**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" $K$-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 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** $\texttt{res} = \texttt{DeCasteljau}(p, s)$
$n = \texttt{length}(p) - 1$
$r = 1 \ominus s$

**for** $j = 0, \ldots, n$ **do**
$b_j^{(n)} = p_j$
**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** $\texttt{res} = \texttt{CompDeCasteljau}(p, s)$
$n = \texttt{length}(p) - 1$
$[r, \rho] = \texttt{TwoSum}(1, -s)$

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

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

$\texttt{res} = b_0^{(0)} \oplus e_0^{(0)}$
**end function**

# 2 de Casteljau's Method

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

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

## 2.1 Condition Number

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

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

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

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

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

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

$$\operatorname{cond}(p(s)) = \frac{\widetilde{p}(s)}{|p(s)|} = \frac{\sum_j |p_j|\,|b_{j,n}(s)|}{|p(s)|}.$$

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 $p_j \geq 0$ we must necessarily have $\operatorname{cond}(p(s)) = 1$. To see an example in use, consider

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

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

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

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

## 2.2 Example of Compensated de Casteljau Failing

Consider

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

at the point $s = \frac{3}{4} + 800u$. In exact arithmetic, we have $p(s) = 2^{38.6}u^3 + \mathcal{O}\left(u^4\right)$. However, using the compensated de Casteljau algorithm causes problems.

| $k$ | $j$ | $b_j^{(k)}$ | $e_j^{(k)}$ |
|---|---|---|---|
| 3 | 0 | $-87\frac{3}{4} + 108032u$ | $-32u$ |
| 3 | 1 | $29\frac{1}{4} + 88768u$ | $32u$ |
| 3 | 2 | $-9\frac{3}{4} - 71200u$ | $0$ |
| 3 | 3 | $3\frac{1}{4} + 37600u$ | $0$ |
| 2 | 0 | $187200u$ | $-15360000u^2$ |
| 2 | 1 | $-62408u$ | $8u - 128000000u^2$ |
| 2 | 2 | $20800u$ | $87040000u^2$ |
| 1 | 0 | $-6u - 199688192u^2$ | $\boxed{6u - 99831808u^2}$ |
| 1 | 1 | $-2u + 66568192u^2$ | $\boxed{2u + 33271808u^2}$ |
| 0 | 0 | $-3u + 7296u^2$ | $\boxed{3u - 7296u^2}$ |

This is due to dropped terms: $e_0^{(1)}$ drops $-90112000000u^3$, $e_1^{(1)}$ drops $172032000000u^3$, $e_0^{(0)}$ drops $319488000000u^3$ (that's **after** the $e_j^{(1)}$ terms are incorrect).

## 2.3 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.

## 2.4 $K$-Fold Error Filtering

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

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

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

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

flops to evaluate a degree $n$ polynomial. (Here $T_n$ is the $n$th 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 $(10 + 6 - 1)(3K - 4)T_n$.

## 3 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.

[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.