High-order Lagrangian Methods and Computations on Curved Elements

by

Danny Hermes

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Applied Mathematics

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Per-Olof Persson, Chair
Professor John Strain
Professor TODO

Summer 2018

The dissertation of Danny Hermes, titled High-order Lagrangian Methods and Computations on Curved Elements, is approved:

Chair		Date	
		Date	
		Date	

University of California, Berkeley

High-order Lagrangian Methods and Computations on Curved Elements

Abstract

High-order Lagrangian Methods and Computations on Curved Elements

by

Danny Hermes

Doctor of Philosophy in Applied Mathematics

University of California, Berkeley

Professor Per-Olof Persson, Chair

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.

To my wife Sharona and my sons Jack and Max.

Contents

List of Figures

List of Tables

Acknowledgments

I want to thank my advisor for his patience and guidance.

Chapter 1

Introduction

The method of characteristics helps transform partial differential equations into ordinary differential equations by dividing the physical domain into a family of curves. For example, the simple transport equation

$$u_t + cu_x = 0 \tag{1.1}$$

can be transformed when restricting to the family of lines $x(t) = x_0 + ct$. On these lines $u(x(t), t)$ is constant, by construction, and so the solution is "transported" from $u(x_0, 0)$ along each characteristic line.

Motivated by this, **Lagrangian methods** treat each point in the physical domain as a "particle" which moves along a characteristic curve over time and then monitor values associated with the particle (heat / energy, velocity, pressure, density, concentration, etc.). They are an effective way to solve PDEs, even with higher order or non-linear terms. For example, if we add a viscosity term to (1.1)

$$u_t + cu_x - \varepsilon u_{xx} = 0 \tag{1.2}$$

then the same characteristics can be used, but the value along each characteristic is no longer constant; instead it satisfies the ODE $\frac{d}{dt}u(x(t), t) = \varepsilon u_{xx}$.

This approach transforms the numerical solution of PDEs into a family of numerical solutions to many independent ODEs. It allows the use of familiar and well understood ODE solvers. In addition, Lagrangian methods often have less restrictive conditions on time steps than Eulerian methods. When solving PDEs on unstructured meshes with Lagrangian methods, the nodes that determine the mesh move (since they are treated like particles) and the mesh adapts. However, mesh adaptivity ([DK87, JH04, BR78]) can cause problems if it causes the mesh to leave the domain being analyzed or if it deforms the mesh until the element quality is too low in some mesh elements. For an example of such deformation (Figure 1.1), consider a PDE of the form

$$u_t + \begin{bmatrix} y^2 \\ 1 \end{bmatrix} \cdot \nabla u + F(u, \nabla u) = 0. \tag{1.3}$$

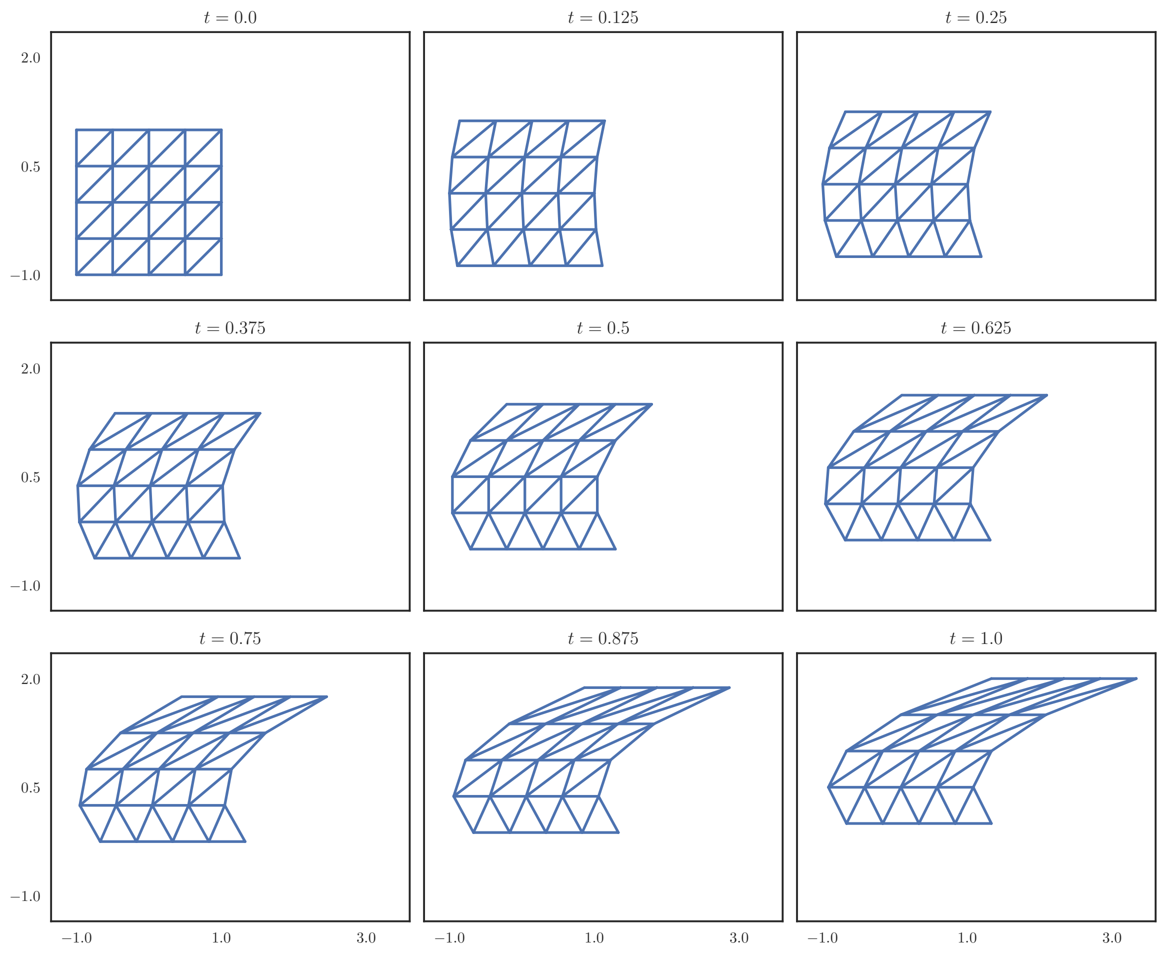

Figure 1.1: Deformation of elements caused by particle motion.

This has characteristics given by

$$x' = y^2,\ y' = 1 \Longrightarrow y(t) = y_0 + t,\ x(t) = x_0 + \frac{1}{3}\left(y(t)^3 - y_0^3\right) \tag{1.4}$$

and after just one second the mesh becomes unusable.

Original ALE paper: [HAC74]

In [FPP$^+$09, FM11], stuff happens

In many applications, the field must be conserved for physical reasons, e.g. mass or energy cannot leave or enter the system.

1.1 CAD

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

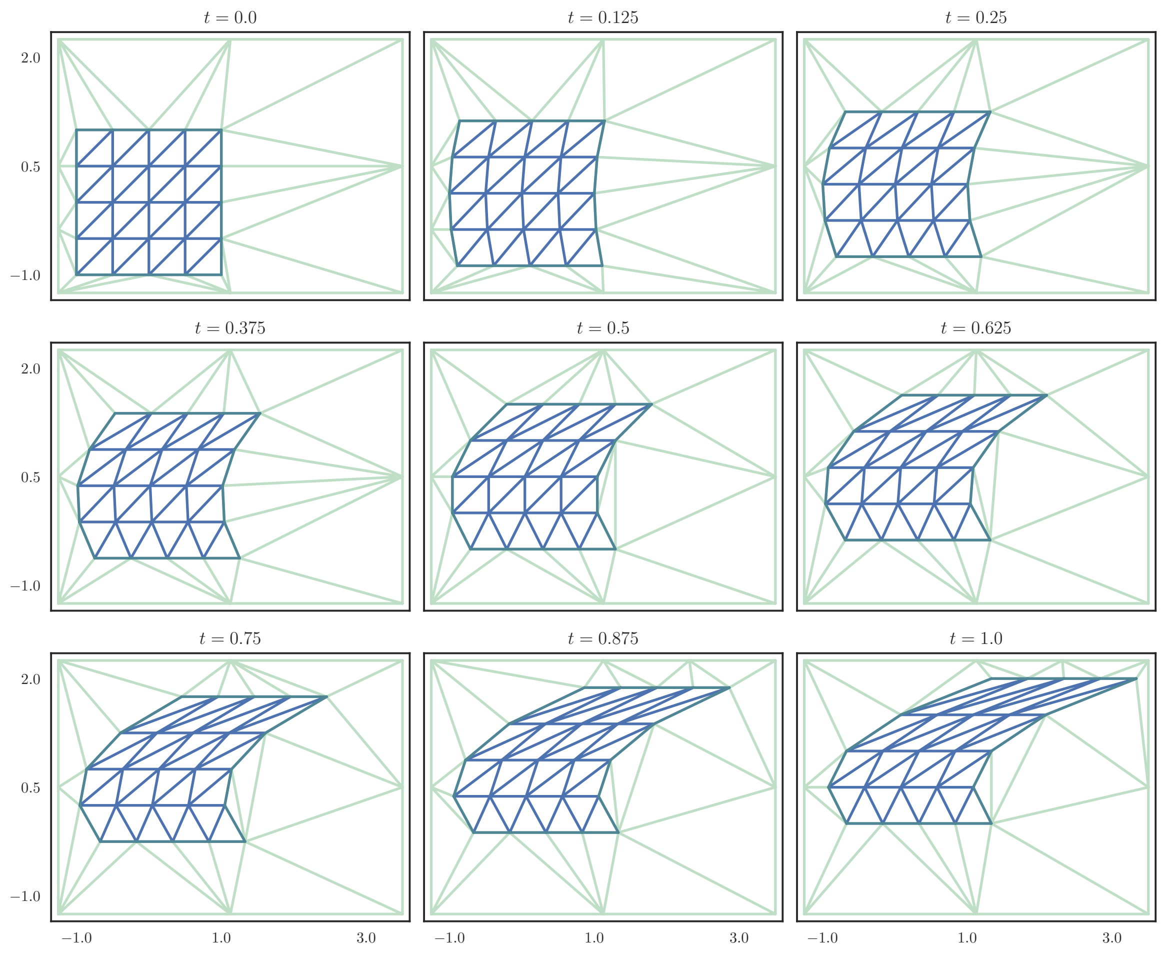

Figure 1.2: Handling partially overlapping meshes.

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 ([FR87, DP15, MP05]) for polynomial evaluation; it is typically more accurate than the monomial basis, for example in Figure 1.3 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 ([MP99]) the following a priori bound:

$$\frac{|p(s) - \texttt{DeCasteljau}(p, s)|}{|p(s)|} \leq \operatorname{cond}(p, s) \times \mathcal{O}(\mathbf{u}). \tag{1.5}$$

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

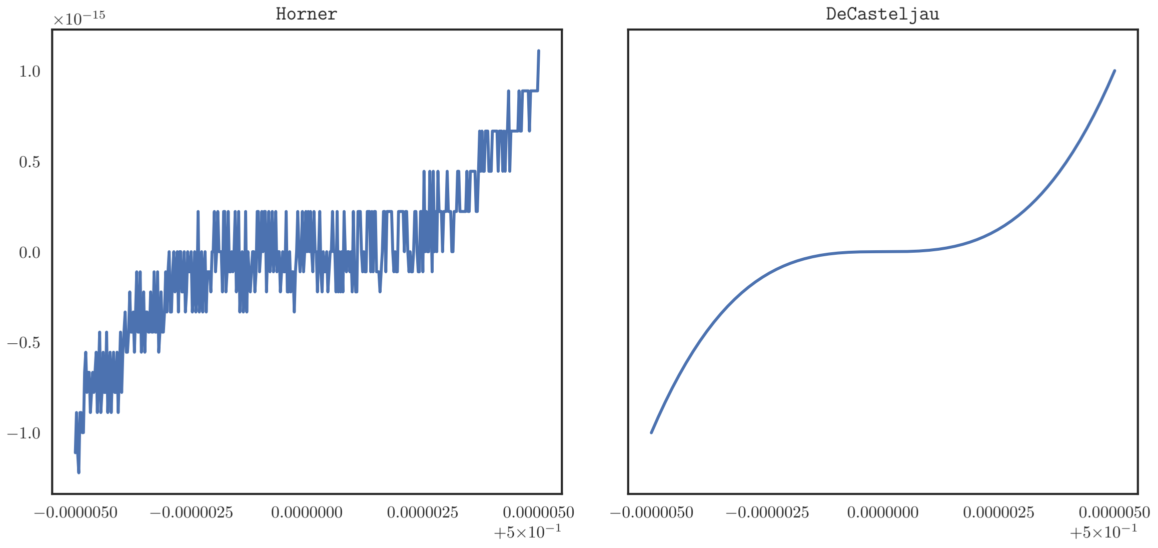

Figure 1.3: Comparing Horner's method to the de Casteljau method for evaluating $p(s) = (2s - 1)^3$ in the neighborhood of its multiple root $1/2$.

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 [ORO05] and open a large number of applications. In [LGL06], a compensated Horner's algorithm was described to evaluate a polynomial in the monomial basis. In [JLCS10], a similar method 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

$$\frac{\left|p(s) - \texttt{CompDeCasteljau}(p, s)\right|}{\left|p(s)\right|} \leq \mathbf{u} + \operatorname{cond}(p, s) \times \mathcal{O}\left(\mathbf{u}^2\right). \tag{1.6}$$

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. Figure 1.4 shows this shift in relative error from `DeCasteljau` to `CompDeCasteljau`.

In [GLL09], 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

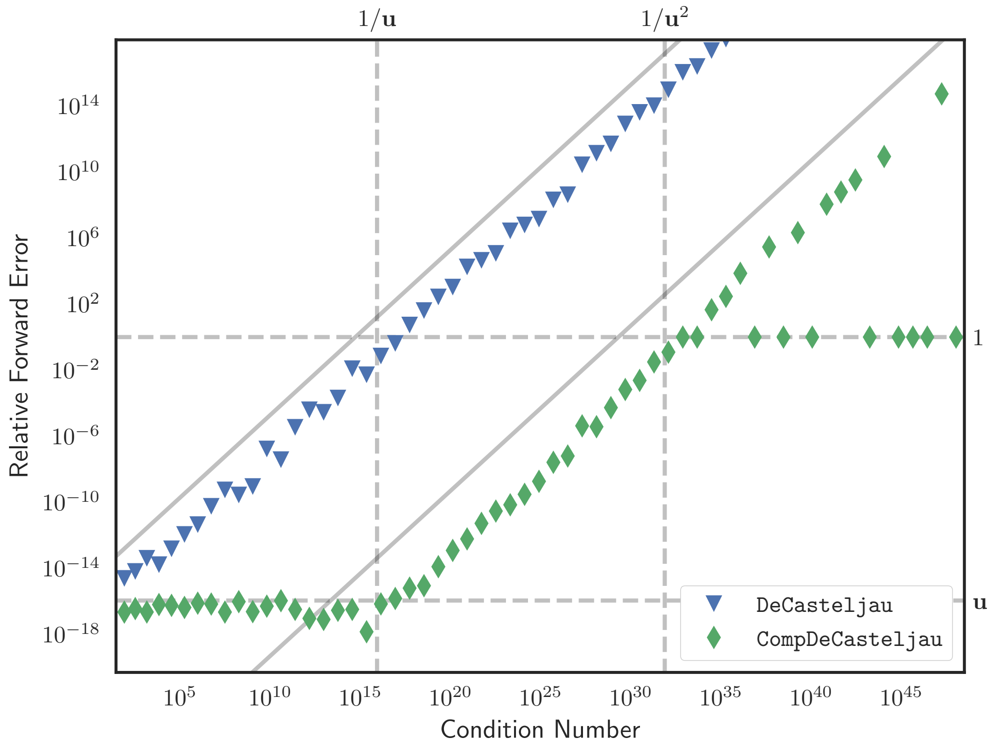

Figure 1.4: Evaluation of $p(s) = (s-1)\,(s-3/4)^7$ represented in Bernstein form.

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 (1.5), the accuracy of the compensated result (1.6) may be arbitrarily bad for ill-conditioned polynomial evaluations. For example, as the condition number grows in Figure 1.4, 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:

$$\frac{|p(s) - \texttt{CompDeCasteljauK}(p, s, K)|}{|p(s)|} \leq \mathbf{u} + \operatorname{cond}(p, s) \times \mathcal{O}\left(\mathbf{u}^K\right). \tag{1.7}$$

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 5.1, the compensated algorithm for polynomial evaluation from [JLCS10] is reviewed and notation is established for the expansion. In Section 5.2, the K-compensated algorithm is provided and a forward error analysis is performed. Finally, in Section 5.3 we perform two numerical experiments to give practical examples of the theoretical error bounds.

Chapter 2

Preliminaries

2.1 General Notation

We'll refer to $\mathbf{R}$ for the reals, $\mathcal{U}$ represents the unit triangle (or unit simplex) in $\mathbf{R}^2$: $\mathcal{U} = \{(s,t) \mid 0 \leq s, t, s+t \leq 1\}$. When dealing to sequences with multiple indices, e.g. $s_{m,n} = m+n$, we'll use bold notation to represent a multi-index: $\mathbf{i} = (m,n)$. The binomial coefficient $\binom{n}{k}$ is equal to $\frac{n!}{k!(n-k)!}$ and the trinomial coefficient $\binom{n}{i,j,k}$ is equal to $\frac{n!}{i!j!k!}$ when $i+j+k = n$. The notation δ_{ij} represents the Kronecker delta, a value which is 1 when $i = j$ and 0 otherwise.

2.2 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 $|\delta_1|, |\delta_2| \leq \mathbf{u}$. 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 [Hig02] for more details). Following [Hig02], we will use the following classic properties in error analysis.

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

2.3 Bézier Curves

A **Bézier curve** is a mapping from the unit interval that is determined by a set of control points $\{\mathbf{p}_j\}_{j=0}^n \subset \mathbf{R}^d$. For a parameter $s \in [0,1]$, there is a corresponding point on the curve:

$$b(s) = \sum_{j=0}^{n} \binom{n}{j} (1-s)^{n-j} s^j \mathbf{p}_j \in \mathbf{R}^d. \tag{2.2}$$

This is a combination of the control points weighted by each Bernstein basis function $B_{j,n}(s) = \binom{n}{j}(1-s)^{n-j}s^j$. Due to the binomial expansion $1 = (s + (1-s))^n = \sum_{j=0}^{n} B_{j,n}(s)$, a Bernstein basis function is in $[0,1]$ when s is as well. Due to this fact, the curve must be contained in the convex hull of it's control points.

2.3.1 de Casteljau Algorithm

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

Algorithm 2.1 *de Casteljau algorithm for polynomial evaluation.*

```
function result = DeCasteljau($b, s$)
    $n$ = 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
```

[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 $b_j^{(2)}$.

Theorem 2.1 ([MP99], 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]

$$|p(s) - \texttt{DeCasteljau}(p, s)| \leq \gamma_{3n} \sum_{j=0}^{n} |b_j| \, B_{j,n}(s). \tag{2.3}$$

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 [MP99], [FR87]):

$$\operatorname{cond}(p, s) = \frac{\widetilde{p}(s)}{|p(s)|}, \tag{2.4}$$

where $\widetilde{p}(s) \coloneqq \sum_{j=0}^{n} |b_j| \, B_{j,n}(s)$.

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

$$b^{(k)} = \left[\begin{array}{ccc} b_0^{(k)} & \cdots & b_k^{(k)} \end{array}\right]^T, \quad \widehat{b}^{(k)} = \left[\begin{array}{ccc} \widehat{b}_0^{(k)} & \cdots & \widehat{b}_k^{(k)} \end{array}\right]^T \tag{2.5}$$

and the reduction matrices:

$$U_k = U_k(s) = \left[\begin{array}{cccccc} 1-s & s & 0 & \cdots & \cdots & 0 \\ 0 & 1-s & s & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & 1-s & s \end{array}\right] \in \mathbf{R}^{k \times (k+1)}. \tag{2.6}$$

With this, we can express ([MP99]) the de Casteljau algorithm as

$$b^{(k)} = U_{k+1} b^{(k+1)} \Longrightarrow b^{(0)} = U_1 \cdots U_n b^{(n)}. \tag{2.7}$$

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

2.4 Bézier Triangles

A **Bézier triangle** ([Far86]) is a mapping from the unit triangle $\mathcal{U}$ and is determined by a control net $\{\mathbf{p}_{i,j,k}\}_{i+j+k=n} \subset \mathbf{R}^d$. For $(s, t) \in \mathcal{U}$ we can define barycentric weights $\lambda_1 = 1 - s - t, \lambda_2 = s, \lambda_3 = t$ so that

$$1 = (\lambda_1 + \lambda_2 + \lambda_3)^n = \sum_{\substack{i+j+k=n \\ i,j,k \geq 0}} \binom{n}{i, j, k} \lambda_1^i \lambda_2^j \lambda_3^k. \tag{2.8}$$

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

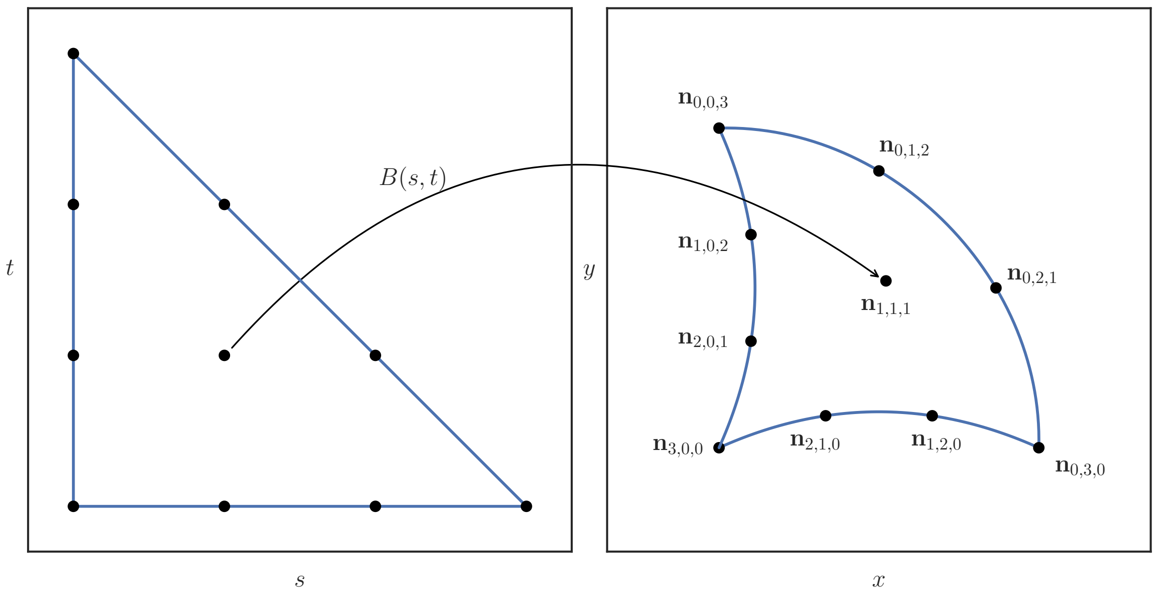

Figure 2.1: Cubic Bézier triangle

Using this we can similarly define a (triangular) Bernstein basis

$$B_{i,j,k}(s,t) = \binom{n}{i,j,k}(1-s-t)^i s^j t^k \tag{2.9}$$

that is in $[0,1]$ when (s,t) is in $\mathcal{U}$. Using this, we define points on the Bézier triangle as a convex combination of the control net:

$$b(s,t) = \sum_{i+j+k=n} \binom{n}{i,j,k} \lambda_1^i \lambda_2^j \lambda_3^k \mathbf{p}_{i,j,k} \in \mathbf{R}^d. \tag{2.10}$$

Rather than defining a Bézier triangle by the control net, it can also be uniquely determined by the image of a standard lattice of points in $\mathcal{U}$: $b\left(j/n, k/n\right) = \mathbf{n}_{i,j,k}$; we'll refer to these as **standard nodes**. Figure 2.1 shows these standard nodes for a cubic triangle. To see the correspondence, when $p = 1$ the standard nodes **are** the control net

$$b(s,t) = \lambda_1 \mathbf{n}_{1,0,0} + \lambda_2 \mathbf{n}_{0,1,0} + \lambda_3 \mathbf{n}_{0,0,1} \tag{2.11}$$

and when $p = 2$

$$\begin{aligned} b(s,t) = \lambda_1 \left(2\lambda_1 - 1\right) \mathbf{n}_{2,0,0} + \lambda_2 \left(2\lambda_2 - 1\right) \mathbf{n}_{0,2,0} + \lambda_3 \left(2\lambda_3 - 1\right) \mathbf{n}_{0,0,2} + \\ 4\lambda_1\lambda_2 \mathbf{n}_{1,1,0} + 4\lambda_2\lambda_3 \mathbf{n}_{0,1,1} + 4\lambda_3\lambda_1 \mathbf{n}_{1,0,1}. \end{aligned} \tag{2.12}$$

However, it's worth noting that the transformation between the control net and the standard nodes has condition number that grows exponentially with n (see [Far91]). This may make working with higher degree triangles prohibitively unstable.

2.5 Curved Elements

We define a curved mesh element $\mathcal{T}$ of degree p to be a Bézier triangle in $\mathbf{R}^2$ of the same degree. We refer to the component functions of $b(s,t)$ (the map that defined $\mathcal{T}$) as $x(s,t)$ and $y(s,t)$.

2.5.1 Shape Functions

When defining shape functions (i.e. a basis with geometric meaning) on a curved element there are (at least) two choices. When the degree of the shape functions is the same as the degree of the Bézier triangle, we say the element $\mathcal{T}$ is **isoparametric**. For the multi-index $\mathbf{i} = (i,j,k)$, we define $\mathbf{u_i} = (j/n, k/n)$ and the corresponding standard node $\mathbf{n_i} = b\left(\mathbf{u_i}\right)$. Given these points, two choices for shape functions present themselves:

- **Pre-Image Basis**: $\varphi_{\mathbf{j}}\left(\mathbf{n_i}\right) = \widehat{\varphi}_{\mathbf{j}}\left(\mathbf{u_i}\right) = \widehat{\varphi}_{\mathbf{j}}\left(b^{-1}\left(\mathbf{n_i}\right)\right)$ where $\widehat{\varphi}_{\mathbf{j}}$ is a canonical basis function on $\mathcal{U}$, i.e. $\widehat{\varphi}_{\mathbf{j}}$ a degree p bivariate polynomial and $\widehat{\varphi}_{\mathbf{j}}\left(\mathbf{u_i}\right) = \delta_{\mathbf{ij}}$
- **Global Coordinates Basis**: $\varphi_{\mathbf{j}}\left(\mathbf{n_i}\right) = \delta_{\mathbf{ij}}$, i.e. a canonical basis function on the standard nodes $\{\mathbf{n_i}\}$.

For example, consider a quadratic Bézier triangle:

$$b(s,t) = \left[\begin{array}{cc} 4(st+s+t) & 4(st+t+1) \end{array}\right]^T \tag{2.13}$$

$$\Longrightarrow \left[\begin{array}{cccccc} \mathbf{n}_{2,0,0} & \mathbf{n}_{1,1,0} & \mathbf{n}_{0,2,0} & \mathbf{n}_{1,0,1} & \mathbf{n}_{0,1,1} & \mathbf{n}_{0,0,2} \end{array}\right] = \left[\begin{array}{cccccc} 0 & 2 & 4 & 2 & 5 & 4 \\ 4 & 4 & 4 & 6 & 7 & 8 \end{array}\right]. \tag{2.14}$$

In the **Global Coordinates Basis**, we have

$$\varphi^G_{0,1,1}(x,y) = \frac{(y-4)(x-y+4)}{6}. \tag{2.15}$$

For the **Pre-Image Basis**, we need the inverse and the canonical basis

$$b^{-1}(x,y) = \left[\begin{array}{cc} \frac{x-y+4}{4} & \frac{y-4}{x-y+8} \end{array}\right] \quad \text{and} \quad \widehat{\varphi}_{0,1,1}(s,t) = 4st \tag{2.16}$$

and together they give

$$\varphi^P_{0,1,1}(x,y) = \frac{(y-4)(x-y+4)}{x-y+8}. \tag{2.17}$$

In general $\varphi^P_{\mathbf{j}}$ may not even be a rational bivariate function; due to composition with b^{-1} we can only guarantee that it is algebraic (i.e. it can be defined as the zero set of polynomials).

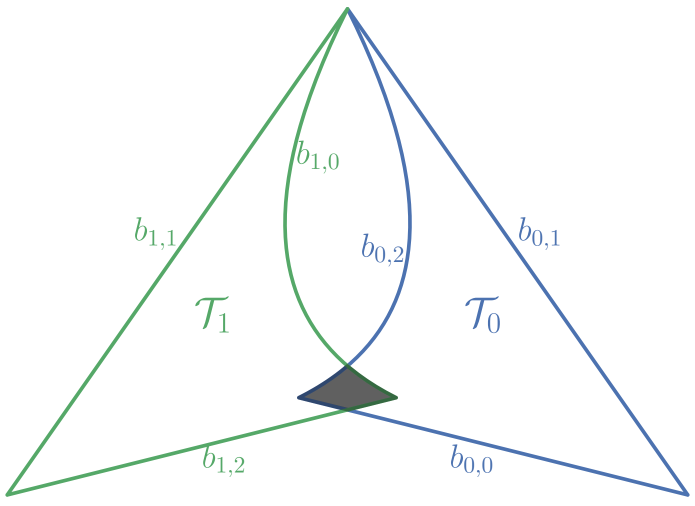

Figure 2.2: Intersection of Bézier triangles

2.5.2 Curved Polygons

When intersecting two curved elements, the resulting surface(s) will be defined by the boundary, alternating between edges of each element. For example, in Figure 2.2, a "curved quadrilateral" is formed when two Bézier triangles $\mathcal{T}_0$ and $\mathcal{T}_1$ are intersected.

A **curved polygon** is defined by a collection of Bézier curves in $\mathbf{R}^2$ that determine the boundary. In order to be a valid polygon, none of the boundary curves may cross and the ends of consecutive edge curves must meet. For our example in Figure 2.2, the triangles have boundaries formed by three Bézier curves: $\partial\mathcal{T}_0 = b_{0,0} \cup b_{0,1} \cup b_{0,2}$ and $\partial\mathcal{T}_1 = b_{1,0} \cup b_{1,1} \cup b_{1,2}$. The intersection $\mathcal{I}$ is defined by its boundary[3]:

$$\partial\mathcal{I} = b_{0,0}\left([0, 1/8]\right) \cup b_{1,2}\left([7/8, 1]\right) \cup b_{1,0}\left([0, 1/7]\right) \cup b_{0,2}\left([6/7, 1]\right). \tag{2.18}$$

Though an intersection can be described in terms of the Bézier triangles, the structure of the control net will be lost. The region will not in general be able to be described by a mapping from a simple space like $\mathcal{U}$.

2.6 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 $\mathbf{F}$. The error-free transformations used in this paper are the `TwoSum` algorithm by Knuth ([Knu97]) and `TwoProd` algorithm by Dekker ([Dek71], Section 5), respectively.

[3]Each specialization of a Bézier curve $b\left([a_1, a_2]\right)$ is itself a Bézier curve.

Theorem 2.1 ([ORO05], Theorem 3.4)**.** For $a, b \in \mathbf{F}$ and $P, \pi, S, \sigma \in \mathbf{F}$, `TwoSum` and `TwoProd` satisfy

$$[S, \sigma] = \texttt{TwoSum}(a, b), \quad S = \mathrm{fl}\left(a + b\right), S + \sigma = a + b, \sigma \leq \mathbf{u}\left|S\right|, \sigma \leq \mathbf{u}\left|a + b\right| \tag{2.19}$$

$$[P, \pi] = \texttt{TwoProd}(a, b), P = \mathrm{fl}\left(a \times b\right), P + \pi = a \times b, \pi \leq \mathbf{u}\left|P\right|, \pi \leq \mathbf{u}\left|a \times b\right|. \tag{2.20}$$

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

Chapter 3

Motivation

3.1 A Case for Characteristics

Content.

3.1.1 Lagrangian

Content.

Chapter 4

Bézier Intersection Problems

4.1 Intersecting Bézier Curves

The problem of intersecting two Bézier curves is a core building block for intersecting two Bézier triangles in $\mathbf{R}^2$. Since a curve is a degree one object, the intersections will either be a curve segment common to both curves (if they coincide) or a finite set of points. Many algorithsm have been described in the literature, both geometric ([SP86, SN90, KLS98]) and algebraic ([MD92]).

In the implementation for this paper, the Bézier subdivision algorithm is used. In the case of a transversal intersection (i.e. one where the tangents to each curve are not parallel and both are non-zero), this algorithm performs very well. However, when curves are tangent, a large number of (false) candidate intersections are detected and convergence of Newton's method slows once in a neighborhood of an actual intersection. Non-transversal intersections have infinite condition number, but transversal intersections with very high condition number can also cause convergence problems.

In the the Bézier subdivision algorithm, we first check if the bounding boxes for the curves are disjoint (Figure 4.1).

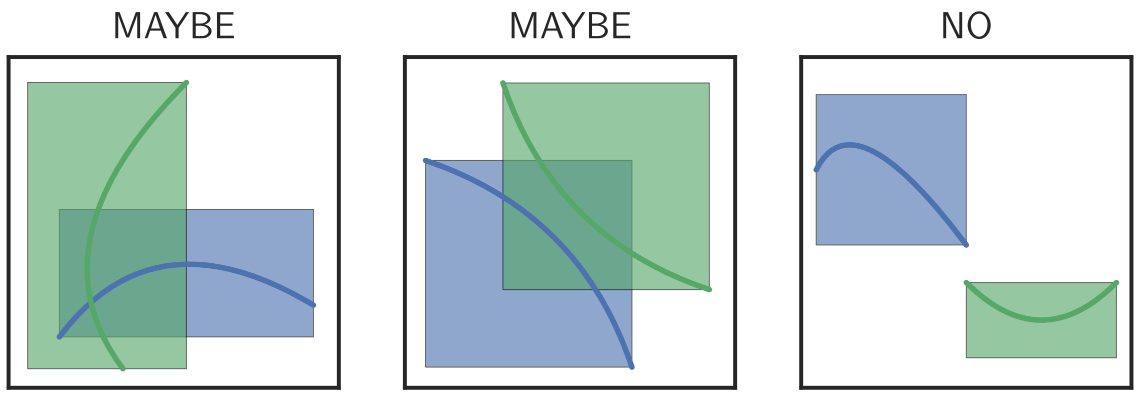

Figure 4.1: Bounding box check.

We use the bounding boxes rather than the convex hulls since they are easier to compute and the intersections of boxes are easier to check. If they are disjoint, the pair can be rejected. If not, each curve $\mathcal{C} = b\left([0, 1]\right)$ is split into two halves by splitting the unit interval: $b\left(\left[0, \frac{1}{2}\right]\right)$ and $b\left(\left[\frac{1}{2}, 1\right]\right)$ (Figure 4.2).

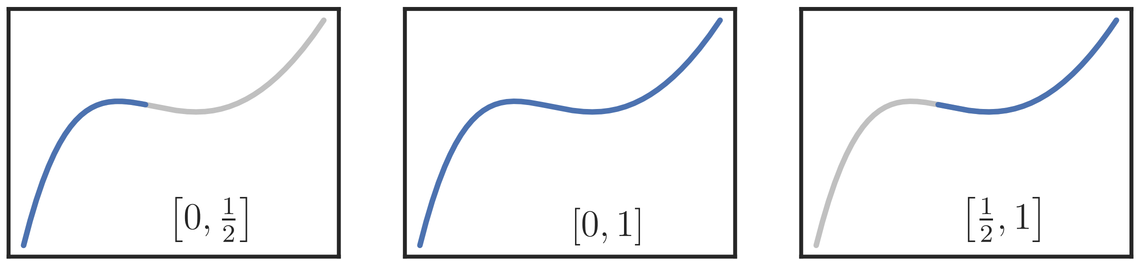

Figure 4.2: Bézier curve subdivision.

As the subdivision continues, some pairs of curve segments may be kept around that won't lead to an intersection (Figure 4.3).

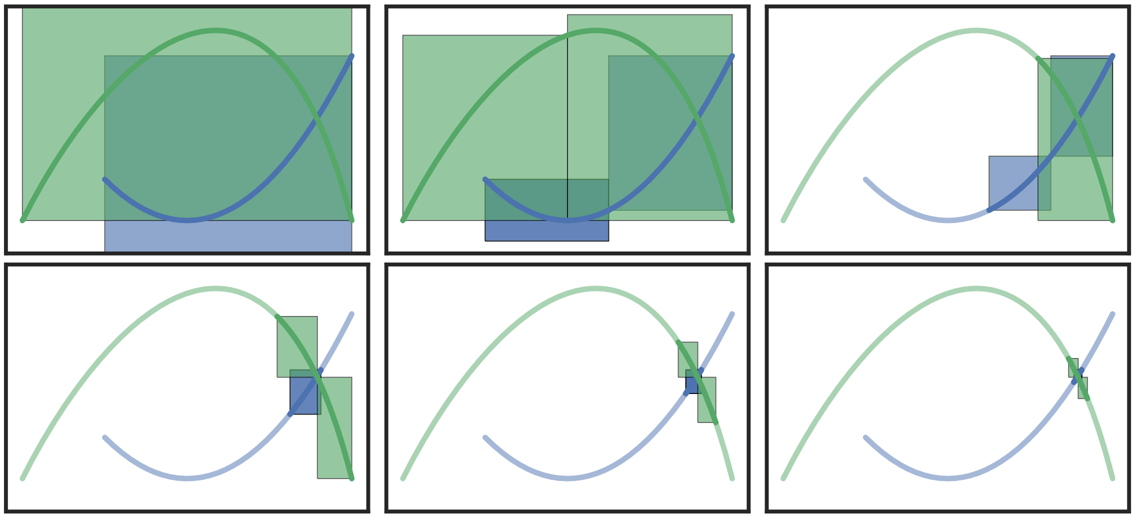

Figure 4.3: Bézier subdivision algorithm.

Once the curve segments are close to linear within a given tolerance (Figure 4.4), the process terminates.

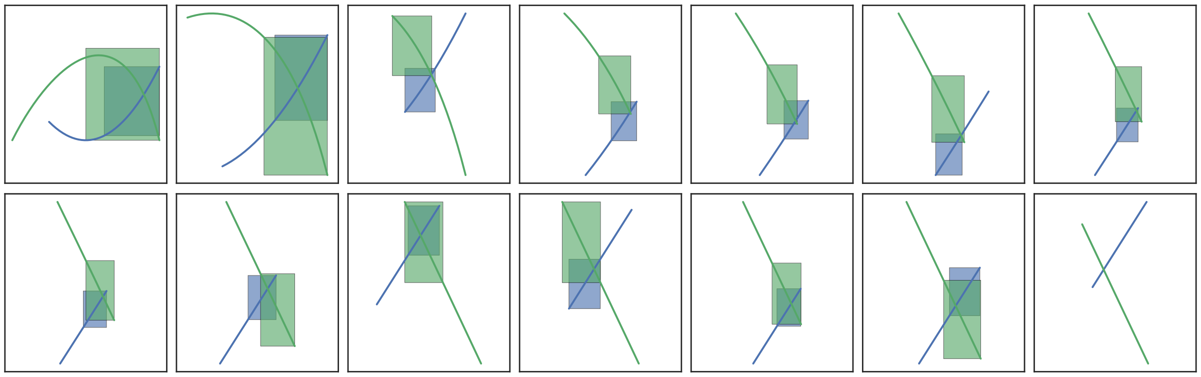

Figure 4.4: Subdividing until linear within tolerance.

Once both curve segments are linear (to tolerance), the intersection is approximated by intersecting the lines connecting the endpoints of each curve segment. This approximation is used as a starting point for Newton's method, to find a root of $F(s,t) = b_0(s) - b_1(t)$. Since $b_0(s), b_1(t) \in \mathbf{R}^2$ we have Jacobian $J = \left[\begin{array}{cc} b_0'(s) & -b_1'(t) \end{array}\right]$. With these, Newton's method is

$$\left[\begin{array}{cc} s_{n+1} & t_{n+1} \end{array}\right]^T = \left[\begin{array}{cc} s_n & t_n \end{array}\right]^T - J_n^{-1} F_n. \tag{4.1}$$

This also gives an indication why convergence issues occur at non-transveral intersections: they are exactly the intersections where the Jacobian is singular.

4.2 Intersecting Bézier Triangles

The chief difficulty in intersecting two surfaces is intersecting their edges. Though this is just a part of the overall algorithm, it proved to be the most difficult to implement.

To intersect two Bézier surfaces, we first find all points where the edges intersect (Figure 4.5).

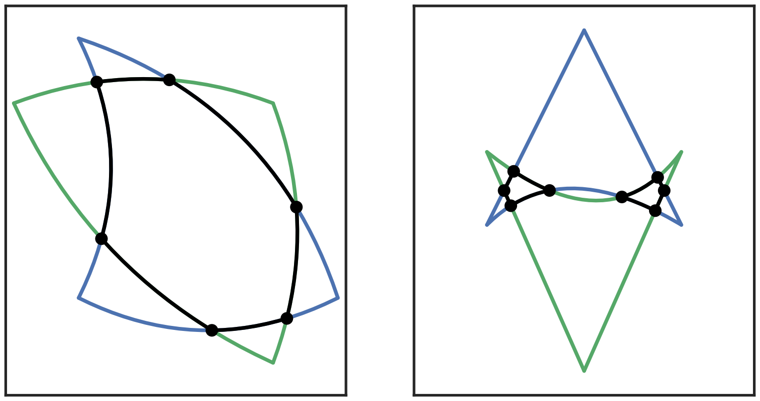

Figure 4.5: Edge intersections during Bézier triangle intersection.

To determine the curve segments that bound the curved polygon region(s) (see 2.5.2) of intersection, we not only need to keep track of the coordinates of intersection, we also need to keep note of **which** edges the intersection occurred on and the parameters along each curve. With this information, we can classify each point of intersection according to which of the two curves forms the boundary of the curved polygon (Figure 4.6). Using the right-hand rule we can compare the tangent vectors on each curve to determine which one is on the interior.

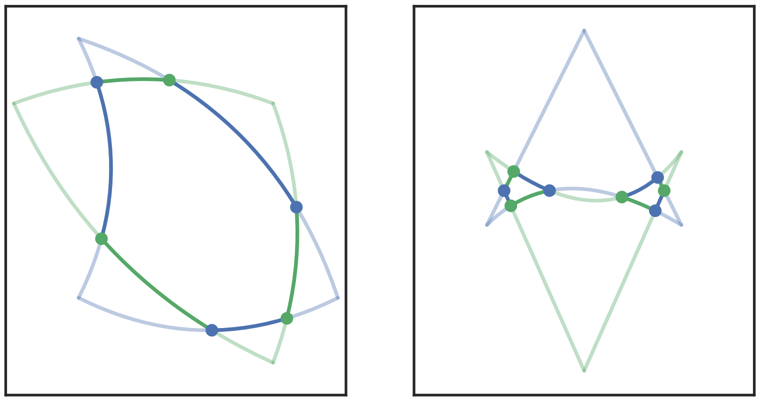

Figure 4.6: Classified intersections during Bézier triangle intersection.

This classification becomes more difficult when the curves are tangent at an intersection, when the intersection occurs at a corner of one of the surfaces or when two intersecting edges are coincident on the same algebraic curve (Figure 4.7).

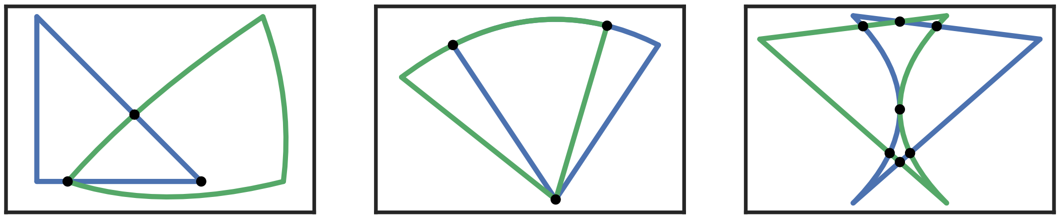

Figure 4.7: Bézier triangle intersection difficulties.

In the case of tangency, the intersection is non-transversal, hence has infinite condition number. In the case of coincident curves, there are infinitely many intersections (along the segment when the curves coincide) so the subdivision process breaks down.

4.2.0.1 Example

Consider two Bézier surfaces (Figure 4.8)

$$b_0(s,t) = \begin{bmatrix} 8s \\ 8t \end{bmatrix} \qquad b_1(s,t) = \begin{bmatrix} 2(6s+t-1) \\ 2(8s^2+8st-8s+3t+2) \end{bmatrix} \tag{4.2}$$

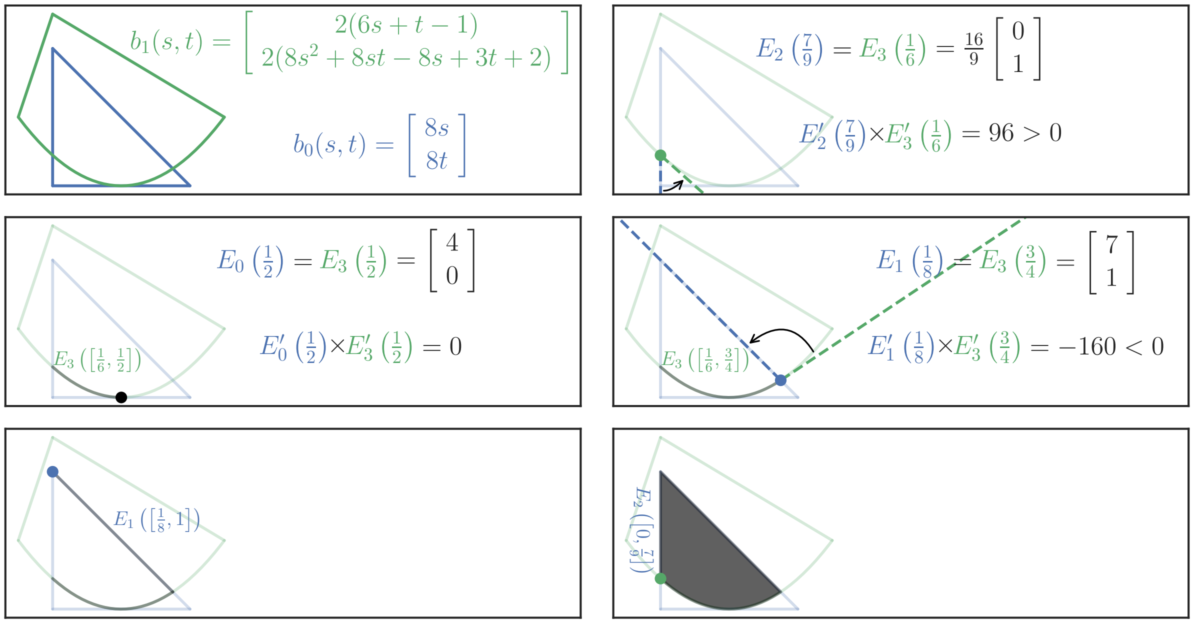
$b_1(s,t) = \begin{bmatrix} 2(6s+t-1) \\ 2(8s^2+8st-8s+3t+2) \end{bmatrix}$
$b_0(s,t) = \begin{bmatrix} 8s \\ 8t \end{bmatrix}$
$E_2\left(\frac{7}{9}\right) = E_3\left(\frac{1}{6}\right) = \frac{16}{9}\begin{bmatrix} 0 \\ 1 \end{bmatrix}$
$E_2'\left(\frac{7}{9}\right) \times E_3'\left(\frac{1}{6}\right) = 96 > 0$
$E_0\left(\frac{1}{2}\right) = E_3\left(\frac{1}{2}\right) = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$
$E_0'\left(\frac{1}{2}\right) \times E_3'\left(\frac{1}{2}\right) = 0$
$E_3\left(\left[\frac{1}{6}, \frac{1}{2}\right]\right)$
$E_1\left(\frac{1}{8}\right) = E_3\left(\frac{3}{4}\right) = \begin{bmatrix} 7 \\ 1 \end{bmatrix}$
$E_1'\left(\frac{1}{8}\right) \times E_3'\left(\frac{3}{4}\right) = -160 < 0$
$E_3\left(\left[\frac{1}{6}, \frac{3}{4}\right]\right)$
$E_1\left(\left[\frac{1}{8}, 1\right]\right)$
$E_2\left(\left[0, \frac{7}{9}\right]\right)$

Figure 4.8: Surface Intersection Example

In the **first step** we find all intersections of the edge curves

$$\begin{aligned} E_0(r) = \left[\begin{array}{c} 8r \\ 0 \end{array}\right], E_1(r) = \left[\begin{array}{c} 8(1-r) \\ 8r \end{array}\right], E_2(r) = \left[\begin{array}{c} 0 \\ 8(1-r) \end{array}\right], \\ E_3(r) = \left[\begin{array}{c} 2(6r-1) \\ 4(2r-1)^2 \end{array}\right], E_4(r) = \left[\begin{array}{c} 10(1-r) \\ 2(3r+2) \end{array}\right], E_5(r) = \left[\begin{array}{c} -2r \\ 2(5-3r) \end{array}\right]. \end{aligned} \tag{4.3}$$

We find three intersections and we classify each of them by comparing the tangent vectors

$$I_1 : E_2\left(\frac{7}{9}\right) = E_3\left(\frac{1}{6}\right) = \frac{16}{9}\left[\begin{array}{c} 0 \\ 1 \end{array}\right] \Longrightarrow E_2'\left(\frac{7}{9}\right) \times E_3'\left(\frac{1}{6}\right) = 96 \tag{4.4}$$

$$I_2 : E_0\left(\frac{1}{2}\right) = E_3\left(\frac{1}{2}\right) = \left[\begin{array}{c} 4 \\ 0 \end{array}\right] \Longrightarrow E_0'\left(\frac{1}{2}\right) \times E_3'\left(\frac{1}{2}\right) = 0 \tag{4.5}$$

$$I_3 : E_1\left(\frac{1}{8}\right) = E_3\left(\frac{3}{4}\right) = \left[\begin{array}{c} 7 \\ 1 \end{array}\right] \Longrightarrow E_1'\left(\frac{1}{8}\right) \times E_3'\left(\frac{3}{4}\right) = -160. \tag{4.6}$$

From here, we construct our curved polygon intersection by drawing from our list of intersections until none remain.

- First consider I_1. Since $E_2' \times E_3' > 0$ at this point, then we consider the curve E_3 to be *interior.*
- After classification, we move along E_3 until we encounter another intersection: I_2
- I_2 is a point of tangency since $E_0'\left(\frac{1}{2}\right) \times E_3'\left(\frac{1}{2}\right) = 0$. Since a tangency has no impact on the underlying intersection geometry, we ignore it and keep moving.
- Continuing to move along E_3, we encounter another intersection: I_3. Since $E_1' \times E_3' < 0$ at this point, we consider the curve E_1 to be *interior* at the intersection. Thus we stop moving along E_3 and we have our first curved segment: $E_3\left(\left[\frac{1}{6}, \frac{3}{4}\right]\right)$
- Finding no other intersections on E_1 we continue until the end of the edge. Now our (ordered) curved segments are:

$$E_3\left(\left[\frac{1}{6}, \frac{3}{4}\right]\right) \longrightarrow E_1\left(\left[\frac{1}{8}, 1\right]\right). \tag{4.7}$$

- Next we stay at the corner and switch to the next curve E_2, moving along that curve until we hit the next intersecton I_1. Now our (ordered) curved segments are:

$$E_3\left(\left[\frac{1}{6}, \frac{3}{4}\right]\right) \longrightarrow E_1\left(\left[\frac{1}{8}, 1\right]\right) \longrightarrow E_2\left(\left[0, \frac{7}{9}\right]\right). \tag{4.8}$$

Since we are now back where we started (at I_1) the process stops

We represent the boundary of the curved polygon as Bézier curves, so to complete the process we reparameterize ([Far01, Ch. 5.4]) each curve onto the relevant interval. For example, E_3 has control points $p_0 = \left[\begin{array}{c} -2 \\ 4 \end{array}\right]$, $p_1 = \left[\begin{array}{c} 4 \\ -4 \end{array}\right]$, $p_2 = \left[\begin{array}{c} 10 \\ 4 \end{array}\right]$ and we reparameterize on $\left[\frac{1}{6}, \frac{3}{4}\right]$ to control points

$$q_0 = E_3\left(\frac{1}{6}\right) = \frac{16}{9}\left[\begin{array}{c} 0 \\ 1 \end{array}\right] \tag{4.9}$$

$$q_1 = \left(1 - \frac{1}{6}\right)\left[\left(1 - \frac{3}{4}\right) p_0 + \frac{3}{4} p_1\right] + \frac{1}{6}\left[\left(1 - \frac{3}{4}\right) p_1 + \frac{3}{4} p_2\right] = \frac{1}{6}\left[\begin{array}{c} 21 \\ -8 \end{array}\right] \tag{4.10}$$

$$q_2 = E_3\left(\frac{3}{4}\right) = \left[\begin{array}{c} 7 \\ 1 \end{array}\right]. \tag{4.11}$$

4.3 Bézier Triangle Inverse

The problem of determining the parameters (s, t) given a point (x, y) in a Bézier triangle...

Chapter 5

Compensated de Casteljau algorithm in K times the working precision

5.1 Compensated de Casteljau

In this section we review the compensated de Casteljau algorithm from [JLCS10]. In order to track the local errors at each update step, we use four EFTs:

$$[\widehat{r}, \rho] = \mathtt{TwoSum}(1, -s) \tag{5.1}$$

$$[P_1, \pi_1] = \mathtt{TwoProd}\left(\widehat{r}, \widehat{b}_j^{(k+1)}\right) \tag{5.2}$$

$$[P_2, \pi_2] = \mathtt{TwoProd}\left(s, \widehat{b}_{j+1}^{(k+1)}\right) \tag{5.3}$$

$$\left[\widehat{b}_j^{(k)}, \sigma_3\right] = \mathtt{TwoSum}(P_1, P_2) \tag{5.4}$$

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

$$\ell_{1,j}^{(k)} = \pi_1 + \pi_2 + \sigma_3 + \rho \cdot \widehat{b}_j^{(k+1)} \tag{5.5}$$

$$(1 - s) \cdot \widehat{b}_j^{(k+1)} + s \cdot \widehat{b}_{j+1}^{(k+1)} = \widehat{b}_j^{(k)} + \ell_{1,j}^{(k)}. \tag{5.6}$$

By defining the global errors at each step

$$\partial b_j^{(k)} = b_j^{(k)} - \widehat{b}_j^{(k)} \tag{5.7}$$

we can see (Figure 5.1) that the local errors accumulate in $\partial b^{(k)}$:

$$\partial b_j^{(k)} = (1 - s) \cdot \partial b_j^{(k+1)} + s \cdot \partial b_{j+1}^{(k+1)} + \ell_{1,j}^{(k)}. \tag{5.8}$$

When computed in exact arithmetic

$$p(s) = \widehat{b}_0^{(0)} + \partial b_0^{(0)} \tag{5.9}$$

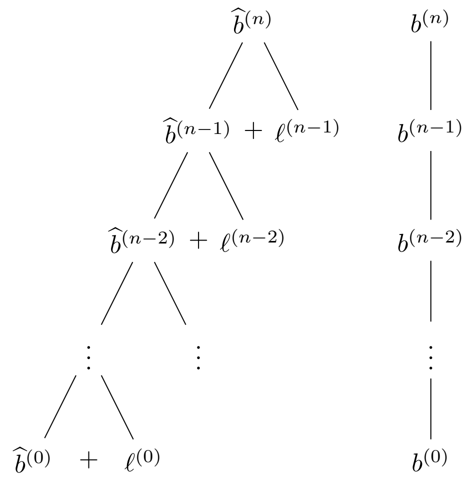
$\widehat{b}^{(n)}$
$b^{(n)}$
$\widehat{b}^{(n-1)} + \ell^{(n-1)}$
$b^{(n-1)}$
$\widehat{b}^{(n-2)} + \ell^{(n-2)}$
$b^{(n-2)}$
$\widehat{b}^{(0)} + \ell^{(0)}$
$b^{(0)}$

Figure 5.1: Local round-off errors

and by using (5.8), 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:

Algorithm 5.1 *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
```

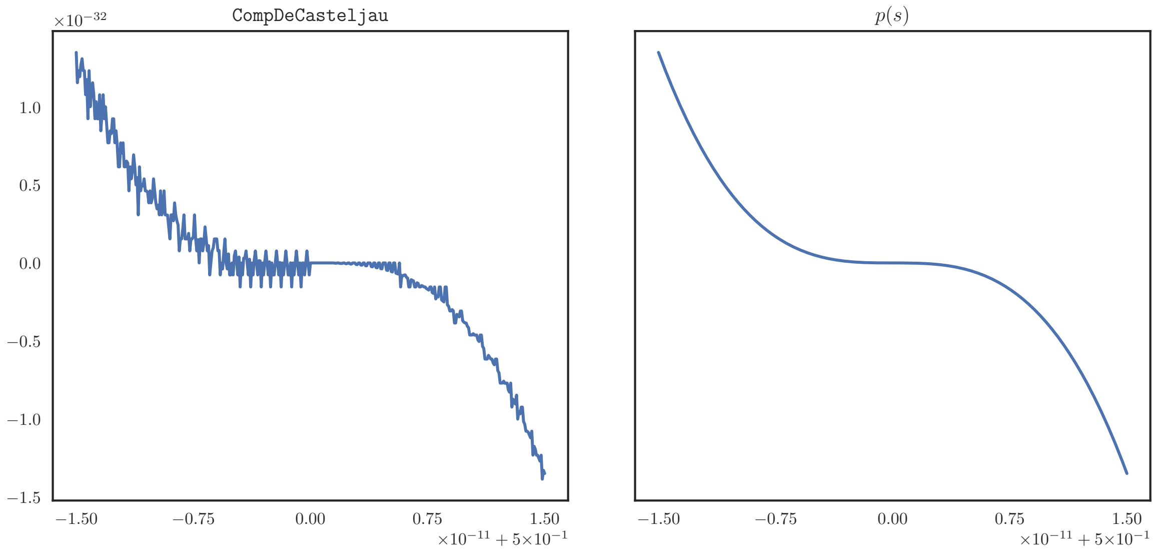

Figure 5.2: The compensated de Castelјau method starts to lose accuracy for $p(s) = (2s-1)^3(s-1)$ in the neighborhood of its multiple root $1/2$.

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

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 5.1) 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 5.1 ([JLCS10], Theorem 5)**.** If no underflow occurs, $n \geq 2$ and $s \in [0, 1]$

$$\frac{\left|p(s) - \texttt{CompDeCasteljau}(p, s)\right|}{|p(s)|} \leq \mathbf{u} + 2\gamma_{3n}^2 \operatorname{cond}(p, s)\,. \tag{5.10}$$

Unfortunately, Figure 5.2 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 $\left|s - \frac{1}{2}\right| \leq \frac{3}{2} \cdot 10^{-11}$ — evaluates to exactly 0 when it should be $\mathcal{O}\left(\mathbf{u}^3\right)$. As shown in Table 5.1, the breakdown occurs because $\widehat{b}_0^{(0)} = -\partial\widehat{b}_0^{(0)} = \mathbf{u}/16$.

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	$0.0625\mathbf{u}$	$-0.0625\mathbf{u}$	$-4(1001\mathbf{u})^3 + 8(1001\mathbf{u})^4$

Table 5.1: Terms computed by `CompDeCasteljau` when evaluating $p(s) = (2s-1)^3(s-1)$ at the point $s = \frac{1}{2} + 1001\mathbf{u}$

5.2 K-Compensated de Casteljau

5.2.1 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:

$$b_j^{(k)} = \widehat{b}_j^{(k)} + \partial b_j^{(k)} \tag{5.11}$$

$$\partial b_j^{(k)} = \widehat{\partial b}_j^{(k)} + \partial^2 b_j^{(k)} \tag{5.12}$$

$$\partial^2 b_j^{(k)} = \widehat{\partial^2 b}_j^{(k)} + \partial^3 b_j^{(k)} \tag{5.13}$$

$$\vdots$$

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:

$$(1-s)\cdot \widehat{b}_j^{(k+1)} + s\cdot \widehat{b}_{j+1}^{(k+1)} = \widehat{b}_j^{(k)} + \ell_{1,j}^{(k)} \tag{5.14}$$

$$(1-s)\cdot \widehat{\partial b}_j^{(k+1)} + s\cdot \widehat{\partial b}_{j+1}^{(k+1)} + \ell_{1,j}^{(k)} = \widehat{\partial b}_j^{(k)} + \ell_{2,j}^{(k)} \tag{5.15}$$

$$(1-s)\cdot \widehat{\partial^2 b}_j^{(k+1)} + s\cdot \widehat{\partial^2 b}_{j+1}^{(k+1)} + \ell_{2,j}^{(k)} = \widehat{\partial^2 b}_j^{(k)} + \ell_{3,j}^{(k)} \tag{5.16}$$

$$\vdots$$

In `CompDeCasteljau` (Algorithm 5.1), 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

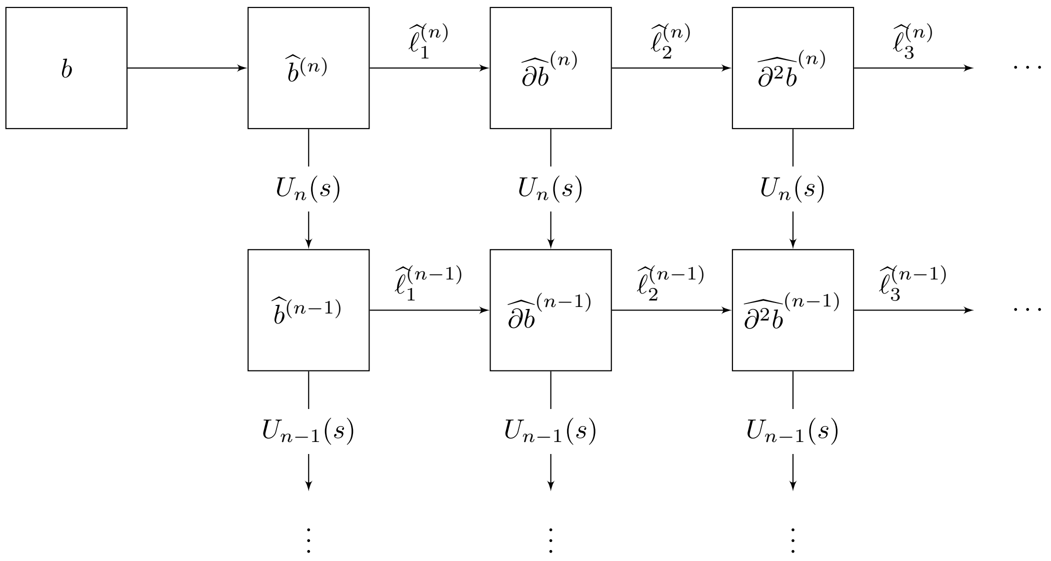

Figure 5.3: Filtering errors

error). In order to obtain results that are as accurate as if computed in K times the working precision, we must continue filtering (see Figure 5.3) 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 $\widehat{\ell}_{F,j}^{(k)}$ from the terms in $\ell_{F,j}^{(k)}$
- an error for the "missing" $\rho \cdot \widehat{\partial^F b}_j^{(k+1)}$ in $(1-s) \cdot \widehat{\partial^F b}_j^{(k+1)}$
- an error from the product $\widehat{r} \otimes \widehat{\partial^F b}_j^{(k+1)}$
- an error from the product $s \otimes \widehat{\partial^F b}_{j+1}^{(k+1)}$
- two errors from the two $\oplus$ when combining the three terms in $\widehat{\ell}_{F,j}^{(k)} \oplus \left(s \otimes \widehat{\partial^F b}_{j+1}^{(k+1)}\right) \oplus \left(\widehat{r} \otimes \widehat{\partial^F b}_j^{(k+1)}\right)$

For example, in (5.5):

$$\ell_{1,j}^{(k)} = \underbrace{\pi_1}_{P_1 = \widehat{r} \otimes \widehat{b}_j^{(k+1)}} + \underbrace{\pi_2}_{P_2 = s \otimes \widehat{b}_{j+1}^{(k+1)}} + \underbrace{\sigma_3}_{P_1 \oplus P_2} + \underbrace{\rho \cdot \widehat{b}_j^{(k+1)}}_{(1-s)\widehat{b}_j^{(k+1)}} \tag{5.17}$$

After each stage, we'll always have

$$\ell_{F,j}^{(k)} = e_1 + \cdots + e_{5F-2} + \rho \cdot \widehat{\partial^{F-1} b}_j^{(k+1)} \tag{5.18}$$

where the terms $e_1, \ldots, e_{5F-2}$ come from using `TwoSum` and `TwoProd` when computing $\widehat{\partial^{F-1} b}_j^{(k)}$ and the ρ term comes from the round-off in $1 \ominus s$ when multiplying $(1-s)$ by $\widehat{\partial^{F-1} b}_j^{(k+1)}$. 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 5.2 *EFT for computing the local error.*

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

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

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

With this EFT in place[1], 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.

Algorithm 5.3 *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
```

[1]And the related `LocalError` in Algorithm A.7

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

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

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

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

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

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

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

Noting that $\ell_{F,j}$ contains $5F-1$ terms, one can show that `CompDeCasteljauK` (Algorithm 5.3) requires

$$(15K^2 + 11K - 34)T_n + 6K^2 - 11K + 11 = \mathcal{O}\left(n^2K^2\right) \tag{5.19}$$

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.

5.2.2 Error bound for polynomial evaluation

Theorem 5.1 ([ORO05], 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:

$$\left| \texttt{SumK}(v, K) - \sum_{j=1}^{n} v_j \right| \leq \left(\mathbf{u} + 3\gamma_{n-1}^2\right) \left| \sum_{j=1}^{n} v_j \right| + \gamma_{2n-2}^K \sum_{j=1}^{n} |v_j| \,. \tag{5.20}$$

Lemma 5.1 ([JLCS10], Theorem 4)**.** The second order error $\partial^2 b_0^{(0)}$ satisfies[2]

$$\left| \partial b_0^{(0)} - \widehat{\partial b}_0^{(0)} \right| = \left| \partial^2 b_0^{(0)} \right| \leq 2\gamma_{3n+2}\gamma_{3(n-1)}\widetilde{p}(s). \tag{5.21}$$

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

$$\widetilde{\ell}_{F,j} \coloneqq |e_1| + \cdots + |e_{5F-2}| + \left| \rho \cdot \widehat{\partial^{F-1} b_j}^{(k+1)} \right|. \tag{5.22}$$

Lemma 5.2. The local error bounds $\widetilde{\ell}_{F,j}$ satisfy:

$$\widetilde{\ell}_{1,j}^{(k)} \leq \gamma_3 \left((1 - s) \left| \widehat{b}_j^{(k+1)} \right| + s \left| \widehat{b}_{j+1}^{(k+1)} \right| \right) \tag{5.23}$$

$$\widetilde{\ell}_{F+1,j}^{(k)} \leq \gamma_3 \left((1 - s) \left| \widehat{\partial^F b_j}^{(k+1)} \right| + s \left| \widehat{\partial^F b_{j+1}}^{(k+1)} \right| \right) + \gamma_{5F} \cdot \widetilde{\ell}_{F,j}^{(k)} \text{ for } F \geq 1. \tag{5.24}$$

As we'll see soon (Lemma 5.4), putting a bound on sums of the form $\sum_{j=0}^{k} \ell_{F,j}^{(k)} B_{j,k}(s)$ will be useful to get an overall bound on the relative error for `CompDeCasteljauK`, so we define $L_{F,k} \coloneqq \sum_{j=0}^{k} \ell_{F,j}^{(k)} B_{j,k}(s)$.

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

Lemma 5.3. For $s \in [0, 1]$, the Bernstein-type error sum defined above satisfies the following bounds:

$$L_{F,n-k} \leq \left[\left(3^F \binom{k}{F-1} + \mathcal{O}\left(k^{F-1}\right)\right) \mathbf{u}^F + \mathcal{O}\left(\mathbf{u}^{F+1}\right)\right] \cdot \widetilde{p}(s) \tag{5.25}$$

$$\sum_{k=0}^{n-1} \gamma_{3k+5F} L_{F,k} \leq \left[\left(3^{F+1} \binom{n}{F+1} + \mathcal{O}\left(n^F\right)\right) \mathbf{u}^{F+1} + \mathcal{O}\left(\mathbf{u}^{F+2}\right)\right] \cdot \widetilde{p}(s). \tag{5.26}$$

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 Lemma 5.2 and Lemma 5.3.

Lemma 5.4. The K order error $\partial^K b_0^{(0)}$ satisfies

$$\left|\partial^{K-1} b_0^{(0)} - \widehat{\partial^{K-1} b_0}^{(0)}\right| = \left|\partial^K b_0^{(0)}\right| \leq \left[\left(3^K \binom{n}{K} + \mathcal{O}\left(n^{K-1}\right)\right) \mathbf{u}^K + \mathcal{O}\left(\mathbf{u}^{K+1}\right)\right] \cdot \widetilde{p}(s). \tag{5.27}$$

Proof. As in (2.7), we can express the compensated de Casteljau algorithm as

$$\partial^F b^{(k)} = U_{k+1} \partial^F b^{(k+1)} + \ell_F^{(k)} \Longrightarrow \partial^F b^{(0)} = \sum_{k=0}^{n-1} U_1 \cdots U_k \ell_F^{(k)} = \sum_{k=0}^{n-1} \left[\sum_{j=0}^{k} \ell_{F,j}^{(k)} B_{j,k}(s)\right]. \tag{5.28}$$

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:

$$\begin{aligned}
\widehat{\partial^F b_j}^{(k)} &= \widehat{\ell}_{F,j}^{(k)} \oplus \left(s \otimes \widehat{\partial^F b_{j+1}}^{(k+1)}\right) \oplus \left(\widehat{r} \otimes \widehat{\partial^F b_j}^{(k+1)}\right) && (5.29)\\
&= (1-s) \cdot \widehat{\partial^F b_j}^{(k+1)} (1+\theta_3) + s \cdot \widehat{\partial^F b_{j+1}}^{(k+1)} (1+\theta_3) + \widehat{\ell}_{F,j}^{(k)} (1+\theta_2) && (5.30)\\
\Longrightarrow \widehat{\partial^F b}^{(k)} &= U_{k+1} \widehat{\partial^F b}^{(k+1)} (1+\theta_3) + \widehat{\ell}_F^{(k)} (1+\theta_2) && (5.31)\\
\Longrightarrow \widehat{\partial^F b}^{(0)} &= \sum_{k=0}^{n-1} U_1 \cdots U_k \widehat{\ell}_F^{(k)} (1+\theta_{3k+2}) = \sum_{k=0}^{n-1} \left[\sum_{j=0}^{k} \widehat{\ell}_{F,j}^{(k)} (1+\theta_{3k+2}) B_{j,k}(s)\right]. && (5.32)
\end{aligned}$$

Since

$$\partial^{F+1} b_0^{(0)} = \partial^F b_0^{(0)} - \widehat{\partial^F b_0}^{(0)} = \sum_{k=0}^{n-1} \sum_{j=0}^{k} \left(\ell_{F,j}^{(k)} - \widehat{\ell}_{F,j}^{(k)} (1+\theta_{3k+2})\right) B_{j,k}(s) \tag{5.33}$$

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

$$\begin{aligned}
\widehat{\ell}_{F,j}^{(k)} &= e_1 \oplus \cdots \oplus e_{5F-2} \oplus \left(\rho \otimes \widehat{\partial^{F-1} b_j}^{(k+1)}\right) && (5.34)\\
&= e_1 \left(1+\theta_{5F-2}\right) + \cdots + e_{5F-2} \left(1+\theta_2\right) + \rho \cdot \widehat{\partial^{F-1} b_j}^{(k+1)} \left(1+\theta_2\right) && (5.35)
\end{aligned}$$

we see that

$$\left|\ell_{F,j}^{(k)} - \widetilde{\ell}_{F,j}^{(k)}(1+\theta_{3k+2})\right| \leq \gamma_{3k+5F}\cdot \widetilde{\ell}_{F,j}^{(k)} \Longrightarrow \left|\partial^{F+1}b_0^{(0)}\right| \leq \sum_{k=0}^{n-1}\gamma_{3k+5F}\sum_{j=0}^{k}\widetilde{\ell}_{F,j}^{(k)}B_{j,k}(s). \tag{5.36}$$

Applying (5.26) directly gives

$$\left|\partial^{F+1}b_0^{(0)}\right| \leq \left[\left(3^{F+1}\binom{n}{F+1} + \mathcal{O}\left(n^F\right)\right)\mathbf{u}^{F+1} + \mathcal{O}\left(\mathbf{u}^{F+2}\right)\right]\cdot \widetilde{p}(s). \tag{5.37}$$

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

Theorem 5.2. If no underflow occurs, $n \geq 2$ and $s \in [0,1]$

$$\frac{|p(s) - \texttt{CompDeCasteljau}(p,s,K)|}{|p(s)|} \leq \left[\mathbf{u} + \mathcal{O}\left(\mathbf{u}^2\right)\right] + \left[\left(3^K\binom{n}{K} + \mathcal{O}\left(n^{K-1}\right)\right)\mathbf{u}^K + \mathcal{O}\left(\mathbf{u}^{K+1}\right)\right]\operatorname{cond}(p,s). \tag{5.38}$$

Proof. Since

$$\texttt{CompDeCasteljau}(p,s,K) = \texttt{SumK}\left(\left[\widehat{b}_0^{(0)},\ldots,\widehat{\partial^{K-1}b_0}^{(0)}\right], K\right), \tag{5.39}$$

applying Theorem 5.1 tells us that

$$\left|\texttt{CompDeCasteljau}(p,s,K) - \sum_{F=0}^{K-1}\widehat{\partial^F b_0}^{(0)}\right| \leq \left(\mathbf{u} + 3\gamma_{n-1}^2\right)\left|\sum_{F=0}^{K-1}\widehat{\partial^F b_0}^{(0)}\right| + \gamma_{2n-2}^K\sum_{F=0}^{K-1}\left|\widehat{\partial^F b_0}^{(0)}\right|. \tag{5.40}$$

Since

$$p(s) = b_0^{(0)} = \widehat{b}_0^{(0)} + \partial b_0^{(0)} = \cdots = \widehat{b}_0^{(0)} + \widehat{\partial b}_0^{(0)} + \cdots + \widehat{\partial^{K-1}b_0}^{(0)} + \partial^K b_0^{(0)} \tag{5.41}$$

we have

$$\left|\sum_{F=0}^{K-1}\widehat{\partial^F b_0}^{(0)}\right| \leq |p(s)| + \left|\partial^K b_0^{(0)}\right| \quad \text{and} \tag{5.42}$$

$$|\texttt{CompDeCasteljau}(p,s,K) - p(s)| \leq \left|\texttt{CompDeCasteljau}(p,s,K) - \sum_{F=0}^{K-1}\widehat{\partial^F b_0}^{(0)}\right| + \left|\partial^K b_0^{(0)}\right|. \tag{5.43}$$

Due to Lemma 5.4, $\partial^F b_0^{(0)} = \mathcal{O}\left(\mathbf{u}^F\right)\widetilde{p}(s)$, hence

$$\left(\mathbf{u} + 3\gamma_{n-1}^2\right)\left|\sum_{F=0}^{K-1}\widehat{\partial^F b_0}^{(0)}\right| \leq \left[\mathbf{u} + \mathcal{O}\left(\mathbf{u}^2\right)\right]|p(s)| + \mathcal{O}\left(\mathbf{u}^{K+1}\right)\widetilde{p}(s) \tag{5.44}$$

$$\gamma_{2n-2}^{K} \sum_{F=0}^{K-1} \left|\widehat{\partial^F b}_0^{(0)}\right| \leq \gamma_{2n-2}^{K} \left|\widehat{b}_0^{(0)}\right| + \mathcal{O}\left(\mathbf{u}^{K+1}\right) \widetilde{p}(s) \tag{5.45}$$

$$\leq \gamma_{2n-2}^{K} \left[|p(s)| + \mathcal{O}\left(\mathbf{u}\right) \widetilde{p}(s)\right] + \mathcal{O}\left(\mathbf{u}^{K+1}\right) \widetilde{p}(s). \tag{5.46}$$

Combining this with (5.40) and (5.43), we see

$$|\texttt{CompDeCasteljau}(p, s, K) - p(s)| \tag{5.47}$$

$$\leq \left[\mathbf{u} + \mathcal{O}\left(\mathbf{u}^2\right)\right] |p(s)| + \left|\partial^K b_0^{(0)}\right| + \mathcal{O}\left(\mathbf{u}^{K+1}\right) \widetilde{p}(s) \tag{5.48}$$

$$\leq \left[\mathbf{u} + \mathcal{O}\left(\mathbf{u}^2\right)\right] |p(s)| + \left[\left(3^K \binom{n}{K} + \mathcal{O}\left(n^{K-1}\right)\right) \mathbf{u}^K + \mathcal{O}\left(\mathbf{u}^{K+1}\right)\right] \widetilde{p}(s). \tag{5.49}$$

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

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

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

See the proof of Lemma 5.3 for more details on where these polynomials come from.

5.3 Numerical experiments

All experiments were performed in IEEE-754 double precision. As in [JLCS10], we consider the evaluation in the neighborhood of the multiple root of $p(s) = (s-1)(s-3/4)^7$, written in Bernstein form. Figure 5.4 shows the evaluation of $p(s)$ at the 401 equally spaced[3] points $\left\{\frac{3}{4} + j\frac{10^{-7}}{2}\right\}_{j=-200}^{200}$ with `DeCasteljau` (Algorithm 2.1), `CompDeCasteljau` (Algorithm 5.1) and `CompDeCasteljau3` (Algorithm 5.3 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 [JLCS10], 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.

[3]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.

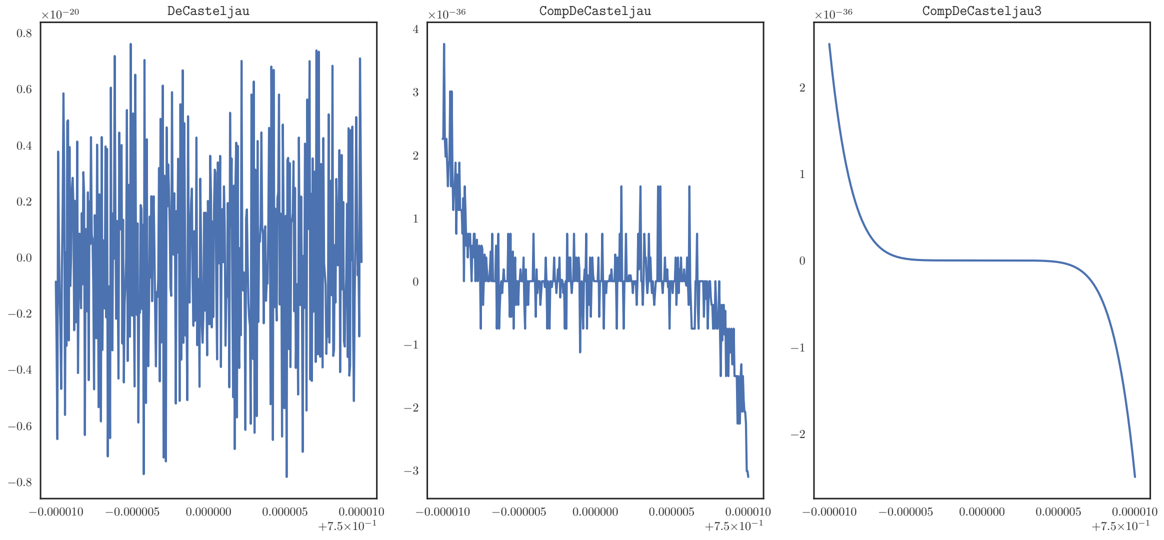

Figure 5.4: Evaluation of $p(s) = (s-1)(s-3/4)^7$ in the neighborhood of its multiple root 3/4.

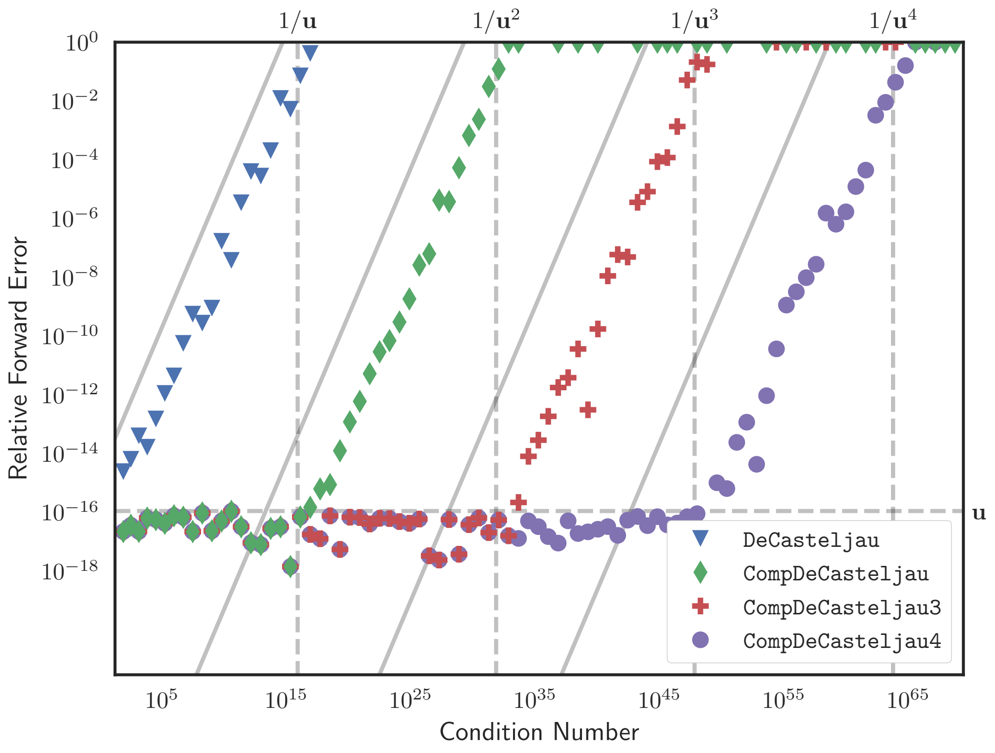

Figure 5.5: Accuracy of evaluation of $p(s) = (s-1)(s-3/4)^7$ represented in Bernstein form.

Figure 5.5 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 $\widetilde{p}(s) = (s - 1)\left(s/2 - 3/4\right)^7$. Once the relative errors and condition numbers are computed as fractions, they are rounded to the nearest IEEE-754 double precision value. As in [JLCS10], we use values $\left\{\frac{3}{4} - (1.3)^j\right\}_{j=-5}^{-90}$[4]. The curves for `DeCasteljau` and `CompDeCasteljau` trace the same paths seen in [JLCS10]. In particular, `CompDeCasteljau` has a relative error that is $\mathcal{O}\left(\mathbf{u}\right)$ until $\operatorname{cond}\left(p, s\right)$ reaches $1/\mathbf{u}$, at which point the relative error increases linearly with the condition number until it becomes $\mathcal{O}\left(1\right)$ when $\operatorname{cond}\left(p, s\right)$ reaches $1/\mathbf{u}^2$. Similarly, the relative error in `CompDeCasteljau3` (Algorithm 5.3 with $K = 3$) is $\mathcal{O}\left(\mathbf{u}\right)$ until $\operatorname{cond}\left(p, s\right)$ reaches $1/\mathbf{u}^2$ at which point the relative error increases linearly to $\mathcal{O}\left(1\right)$ when $\operatorname{cond}\left(p, s\right)$ reaches $1/\mathbf{u}^3$ and the relative error in `CompDeCasteljau4` (Algorithm 5.3 with $K = 4$) is $\mathcal{O}\left(\mathbf{u}\right)$ until $\operatorname{cond}\left(p, s\right)$ reaches $1/\mathbf{u}^3$ at which point the relative error increases linearly to $\mathcal{O}\left(1\right)$ when $\operatorname{cond}\left(p, s\right)$ reaches $1/\mathbf{u}^4$.

[4]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 $\operatorname{cond}\left(p, s\right)$ as j decreases away from 0 and because it results in "random" changes in the bits of 0.75 that are impacted by subtracting $(1.3)^j$.

Bibliography

[BR78] I. Babuška and W. C. Rheinboldt. Error estimates for adaptive finite element computations. *SIAM Journal on Numerical Analysis*, 15(4):736–754, 1978.

[Dek71] T. J. Dekker. A floating-point technique for extending the available precision. *Numerische Mathematik*, 18(3):224–242, Jun 1971.

[DK87] John K. Dukowicz and John W. Kodis. Accurate conservative remapping (rezoning) for arbitrary lagrangian-eulerian computations. *SIAM Journal on Scientific and Statistical Computing*, 8(3):305–321, may 1987.

[DP15] Jorge Delgado and J.M. Peña. Accurate evaluation of Bézier curves and surfaces and the Bernstein-Fourier algorithm. *Applied Mathematics and Computation*, 271:113–122, Nov 2015.

[Far86] Gerald Farin. Triangular Bernstein-Bézier patches. *Computer Aided Geometric Design*, 3(2):83–127, Aug 1986.

[Far91] R.T. Farouki. On the stability of transformations between power and Bernstein polynomial forms. *Computer Aided Geometric Design*, 8(1):29–36, Feb 1991.

[Far01] Gerald Farin. *Curves and Surfaces for CAGD, Fifth Edition: A Practical Guide (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann, 2001.

[FM11] P.E. Farrell and J.R. Maddison. Conservative interpolation between volume meshes by local Galerkin projection. *Computer Methods in Applied Mechanics and Engineering*, 200(1-4):89–100, Jan 2011.

[FPP$^+$09] P.E. Farrell, M.D. Piggott, C.C. Pain, G.J. Gorman, and C.R. Wilson. Conservative interpolation between unstructured meshes via supermesh construction. *Computer Methods in Applied Mechanics and Engineering*, 198(33-36):2632–2642, Jul 2009.

[FR87] R.T. Farouki and V.T. Rajan. On the numerical condition of polynomials in Bernstein form. *Computer Aided Geometric Design*, 4(3):191–216, Nov 1987.

[GLL09] Stef Graillat, Philippe Langlois, and Nicolas Louvet. Algorithms for accurate, validated and fast polynomial evaluation. *Japan Journal of Industrial and Applied Mathematics*, 26(2-3):191–214, Oct 2009.

[HAC74] C.W Hirt, A.A Amsden, and J.L Cook. An arbitrary lagrangian-eulerian computing method for all flow speeds. *Journal of Computational Physics*, 14(3):227–253, mar 1974.

[Hig02] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms.* Society for Industrial and Applied Mathematics, jan 2002.

[JH04] Xiangmin Jiao and Michael T. Heath. Common-refinement-based data transfer between non-matching meshes in multiphysics simulations. *International Journal for Numerical Methods in Engineering*, 61(14):2402–2427, 2004.

[JLCS10] Hao Jiang, Shengguo Li, Lizhi Cheng, and Fang Su. Accurate evaluation of a polynomial and its derivative in Bernstein form. *Computers & Mathematics with Applications*, 60(3):744–755, Aug 2010.

[KLS98] Deok-Soo Kim, Soon-Woong Lee, and Hayong Shin. A cocktail algorithm for planar Bézier curve intersections. *Computer-Aided Design*, 30(13):1047–1051, Nov 1998.

[Knu97] Donald E. Knuth. *Art of Computer Programming, Volume 2: Seminumerical Algorithms (3rd Edition).* Addison-Wesley Professional, 1997.

[LGL06] Philippe Langlois, Stef Graillat, and Nicolas Louvet. Compensated Horner Scheme. In Bruno Buchberger, Shin'ichi Oishi, Michael Plum, and Sigfried M. Rump, editors, *Algebraic and Numerical Algorithms and Computer-assisted Proofs*, number 05391 in Dagstuhl Seminar Proceedings, pages 1–29, Dagstuhl, Germany, 2006. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[MD92] Dinesh Manocha and James W. Demmel. Algorithms for intersecting parametric and algebraic curves. Technical Report UCB/CSD-92-698, EECS Department, University of California, Berkeley, Aug 1992.

[MP99] E. Mainar and J.M. Peña. Error analysis of corner cutting algorithms. *Numerical Algorithms*, 22(1):41–52, 1999.

[MP05] E. Mainar and J. M. Peña. Running Error Analysis of Evaluation Algorithms for Bivariate Polynomials in Barycentric Bernstein Form. *Computing*, 77(1):97–111, Dec 2005.

[ORO05] Takeshi Ogita, Siegfried M. Rump, and Shin'ichi Oishi. Accurate Sum and Dot Product. *SIAM Journal on Scientific Computing*, 26(6):1955–1988, Jan 2005.

[SN90] T.W. Sederberg and T. Nishita. Curve intersection using Bézier clipping. *Computer-Aided Design*, 22(9):538–549, Nov 1990.

[SP86] Thomas W Sederberg and Scott R Parry. Comparison of three curve intersection algorithms. *Computer-Aided Design*, 18(1):58–63, Jan 1986.

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-multipy-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 [ORO05] 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 VecSum($p$)
    $n = \texttt{length}(p)$
    for $j = 2, \ldots, n$ do
        $[p_j, p_{j-1}] = \texttt{TwoSum}\,(p_j, p_{j-1})$
    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 result = SumK($p, K$)
    for $j = 1, \ldots, K - 1$ do
        $p = \texttt{VecSum}(p)$
    end for
    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 5.2:

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
```

B

Proof Details

Proof of Lemma 5.2. We'll start with the $F = 1$ case. Recall where the terms originate:

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

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

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

Hence Theorem 2.1 tells us that

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

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

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

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

$$|e_3| \leq \mathbf{u} \left|P_1\right| + \mathbf{u} \left|P_2\right| \tag{B.8}$$

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

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

$$\begin{aligned}
\widetilde{\ell}_{1,j}^{(k)} &= |e_1| + |e_2| + |e_3| + \left|\rho \cdot \widehat{b}_j^{(k+1)}\right| && \text{(B.10)}\\
&\leq (2 + \mathbf{u}) \left(|e_1| + |e_2|\right) + (1 + \mathbf{u})(1 - s) \left|\widehat{b}_j^{(k+1)}\right| && \text{(B.11)}\\
&\leq \left[(1 + \mathbf{u})^3 - 1\right] (1 - s) \left|\widehat{b}_j^{(k+1)}\right| + \left[(1 + \mathbf{u})^2 - 1\right] s \left|\widehat{b}_{j+1}^{(k+1)}\right| && \text{(B.12)}\\
&\leq \gamma_3 \left((1 - s) \left|\widehat{b}_j^{(k+1)}\right| + s \left|\widehat{b}_{j+1}^{(k+1)}\right|\right). && \text{(B.13)}
\end{aligned}$$

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:

$$\begin{aligned}
[S_1, e_1] &= \texttt{TwoSum}\left(e'_1, e'_2\right) && \text{(B.14)}\\
[S_2, e_2] &= \texttt{TwoSum}\left(S_1, e'_3\right) && \text{(B.15)}\\
&\vdots\\
[S_{5F-3}, e_{5F-3}] &= \texttt{TwoSum}\left(S_{5F-4}, e'_{5F-2}\right) && \text{(B.16)}\\
[P_{5F-2}, e_{5F-2}] &= \texttt{TwoProd}\left(\rho, \widehat{\partial^{F-1} b}_j^{(k+1)}\right) && \text{(B.17)}\\
\left[\widehat{\ell}_{F,j}^{(k)}, e_{5F-1}\right] &= \texttt{TwoSum}\left(S_{5F-3}, P_{5F-2}\right) && \text{(B.18)}\\
[P_{5F}, e_{5F}] &= \texttt{TwoProd}\left(s, \widehat{\partial^F b}_{j+1}^{(k+1)}\right) && \text{(B.19)}\\
[S_{5F+1}, e_{5F+1}] &= \texttt{TwoSum}\left(\widehat{\ell}_{F,j}^{(k)}, P_{5F}\right) && \text{(B.20)}\\
[P_{5F+2}, e_{5F+2}] &= \texttt{TwoProd}\left(\rho, \widehat{\partial^F b}_j^{(k+1)}\right) && \text{(B.21)}\\
\left[\widehat{\partial^F b}_j^{(k)}, e_{5F+3}\right] &= \texttt{TwoSum}\left(S_{5F+1}, P_{5F+2}\right). && \text{(B.22)}
\end{aligned}$$

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

$$\begin{aligned}
\widetilde{\ell}_{F+1,j}^{(k)} &= |e_1| + \cdots + |e_{5F+3}| + \left|\rho \cdot \widehat{\partial^F b}_j^{(k+1)}\right| && \text{(B.23)}\\
&\leq \gamma_{5F}\,|e'_1| + \gamma_{5F}\,|e'_2| + \gamma_{5F-1}\,|e'_3| + \cdots + \gamma_4\,\left|e'_{5F-2}\right| + \gamma_4 \left|\rho \cdot \widehat{\partial^{F-1} b}_j^{(k+1)}\right| && \text{(B.24)}\\
&\qquad + \gamma_3(1-s)\left|\widehat{\partial^F b}_j^{(k+1)}\right| + \gamma_3 s \left|\widehat{\partial^F b}_{j+1}^{(k+1)}\right| && \text{(B.25)}\\
&\leq \gamma_3\left((1-s)\left|\widehat{\partial^F b}_j^{(k+1)}\right| + s\left|\widehat{\partial^F b}_{j+1}^{(k+1)}\right|\right) + \gamma_{5F} \cdot \widetilde{\ell}_{F,j}^{(k)} && \text{(B.26)}
\end{aligned}$$

as desired. ■

Proof of Lemma 5.3. First, note that for **any** sequence $v_0, \ldots, v_{k+1}$ 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 (5.23), 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 $L_{F,k}$:

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

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

Hence Lemma 5.2 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

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

Since $\ell_{F,j}^{(k)}$ 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 $\left| \widehat{\ell}_{F,j}^{(k)} \right| \leq (1+\gamma_{5F-2}) \widetilde{\ell}_{F,j}^{(k)}$. Combined with (B.35) and the fact that there is no local error when $F=0$, this means

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

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

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 $D_{0,n-k} \leq (1+\gamma_{3k}) \cdot \widetilde{p}(s)$ and from (B.31) that $L_{1,n-k} \leq \gamma_3 \left(1+\gamma_{3(k-1)}\right) \cdot \widetilde{p}(s)$.

To show the bounds for higher values of F, we'll assume we have bounds of the form $D_{F,n-k} \leq \left(q_F(k) \mathbf{u}^F + \mathcal{O}\left(\mathbf{u}^{F+1}\right) \right) \cdot \widetilde{p}(s)$ and $L_{F,n-k} \leq \left(r_F(k) \mathbf{u}^F + \mathcal{O}\left(\mathbf{u}^{F+1}\right) \right) \cdot \widetilde{p}(s)$ 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

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

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 $\gamma_{3k+5F} L_{F,k} \leq (3k+5F)L_{F,k}\mathbf{u} + \mathcal{O}\left(\mathbf{u}^{F+2}\right)\widetilde{p}(s)$ 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

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

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