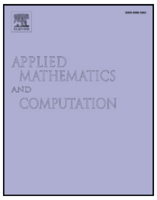
Applied Mathematics and Computation 357 (2019) 57–74

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc



APPLIED MATHEMATICS AND COMPUTATION

Compensated de Casteljau algorithm in K times the working precision

Danny Hermes

UC Berkeley, 970 Evans Hall #3840, Berkeley, CA 94720-3840 USA

ARTICLE INFO

Keywords:
Polynomial evaluation
Compensated algorithm
Floating-point arithmetic
Bernstein polynomial
Error-free transformation
Round-off error

ABSTRACT

In computer aided geometric design a polynomial is usually represented in Bernstein form. This paper presents a family of compensated algorithms to accurately evaluate a polynomial in Bernstein form with floating point coefficients. The principle is to apply error-free transformations to improve the traditional de Casteljau algorithm. At each stage of computation, round-off error is passed on to first order errors, then to second order errors, and so on. After the computation has been "filtered" $(K-1)$ times via this process, the resulting output is as accurate as the de Casteljau algorithm performed in K times the working precision. Forward error analysis and numerical experiments illustrate the accuracy of this family of algorithms.

© 2019 Elsevier Inc. All rights reserved.

1. Introduction

In computer aided geometric design, polynomials are usually expressed in Bernstein form. Polynomials in this form are usually evaluated by the de Casteljau algorithm. This algorithm has a round-off error bound which grows only linearly with degree, even though the number of arithmetic operations grows quadratically. The Bernstein basis is optimally suited ([1–3]) for polynomial evaluation; it is typically more accurate than the monomial basis, for example in Fig. 1.1 evaluation via Horner's method produces a jagged curve for points near a triple root, but the de Casteljau algorithm produces a smooth curve. Nevertheless the de Casteljau algorithm returns results arbitrarily less accurate than the working precision $\mathbf{u}$ when evaluating $p(s)$ is ill-conditioned. The relative accuracy of the computed evaluation with the de Casteljau algorithm (`DeCasteljau`) satisfies ([4]) the following a priori bound:

$$\frac{|p(s) - \texttt{DeCasteljau}(p, s)|}{|p(s)|} \leq \operatorname{cond}(p, s) \times \mathcal{O}(\mathbf{u}). \tag{1.1}$$

In the right-hand side of this inequality, $\mathbf{u}$ is the computing precision and the condition number $\operatorname{cond}(p, s) \geq 1$ only depends on s and the Bernstein coefficients of p — its expression will be given further.

For ill-conditioned problems, such as evaluating $p(s)$ near a multiple root, the condition number may be arbitrarily large, i.e. $\operatorname{cond}(p, s) > 1/\mathbf{u}$, in which case most or all of the computed digits will be incorrect. In some cases, even the order of magnitude of the computed value of $p(s)$ can be incorrect.

To address ill-conditioned problems, error-free transformations (EFT) can be applied in *compensated algorithms* to account for round-off. Error-free transformations were studied in great detail in [5] and open a large number of applications. In [6], a compensated Horner's algorithm was described to evaluate a polynomial in the monomial basis. In [7], a similar method

E-mail address: dhermes@berkeley.edu

https://doi.org/10.1016/j.amc.2019.03.047
0096-3003/© 2019 Elsevier Inc. All rights reserved.

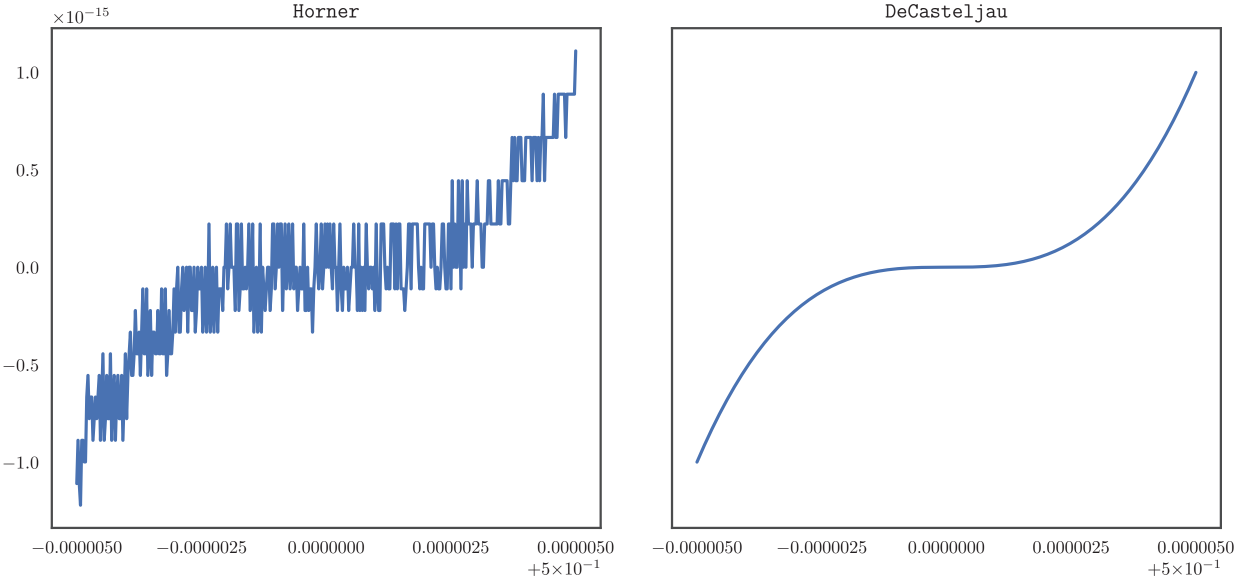


Fig. 1.1. Comparing Horner's method to the de Casteljau method for evaluating $p(s) = (2s - 1)^3$ in the neighborhood of its multiple root 1/2.

was described to perform a compensated version of the de Casteljau algorithm. In both cases, the $\operatorname{cond}(p, s)$ factor is moved from $\mathbf{u}$ to $\mathbf{u}^2$ and the computed value is as accurate as if the computations were done in twice the working precision. For example, the compensated de Casteljau algorithm (`CompDeCasteljau`) satisfies

$$\frac{|p(s) - \texttt{CompDeCasteljau}(p, s)|}{|p(s)|} \leq \mathbf{u} + \operatorname{cond}(p, s) \times \mathcal{O}(\mathbf{u}^2). \tag{1.2}$$

For problems with $\operatorname{cond}(p, s) < 1/\mathbf{u}^2$, the relative error is $\mathbf{u}$, i.e. accurate to full precision, aside from rounding to the nearest floating point number. Fig. 1.2 shows this shift in relative error from `DeCasteljau` to `CompDeCasteljau`.

In [8], the authors generalized the compensated Horner's algorithm to produce a method for evaluating a polynomial as if the computations were done in K times the working precision for any $K \geq 2$. This result motivates this paper, though the approach there is somewhat different than ours. They perform each computation with error-free transformations and interpret the errors as coefficients of new polynomials. They then evaluate the error polynomials, which (recursively) generate second order error polynomials and so on. This recursive property causes the number of operations to grow exponentially in K. Here, we instead have a fixed number of error groups, each corresponding to round-off from the group above it. For example, when $(1 - s)b_j^{(n)} + sb_{j+1}^{(n)}$ is computed in floating point, any error is filtered down to the error group below it.

As in (1.1), the accuracy of the compensated result (1.2) may be arbitrarily bad for ill-conditioned polynomial evaluations. For example, as the condition number grows in Fig. 1.2, some points have relative error exactly equal to 1; this indicates that $\texttt{CompDeCasteljau}(p, s) = 0$, which is a complete failure to evaluate the order of magnitude of $p(s)$. For root-finding problems $\texttt{CompDeCasteljau}(p, s) = 0$ when $p(s) \neq 0$ can cause premature convergence and incorrect results. We describe how to defer rounding into progressively smaller error groups and improve the accuracy of the computed result by a factor of $\mathbf{u}$ for every error group added. So we derive `CompDeCasteljauK`, a K-fold compensated de Casteljau algorithm that satisfies the following a priori bound for any arbitrary integer K:

$$\frac{|p(s) - \texttt{CompDeCasteljauK}(p, s, K)|}{|p(s)|} \leq \mathbf{u} + \operatorname{cond}(p, s) \times \mathcal{O}\left(\mathbf{u}^K\right). \tag{1.3}$$

This means that the computed value with `CompDeCasteljauK` is now as accurate as the result of the de Casteljau algorithm performed in K times the working precision with a final rounding back to the working precision.

The paper is organized as follows. Section 2 establishes notation for error analysis with floating point operations, reviews results about error-free transformations and reviews the de Casteljau algorithm. In Section 3, the compensated algorithm for polynomial evaluation from [7] is reviewed and notation is established for the expansion. In Section 4, the K-compensated algorithm is provided and a forward error analysis is performed. Finally, in Section 5 we perform two numerical experiments to give practical examples of the theoretical error bounds.

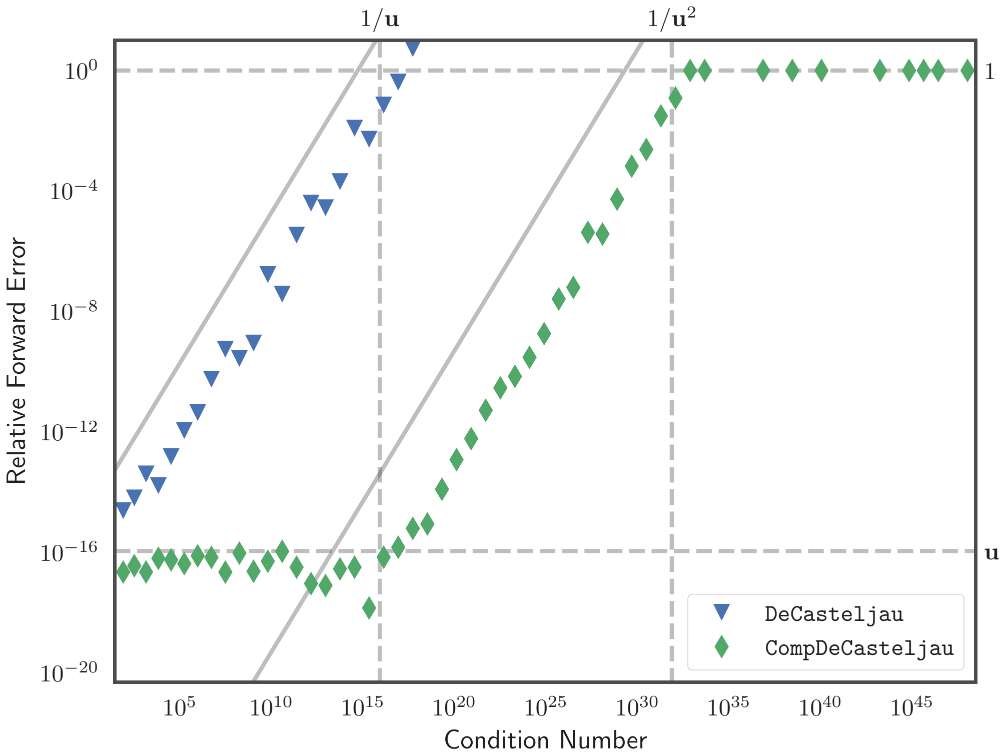


Fig. 1.2. Evaluation of $p(s) = (s-1)(s-3/4)^7$ represented in Bernstein form.

2. Basic notation and results

2.1. Floating point and forward error analysis

We assume all floating point operations obey

$$a \star b = \mathrm{fl}\,(a \circ b) = (a \circ b)(1+\delta_1) = (a \circ b)/(1+\delta_2) \tag{2.1}$$

where $\star \in \{\oplus, \ominus, \otimes, \oslash\}$, $\circ \in \{+, -, \times, \div\}$ and $|\delta_1|, |\delta_2| \le \mathbf{u}$. The symbol $\mathbf{u}$ is the unit round-off and $\star$ is a floating point operation, e.g. $a \oplus b = \mathrm{fl}\,(a+b)$. (For IEEE-754 floating point double precision, $\mathbf{u} = 2^{-53}$.) We denote the computed result of $\alpha \in \mathbf{R}$ in floating point arithmetic by $\widehat{\alpha}$ or $\mathrm{fl}\,(\alpha)$ and use $\mathbf{F}$ as the set of all floating point numbers (see [9] for more details). Following Higham [9], we will use the following classic properties in error analysis.

1. If $\delta_i \le \mathbf{u}$, $\rho_i = \pm 1$, then $\prod_{i=1}^{n}(1+\delta_i)^{\rho_i} = 1+\theta_n$,
2. $|\theta_n| \le \gamma_n := n\mathbf{u}/(1-n\mathbf{u})$,
3. $(1+\theta_k)(1+\theta_j) = 1+\theta_{k+j}$,
4. $\gamma_k + \gamma_j + \gamma_k\gamma_j \le \gamma_{k+j} \iff (1+\gamma_k)(1+\gamma_j) \le 1+\gamma_{k+j}$,
5. $(1+\mathbf{u})^j \le 1/(1-j\mathbf{u}) \iff (1+\mathbf{u})^j - 1 \le \gamma_j$.

2.2. Error-free transformation

An error-free transformation is a computational method where both the computed result and the round-off error are returned. It is considered “free” of error if the round-off can be represented exactly as an element or elements of **F**. The error-free transformations used in this paper are the `TwoSum` algorithm by Knuth ([10]) and `TwoProd` algorithm by Dekker ([11], Section 5), respectively.

Theorem 2.1 ([5], Theorem 3.4). *For $a, b \in \mathbf{F}$ and $P, \pi, S, \sigma \in \mathbf{F}$,* `TwoSum` *and* `TwoProd` *satisfy*

$$[S, \sigma] = \texttt{TwoSum}(a, b), S = \mathrm{fl}\,(a+b), S+\sigma = a+b, \sigma \le \mathbf{u}|S|, \sigma \le \mathbf{u}|a+b| \tag{2.2}$$

$$[P, \pi] = \texttt{TwoProd}(a, b), P = \mathrm{fl}\,(a \times b), P+\pi = a \times b, \pi \le \mathbf{u}|P|, \pi \le \mathbf{u}|a \times b|. \tag{2.3}$$

The letters σ and π are used to indicate that the errors came from sum and product, respectively. See Appendix A for implementation details.

2.3. de Casteljau algorithm

Next, we recall[1] the de Casteljau algorithm:

Algorithm 2.1 de Casteljau algorithm for polynomial evaluation.

```
function result = DeCasteljau($b, s$)
    $n = \texttt{length}(b) - 1$
    $\widehat{r} = 1 \ominus s$

    for $j = 0, \ldots, n$ do
        $\widehat{b}_j^{(n)} = b_j$
    end for

    for $k = n - 1, \ldots, 0$ do
        for $j = 0, \ldots, k$ do
            $\widehat{b}_j^{(k)} = \left(\widehat{r} \otimes \widehat{b}_j^{(k+1)}\right) \oplus \left(s \otimes \widehat{b}_{j+1}^{(k+1)}\right)$
        end for
    end for

    result = $\widehat{b}_0^{(0)}$
end function
```

Theorem 2.2 ([4], Corollary 3.2). *If $p(s) = \sum_{j=0}^{n} b_j B_{j,n}(s)$ and $\texttt{DeCasteljau}(p, s)$ is the value computed by the de Casteljau algorithm then*[2]

$$|p(s) - \texttt{DeCasteljau}(p, s)| \leq \gamma_{3n} \sum_{j=0}^{n} \left|b_j\right| B_{j,n}(s). \tag{2.4}$$

The relative condition number of the evaluation of $p(s) = \sum_{j=0}^{n} b_j B_{j,n}(s)$ in Bernstein form used in this paper is (see [1,4]):

$$\text{cond}\,(p, s) = \frac{\widetilde{p}(s)}{|p(s)|}, \tag{2.5}$$

where $B_{j,n}(s) = \binom{n}{j}(1-s)^{n-j}s^j \geq 0$ and $\widetilde{p}(s) := \sum_{j=0}^{n} \left|b_j\right| B_{j,n}(s)$.

To be able to express the algorithm in matrix form, we define the vectors

$$b^{(k)} = \begin{bmatrix} b_0^{(k)} & \cdots & b_k^{(k)} \end{bmatrix}^T, \quad \widehat{b}^{(k)} = \begin{bmatrix} \widehat{b}_0^{(k)} & \cdots & \widehat{b}_k^{(k)} \end{bmatrix}^T \tag{2.6}$$

and the reduction matrices:

$$U_k = U_k(s) = \begin{bmatrix} 1-s & s & 0 & \cdots & \cdots & 0 \\ 0 & 1-s & s & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & 1-s & s \end{bmatrix} \in \mathbf{R}^{k \times (k+1)}. \tag{2.7}$$

With this, we can express ([4]) the de Casteljau algorithm as

$$b^{(k)} = U_{k+1} b^{(k+1)} \Longrightarrow b^{(0)} = U_1 \cdots U_n b^{(n)}. \tag{2.8}$$

In general, for a sequence $v_0, \ldots, v_n$ we'll refer to v as the vector containing all of the values: $v = \begin{bmatrix} v_0 & \cdots & v_n \end{bmatrix}^T$.

3. Compensated de Casteljau

In this section we review the compensated de Casteljau algorithm from [7]. In order to track the local errors at each update step, we use four EFTs:

$$[\widehat{r}, \rho] = \texttt{TwoSum}(1, -s) \tag{3.1}$$

[1] We have used slightly non-standard notation for the terms produced by the de Casteljau algorithm: we start the superscript at n and count down to 0 as is typically done when describing Horner's algorithm. For example, we use $b_j^{(n-2)}$ instead of $b_j^{(2)}$.

[2] In the original paper the factor on $\widetilde{p}(s)$ is γ_{2n}, but the authors did not consider round-off when computing $1 \ominus s$.

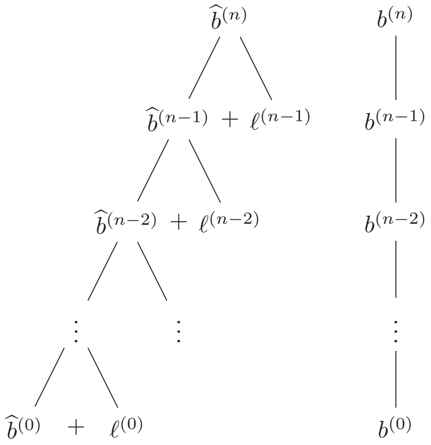

$\widehat{b}^{(n)}$
$b^{(n)}$
$\widehat{b}^{(n-1)} + \ell^{(n-1)}$
$b^{(n-1)}$
$\widehat{b}^{(n-2)} + \ell^{(n-2)}$
$b^{(n-2)}$
$\widehat{b}^{(0)} + \ell^{(0)}$
$b^{(0)}$

Fig. 3.1. Local round-off errors.

Algorithm 3.1 Compensated de Casteljau algorithm for polynomial evaluation.

```
function result = CompDeCasteljau($b, s$)
    $n$ = length($b$) $- 1$
    $[\widehat{r}, \rho]$ = TwoSum($1, -s$)

    for $j = 0, \ldots, n$ do
        $\widehat{b}_j^{(n)} = b_j$
        $\widehat{\partial b}_j^{(n)} = 0$
    end for

    for $k = n-1, \ldots, 0$ do
        for $j = 0, \ldots, k$ do
            $[P_1, \pi_1]$ = TwoProd$\left(\widehat{r}, \widehat{b}_j^{(k+1)}\right)$
            $[P_2, \pi_2]$ = TwoProd$\left(s, \widehat{b}_{j+1}^{(k+1)}\right)$
            $\left[\widehat{b}_j^{(k)}, \sigma_3\right]$ = TwoSum$(P_1, P_2)$
            $\widehat{\ell}_{1,j}^{(k)} = \pi_1 \oplus \pi_2 \oplus \sigma_3 \oplus \left(\rho \otimes \widehat{b}_j^{(k+1)}\right)$
            $\widehat{\partial b}_j^{(k)} = \widehat{\ell}_{1,j}^{(k)} \oplus \left(s \otimes \widehat{\partial b}_{j+1}^{(k+1)}\right) \oplus \left(\widehat{r} \otimes \widehat{\partial b}_j^{(k+1)}\right)$
        end for
    end for

    result = $\widehat{b}_0^{(0)} \oplus \widehat{\partial b}_0^{(0)}$
end function
```

$$[P_1, \pi_1] = \texttt{TwoProd}\left(\widehat{r}, \widehat{b}_j^{(k+1)}\right) \tag{3.2}$$

$$[P_2, \pi_2] = \texttt{TwoProd}\left(s, \widehat{b}_{j+1}^{(k+1)}\right) \tag{3.3}$$

$$\left[\widehat{b}_j^{(k)}, \sigma_3\right] = \texttt{TwoSum}(P_1, P_2) \tag{3.4}$$

With these, we can exactly describe the local error between the exact update and computed update:

$$\ell_{1,j}^{(k)} = \pi_1 + \pi_2 + \sigma_3 + \rho \cdot \widehat{b}_j^{(k+1)} \tag{3.5}$$

$$(1-s) \cdot \widehat{b}_j^{(k+1)} + s \cdot \widehat{b}_{j+1}^{(k+1)} = \widehat{b}_j^{(k)} + \ell_{1,j}^{(k)}. \tag{3.6}$$

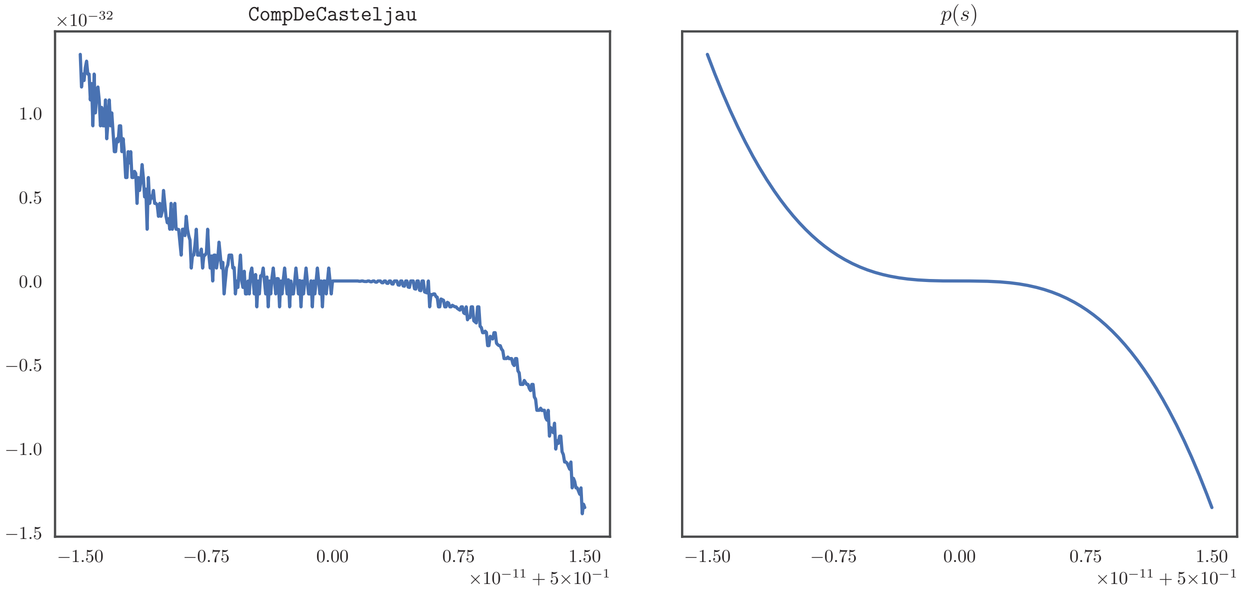


Fig. 3.2. The compensated de Casteljau method starts to lose accuracy for $p(s) = (2s - 1)^3 (s - 1)$ in the neighborhood of its multiple root 1/2.

By defining the global errors at each step

$$\partial b_j^{(k)} = b_j^{(k)} - \widehat{b}_j^{(k)} \tag{3.7}$$

we can see (Fig. 3.1) that the local errors accumulate in $\partial b^{(k)}$:

$$\partial b_j^{(k)} = (1 - s) \cdot \partial b_j^{(k+1)} + s \cdot \partial b_{j+1}^{(k+1)} + \ell_{1,j}^{(k)}. \tag{3.8}$$

When computed in exact arithmetic

$$p(s) = \widehat{b}_0^{(0)} + \partial b_0^{(0)} \tag{3.9}$$

and by using (3.8), we can continue to compute approximations of $\partial b_j^{(k)}$. The idea behind the compensated de Casteljau algorithm is to compute both the local error and the updates of the global error with floating point operations:

When comparing this computed error to the exact error, the difference depends only on s and the Bernstein coefficients of p. Using a bound (Lemma 4.1) on the round-off error when computing $\partial b^{(0)}$, the algorithm can be shown to be as accurate as if the computations were done in twice the working precision:

Theorem 3.1 ([7], Theorem 5)**.** *If no underflow occurs, $n \geq 2$ and $s \in [0, 1]$*

$$\frac{|p(s) - \texttt{CompDeCasteljau}(p, s)|}{|p(s)|} \leq \mathbf{u} + 2\gamma_{3n}^2 \operatorname{cond}(p, s). \tag{3.10}$$

Unfortunately, Fig. 3.2 shows how CompDeCasteljau starts to break down in a region of high condition number (caused by a multiple root with multiplicity higher than two). For example, the point $s = \frac{1}{2} + 1001\mathbf{u}$ — which is in the plotted region $|s - \frac{1}{2}| \leq \frac{3}{2} \cdot 10^{-11}$ — evaluates to exactly 0 when it should be $\mathcal{O}(\mathbf{u}^3)$. As shown in Table 3.1, the breakdown occurs because $\widehat{b}_0^{(0)} = -\widehat{\partial b}_0^{(0)} = \mathbf{u}/16$.

4. *K*-compensated de Casteljau

4.1. Algorithm specified

In order to raise from twice the working precision to K times the working precision, we continue using EFTs when computing $\widehat{\partial b}^{(k)}$. By tracking the round-off from each floating point evaluation via an EFT, we can form a cascade of global errors:

$$b_j^{(k)} = \widehat{b}_j^{(k)} + \partial b_j^{(k)} \tag{4.1}$$

$$\partial b_j^{(k)} = \widehat{\partial b}_j^{(k)} + \partial^2 b_j^{(k)} \tag{4.2}$$

Table 3.1
Terms computed by CompDeCasteljau when evaluating $p(s) = (2s - 1)^3(s - 1)$ at the point $s = \frac{1}{2} + 1001\ \mathbf{u}$.

k	j	$\widehat{b}_j^{(k)}$	$\widehat{\partial b}_j^{(k)}$	$\partial b_j^{(k)} - \widehat{\partial b}_j^{(k)}$
3	0	$0.125 - 1.75\ (1001\ \mathbf{u}) - 0.25\ \mathbf{u}$	$0.25\mathbf{u}$	0
3	1	$-0.125 + 1.25(1001\ \mathbf{u}) + 0.25\ \mathbf{u}$	$-0.25\mathbf{u}$	0
3	2	$0.125 - 0.75\ (1001\ \mathbf{u})$	0	0
3	3	$-0.125 + 0.25\ (1001\ \mathbf{u})$	0	0
2	0	$-0.5\ (1001\ \mathbf{u})$	$3\ (1001\ \mathbf{u})^2$	0
2	1	$0.5(1001\ \mathbf{u}) + 0.125\ \mathbf{u}$	$-0.125\mathbf{u} - 2\ (1001\ \mathbf{u})^2$	0
2	2	$-0.5\ (1001\ \mathbf{u})$	$(1001\ \mathbf{u})^2$	0
1	0	$0.0625\mathbf{u} + (1001\ \mathbf{u})^2 + 239\mathbf{u}^2$	$-0.0625\mathbf{u} + 0.5\ (1001\ \mathbf{u})^2 - 239\ \mathbf{u}^2$	$-5\ (1001\mathbf{u})^3$
1	1	$0.0625\mathbf{u} - (1001\ \mathbf{u})^2 - 239\mathbf{u}^2$	$-0.0625\mathbf{u} - 0.5\ (1001\ \mathbf{u})^2 + 239\ \mathbf{u}^2$	$3\ (1001\mathbf{u})^3$
0	0	$0.0625\ \mathbf{u}$	$-0.0625\ \mathbf{u}$	$-4\ (1001\ \mathbf{u})^3 + 8\ (1001\ \mathbf{u})^4$

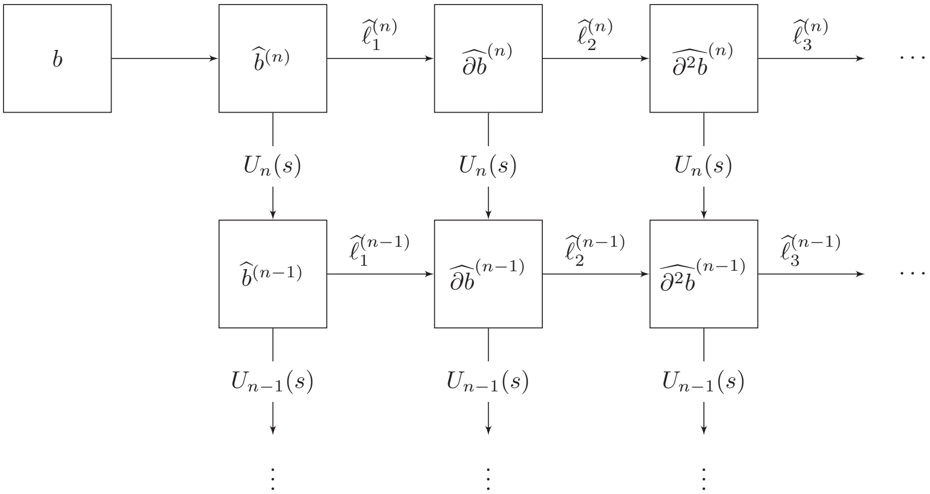


Fig. 4.1. Filtering errors.

$$\partial^2 b_j^{(k)} = \widehat{\partial^2 b}_j^{(k)} + \partial^3 b_j^{(k)} \tag{4.3}$$

$$\vdots$$

In the same way local error can be tracked when updating $\widehat{b}_j^{(k)}$, it can be tracked for updates that happen down the cascade:

$$(1 - s) \cdot \widehat{b}_j^{(k+1)} + s \cdot \widehat{b}_{j+1}^{(k+1)} = \widehat{b}_j^{(k)} + \ell_{1,j}^{(k)} \tag{4.4}$$

$$(1 - s) \cdot \widehat{\partial b}_j^{(k+1)} + s \cdot \widehat{\partial b}_{j+1}^{(k+1)} + \ell_{1,j}^{(k)} = \widehat{\partial b}_j^{(k)} + \ell_{2,j}^{(k)} \tag{4.5}$$

$$(1 - s) \cdot \widehat{\partial^2 b}_j^{(k+1)} + s \cdot \widehat{\partial^2 b}_{j+1}^{(k+1)} + \ell_{2,j}^{(k)} = \widehat{\partial^2 b}_j^{(k)} + \ell_{3,j}^{(k)} \tag{4.6}$$

$$\vdots$$

In CompDeCasteljau (Algorithm 3.1), after a single stage of error filtering we "give up" and use $\widehat{\partial b}$ instead of ∂b (without keeping around any information about the round-off error). In order to obtain results that are as accurate as if computed in K times the working precision, we must continue filtering (see Fig. 4.1) errors down $(K - 1)$ times, and only at the final level do we accept the rounded $\widehat{\partial^{K-1} b}$ in place of the exact $\partial^{K-1} b$.

When computing $\widehat{\partial^F b}$ (i.e. the error after F stages of filtering) there will be several sources of round-off. In particular, there will be

- errors when computing $\widehat{\ell}^{(k)}_{F,j}$ from the terms in $\ell^{(k)}_{F,j}$
- an error for the "missing" $\rho \cdot \widehat{\partial^F b}^{(k+1)}_j$ in $(1-s) \cdot \widehat{\partial^F b}^{(k+1)}_j$
- an error from the product $\widehat{r} \otimes \widehat{\partial^F b}^{(k+1)}_j$
- an error from the product $s \otimes \widehat{\partial^F b}^{(k+1)}_{j+1}$
- two errors from the two $\oplus$ when combining the three terms in $\widehat{\ell}^{(k)}_{F,j} \oplus \left(s \otimes \widehat{\partial^F b}^{(k+1)}_{j+1}\right) \oplus \left(\widehat{r} \otimes \widehat{\partial^F b}^{(k+1)}_j\right)$

For example, in (3.5):

$$\ell^{(k)}_{1,j} = \underbrace{\pi_1}_{P_1=\widehat{r}\otimes\widehat{b}^{(k+1)}_j} + \underbrace{\pi_2}_{P_2=s\otimes\widehat{b}^{(k+1)}_{j+1}} + \underbrace{\sigma_3}_{P_1\oplus P_2} + \underbrace{\rho\cdot\widehat{b}^{(k+1)}_j}_{(1-s)\widehat{b}^{(k+1)}_j} \tag{4.7}$$

After each stage, we'll always have

$$\ell^{(k)}_{F,j} = e_1 + \cdots + e_{5F-2} + \rho \cdot \widehat{\partial^{F-1} b}^{(k+1)}_j \tag{4.8}$$

where the terms $e_1, \ldots, e_{5F-2}$ come from using TwoSum and TwoProd when computing $\widehat{\partial^{F-1} b}^{(k)}_j$ and the ρ term comes from the round-off in $1 \ominus s$ when multiplying $(1-s)$ by $\widehat{\partial^{F-1} b}^{(k+1)}_j$. With this in mind, we can define an EFT (LocalErrorEFT) that computes $\widehat{\ell}$ and tracks all round-off errors generated in the process:

Algorithm 4.1 EFT for computing the local error.

```
function $[\eta, \widehat{\ell}]$ = LocalErrorEFT($e, \rho, \delta b$)
    $L$ = length($e$)

    $[\widehat{\ell}, \eta_1]$ = TwoSum($e_1, e_2$)
    for $j = 3, \ldots, L$ do
        $[\widehat{\ell}, \eta_{j-1}]$ = TwoSum($\widehat{\ell}, e_j$)
    end for

    $[P, \eta_L]$ = TwoProd($\rho, \delta b$)
    $[\widehat{\ell}, \eta_{L+1}]$ = TwoSum($\widehat{\ell}, P$)
end function
```

With this EFT in place,[3] we can perform $(K-1)$ error filtrations. Once we've computed the K stages of global errors, they can be combined with SumK (Algorithm A.6) to produce a sum that is as accurate as if computed in K times the working precision.

Noting that ℓ_{F_j} contains $5F-1$ terms, one can show that CompDeCasteljauK (Algorithm 4.2) requires

$$(15K^2 + 11K - 34)T_n + 6K^2 - 11K + 11 = \mathcal{O}\left(n^2 K^2\right) \tag{4.9}$$

flops to evaluate a degree n polynomial, where T_n is the nth triangular number. As a comparison, the non-compensated form of de Casteljau requires $3T_n + 1$ flops. In total this will require $(3K-4)T_n$ uses of TwoProd. On hardware that supports FMA, TwoProdFMA (Algorithm A.4) can be used instead, lowering the flop count by $15(3K-4)T_n$. Another way to lower the total flop count is to just use $\widehat{b}^{(0)}_0 \oplus \cdots \oplus \widehat{\partial^{K-1} b}^{(0)}_0$ instead of SumK; this will reduce the total by $6(K-1)^2$ flops. When using a standard sum, the results produced are (empirically) identical to those with SumK. This makes sense: the whole point of SumK is to filter errors in a summation so that the final operation produces a sum of the form $v_1 \oplus \cdots \oplus v_K$ where each term is smaller than the previous by a factor of $\mathbf{u}$. This property is already satisfied for the $\widehat{\partial^F b}^{(0)}_0$ so in practice the K-compensated summation is likely not needed.

[3] And the related LocalError in Algorithm A.7.

Algorithm 4.2 *K-compensated de Casteljau algorithm.*

```
function result = CompDeCasteljauK($b, s, K$)
    $n = $ length($b$) $- 1$
    $[\widehat{r}, \rho] = $ TwoSum($1, -s$)

    for $j = 0, \ldots, n$ do
        $\widehat{b}_j^{(n)} = b_j$
        for $F = 1, \ldots, K - 1$ do
            $\widehat{\partial^F b_j}^{(n)} = 0$
        end for
    end for

    for $k = n - 1, \ldots, 0$ do
        for $j = 0, \ldots, k$ do
            $[P_1, \pi_1] = $ TwoProd$\left(\widehat{r}, \widehat{b}_j^{(k+1)}\right)$
            $[P_2, \pi_2] = $ TwoProd$\left(s, \widehat{b}_{j+1}^{(k+1)}\right)$
            $\left[\widehat{b}_j^{(k)}, \sigma_3\right] = $ TwoSum($P_1, P_2$)

            $e = [\pi_1, \pi_2, \sigma_3]$
            $\delta b = \widehat{b}_j^{(k+1)}$

            for $F = 1, \ldots, K - 2$ do
                $\left[\eta, \widehat{\ell}\right] = $ LocalErrorEFT($e, \rho, \delta b$)
                $L = $ length($\eta$)

                $[P_1, \eta_{L+1}] = $ TwoProd$\left(s, \widehat{\partial^F b_{j+1}}^{(k+1)}\right)$
                $[S_2, \eta_{L+2}] = $ TwoSum$\left(\widehat{\ell}, P_1\right)$
                $[P_3, \eta_{L+3}] = $ TwoProd$\left(\widehat{r}, \widehat{\partial^F b_j}^{(k+1)}\right)$
                $\left[\widehat{\partial^F b_j}^{(k)}, \eta_{L+4}\right] = $ TwoSum($S_2, P_3$)

                $e = \eta$
                $\delta b = \widehat{\partial^F b_j}^{(k+1)}$
            end for

            $\widehat{\ell} = $ LocalError($e, \rho, \delta b$)
            $\widehat{\partial^{K-1} b_j}^{(k)} = \widehat{\ell} \oplus \left(s \otimes \widehat{\partial^{K-1} b_{j+1}}^{(k+1)}\right) \oplus \left(\widehat{r} \otimes \widehat{\partial^{K-1} b_j}^{(k+1)}\right)$
        end for
    end for

    result = SumK$\left(\left[\widehat{b}_0^{(0)}, \ldots, \widehat{\partial^{K-1} b_0}^{(0)}\right], K\right)$
end function
```

4.2. Error bound for polynomial evaluation

Theorem 4.1 ([5], Proposition 4.10). *A summation can be computed* (SumK, Algorithm A.6) *with results that are as accurate as if computed in K times the working precision. When computed this way, the result satisfies:*

$$\left| \mathtt{SumK}(v, K) - \sum_{j=1}^{n} v_j \right| \leq \left(\mathbf{u} + 3\gamma_{n-1}^2\right) \left| \sum_{j=1}^{n} v_j \right| + \gamma_{2n-2}^K \sum_{j=1}^{n} \left| v_j \right|. \tag{4.10}$$

Lemma 4.1 ([7], Theorem 4)**.** *The second order error* $\partial^2 b_0^{(0)}$ *satisfies*[4]

$$\left|\partial b_0^{(0)} - \widehat{\partial b_0}^{(0)}\right| = \left|\partial^2 b_0^{(0)}\right| \le 2\gamma_{3n+2}\gamma_{3(n-1)}\widetilde{p}(s). \tag{4.11}$$

To enable a bound on the K order error $\partial^K b_0^{(0)}$, it's necessary to understand the difference between the exact local errors $\ell_{F,j}$ and the computed equivalents $\widehat{\ell}_{F,j}$. To do this, we define

$$\widetilde{\ell}_{F,j} := |e_1| + \cdots + |e_{5F-2}| + \left|\rho \cdot \widehat{\partial^{F-1} b_j}^{(k+1)}\right|. \tag{4.12}$$

Lemma 4.2. *The local error bounds* $\widetilde{\ell}_{F,j}$ *satisfy:*

$$\widetilde{\ell}_{1,j}^{(k)} \le \gamma_3\left((1-s)\left|\widehat{b}_j^{(k+1)}\right| + s\left|\widehat{b}_{j+1}^{(k+1)}\right|\right) \tag{4.13}$$

$$\widetilde{\ell}_{F+1,j}^{(k)} \le \gamma_3\left((1-s)\left|\widehat{\partial^F b_j}^{(k+1)}\right| + s\left|\widehat{\partial^F b_{j+1}}^{(k+1)}\right|\right) + \gamma_{5F}\cdot\widetilde{\ell}_{F,j}^{(k)} \text{ for } F \ge 1. \tag{4.14}$$

As we'll see soon (Lemma 4.4), putting a bound on sums of the form $\sum_{j=0}^{k}\widetilde{\ell}_{F,j}^{(k)}B_{j,k}(s)$ will be useful to get an overall bound on the relative error for `CompDeCasteljauK`, so we define $L_{F,k} := \sum_{j=0}^{k}\widetilde{\ell}_{F,j}^{(k)}B_{j,k}(s)$.

Lemma 4.3. *For* $s \in [0, 1]$, *the Bernstein-type error sum defined above satisfies the following bounds:*

$$L_{F,n-k} \le \left[\left(3^F\binom{k}{F-1} + \mathcal{O}\left(k^{F-1}\right)\right)\mathbf{u}^F + \mathcal{O}\left(\mathbf{u}^{F+1}\right)\right]\cdot\widetilde{p}(s) \tag{4.15}$$

$$\sum_{k=0}^{n-1}\gamma_{3k+5F}L_{F,k} \le \left[\left(3^{F+1}\binom{n}{F+1} + \mathcal{O}\left(n^F\right)\right)\mathbf{u}^{F+1} + \mathcal{O}\left(\mathbf{u}^{F+2}\right)\right]\cdot\widetilde{p}(s). \tag{4.16}$$

In particular, this means that $\sum_{k=0}^{n-1}\gamma_{3k+5F}L_{F,k} = \mathcal{O}\left((3n\mathbf{u})^{F+1}\right)\cdot\widetilde{p}(s)$.

See Appendix B for details on proving Lemmas 4.2 and 4.3.

Lemma 4.4. *The K order error* $\partial^K b_0^{(0)}$ *satisfies*

$$\left|\partial^{K-1}b_0^{(0)} - \widehat{\partial^{K-1}b_0}^{(0)}\right| = \left|\partial^K b_0^{(0)}\right| \le \left[\left(3^K\binom{n}{K} + \mathcal{O}\left(n^{K-1}\right)\right)\mathbf{u}^K + \mathcal{O}\left(\mathbf{u}^{K+1}\right)\right]\cdot\widetilde{p}(s). \tag{4.17}$$

Proof. As in (2.8), we can express the compensated de Casteljau algorithm as

$$\partial^F b^{(k)} = U_{k+1}\partial^F b^{(k+1)} + \ell_F^{(k)} \Longrightarrow \partial^F b^{(0)} = \sum_{k=0}^{n-1}U_1\cdots U_k\ell_F^{(k)} = \sum_{k=0}^{n-1}\left[\sum_{j=0}^{k}\ell_{F,j}^{(k)}B_{j,k}(s)\right]. \tag{4.18}$$

For the inexact equivalent of these things, first note that $\widehat{r} = (1-s)(1+\delta)$. Due to this, we put the $\widehat{r}$ term at the end of each update step to reduce the amount of round-off:

$$\widehat{\partial^F b_j}^{(k)} = \widehat{\ell}_{F,j}^{(k)} \oplus \left(s \otimes \widehat{\partial^F b_{j+1}}^{(k+1)}\right) \oplus \left(\widehat{r} \otimes \widehat{\partial^F b_j}^{(k+1)}\right) \tag{4.19}$$

$$= (1-s)\cdot\widehat{\partial^F b_j}^{(k+1)}(1+\theta_3) + s\cdot\widehat{\partial^F b_{j+1}}^{(k+1)}(1+\theta_3) + \widehat{\ell}_{F,j}^{(k)}(1+\theta_2) \tag{4.20}$$

$$\Longrightarrow \widehat{\partial^F b}^{(k)} = U_{k+1}\widehat{\partial^F b}^{(k+1)}(1+\theta_3) + \widehat{\ell}_F^{(k)}(1+\theta_2) \tag{4.21}$$

$$\Longrightarrow \widehat{\partial^F b}^{(0)} = \sum_{k=0}^{n-1}U_1\cdots U_k\widehat{\ell}_F^{(k)}(1+\theta_{3k+2}) = \sum_{k=0}^{n-1}\left[\sum_{j=0}^{k}\widehat{\ell}_{F,j}^{(k)}(1+\theta_{3k+2})B_{j,k}(s)\right]. \tag{4.22}$$

Since

$$\partial^{F+1}b_0^{(0)} = \partial^F b_0^{(0)} - \widehat{\partial^F b_0}^{(0)} = \sum_{k=0}^{n-1}\sum_{j=0}^{k}\left(\ell_{F,j}^{(k)} - \widehat{\ell}_{F,j}^{(k)}(1+\theta_{3k+2})\right)B_{j,k}(s) \tag{4.23}$$

[4] The authors missed one round-off error so used γ_{3n+1} where γ_{3n+2} would have followed from their arguments.

it's useful to put a bound on $\ell^{(k)}_{F,j} - \widehat{\ell}^{(k)}_{F,j}(1+\theta_{3k+2})$. Via

$$\widehat{\ell}^{(k)}_{F,j} = e_1 \oplus \cdots \oplus e_{5F-2} \oplus \left(\rho \otimes \widehat{\partial^{F-1} b}^{(k+1)}_j\right) \tag{4.24}$$

$$= e_1(1+\theta_{5F-2}) + \cdots + e_{5F-2}(1+\theta_2) + \rho \cdot \widehat{\partial^{F-1} b}^{(k+1)}_j (1+\theta_2) \tag{4.25}$$

we see that

$$\left|\ell^{(k)}_{F,j} - \widehat{\ell}^{(k)}_{F,j}(1+\theta_{3k+2})\right| \le \gamma_{3k+5F} \cdot \widetilde{\ell}^{(k)}_{F,j} \Longrightarrow \left|\partial^{F+1} b_0^{(0)}\right| \le \sum_{k=0}^{n-1} \gamma_{3k+5F} \sum_{j=0}^{k} \widetilde{\ell}^{(k)}_{F,j} B_{j,k}(s). \tag{4.26}$$

Applying (4.16) directly gives

$$\left|\partial^{F+1} b_0^{(0)}\right| \le \left[\left(3^{F+1}\binom{n}{F+1} + \mathcal{O}\left(n^F\right)\right)\mathbf{u}^{F+1} + \mathcal{O}\left(\mathbf{u}^{F+2}\right)\right] \cdot \widetilde{p}(s). \tag{4.27}$$

Letting $K = F+1$ we have our result. □

Theorem 4.2. *If no underflow occurs, $n \ge 2$ and $s \in [0, 1]$*

$$\frac{|p(s) - \texttt{CompDeCasteljau}(p, s, K)|}{|p(s)|} \le \left[\mathbf{u} + \mathcal{O}\left(\mathbf{u}^2\right)\right] + \left[\left(3^K \binom{n}{K} + \mathcal{O}\left(n^{K-1}\right)\right)\mathbf{u}^K + \mathcal{O}\left(\mathbf{u}^{K+1}\right)\right] \operatorname{cond}(p, s). \tag{4.28}$$

Proof. Since

$$\texttt{CompDeCasteljau}(p, s, K) = \texttt{SumK}\left(\left[\widehat{b}_0^{(0)}, \ldots, \widehat{\partial^{K-1} b_0}^{(0)}\right], K\right), \tag{4.29}$$

applying Theorem 4.1 tells us that

$$\left|\texttt{CompDeCasteljau}(p, s, K) - \sum_{F=0}^{K-1} \widehat{\partial^F b_0}^{(0)}\right| \le \left(\mathbf{u} + 3\gamma_{n-1}^2\right)\left|\sum_{F=0}^{K-1} \widehat{\partial^F b_0}^{(0)}\right| + \gamma_{2n-2}^K \sum_{F=0}^{K-1} \left|\widehat{\partial^F b_0}^{(0)}\right|. \tag{4.30}$$

Since

$$p(s) = b_0^{(0)} = \widehat{b}_0^{(0)} + \partial b_0^{(0)} = \cdots = \widehat{b}_0^{(0)} + \widehat{\partial b_0}^{(0)} + \cdots + \widehat{\partial^{K-1} b_0}^{(0)} + \partial^K b_0^{(0)} \tag{4.31}$$

we have

$$\left|\sum_{F=0}^{K-1} \widehat{\partial^F b_0}^{(0)}\right| \le |p(s)| + \left|\partial^K b_0^{(0)}\right| \quad \text{and} \tag{4.32}$$

$$\left|\texttt{CompDeCasteljau}(p, s, K) - p(s)\right| \le \left|\texttt{CompDeCasteljau}(p, s, K) - \sum_{F=0}^{K-1} \widehat{\partial^F b_0}^{(0)}\right| + \left|\partial^K b_0^{(0)}\right|. \tag{4.33}$$

Due to Lemma 4.4, $\partial^F b_0^{(0)} = \mathcal{O}\left(\mathbf{u}^F\right)\widetilde{p}(s)$, hence

$$\left(\mathbf{u} + 3\gamma_{n-1}^2\right)\left|\sum_{F=0}^{K-1} \widehat{\partial^F b_0}^{(0)}\right| \le \left[\mathbf{u} + \mathcal{O}\left(\mathbf{u}^2\right)\right]|p(s)| + \mathcal{O}\left(\mathbf{u}^{K+1}\right)\widetilde{p}(s) \tag{4.34}$$

$$\gamma_{2n-2}^K \sum_{F=0}^{K-1} \left|\widehat{\partial^F b_0}^{(0)}\right| \le \gamma_{2n-2}^K \left|\widehat{b}_0^{(0)}\right| + \mathcal{O}\left(\mathbf{u}^{K+1}\right)\widetilde{p}(s) \tag{4.35}$$

$$\le \gamma_{2n-2}^K\left[|p(s)| + \mathcal{O}(\mathbf{u})\widetilde{p}(s)\right] + \mathcal{O}\left(\mathbf{u}^{K+1}\right)\widetilde{p}(s). \tag{4.36}$$

Combining this with (4.30) and (4.33), we see

$$\left|\texttt{CompDeCasteljau}(p, s, K) - p(s)\right| \tag{4.37}$$

$$\le \left[\mathbf{u} + \mathcal{O}\left(\mathbf{u}^2\right)\right]|p(s)| + \left|\partial^K b_0^{(0)}\right| + \mathcal{O}\left(\mathbf{u}^{K+1}\right)\widetilde{p}(s) \tag{4.38}$$

$$\le \left[\mathbf{u} + \mathcal{O}\left(\mathbf{u}^2\right)\right]|p(s)| + \left[\left(3^K \binom{n}{K} + \mathcal{O}\left(n^{K-1}\right)\right)\mathbf{u}^K + \mathcal{O}\left(\mathbf{u}^{K+1}\right)\right]\widetilde{p}(s). \tag{4.39}$$

Dividing this by $|p(s)|$, we have our result. □

For the first few values of K the coefficient of $\mathrm{cond}\,(p, s)$ in the bound is

K	Method	Multiplier
1	`DeCasteljau`	$3\ \binom{n}{1}\ \mathbf{u} = 3\mathrm{n}\ \mathbf{u} \approx \gamma_{3n}$
2	`CompDeCasteljau`	$[9\ \binom{n}{2} + 15\ \binom{n}{1}]\mathbf{u}^2 = \frac{3n(3n+7)}{2}\ \mathbf{u}^2 \approx \frac{1}{4}.\ 2\ \gamma_{3n}^2$
3	`CompDeCasteljau3`	$[27\ \binom{n}{3} + 135\ \binom{n}{2} + 150\ \binom{n}{1}]\ \mathbf{u}^3 = \frac{3n(3n^2+36n+61)}{2}\ \mathbf{u}^3$
4	`CompDeCasteljau4`	$[81\ \binom{n}{4} + 810\ \binom{n}{3} + 2475\ \binom{n}{2} + 2250\ \binom{n}{1}]\ \mathbf{u}^4$

See the proof of Lemma 4.3 for more details on where these polynomials come from.

5. Numerical experiments

All experiments were performed in IEEE-754 double precision. As in [7], we consider the evaluation in the neighborhood of the multiple root of $p(s) = (s-1)(s-3/4)^7$, written in Bernstein form. Fig. 5.1 shows the evaluation of $p(s)$ at the 401 equally spaced[5] points $\left\{\frac{3}{4} + j\frac{10^{-7}}{2}\right\}_{j=-200}^{200}$ with `DeCasteljau` (Algorithm 2.1), `CompDeCasteljau` (Algorithm 3.1) and `CompDeCasteljau3` (Algorithm 4.2 with $K = 3$). We see that `DeCasteljau` fails to get the magnitude correct, `CompDeCasteljau` has the right shape but lots of noise and `CompDeCasteljau3` is able to smoothly evaluate the function. This is in contrast to a similar figure in [7], where the plot was smooth for the 400 equally spaced points $\left\{\frac{3}{4} + \frac{10^{-4}}{2}\frac{2j-399}{399}\right\}_{j=0}^{399}$. The primary difference is that as the interval shrinks by a factor of $\approx \frac{10^{-4}}{10^{-7}} = 10^3$, the condition number goes up by $\approx 10^{21}$ and `CompDeCasteljau` is no longer accurate.

Fig. 5.2 shows the relative forward errors compared against the condition number. To compute relative errors, each input and coefficient is converted to a fraction (i.e. infinite precision) and $p(s)$ is computed exactly as a fraction, then compared to the corresponding computed values. Similar tools are used to **exactly** compute the condition number, though here we can rely on the fact that $\widetilde{p}(s) = (s-1)(s/2-3/4)^7$. Once the relative errors and condition numbers are computed as fractions, they are rounded to the nearest IEEE-754 double precision value. As in [7], we use values $\left\{\frac{3}{4} - (1.3)^j\right\}_{j=-5}^{-90}$.[6] The curves for `DeCasteljau` and `CompDeCasteljau` trace the same paths seen in [7]. In particular, `CompDeCasteljau` has a relative error that is $\mathcal{O}(\mathbf{u})$ until $\mathrm{cond}\,(p, s)$ reaches $1/\mathbf{u}$, at which point the relative error increases linearly with the condition number until it becomes $\mathcal{O}(1)$ when $\mathrm{cond}\,(p, s)$ reaches $1/\mathbf{u}^2$. Similarly, the relative error in `CompDeCasteljau3` (Algorithm 4.2 with $K = 3$) is $\mathcal{O}(\mathbf{u})$ until $\mathrm{cond}\,(p, s)$ reaches $1/\mathbf{u}^2$ at which point the relative error increases linearly to

[5] It's worth noting that 0.1 cannot be represented exactly in IEEE-754 double precision (or any binary arithmetic for that matter). Hence (most of) the points of the form $a + b \cdot 10^{-c}$ can only be approximately represented.

[6] As with 0.1, it's worth noting that $(1.3)^j$ can't be represented exactly in IEEE-754 double precision. However, this geometric series still serves a useful purpose since it continues to raise $\mathrm{cond}\,(p, s)$ as j decreases away from 0 and because it results in "random" changes in the bits of 0.75 that are impacted by subtracting $(1.3)^j$.

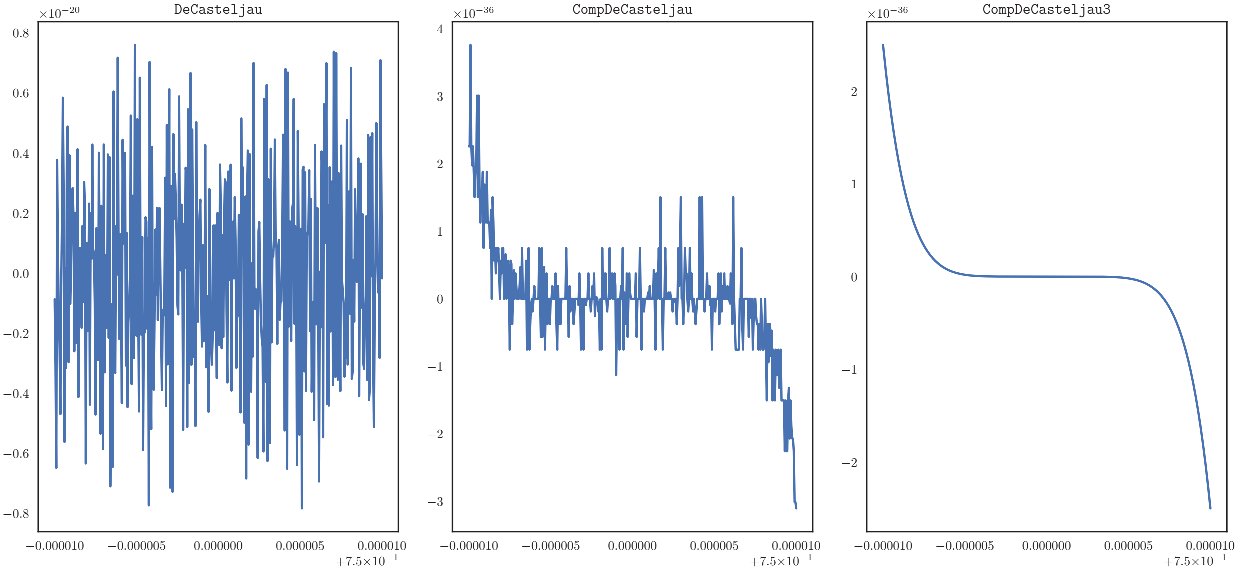


Fig. 5.1. Evaluation of $p(s) = (s-1)(s-3/4)^7$ in the neighborhood of its multiple root 3/4.

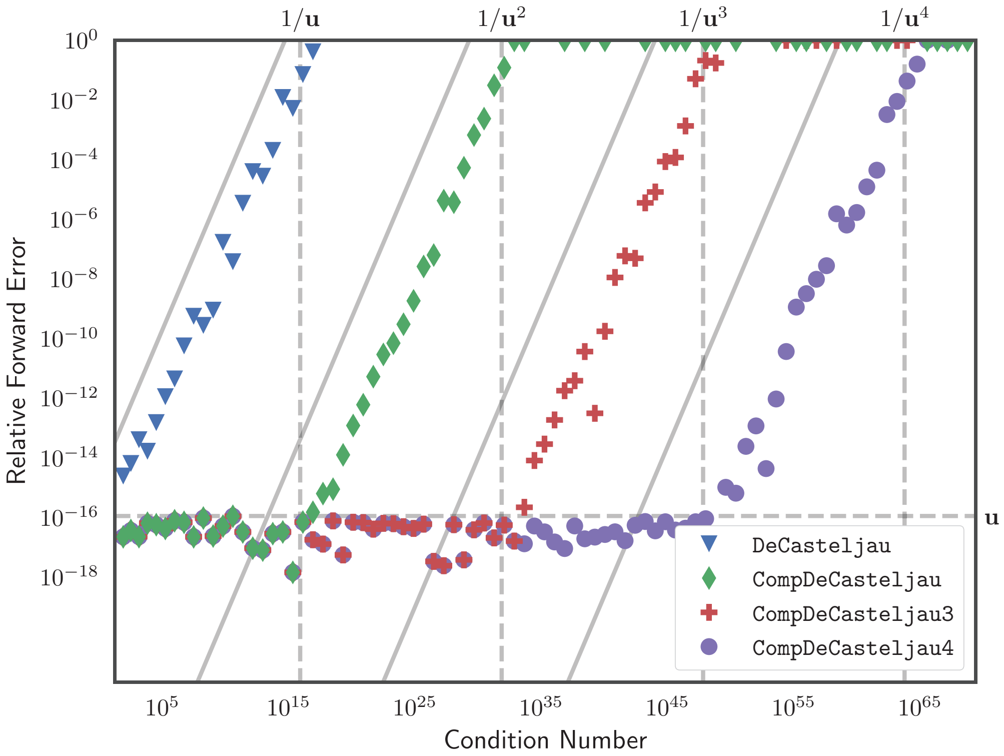


Fig. 5.2. Accuracy of evaluation of $p(s) = (s-1)(s-3/4)^7$ represented in Bernstein form.

$\mathcal{O}(1)$ when cond (p, s) reaches $1/\mathbf{u}^3$ and the relative error in `CompDeCasteljau4` (Algorithm 4.2 with $K = 4$) is $\mathcal{O}(\mathbf{u})$ until cond (p, s) reaches $1/\mathbf{u}^3$ at which point the relative error increases linearly to $\mathcal{O}(1)$ when cond (p, s) reaches $1/\mathbf{u}^4$.

6. Future work

The family of algorithms described in this paper have been implemented in C, C++ and Python by the author ([12]). A more complete compensated algorithms library (similar to [13]) could be quite useful. For example, such a library could include the algorithms in the existing literature such as the K-compensated algorithm for Horner's method from [8].

Acknowledgements

The author would like to thank Ogita et al. [5], Jiang et al. [7] and Graillat et al. [8] for their papers that motivated this work. In particular, the work of Ogita et al. [5] reignited the path to compensated algorithms set forth in [10,11,14].

Appendix A. Algorithms

Find here concrete implementation details on the EFTs described in Theorem 2.1. They do not use branches, nor access to the mantissa that can be time-consuming.

Algorithm A.1 EFT of the sum of two floating point numbers.

```
function $[S, \sigma] = \texttt{TwoSum}(a, b)$
    $S = a \oplus b$
    $z = S \ominus a$
    $\sigma = (a \ominus (S \ominus z)) \oplus (b \ominus z)$
end function
```

In order to avoid branching to check which among $|a|$, $|b|$ is largest, TwoSum uses 6 flops rather than 3.

Algorithm A.2 Splitting of a floating point number into two parts.

```
function $[h, \ell] = \texttt{Split}(a)$
    $z = a \otimes (2^r + 1)$
    $h = z \ominus (z \ominus a)$
    $\ell = a \ominus h$
end function
```

For IEEE-754 double precision floating point number, $r = 27$ so $2^r + 1$ will be known before Split is called. In all, Split uses 4 flops.

Algorithm A.3 EFT of the product of two floating point numbers.

```
function $[P, \pi] = \texttt{TwoProd}(a, b)$
    $P = a \otimes b$
    $[a_h, a_\ell] = \texttt{Split}(a)$
    $[b_h, b_\ell] = \texttt{Split}(b)$
    $\pi = a_\ell \otimes b_\ell \ominus (((P \ominus a_h \otimes b_h) \ominus a_\ell \otimes b_h) \ominus a_h \otimes b_\ell)$
end function
```

This implementation of TwoProd requires 17 flops. For processors that provide a fused-multiply-add operator (FMA), TwoProd can be rewritten to use only 2 flops:

Algorithm A.4 EFT of the sum of two floating point numbers with a FMA.

```
function $[P, \pi] = \texttt{TwoProdFMA}(a, b)$
    $P = a \otimes b$
    $\pi = \texttt{FMA}(a, b, -P)$
end function
```

The following algorithms from [5] can be used as a compensated method for computing a sum of numbers. The first is a vector transformation that is used as a helper:

Algorithm A.5 Error-free vector transformation for summation.

```
function $\texttt{VecSum}(p)$
    $n = \texttt{length}(p)$
    for $j = 2, \ldots, n$ do
        $\left[p_j, p_{j-1}\right] = \texttt{TwoSum}\left(p_j, p_{j-1}\right)$
    end for
end function
```

The second (SumK) computes a sum with results that are as accurate as if computed in K times the working precision. It requires $(6K - 5)(n - 1)$ floating point operations.

Algorithm A.6 Summation as in K-fold precision by $(K - 1)$-fold error-free vector transformation.

```
function $\texttt{result} = \texttt{SumK}(p, K)$
    for $j = 1, \ldots, K - 1$ do
        $p = \texttt{VecSum}(p)$
    end for
    $\texttt{result} = p_1 \oplus p_2 \oplus \cdots \oplus p_n$
end function
```

Since the final error $\widehat{\partial^{K-1} b}$ will not track the errors during computation, we have a non-EFT version of Algorithm 4.1:

Algorithm A.7 Compute the local error (non-EFT).

```
function $\widehat{\ell} = \texttt{LocalError}(e, \rho, \delta b)$
    $L = \texttt{length}(e)$

    $\widehat{\ell} = e_1 \oplus e_2$
    for $j = 3, \ldots, L$ do
        $\widehat{\ell} = \widehat{\ell} \oplus e_j$
    end for

    $\widehat{\ell} = \widehat{\ell} \oplus (\rho \otimes \delta b)$
end function
```

Appendix B. Proof details

Proof of Lemma 4.2. We'll start with the $F = 1$ case. Recall where the terms originate:

$$[P_1, e_1] = \texttt{TwoProd}\left(\widehat{r}, \widehat{b}_j^{(k+1)}\right) \tag{B.1}$$

$$[P_2, e_2] = \texttt{TwoProd}\left(s, \widehat{b}_{j+1}^{(k+1)}\right) \tag{B.2}$$

$$\left[\widehat{b}_j^{(k)}, e_3\right] = \texttt{TwoSum}(P_1, P_2). \tag{B.3}$$

Hence Theorem 2.1 tells us that

$$|P_1| \le (1 + \mathbf{u})\left|\widehat{r} \cdot \widehat{b}_j^{(k+1)}\right| \le (1 + \mathbf{u})^2 (1 - s)\left|\widehat{b}_j^{(k+1)}\right| \tag{B.4}$$

$$|e_1| \le \mathbf{u}\left|\widehat{r} \cdot \widehat{b}_j^{(k+1)}\right| \le \mathbf{u}(1 + \mathbf{u})(1 - s)\left|\widehat{b}_j^{(k+1)}\right| \tag{B.5}$$

$$|P_2| \le (1 + \mathbf{u}) s\left|\widehat{b}_{j+1}^{(k+1)}\right| \tag{B.6}$$

$$|e_2| \le \mathbf{u} s\left|\widehat{b}_{j+1}^{(k+1)}\right| \tag{B.7}$$

$$|e_3| \le \mathbf{u}|P_1| + \mathbf{u}|P_2| \tag{B.8}$$

$$\left|\rho \cdot \widehat{b}_j^{(k+1)}\right| \le (1 + \mathbf{u})(1 - s)\left|\widehat{b}_j^{(k+1)}\right|. \tag{B.9}$$

In general, we can swap $\mathbf{u}|P_j|$ for $(1 + \mathbf{u})|e_j|$ based on how closely related the bound on the result and the bound on the error are. Thus

$$\widetilde{\ell}_{1,j}^{(k)} = |e_1| + |e_2| + |e_3| + \left|\rho \cdot \widehat{b}_j^{(k+1)}\right| \tag{B.10}$$

$$\le (2 + \mathbf{u})(|e_1| + |e_2|) + (1 + \mathbf{u})(1 - s)\left|\widehat{b}_j^{(k+1)}\right| \tag{B.11}$$

$$\le \left[(1 + \mathbf{u})^3 - 1\right](1 - s)\left|\widehat{b}_j^{(k+1)}\right| + \left[(1 + \mathbf{u})^2 - 1\right] s\left|\widehat{b}_{j+1}^{(k+1)}\right| \tag{B.12}$$

$$\le \gamma_3\left((1 - s)\left|\widehat{b}_j^{(k+1)}\right| + s\left|\widehat{b}_{j+1}^{(k+1)}\right|\right). \tag{B.13}$$

For $\widetilde{\ell}_{F+1}$, we want to relate the "current" errors $e_1, \ldots, e_{5F+3}$ to the "previous" errors $e'_1, \ldots, e'_{5F-2}$ that show up in $\widetilde{\ell}_F$. In the same fashion as above, we track where the current errors come from:

$$[S_1, e_1] = \texttt{TwoSum}\left(e'_1, e'_2\right) \tag{B.14}$$

$$[S_2, e_2] = \texttt{TwoSum}\left(S_1, e'_3\right) \tag{B.15}$$

$$\vdots$$

$$[S_{5F-3}, e_{5F-3}] = \texttt{TwoSum}\left(S_{5F-4}, e'_{5F-2}\right) \tag{B.16}$$

$$[P_{5F-2}, e_{5F-2}] = \texttt{TwoProd}\left(\rho, \widehat{\partial^{F-1} b}_j^{(k+1)}\right) \tag{B.17}$$

$$\left[\widehat{\ell}_{F,j}^{(k)}, e_{5F-1}\right] = \texttt{TwoSum}(S_{5F-3}, P_{5F-2}) \tag{B.18}$$

$$[P_{5F}, e_{5F}] = \texttt{TwoProd}\left(s, \widehat{\partial^F b}_{j+1}^{(k+1)}\right) \tag{B.19}$$

$$[S_{5F+1}, e_{5F+1}] = \texttt{TwoSum}\left(\widehat{\ell}_{F,j}^{(k)}, P_{5F}\right) \tag{B.20}$$

$$[P_{5F+2}, e_{5F+2}] = \texttt{TwoProd}\left(\rho, \widehat{\partial^F b}_j^{(k+1)}\right) \tag{B.21}$$

$$\left[\widehat{\partial^F b}_j^{(k)}, e_{5F+3}\right] = \texttt{TwoSum}(S_{5F+1}, P_{5F+2}). \tag{B.22}$$

Arguing as we did above, we start with $|e_1| \leq \mathbf{u}\left|e'_1\right| + \mathbf{u}\left|e'_2\right|$ and build each bound recursively based on the previous, e.g. $|e_2| \leq \mathbf{u}|S_1| + \mathbf{u}\left|e'_3\right| \leq (1+\mathbf{u})\mathbf{u}\left|e'_1\right| + (1+\mathbf{u})\mathbf{u}\left|e'_2\right| + \mathbf{u}\left|e'_3\right|$. Proceeding in this fashion, we find

$$\widetilde{\ell}_{F+1,j}^{(k)} = |e_1| + \cdots + |e_{5F+3}| + \left|\rho \cdot \widehat{\partial^F b}_j^{(k+1)}\right| \tag{B.23}$$

$$\leq \gamma_{5F}\left|e'_1\right| + \gamma_{5F}\left|e'_2\right| + \gamma_{5F-1}\left|e'_3\right| + \cdots + \gamma_4\left|e'_{5F-2}\right| + \gamma_4\left|\rho \cdot \widehat{\partial^{F-1} b}_j^{(k+1)}\right| \tag{B.24}$$

$$+ \gamma_3(1-s)\left|\widehat{\partial^F b}_j^{(k+1)}\right| + \gamma_3\ s\left|\widehat{\partial^F b}_{j+1}^{(k+1)}\right| \tag{B.25}$$

$$\leq \gamma_3\left((1-s)\left|\widehat{\partial^F b}_j^{(k+1)}\right| + s\left|\widehat{\partial^F b}_{j+1}^{(k+1)}\right|\right) + \gamma_{5F} \cdot \widetilde{\ell}_{F,j}^{(k)} \tag{B.26}$$

as desired. □

Proof of Lemma 4.3. First, note that for **any** sequence $v_0, \dots, v_{k+1}$ we must have

$$\sum_{j=0}^{k}\left[(1-s)v_j + s v_{j+1}\right]B_{j,k}(s) = \sum_{j=0}^{k+1} v_j B_{j,k+1}(s). \tag{B.27}$$

For example of this in use, via (4.13), we have

$$L_{1,k} \leq \gamma_3 \sum_{j=0}^{k+1}\left|\widehat{b}_j^{(k+1)}\right| B_{j,k+1}(s). \tag{B.28}$$

In order to work with sums of this form, we define Bernstein-type sums related to $L_{F,k}$:

$$D_{0,k} := \sum_{j=0}^{k}\left|\widehat{b}_j^{(k)}\right| B_{j,k}(s) \tag{B.29}$$

$$D_{F,k} := \sum_{j=0}^{k}\left|\widehat{\partial^F b}_j^{(k)}\right| B_{j,k}(s). \tag{B.30}$$

Hence Lemma 4.2 gives

$$L_{1,k} \leq \gamma_3 D_{0,k+1} \tag{B.31}$$

$$L_{F+1,k} \leq \gamma_3 D_{F,k+1} + \gamma_{5F} L_{F,k} \tag{B.32}$$

In addition, for $F \geq 1$ since

$$\widehat{\partial^F b}_j^{(k)} = \widehat{\ell}_{F,j}^{(k)} \oplus \left(s \otimes \widehat{\partial^F b}_{j+1}^{(k+1)}\right) \oplus \left((1 \ominus s) \otimes \widehat{\partial^F b}_j^{(k+1)}\right) \tag{B.33}$$

$$= (1-s) \cdot \widehat{\partial^F b}_j^{(k+1)}(1+\theta_3) + s \cdot \widehat{\partial^F b}_{j+1}^{(k+1)}(1+\theta_3) + \widehat{\ell}_{F,j}^{(k)}(1+\theta_2) \tag{B.34}$$

we have

$$D_{F,k} \le (1+\gamma_3)D_{F,k+1} + (1+\gamma_2)\sum_{j=0}^{k}\left|\widehat{\ell}^{(k)}_{F,j}\right| B_{j,k}(s). \tag{B.35}$$

Since $\ell^{(k)}_{F,j}$ has $5F-1$ terms (only the last of which involves a product), the terms in the computed value will be involved in at most $5F-2$ flops, hence $|\widehat{\ell}^{(k)}_{F,j}| \le (1+\gamma_{5F-2})\widetilde{\ell}^{(k)}_{F,j}$. Combined with (B.35) and the fact that there is no local error when $F=0$, this means

$$D_{0,k} \le (1+\gamma_3)D_{0,k+1} \tag{B.36}$$

$$D_{F,k} \le (1+\gamma_3)D_{F,k+1} + (1+\gamma_{5F})L_{F,k}. \tag{B.37}$$

The four inequalities (B.31), (B.32), (B.36) and (B.37) allow us to write all bounds in terms of $D_{0,n} = \widetilde{p}(s)$ and $D_{F,n} = 0$. From (B.36) we can conclude that $D_{0,n-k} \le (1+\gamma_{3k})\cdot\widetilde{p}(s)$ and from (B.31) that $L_{1,n-k} \le \gamma_3(1+\gamma_{3(k-1)})\cdot\widetilde{p}(s)$.

To show the bounds for higher values of F, we'll assume we have bounds of the form $D_{F,n-k} \le (q_F(k)\mathbf{u}^F + \mathcal{O}(\mathbf{u}^{F+1}))\cdot\widetilde{p}(s)$ and $L_{F,n-k} \le (r_F(k)\mathbf{u}^F + \mathcal{O}(\mathbf{u}^{F+1}))\cdot\widetilde{p}(s)$ for two families of polynomials $q_F(k)$, $r_F(k)$. We have $q_0(k)=1$ and $r_1(k)=3$ as our base cases and can build from there. To satisfy (B.37), we'd like $q_F(k) = q_F(k-1) + r_F(k)$ and for (B.32) $r_{F+1}(k) = 3q_F(k-1) + 5Fr_F(k)$. Since the forward difference $\Delta q_F(k) = r_F(k+1)$ is known, we can inductively solve for q_F in terms of $q_F(0)$. But $D_{F,n}=0$ gives $q_F(0)=0$.

For example, since we have $r_1(k) = 3\binom{k}{0}$ we'll have $q_1(k) = 3\binom{k}{1}$. Once this is known

$$r_2(k) = 3q_1(k-1) + 5r_1(k) = 3\cdot 3\binom{k-1}{1} + 5\cdot 3\binom{k}{0} = 9\binom{k}{1} + 6\binom{k}{0}. \tag{B.38}$$

If we write these polynomials in the "falling factorial" basis of forward differences, then we can show that

$$r_F(k) = 3^F\binom{k}{F} + \cdots \tag{B.39}$$

which will complete the proof of the first inequality. To see this, first note that for a polynomial in this basis $f(k) = A\binom{k}{d} + B\binom{k}{d-1} + C\binom{k}{d-2} + D\binom{k}{d-3} + \cdots$ we have

$$f(k+1) = A\binom{k}{d} + (A+B)\binom{k}{d-1} + (B+C)\binom{k}{d-2} + (C+D)\binom{k}{d-3} + \cdots \tag{B.40}$$

$$f(k-1) = A\binom{k}{d} + (B-A)\binom{k}{d-1} + (C-B+A)\binom{k}{d-2} + (D-C+B-A)\binom{k}{d-3} + \cdots \tag{B.41}$$

Using these, we can show that if $r_F(k) = \sum_{j=0}^{F-1} c_j\binom{k}{j}$ then

$$q_F(k) = c_{F-1}\binom{k}{F} + \sum_{j=1}^{F-1}(c_j + c_{j-1})\binom{k}{j} \tag{B.42}$$

$$r_{F+1}(k) = 3\left[-c_0\binom{k}{0} + \sum_{j=1}^{F} c_{j-1}\binom{k}{j}\right] + 5F\left[\sum_{j=0}^{F-1} c_j\binom{k}{j}\right] = 3c_{F-1}\binom{k}{F} + \cdots \tag{B.43}$$

Under the inductive hypothesis $c_{F-1} = 3^F$ so that the lead term in $r_{F+1}(k)$ is $3c_{F-1}\binom{k}{F} = 3^{F+1}\binom{k}{F}$.

For the second inequality, we'll show that

$$\sum_{k=0}^{n-1}\gamma_{3k+5F}L_{F,k} \le \left[q_{F+1}(n)\mathbf{u}^{F+1} + \mathcal{O}\left(\mathbf{u}^{F+2}\right)\right]\cdot\widetilde{p}(s) \tag{B.44}$$

and then we'll have our result since we showed above that $q_{F+1}(n) = 3^{F+1}\binom{n}{F+1} + \mathcal{O}\left(n^F\right)$. Since $\gamma_{3k+5F}L_{F,k} \le (3k+5F)L_{F,k}\mathbf{u} + \mathcal{O}(\mathbf{u}^{F+2})\widetilde{p}(s)$ it's enough to consider

$$\sum_{k=0}^{n-1}(3k+5F)r_F(n-k) = \sum_{k=1}^{n}(3(n-k)+5F)r_F(k). \tag{B.45}$$

Since $q_F(k) = q_F(k-1) + r_F(k)$ and $q_F(0)=0$ we have $q_F(n) = \sum_{k=1}^{n} r_F(k)$ thus

$$q_{F+1}(n) = \sum_{k=1}^{n} r_{F+1}(k) = \sum_{k=1}^{n} 3q_F(k-1) + 5Fr_F(k) = \sum_{k=1}^{n} 3\left[\sum_{j=1}^{k-1} r_F(j)\right] + 5Fr_F(k). \tag{B.46}$$

Swapping the order of summation and grouping like terms, we have our result. □

References

[1] R. Farouki, V. Rajan, On the numerical condition of polynomials in Bernstein form, Comput. Aided Geom. Des. 4 (3) (1987) 191–216, doi:10.1016/0167-8396(87)90012-4.

[2] J. Delgado, J. Peña, Accurate evaluation of Bézier curves and surfaces and the Bernstein–Fourier algorithm, Appl. Math. Comput. 271 (2015) 113–122, doi:10.1016/j.amc.2015.08.086.

[3] E. Mainar, J.M. Peña, Running Error Analysis of Evaluation Algorithms for Bivariate Polynomials in Barycentric Bernstein Form, Computing 77 (1) (2005) 97–111, doi:10.1007/s00607-005-0149-8.

[4] E. Mainar, J. Peña, Error analysis of corner cutting algorithms, Numer. Algorithms 22 (1) (1999) 41–52, doi:10.1023/a:1019190220312.

[5] T. Ogita, S.M. Rump, S. Oishi, Accurate Sum and Dot Product, SIAM J. Scientif. Comput. 26 (6) (2005) 1955–1988, doi:10.1137/030601818.

[6] P. Langlois, S. Graillat, N. Louvet, Compensated Horner Scheme, in: B. Buchberger, S. Oishi, M. Plum, S.M. Rump (Eds.), Algebraic and Numerical Algorithms and Computer-assisted Proofs, No. 05391 in Dagstuhl Seminar Proceedings, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, Dagstuhl, Germany, 2006, pp. 1–29. http://drops.dagstuhl.de/opus/volltexte/2006/442

[7] H. Jiang, S. Li, L. Cheng, F. Su, Accurate evaluation of a polynomial and its derivative in Bernstein form, Comput. Math. Appl. 60 (3) (2010) 744–755, doi:10.1016/j.camwa.2010.05.021.

[8] S. Graillat, P. Langlois, N. Louvet, Algorithms for accurate, validated and fast polynomial evaluation, Jpn. J. Ind. Appl.Math. 26 (2–3) (2009) 191–214, doi:10.1007/bf03186531.

[9] N.J. Higham, Accuracy and Stability of Numerical Algorithms, Society for Industrial and Applied Mathematics, 2002, doi:10.1137/1.9780898718027.

[10] D.E. Knuth, The Art of Computer Programming, Volume 2: Seminumerical Algorithms, Addison Wesley, 1969.

[11] T.J. Dekker, A floating-point technique for extending the available precision, Numerische Mathematik 18 (3) (1971) 224–242, doi:10.1007/bf01397083.

[12] D. Hermes, dhermes/k-compensated-de-casteljau: 2018.08.28, 2018, https://zenodo.org/record/1405259.

[13] R. Barrio, P. Du, H. Jiang, S. Serrano, Orthopoly: A library for accurate evaluation of series of classical orthogonal polynomials and their derivatives, Comput. Phys. Commun. 231 (2018) 146–162, doi:10.1016/j.cpc.2018.05.004.

[14] I. Babuška, Numerical stability in mathematical analysis, in: Information Processing 68: Proceedings of IFIP Congress 1968, North-Holland, 1968, pp. 11–23.