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.

Contents

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. 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:

$$\frac{|p(s) - \texttt{DeCasteljau}(p, s)|}{|p(s)|} \leq \operatorname{cond}(p, s) \times \mathcal{O}(\mathbf{u}). \tag{1}$$

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 can be applied in *compensated algorithms* to account for roundoff. 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

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

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.

In [GLL09], the authors built on 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 roundoff 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), the accuracy of the compensated result (2) may be arbitrarily bad for ill-conditioned polynomial evaluations. 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 prior bound for any arbitrary integer K:

$$\frac{|p(s) - \texttt{CompDeCasteljauK}(p, s)|}{|p(s)|} \leq \mathbf{u} + \operatorname{cond}(p, s) \times \mathcal{O}\left(\mathbf{u}^K\right). \tag{3}$$

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 this working precision.

The paper is organized as follows. Section 2 introduces some basic notations and results about floating point arithmetic, error-free transformations and the de Casteljau algorithm. In Section 3, the compensated algorithm for polynomial evaluation from [JLCS10] 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 give numerical tests to illustrate the practical efficiency of our algorithms.

2 Basic notation and results

Placeholder

3 Compensated de Casteljau

Placeholder

4 K Compensated de Casteljau

Placeholder

5 Numerical experiments

Placeholder

6 IEEE Stuff

Algorithm 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

Algorithm 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

Algorithm 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

Algorithm 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

Algorithm 5 *de Casteljau algorithm for polynomial evaluation.*

function $\texttt{res} = \texttt{DeCasteljau}(\mathbf{b}, s)$
$n = \texttt{length}(b) - 1$
$r = 1 \ominus s$

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

for $k = n - 1, \ldots, 0$ **do**
for $j = 0, \ldots, k$ **do**
$b_j^{(k)} = \left(r \otimes b_j^{(k+1)}\right) \oplus \left(s \otimes b_{j+1}^{(k+1)}\right)$
end for
end for

$\texttt{res} = b_0^{(0)}$
end function

Algorithm 6 *Compensated de Casteljau algorithm for polynomial evaluation.*

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

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

  for $k = n - 1, \ldots, 0$ **do**
    for $j = 0, \ldots, k$ **do**
      $[P_1, \pi_1] =$ `TwoProd` $\left(r, b_j^{(k+1)}\right)$
      $[P_2, \pi_2] =$ `TwoProd` $\left(s, b_{j+1}^{(k+1)}\right)$
      $\left[b_j^{(k)}, \sigma_3\right] =$ `TwoSum`(P_1, P_2)
      $w = \pi_1 \oplus \pi_2 \oplus \sigma_3 \oplus \left(\rho \otimes b_j^{(k+1)}\right)$
      $\partial b_j^{(k)} = w \oplus \left(s \otimes \partial b_{j+1}^{(k+1)}\right) \oplus \left(r \otimes \partial b_j^{(k+1)}\right)$
    end for
  end for

  `res` $= b_0^{(0)} \oplus \partial b_0^{(0)}$
end function

7 de Casteljau's Method

Consider de Casteljau's method to evaluate a degree n polynomial in Bernstein-Bézier form with control points b_j:

$$\begin{aligned} b_j^{(n)} &= b_j \\ b_j^{(k)} &= (1 - s) b_j^{(k+1)} + s b_{j+1}^{(k+1)} \\ b(s) &= b_0^{(0)}. \end{aligned}$$

7.1 Condition Number

For a polynomial $p(x)$ in the power basis, we have ([LGL06]):

$$\operatorname{cond}(p(x)) = \frac{\widetilde{p}(|x|)}{|p(x)|} = \frac{\sum_j |a_j| \, |x|^j}{|p(x)|}. \tag{4}$$

In particular, this means that if $x \geq 0$ and each $a_j \geq 0$ we must necessarily have $\operatorname{cond}(p(x)) = 1$. To see an example in use, consider $p(x) = (x - 1)^n$ and input values of the form $x = 1 + \delta$ (with $|\delta| \ll 1$). Since $a_j = \binom{n}{j}(-1)^{n-j}$ we have $\widetilde{p}(x) = (x + 1)^n$ hence

$$\operatorname{cond}(p(1 + \delta)) = \frac{(2 + \delta)^n}{|\delta|^n} = \left|1 + \frac{2}{\delta}\right|^n. \tag{5}$$

As $\delta \to 0$, this value approaches ∞ (as expected).

For a polynomial $p(s)$ in Bernstein form, we have ([JLCS10]):

$$\operatorname{cond}(p(s)) = \frac{\widetilde{p}(s)}{|p(s)|} = \frac{\sum_j |b_j| \, |B_{j,n}(s)|}{|p(s)|}. \tag{6}$$

The Bernstein form is suited for $s \in [0, 1]$, which means $B_{j,n}(s) \geq 0$ typically. If $s \in [0, 1]$ and each $b_j \geq 0$ we must necessarily have $\operatorname{cond}(p(s)) = 1$. To see an example in use, consider

$$p(s) = (1 - 2s)^n = [(1 - s) - s]^n = \sum_j \binom{n}{j} (1 - s)^{n-j}(-s)^j = \sum_j (-1)^j B_{j,n}(s) \tag{7}$$

and input values of the form $x = \frac{1}{2} + \delta$ (with $|\delta| \ll \frac{1}{2}$). Since $b_j = (-1)^j$ we have $\widetilde{p}(s) = [(1 - s) + s]^n = 1$

$$\operatorname{cond}\left(p\left(\frac{1}{2} + \delta\right)\right) = \frac{1}{|2\delta|^n}. \tag{8}$$

As $\delta \to 0$, this value approaches ∞ (as expected).

7.2 Example of Compensated de Casteljau Failing

Consider

$$p(s) = (4s - 3)^3(8s + 7) = (-189)B_{0,4}(s) + (-54)B_{1,4}(s) + 57B_{2,4}(s) + (-32)B_{3,4}(s) + 15B_{4,4}(s) \tag{9}$$

at the point $s = \frac{3}{4} + 800u$. In exact arithmetic, we have $p(s) = 13(3200u)^3 + \mathcal{O}\left(u^4\right)$. However, using the compensated de Casteljau algorithm results in $b_0^{(0)} + \partial b_0^{(0)} = 0$:

k	j	$b_j^{(k)}$	$\partial b_j^{(k)}$	$\partial^2 b_j^{(k)}$	$\partial^3 b_j^{(k)}$
3	0	$-87\frac{3}{4} + 108032u$	$-32u$	0	0
3	1	$29\frac{1}{4} + 88768u$	$32u$	0	0
3	2	$-9\frac{3}{4} - 71200u$	0	0	0
3	3	$3\frac{1}{4} + 37600u$	0	0	0
2	0	$187200u$	$-15360000u^2$	0	0
2	1	$-62408u$	$8u - 128000000u^2$	0	0
2	2	$20800u$	$87040000u^2$	0	0
1	0	$-6u - 199688192u^2$	$6u - 99831808u^2$	$-90112000000u^3$	0
1	1	$-2u + 66568192u^2$	$2u + 33271808u^2$	$172032000000u^3$	0
0	0	$-3u + 7296u^2$	$3u - 7296u^2$	$13(3200u)^3 + 3048(512u)^4$	$962(128u)^4$

7.3 Selection of Test Cases

From [DP15] (end of Section 3):

> We can observe that, in this case, the algorithm with a good behavior everywhere is the de Casteljau algorithm

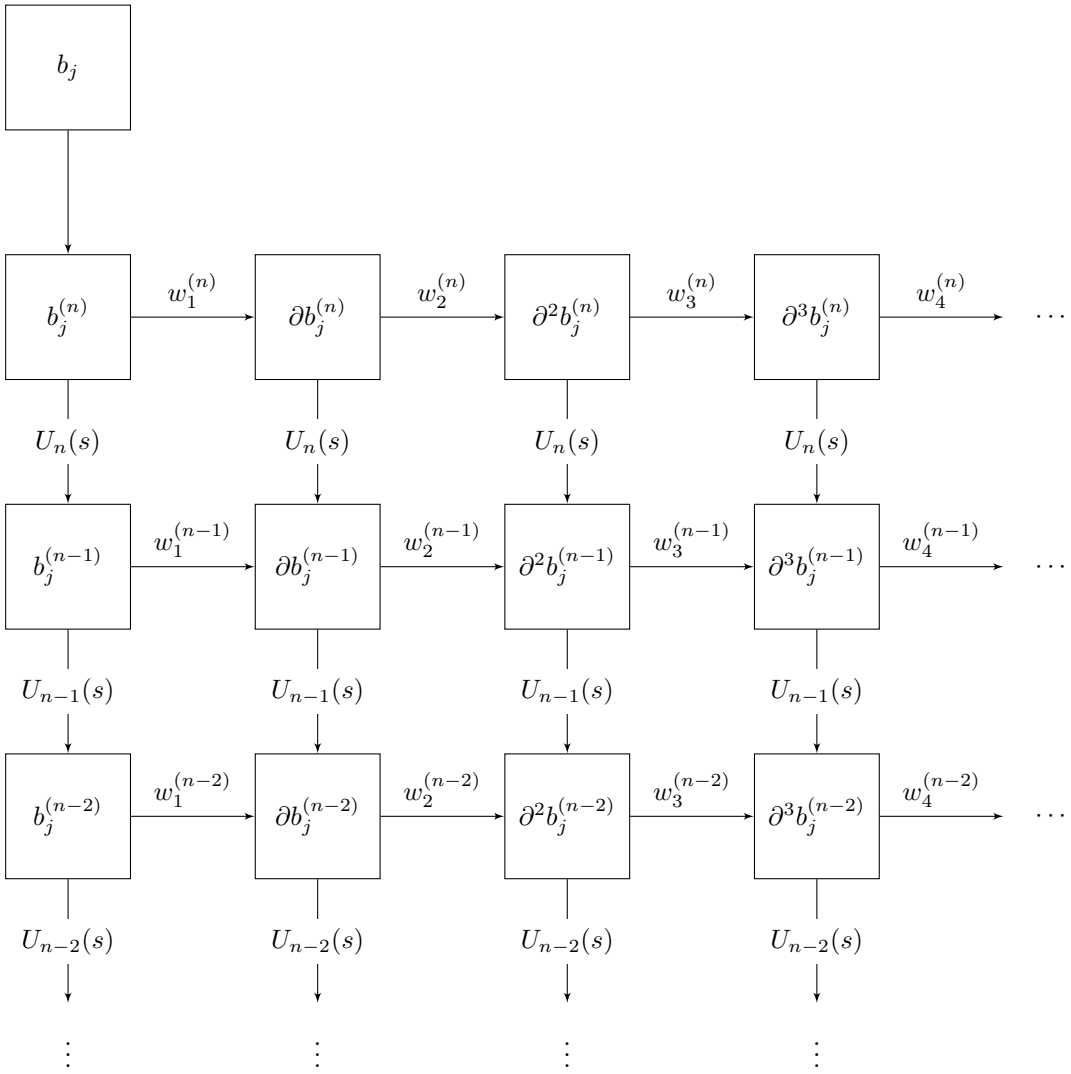
In the same paper (when referring to [Bez13] at the beginning of Section 2):

> assuming that all control points are positive. This assumption avoided ill-conditioned polynomials. In this section, we shall show that this is a natural assumption in Computer Aided Geometric Design (from now on, C.A.G.D.) and that it permits to assure high relative precision for the evaluation through a large family of representations in C.A.G.D.

From the same author, in [MP05] (towards the end of Section 5, at the bottom of page 109):

> Let us observe that in this case, the de Casteljau algorithm presents better stability properties for the evaluation near the roots. In fact, the de Casteljau algorithm has good behaviour even when using simple precision, although the running error bound is not so accurate in points close to the roots.

7.4 K Error Filtering



7.5 Operation Counts

After implementing for $K = 2, 3, \ldots, 12$ and instrumenting all relevant floating point operations, the K-fold Horner requires

$$(5 \cdot 2^K - 8)n + \left((K+8)2^K - 12K - 6\right) = \mathcal{O}\left((n+K)2^K\right) \tag{10}$$

flops to evaluate a degree n polynomial (this only applies when $n \geq K - 1$). As a comparison, the non-compensated form of Horner requires $2n$ flops. Of these, $\left(2^{K-1} - 1\right)n - 2^{K-1}(K-3) - 2$ are FMA (fused-multiply-add) instructions.

After implementing for $K = 2, 3, 4, 5$ and instrumenting all relevant floating point operations, the K-fold de Casteljau requires

$$(15K^2 - 34K + 26)T_n + K + 5 = \mathcal{O}\left(n^2K^2\right) \tag{11}$$

flops to evaluate a degree n polynomial. (Here T_n is the nth triangular number.) As a comparison, the non-compensated form of de Casteljau requires $3T_n + 1$ flops. Of these, $(3K-4)T_n$ are FMA instructions. On hardware that doesn't support FMA, every FMA will be exchanged for 10 $\ominus$'s and 6 $\otimes$'s so the count will increase by $(10 + 6 - 1)(3K - 4)T_n$.

8 Bogus Section for Refs

Here they are, for now

- Compensated Horner ($K = 2$) ([LGL06])
- Compensated de Casteljau ([JLCS10])
- Newton with compensated Horner ([Gra08])
- K-fold Sum ([ORO05])
- K-fold Horner ([GLL09])

References

[Bez13] Licio Hernanes Bezerra. Efficient computation of Bézier curves from their Bernstein–Fourier representation. *Applied Mathematics and Computation*, 220:235–238, Sep 2013.

[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.

[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.

[Gra08] Stef Graillat. Accurate simple zeros of polynomials in floating point arithmetic. *Computers & Mathematics with Applications*, 56(4):1114–1120, Aug 2008.

[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.

[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, Dagstuhl, Germany, 2006. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.

[MP99] E. Mainar and J.M. Peña. *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.