitudes between 10^{-14}

and 10^{-13} for all methods except the barycentric form with uniform nodes. This latter remains stable, as its conditioning function κ_Q is small, apart from the initial spike, and its nodes are far from the instability regions of the RVS algorithm. In contrast, Chebyshev nodes compromise the stability of the method due to the computation of one interpolation point with the RVS algorithm where it is unstable, specifically for the node in $[0.1, 0.2]$. Furthermore, while both uniform and Chebyshev nodes include $t = 0.5$, the RVS is exact at this point with a relative error of 0, thus preserving the stability of the barycentric method with uniform nodes. However, the computation of this node with the Chebyshev formula is not exact, resulting in perturbed data.

5 Conclusion

We conducted a comprehensive comparison of the most common algorithms used to evaluate a rational Bézier curve in terms of both efficiency and numerical stability. Our analysis and numerical experiments reveal that the fastest algorithms are those employing a Horner-like scheme for the evaluation and those defined in barycentric form. Specifically, while the former is advantageous for scenarios that require the evaluation of the curve at few evaluation points, not exceeding 200, the barycentric form becomes the preferred choice when dealing with a larger number of evaluation points. This is because the preprocessing step required by the barycentric algorithms is executed only once, thus its runtime becomes negligible for a significant number of evaluations.

Regarding the numerical stability, we derived an upper bound on the relative error of the different methods and showed, both theoretically and empirically, that it depends on certain conditioning functions. Specifically, algorithms that use the Bernstein basis depend on the same conditioning function, therefore they have consistent numerical behaviours. Instead, conversion to another basis can lead to different relative errors. In fact, there are scenarios where all algorithms are stable, except from those given by the Wang–Ball and the barycentric basis. However, we proved that, if the Bernstein basis gives a good conditioning function, then also the basis related to the barycentric algorithm with Chebyshev nodes behaves well, as long as the ratio between the largest and smallest control points P_i is small. Lastly, even classical algorithms based on the Bernstein basis may fail if the associated conditioning function is large, particularly when control points have different signs. In such cases, it is possible that the conversion to the barycentric form with nodes located away from instability areas can yield better results.

Acknowledgements

This work was supported by the Swiss National Science Foundation (SNSF) under project No. 188577 and the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska–Curie grant agreement No. 860843.

References

[1] L. H. Bezerra. Vandermonde factorizations of a regular Hankel matrix and their application to the computation of Bézier curves. *SIAM Journal on Matrix Analysis and Applications*, 33(2):411–432, 2012.

[2] L. H. Bezerra. Efficient computation of Bézier curves from their Bernstein–Fourier representation. *Applied Mathematics and Computation*, 220:235–238, 2013.

[3] L. H. Bezerra and L. K. Sacht. On computing Bézier curves by Pascal matrix methods. *Applied Mathematics and Computation*, 217(24):10118–10128, Aug. 2011.

[4] P. Bézier. Définition numérique des courbes et surfaces (I). *Automatisme*, 11(4):625–632, 1966.

[5] P. Bézier. Définition numérique des courbes et surfaces (II). *Automatisme*, 12:17–21, 1967.

[6] W. Boehm and A. Müller. On de Casteljau's algorithm. *Computer Aided Geometric Design*, 16(7):587–605, Aug. 1999.

[7] P. de Casteljau. Outillages méthodes calcul. Technical report, André Citroën Automobile SA, 1959.

[8] N. Dejdumrong. Rational DP-Ball curves. In *International Conference on Computer Graphics, Imaging and Visualisation*, CGIV'06, pages 478–483, Sydney, July 2006. IEEE.

[9] N. Dejdumrong. Efficient algorithms for non-rational and rational Bézier curves. In *International Conference on Computer Graphics, Imaging and Visualisation*, CGIV'08, pages 109–114, Penang, Aug. 2008. IEEE.

[10] N. Dejdumrong, H. N. Phien, H. L. Tien, and K. M. Lay. Rational Wang–Ball curves. *International Journal of Mathematical Education in Science and Technology*, 32(4):565–584, 2001.

[11] J. Delgado and J. M. Peña. A shape preserving representation with an evaluation algorithm of linear complexity. *Computer Aided Geometric Design*, 20(1):1–10, Mar. 2003.

[12] J. Delgado and J. M. Peña. A shape preserving representation for rational curves with efficient evaluation algorithm. In M. Sarfraz, editor, *Advances in Geometric Modeling*, chapter 3, pages 39–54. John Wiley & Sons, Chichester, 2004. ISBN 978-0-470-859377.

[13] G. Farin. Algorithms for rational Bézier curves. *Computer-Aided Design*, 15(2):73–77, Mar. 1983.

[14] G. Farin. *Curves and Surfaces for CAGD: A Practical Guide*. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. Morgan Kaufmann, San Francisco, 5th edition, 2001.

[15] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions of Mathematical Software*, 33(2):Article 13, 15 pages, June 2007.

[16] C. Fuda, R. Campagna, and K. Hormann. On the numerical stability of linear barycentric rational interpolation. *Numerische Mathematik*, 152(4):761–786, Dec. 2022. [PDF]

[17] R. Goldman. *Pyramid Algorithms*. The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann, San Francisco, 2003. ISBN 978-1-55860-354-7.

[18] G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010. [Online; accessed 9-May-2024].

[19] S.-M. Hu, G.-Z. Wang, and T.-G. Jin. Properties of two types of generalized Ball curves. *Computer-Aided Design*, 28(2):125–133, Feb. 1996.

[20] H. N. Phien and N. Dejdumrong. Efficient algorithms for Bézier curves. *Computer Aided Geometric Design*, 17(3):247–250, Mar. 2000.

[21] A. Ramanantoanina and K. Hormann. New shape control tools for rational Bézier curve design. *Computer Aided Geometric Design*, 88:Article 102003, 11 pages, June 2021. [PDF]

[22] H. E. Salzer. Lagrangian interpolation at the Chebyshev points $x_{n,\nu} \equiv \cos(\nu\pi/n)$, $\nu = 0(1)n$; some unnoted advantages. *The Computer Journal*, 15(2):156–159, May 1972.

[23] L. L. Schumaker and W. Volk. Efficient evaluation of multivariate polynomials. *Computer Aided Geometric Design*, 3(2):149–154, Aug. 1986.

[24] S. J. Smith. Lebesgue constants in polynomial interpolation. *Annales Mathematicae et Informaticae*, 33:109–123, 2006.

[25] L. N. Trefethen and D. Bau. *Numerical Linear Algebra*. SIAM, Philadelphia, 1997. ISBN 978-0-89871-361-9.

[26] G. Wang. Ball curve of high degree and its geometric properties. *Applied Mathematics: A Journal of Chinese Universities*, 2(1):126–140, 1987.

[27] J. Warren. An efficient algorithm for evaluating polynomials in the Pòlya basis. In H. Hagen, G. Farin, and H. Noltemeier, editors, *Geometric Modelling*, volume 10 of *Computing Supplement*, pages 357–361. Springer, Vienna, 1995.

[28] P. Woźny and F. Chudy. Linear-time geometric algorithm for evaluating Bézier curves. *Computer-Aided Design*, 118:Article 102760, 6 pages, Jan. 2020.

A Appendix: Algorithms

A.1 de Casteljau algorithm

Algorithm 1 ToHomogeneous $(P_0, \dots, P_n, w_0, \dots, w_n)$

for $k \leftarrow 0(1)n$ **do**
  $\hat{P}_k \leftarrow (w_k P_k, w_k)$
return $\hat{P}_0, \dots, \hat{P}_n$

Algorithm 2 DeCasteljau $(P_0, \dots, P_n, t)$

for $k \leftarrow 0(1)n$ **do**
  $\hat{P}_k \leftarrow P_k$
$t_1 \leftarrow 1 - t$
for $r \leftarrow 1(1)n$ **do**
  for $k \leftarrow 0(1)(n-r)$ **do**
    $\hat{P}_k \leftarrow \hat{P}_k t_1 + t \hat{P}_{k+1}$.
$P \leftarrow \text{proj}(\hat{P}_0)$ ▷ central projection
return P

Algorithm 3 RationalDeCasteljau $(P_0,\ldots,P_n,w_0,\ldots,w_n,t)$

$t_1 \leftarrow 1-t$
for $r \leftarrow 1(1)n$ **do**
  for $k \leftarrow 0(1)(n-r)$ **do**
    $u \leftarrow t_1 w_k$
    $v \leftarrow t\, w_{k+1}$
    $w_k \leftarrow u+v$
    $c_1 \leftarrow u/w_k$
    $c_2 \leftarrow 1-c_1$
    $P_k \leftarrow P_k c_1 + c_2 P_{k+1}$.
return P_0

A.2 VS algorithm

Algorithm 4 PreprocessingOfVSandHornerBezier $(P_0,\ldots,P_n,w_0,\ldots,w_n)$

$b \leftarrow 1$
$P_0 \leftarrow w_0 P_0$
$P_n \leftarrow w_n P_n$
for $k = 1(1)(n-1)$ **do**
  $b \leftarrow b(n+1-k)$
  $b \leftarrow b/k$
  $w_k \leftarrow b\, w_k$
  $P_k \leftarrow w_k P_k$
return $(P_0,\ldots,P_n,w_0,\ldots,w_n)$

Algorithm 5 VS $(P_0,\ldots,P_n,w_0,\ldots,w_n,t)$

if $t \le 1/2$ **then**
  $s \leftarrow t/(1-t)$
  $d \leftarrow w_n$
  $N \leftarrow P_n$
  for $k = 1(1)n$ **do**
    $n_k \leftarrow n-k$
    $N \leftarrow N s + P_{n_k}$
    $d \leftarrow d s + w_{n_k}$
else if $t > 1/2$ **then**
  $s \leftarrow (1-t)/t$
  $d \leftarrow w_0$
  $N \leftarrow P_0$
  for $k = 1(1)n$ **do**
    $N \leftarrow N s + P_k$
    $d \leftarrow d s + w_k$
return N/d

A.3 Horner Bézier algorithm

Algorithm 6 HornerBezier $(P_0,\ldots,P_n,w_0,\ldots,w_n,t)$

$s \leftarrow 1-t$
$t_k \leftarrow 1$
$d \leftarrow s\,w_0$
$N \leftarrow s\,P_0$
for $k \leftarrow 1(1)(n-1)$ **do**
  $t_k \leftarrow t_k\,t$
  $N \leftarrow (N + t_k P_k)s$
  $d \leftarrow (d + t_k\,w_k)s$
$t_k \leftarrow t_k\,t$
$N \leftarrow N + t_k P_n$
$d \leftarrow d + t_k\,w_n$
return N/d

A.4 Geometric algorithm

Algorithm 7 Geometric $(P_0,\ldots,P_n,w_0,\ldots,w_n,t)$

$h \leftarrow 1$
$u \leftarrow 1-t$
$n_1 \leftarrow n+1$
$N \leftarrow P_0$
if $t \le 1/2$ **then**
  $u \leftarrow t/u$
  for $k \leftarrow 1(1)n$ **do**
    $h \leftarrow h\,u(n_1-k)w_k$
    $h \leftarrow h/(k\,w_{k-1}+h)$
    $h_1 \leftarrow 1-h$
    $N \leftarrow h_1 N + h P_k$
else if $t > 1/2$ **then**
  $u \leftarrow u/t$
  for $k \leftarrow 1(1)n$ **do**
    $h \leftarrow h(n_1-k)w_k$
    $h \leftarrow h/(k\,u\,w_{k-1}+h)$
    $h_1 \leftarrow 1-h$
    $N \leftarrow h_1 N + h P_k$
return N

A.5 Wang–Ball algorithm

Algorithm 8 ACcoefficients (n)

$p_2 \leftarrow 1$
$M \leftarrow \mathbf{1}_{n+1}$ ▷ identity matrix of size $(n+1)\times(n+1)$
$N_i \leftarrow n-2-2i$
for $i \leftarrow 0(1)(\lceil n/2\rceil-1)$ **do**
  for $k \leftarrow 0(1)n$ **do**
    if $i < k$ and $N_i \ge 0$ **then**
      $M_{k,i} \leftarrow M_{k-1,i}(N_i-k+i-1)$
      $M_{k,i} \leftarrow M_{k,i}/(k-i)$
  for $k \leftarrow 0(1)n$ **do**
    $M_{k,i} \leftarrow M_{k,i}\,p_2$
    $M_{n-k,n-i} \leftarrow M_{k,i}$
  $p_2 \leftarrow 2p_2$
return M

Algorithm 9 ToWangBall $(P_0,\ldots,P_n,w_0,\ldots,w_n)$

$(\hat{P}_0,\ldots,\hat{P}_n) \leftarrow$ ToHomogeneous $(P_0,\ldots,P_n,w_0,\ldots,w_n)$
for $i \leftarrow 0(1)n$ **do**
    $\hat{R}_i \leftarrow 0$
$b \leftarrow 1$
$c \leftarrow 1$
$\hat{R}_0 \leftarrow \hat{P}_0$
$\hat{R}_n \leftarrow \hat{P}_n$
$M \leftarrow$ ACcoefficients (n)
$k \leftarrow 1$
while $k \le n-k$ **do**
    $b \leftarrow b(n-k+1)/k$
    $c \leftarrow c/2$
    $\hat{R}_k \leftarrow (b\hat{P}_k - \langle M_k, (\hat{R}_0,\ldots,\hat{R}_n)\rangle)c$ ▷ $\langle a,b\rangle$ is a scalar product
    if $k = n-k$ **then**
        continue
    $K \leftarrow n-k$
    $\hat{R}_K \leftarrow (b\hat{P}_K - \langle M_K, (\hat{R}_0,\ldots,\hat{R}_n)\rangle)c$
    $k \leftarrow k+1$
for $i \leftarrow 0(1)n$ **do**
    $R_i \leftarrow \text{proj}(\hat{R}_i)$
    $v_i \leftarrow \hat{R}_i^z$ ▷ z-coordinate of $\hat{R}_i$
return $(R_0,\ldots,R_n,v_0,\ldots,v_n)$

Algorithm 10 WangBall($R_0, \ldots, R_n, v_0, \ldots, v_n, t$)

$k \leftarrow n$
$J \leftarrow 0$
$s \leftarrow 1 - t$
while $k > 2$ **do**
  if k is odd **then**
    $k_1 \leftarrow (k-1)/2$
    $k_2 \leftarrow (k+1)/2$
    $a \leftarrow s\, v_{k_1}$
    $b \leftarrow t\, v_{k_2}$
    $v_{k_1} \leftarrow a + b$
    $R_{k_1} \leftarrow (R_{k_1} a + b\, R_{k_2})/v_{k_1}$
    if $J = 0$ **then**
      $J \leftarrow k_2 + 1$
  else
    $k_2 \leftarrow k/2$
    if $J = 0$ **then**
      $J \leftarrow k_2 + 1$
    $a \leftarrow s\, v_{k_2-1}$
    $b \leftarrow t\, v_{k_2}$
    $v_{k_2-1} \leftarrow a + b$
    $R_{k_2-1} \leftarrow (R_{k_2-1} a + b\, R_{k_2})/v_{k_2-1}$
    $a \leftarrow s\, v_{k_2}$
    $b \leftarrow t\, v_J$
    $v_{k_2} \leftarrow a + b$
    $R_{k_2} \leftarrow (R_{k_2} a + b\, R_J)/v_{k_2}$
    $J \leftarrow J + 1$
  $k \leftarrow k - 1$
$a \leftarrow s\, v_0$
$b \leftarrow t\, v_1$
$c \leftarrow s\, v_1$
$d \leftarrow t\, v_n$
$w_q \leftarrow a + b$
$w_r \leftarrow c + d$
$e \leftarrow s\, w_q$
$f \leftarrow t\, w_r$
$w_w \leftarrow e + f$
$Q \leftarrow (R_0 a + b\, R_1)/w_q$
$V \leftarrow (R_1 c + d\, R_n)/w_r$
$R \leftarrow (Q e + f V)/w_w$
return R

A.6 Bernstein–Fourier algorithm

Algorithm 11 RealProduct (u, v)

$a \leftarrow \mathrm{Re}(u)\mathrm{Re}(v)$
$b \leftarrow \mathrm{Im}(u)\mathrm{Im}(v)$
return $a - b$

Algorithm 12 ToHomogeneousAndIfft ($P_0, \ldots, P_n, v_0, \ldots, v_n$)

for $i \leftarrow 0(1)n$ **do**
  $\hat{S}_i \leftarrow (v_i P_i, v_i)$
$(S_0, \ldots, S_n) \leftarrow \mathrm{ifft}\,(S_0, \ldots, S_n)$
return $(S_0, \ldots, S_n)$

Algorithm 13 BernsteinFourier $(S_0,\ldots,S_n,\zeta_1,\ldots,\zeta_n,t)$

$p \leftarrow 0$
$t_1 \leftarrow 1-t$
if n is even **then**
  $N \leftarrow n/2+1$
else
  $N \leftarrow (n+1)/2$
for $i \leftarrow 1(1)(N-1)$ **do**
  $u \leftarrow \zeta_i t + t_1$
  $u \leftarrow \mathrm{pow}(u,n)$
  $p \leftarrow p + \mathrm{RealProduct}(u,Q_i)$
$p \leftarrow 2p+Q_0$
if n is odd **then**
  $u \leftarrow 1-2t$
  $u \leftarrow \mathrm{pow}(u,n)$
  $p \leftarrow p+uQ_N$
return $\mathrm{proj}(p)$

Algorithm 14 BernsteinFourier2 $(S_0,\ldots,S_n,\zeta_1,\ldots,\zeta_n,t)$

$p_t \leftarrow 0$
$p_s \leftarrow 0$
$t_1 \leftarrow 1-t$
if n is even **then**
  $N \leftarrow n/2+1$
else
  $N \leftarrow (n+1)/2$
for $i \leftarrow 1(1)(N-1)$ **do**
  $u \leftarrow \zeta_i t + t_1$
  $u \leftarrow \mathrm{pow}(u,n)$
  $p_t \leftarrow p_t + \mathrm{RealProduct}(u,Q_i)$
  $u \leftarrow \overline{u}/\omega_i$
  $p_s \leftarrow p_s + \mathrm{RealProduct}(u,Q_i)$
$p_t \leftarrow 2p_t+Q_0$
$p_s \leftarrow 2p_s+Q_0$
if n is odd **then**
  $u \leftarrow 1-2t$
  $u \leftarrow \mathrm{pow}(u,n)$
  $p_t \leftarrow p_t+uQ_N$
  $p_s \leftarrow p_s-uQ_N$
return $(\mathrm{proj}(p_t),\mathrm{proj}(p_s))$

A.7 Barycentric form

Algorithm 15 AdaptedVS $(P_0,\ldots,P_n,w_0,\ldots,w_n,t)$

$c \leftarrow 1$
if $t \le 1/2$ **then**
  $t_1 \leftarrow 1-t$
  $s \leftarrow t/t_1$
  $c \leftarrow \mathrm{pow}(t_1,n)$
  $d \leftarrow \omega_n$
  $N \leftarrow P_n$
  for $k = 1(1)n$ **do**
    $n_k \leftarrow n-k$
    $N \leftarrow N\,s + P_{n_k}$
    $d \leftarrow d\,s + w_{n_k}$
else if $t > 1/2$ **then**
  $s \leftarrow (1-t)/t$
  $c \leftarrow \mathrm{pow}(t,n)$
  $d \leftarrow \omega_0$
  $N \leftarrow P_0$
  for $k = 1(1)n$ **do**
    $N \leftarrow N\,s + P_k$
    $d \leftarrow d\,s + w_k$
return $(N/d, c\,d)$

Algorithm 16 ToBarycentric $(P_0,\ldots,P_n,w_0,\ldots,w_n)$

$b \leftarrow 1$
$\mathrm{sgn} \leftarrow 1$
$(P_0,\ldots,P_n,w_0,\ldots,w_n) \leftarrow$ PreprocessingOfVSandHornerBezier $(P_0,\ldots,P_n,w_0,\ldots,w_n)$
for $k = 0(1)n$ **do**
  if UNIFORM **then**
    $t_k \leftarrow k/n$
    $Q_k, z \leftarrow$ AdaptedVS $(P_0,\ldots,P_n,w_0,\ldots,w_n,t_k)$
    $u_k \leftarrow \mathrm{sgn}\,b\,z$
    $b \leftarrow b(n+1-(k+1))$
    $b \leftarrow b/(k+1)$
  else if CHEBYSHEV **then**
    $t_k \leftarrow \cos(k\pi/n)$
    $Q_k, z \leftarrow$ AdaptedVS $(P_0,\ldots,P_n,w_0,\ldots,w_n,t_k)$
    $b \leftarrow 1$
    if $k = 0$ or $k = n$ **then**
      $b \leftarrow 0.5$
    $u_k \leftarrow \mathrm{sgn}\,b\,z$
  $\mathrm{sgn} \leftarrow -\mathrm{sgn}$
return $(Q_0,\ldots,Q_n,u_0,\ldots,u_n,t_0,\ldots,t_n)$

Algorithm 17 Barycentric $(Q_0,\ldots,Q_n,u_0,\ldots,u_n,t_0,\ldots,t_n,t)$

$Q \leftarrow 0$
$d \leftarrow 0$
for $k \leftarrow 0(1)n$ **do**
    $u \leftarrow t - t_k$
    if $u = 0$ **then**
        return Q_k
    $u \leftarrow u_k / u$
    $Q \leftarrow Q + uQ_k$
    $d \leftarrow d + u$
$Q \leftarrow Q/d$
return Q

Algorithm 18 Barycentric2 $(Q_0,\ldots,Q_n,u_0,\ldots,u_n,t_0,\ldots,t_n,t)$

$Q_1 \leftarrow 0$
$d_1 \leftarrow 0$
$Q_2 \leftarrow 0$
$d_2 \leftarrow 0$
for $k \leftarrow 0(1)n$ **do**
    $u \leftarrow t - t_k$
    $n_k \leftarrow n - k$
    if $u = 0$ **then**
        return (Q_k, Q_{n_k})
    $v \leftarrow u_{n_k} / u$
    $u \leftarrow u_k / u$
    $Q_1 \leftarrow Q_1 + uQ_k$
    $d_1 \leftarrow d_1 + u$
    $Q_2 \leftarrow Q_2 + vQ_{n_k}$
    $d_2 \leftarrow d_2 + v$
$Q_1 \leftarrow Q_1/d_1$
$Q_2 \leftarrow Q_2/d_2$
return (Q_1, Q_2)