Algorithm 960: POLYNOMIAL: An object-oriented Matlab library of fast and efficient algorithms for polynomials

JORGE DELGADO, Universidad de Zaragoza
and JUAN MANUEL PEÑA, Universidad de Zaragoza

The design and implementation of a Matlab object-oriented software library for working with polynomials is presented. The construction and evaluation of polynomials in Bernstein form are motivated and justified. Efficient constructions for the coefficients of a polynomial in Bernstein form when the polynomial is not given with this representation are provided. The presented adaptative evaluation algorithm uses VS (Volk and Schumaker) algorithm, de Casteljau algorithm and a compensated VS algorithm. In addition, we have completed the library with other algorithms to perform other usual operations with polynomials in Bernstein form.

CCS Concepts:•**Mathematics of computing** → **Mathematical software; Computations on polynomials;**

Additional Key Words and Phrases: Bernstein basis, polynomial algorithms, numerical stability

ACM Reference Format:
Jorge Delgado and Juan Manuel Peña, 2015. Algorithm ???: POLYNOMIAL: An object-oriented Matlab library of fast and efficient algorithms for polynomials. *ACM Trans. Math. Softw.* ?, ?, Article A (January 13), 19 pages.
DOI: 0000001.0000001

1. INTRODUCTION

In this article we present the design and implementation of a Matlab object-oriented software library for working with polynomials. We focus our efforts mainly on providing algorithms for a fast and accurate evaluation of polynomials in Bernstein form and for an efficient construction of polynomials in this form.

Horner's algorithm is the usual method for evaluating polynomials. It has linear complexity. In some recent articles it has been shown that this algorithm is outperformed by other algorithms from the point of view of accuracy (see [Delgado and Peña 2009]). In particular, from [Farouki and Rajan 1987], [Mainar and Peña 1999], [Farouki and Goodman 1996] and [Delgado and Peña 2009] it can be concluded that the de Casteljau algorithm is the most accurate algorithm among some polynomial evaluation algorithms, including the Horner algorithm. The de Casteljau algorithm evaluates polynomials represented in the Bernstein form, and is the usual polynomial evaluation algorithm in Computer Aided Geometric Design (C.A.G.D.). In [Tsai and Farouki 2001] Tsai and Farouki presented a C++ object-oriented library of numerical algorithms for polynomials in Bernstein form including the de Casteljau algorithm. Nevertheless, the de Casteljau algorithm evaluates a polynomial of degree n with $\mathcal{O}(n^2)$ elementary operations in contrast to the $\mathcal{O}(n)$ elementary operations of the Horner algorithm. In [Delgado and Peña 2009] it was also analyzed the Volk and Schumaker (VS) algorithm

Authors' addresses: J. Delgado, Departamento de Matemática Aplicada, Universidad de Zaragoza, Escuela Universitaria Politécnica de Teruel, C/ Ciudad Escolar s/n, 44003 - Teruel (Spain); J.M. Peña, Departamento de Matemática Aplicada, Universidad de Zaragoza, 50006-Zaragoza, Spain.

(see [Schumaker and Volk 1986]), which, like de Casteljau, evaluates polynomials in the Bernstein form, and, like Horner, is a nested algorithm. As we recall in Section 2, the conditioning for the representations used by the de Casteljau and VS algorithms coincides. Besides, the VS algorithm evaluates a polynomial of degree n represented in Bernstein form with $\mathcal{O}(n)$ elementary operations like Horner method. In [Delgado and Peña 2009] it was proved that the de Casteljau and VS algorithms are more accurate than Horner algorithm. In fact, Example 3.7 shows this property even for polynomials represented in monomial form through a conversion algorithm from monomial to Bernstein basis that we include in Section 3. In addition, in [Tsai and Farouki 2001] the algorithm for the division of polynomials can present an undesired behaviour for polynomials whose true degree is lower than the degree of the Bernstein basis used in its representation. In Subsection 3.3 we explain this problem and propose a solution.

The article has the following layout. Section 2 recalls the error analysis for Horner, de Casteljau and VS algorithms. We also recall the compensated algorithms recently introduced in [Langlois and Louvet 2007] and [Jiang et al. 2010] for Horner and de Casteljau algorithms. In addition to the three usual ways of constructing a polynomial (default, copy and straightforward by providing the coefficients), in Section 3 we also provide two efficient constructions of a polynomial in Bernstein form when the polynomial is not given with this representation. The first construction computes with high accuracy the coefficients from the interpolation conditions. The second construction uses the conversion algorithm from monomial to Bernstein bases mentioned above.

A compensated VS algorithm with a dynamic error estimate is presented in Section 4, and a sketch of the proof is given in the accompanying electronic appendix. It is used in the cases where the VS algorithm is not accurate enough (detected by its relative running error) and where de Casteljau algorithm is too expensive (due to the high degree of the polynomial). Examples show that this estimate provides reliable approximation of the true error when evaluating bad conditioned polynomials. Section 5 includes the corresponding adaptative evaluation algorithm and the examples justifying it.

The structure of the software library, the implementation of functions with common operations with polynomials and some issues about the design and implementation related to Matlab peculiarities may be found in the user manual accompanying the software.

2. BACKGROUND ON EFFICIENT POLYNOMIAL EVALUATION ALGORITHMS

In this section, we survey results on efficient polynomial evaluation algorithms, including the running error bounds used in the adaptative evaluation algorithm presented in this article, which provides its corner stones.

Let $\mathcal{U} = (u_0, \ldots, u_n)$ be a basis of the space of polynomials $\mathcal{P}_n$ of degree at most n. If $p \in \mathcal{P}_n$ then there exists a unique real sequence of real coefficients $c = (c_0, \ldots, c_n)$ such that $p(t) = \sum_{i=0}^{n} c_i u_i(t)$ for all $t \in I$. Given an algorithm for the evaluation of a polynomial $p(t)$ of this form, one obtains the computed value $fl(p(t))$ in floating point arithmetic. In practical computations it is also desirable to obtain an error bound or estimate for the approximation of the exact evaluation $p(t)$ given by $fl(p(t))$. When we evaluate a polynomial by an algorithm, the success on the accuracy of the obtained approximation depends first, on the calculations performed by the algorithm, that is, the backward error, and second, on the difficulty of the evaluated polynomial, that is, the condition number of the polynomial with respect to the representation used in the evaluation algorithm.

We can consider that the computed $fl(p(t))$ can be expressed as $fl(p(t)) = \sum_{i=0}^{n}(1+\delta_i)c_i u_i(t)$, where $\delta = (\delta_i)_{i=0}^{n}$ is a perturbation in c. Then for any $t \in I$

$$|p(t) - fl(p(t))| = \Big|\sum_{i=0}^{n} \delta_i c_i u_i(t)\Big| \leq ||\delta||_\infty \sum_{i=0}^{n} |c_i u_i(t)|. \quad (1)$$

The number $S_{\mathcal{U}}(p(t)) := \sum_{i=0}^{n} |c_i u_i(t)|$, acts as a condition number for the evaluation of p at the point t using the basis $\mathcal{U}$ (see [Farouki and Goodman 1996], [Farouki and Rajan 1987], [Lyche and Peña 2004], [Peña 2002] and [Peña 2006]). Then (1) can be interpreted as an upper bound of the forward error of evaluation of the form of a product of the backward error and the condition number (cf. [Higham 2002]).

The most well-known polynomial evaluation algorithm is the Horner algorithm. It uses $m_n := (m_0^n(t), m_1^n(t), \ldots, m_n^n(t))$, $t \in [0,1]$, the monomial basis of the space $\mathcal{P}_n$ given by $m_i^n(t) = t^i$, $i = 0, 1, \ldots, n$, and let us consider $p(t) = \sum_{i=0}^{n} c_i\, m_i^n(t)$. But there are also other polynomials evaluation algorithms: de Casteljau and VS algorithms for polynomials represented using the Bernstein basis (see [Farin 2002] and [Schumaker and Volk 1986]), the DP algorithm for polynomials represented in the DP basis (see [Delgado and Peña 2003] and [Delgado and Peña 2006]), the Clenshaw algorithm for polynomials represented in orthogonal bases (see [Delgado and Peña 2009]), . . . Let us now consider the Bernstein basis $b_n := (b_0^n(t), b_1^n(t), \ldots, b_n^n(t))$, $t \in [0,1]$, of $\mathcal{P}_n$ given by

$$b_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}, \quad i = 0, 1, \ldots, n, \quad (2)$$

and let $p(t) = \sum_{i=0}^{n} c_i\, b_i^n(t) \in \mathcal{P}_n$. This representation is associated with a well-known evaluation algorithm in C.A.G.D.: the de Casteljau algorithm. In order to evaluate polynomials in Bernstein form with the low computational cost of nested algorithms, the following basis and algorithms can be used. The VS basis $z_n := (z_0^n(t), z_1^n(t), \ldots, z_n^n(t))$, $t \in [0,1]$, of $\mathcal{P}_n$ is given by $z_i^n(t) = t^i(1-t)^{n-i}$, $i = 0, 1, \ldots, n$, and let $p(t) = \sum_{i=0}^{n} c_i\, z_i^n(t) \in \mathcal{P}_n$. Since each function of the VS basis is a multiple of the corresponding Bernstein polynomial, the corresponding condition numbers coincide $S_{z_n}(p(t)) = S_{b_n}(p(t))$ for all $p \in \mathcal{P}_n$ and $t \in [0,1]$. In [Farouki and Goodman 1996] it was proved that the Bernstein basis is optimally conditioned for polynomial evaluation among the bases of nonnegative polynomials on $[0,1]$ in the sense that there does not exist (up to positive scaling) another basis of nonnegative polynomials on $[0,1]$ that is better conditioned at every point $t \in [0,1]$ for every polynomial $p(t)$ of the space (see [Farouki and Goodman 1996]). It is also known that $(S_{z_n}(p(t)) =) S_{b_n}(p(t)) \leq S_{m_n}(p(t))$ at every point $t \in [0,1]$. On the other hand, in [Delgado and Peña 2009] it has been performed a comparison between the polynomial evaluation algorithms mentioned above. It shows that the algorithms associated to the Bernstein bases are the most accurate algorithms, that is, de Casteljau and VS algorithms.

The VS algorithm has linear time complexity, whereas the de Casteljau algorithm has quadratic time complexity. Nevertheless, in [Delgado and Peña 2009] it was also shown that, in the case of extremely ill-conditioned polynomials, the de Casteljau algorithm can outperform VS algorithm in terms of accuracy.

Taking into account the forward error bounds in pp. 40-41 of [Higham 2002], [Mainar and Peña 1999] and Theorem 4.2 of [Delgado and Peña 2009], and the relative error bound in Theorem 3.1 of [Delgado and Peña 2009], we get the following forward error bounds for the relative errors when evaluating the polynomial by the Horner, de Casteljau and VS algorithms:

$$\left|\frac{fl(p(t)) - p(t)}{p(t)}\right| \leq k \cdot n \cdot u \frac{S_{\mathcal{U}}(p(t))}{|fl(p(t))|} + \mathcal{O}(u^2), \quad (3)$$

assuming that $|fl(p(t))| > u\,k \cdot n \cdot S_{\mathcal{U}}(p(t))$ and $k\,n\,u < 1$, with $k = 2$ for Horner and de Casteljau, $k = 4$ for VS, $\mathcal{U} = m_n$ for Horner and $\mathcal{U} = b_n$ for de Casteljau and VS, where u is the unit roundoff. In conclusion, the relative error bounds of de Casteljau and VS algorithms are lower than that of Horner algorithm due to the better conditioning of their bases. This is confirmed by the numerical experiments in this article and in [Delgado and Peña 2009].

Running error bounds for these algorithms were also obtained in [Mainar and Peña 1999] and [Delgado and Peña 2009]. In Algorithm 1 we recall the de Casteljau case, which will be used in the implementation of our adaptative evaluation algorithm, as recalled in Section 5. With the symbols $\oplus$, $\ominus$, $\otimes$ and $\oslash$ we represent the floating point addition, subtraction, multiplication and division.

Algorithm 1 *De Casteljau algorithm* for the evaluation of $p \in \mathcal{P}_n$ at t

```
Require: $t \in [0,1]$ and $(c_i)_{i=0}^n$
Ensure: $res \approx \sum_{i=0}^n c_i b_i^n(t)$
  $f_j^0(t) := c_j, \quad j = 0, \ldots, n$
  $f_j^r(t) = (1-t) \otimes f_j^{r-1}(t) \oplus t \otimes f_{j+1}^{r-1}(t), \quad j = 0, \ldots, n-r, \quad r = 1, \ldots, n$
  $res = f_0^n(t)$
```

The following result with a running relative error bound is a consequence of Theorem 5.2 of [Delgado and Peña 2009] and (3).

THEOREM 2.1. *Let us consider the Bernstein basis b_n and the associate de Casteljau algorithm (Algorithm 1). Let $p(t) = \sum_{i=0}^n c_i\, b_i^n(t)$ and assume that $2nu < 1$, where u is the unit roundoff. Then we have that $|res - p(t)| \leq u\,\pi_0^n$, where π_0^n is given by $\pi_j^0 = 0$ for all $j \in \{0, 1, \ldots, n\}$ and $\pi_j^r = (1-t)\pi_j^{r-1} + t\pi_{j+1}^{r-1} + (1-t)|f_j^{r-1}(t)| + t|f_{j+1}^{r-1}(t)| + |f_j^r(t)|$ for all $r \in \{1, \ldots, n\}$ and $j \in \{0, 1, \ldots, n-r\}$. Moreover, if $|res| > u\,\pi_0^n$, then*

$$\left|\frac{res - p(t)}{p(t)}\right| \leq u\,\frac{\pi_0^n}{|res|} + \mathcal{O}(u^2). \tag{4}$$

As for the VS algorithm, presented in Section 4 (Algorithm 3), the next result follows from Theorem 4.3 of [Delgado and Peña 2009] and (3).

THEOREM 2.2. *Let us consider the VS basis z_n and the associated VS algorithm (Algorithm 3). Let $p(t) = \sum_{i=0}^n c_i\, z_i^n(t)$ and assume that $4nu < 1$, where u is the unit roundoff. Then $|p(t) - res| \leq u\,\widetilde{\pi} + \mathcal{O}(u^2)$, where, if $t \geq 1/2$, $\widetilde{\pi} = fl(t^n)\,\pi_n + |p_n|(n-1)fl(t^n) + |res|$, $\pi_0 = 0$ and, for $i = 1, \ldots, n$, $\pi_i = \pi_{i-1}\,q + 2|p_{i-1}|\,q + |p_i|$, and, if $t < 1/2$, then $\widetilde{\pi} = fl((1-t)^n)\,\pi_0 + |p_0|(n-1)fl((1-t)^n) + |res|$, $\pi_n = 0$ and for $i = 1, \ldots, n$, $\pi_{n-i} = \pi_{n+1-i}\,q + 2|p_{n+1-i}|\,q + |p_{n-i}|$. Moreover, if $|res| > u\,\widetilde{\pi}$, then*

$$\left|\frac{res - p(t)}{p(t)}\right| \leq u\,\frac{\widetilde{\pi}}{|res|} + \mathcal{O}(u^2). \tag{5}$$

Given an algebraic expression defined by additions, subtractions, multiplications and divisions and assuming that each initial real datum is known to high relative accuracy, then it is well known that the algebraic expression can be computed to high relative accuracy if it is defined by sums of numbers of the same sign, products and quotients (cf. p. 52 of [Demmel et al. 1999]). In other words, the only forbidden operation is true subtraction, due to possible cancellation in leading digits. Moreover, in (well–implemented) floating point arithmetic high relative accuracy is also preserved even when we perform true subtractions when the operands are original (and so, exact) data (cf. p. 53 of [Demmel et al. 1999]). Observe that the linear approximation to the running relative error bounds of the two previous theorems can be calculated to high relative accuracy.

Graillat, Langlois and Louvet presented a *compensated* Horner algorithm for the evaluation of a polynomial represented in the monomial basis [Langlois et al. 2005], [Langlois and Louvet 2007]. The compensated algorithm is accurate for not too ill-conditioned polynomials. In fact, the compensated version of an algorithm delays the effects of the bad conditioning in the accuracy of the results. The key tool to obtain more accurate results is to apply what Ogita, Rump and Oishi call error-free transformations (see [Ogita et al. 2005]). A compensated version of the de Casteljau algorithm for the evaluation of a polynomial in Bernstein form was derived in [Jiang et al. 2010]. Let us recall the typical improvement of these compensated algorithms for the evaluation accuracy. If we have an evaluation algorithm of a polynomial $p(t)$ represented in a basis $\mathcal{U}$ with a forward error bound of the form:

$$|p(t) - fl(p(t))| \leq (|p(t)|S_{\mathcal{U}}(p(t))) \times \mathcal{O}(u),$$

then the compensated algorithm produces a computed evaluation $fl(p(t))$ satisfying

$$|p(t) - fl(p(t))| \leq |p(t)|u + (|p(t)|S_{\mathcal{U}}(p(t))) \times \mathcal{O}(u^2).$$

In this article we propose an adaptative evaluation algorithm using the VS algorithm except when the relative error bound requires more accuracy. In this case, we apply either the de Casteljau algorithm or the compensated VS algorithm, presented in Section 4, depending on the computational cost. Since VS and de Casteljau algorithms require a polynomial represented in the Bernstein form, we devote the following section to efficient constructions of polynomials in this form.

3. EFFICIENT CONSTRUCTIONS OF POLYNOMIALS IN BERNSTEIN FORM

We have seen in the previous section that the most accurate algorithms for the evaluation of polynomials use the Bernstein representation of polynomials. Hence, all the three nonstraightforward constructors for polynomials we will provide in the software library aim to providing the coefficients $c_0, \dots, c_n$ of the polynomial $p(t) = \sum_{i=0}^{n} c_i b_i^n(t)$ with high accuracy. In this section, we present recent algorithms existing in the literature for constructing with high accuracy the Bernstein representation of a polynomial from its interpolation conditions and a new method for converting a polynomial in monomial form to its Bernstein representation. In Subsection 3.3 we show how to construct the polynomial in Bernstein form resulting from the division of two polynomials.

3.1. Recent algorithms for constructing the Bernstein representation of a polynomial from its interpolation conditions

Given the interpolation conditions let us construct the corresponding interpolation polynomial in Bernstein form. Given a sequence of parameters $\mathbf{t} = (t_i)_{0\leq i\leq n}$ verifying $0 < t_0 < t_1 < \cdots < t_n < 1$ and a sequence of points $q = (q_i)_{0\leq i\leq n}$, it is well known that there exists a unique $p(t) \in \mathcal{P}_n$ satisfying $p(t_i) = q_i$ for $0 \leq i \leq n$. The constructor we provide, given $\mathbf{t}$ and q, computes with high accuracy the coefficients vector $c = (c_0, c_1, \dots, c_n)$ such that the polynomial $p(t) = \sum_{i=0}^{n} c_i b_i^n(t)$ satisfies the interpolation conditions. Given a polynomial in the Bernstein form $p(t) = \sum_{i=0}^{n} c_i b_i^n(t)$, the interpolation conditions can be formulated as the following Bernstein-Vandermonde linear system of equations (BV linear system): $B(c_0, c_1, \dots, c_n)^T = (q_0, q_1, \dots, q_n)^T$, where $B = M\begin{pmatrix} b_0^n, b_1^n, \dots, b_n^n \\ t_0, t_1, \dots, t_n \end{pmatrix}$ is the collocation matrix of the basis $(b_0^n, b_1^n, \dots, b_n^n)$ at

$t_0, t_1, \ldots, t_n$. From now on, we will refer to a matrix like B as a Bernstein-Vandermonde (BV) matrix. A matrix is said to be totally positive (TP) if all its minors are nonnegative. It is well known that a nonsingular TP matrix and its inverse can be factorized as the product of bidiagonal matrices (see [Gasca and Peña 1996]). BV matrices are nonsingular stochastic TP matrices (see [Marco and Martínez 2007]). In [Koev 2007], assuming that bidiagonal decomposition of the inverse A^{-1} of a TP matrix A is given with high relative accuracy, Koev presents algorithm for solving linear systems of equations of the form $Ax = b$, with high accuracy. In [Koev 2013] we can get the library `TNTool` developed by Koev. This software is distributed under the GNU General Public License. The library implements in Matlab the algorithms in [Koev 2007] and [Koev 2005] for TP matrices. In particular, the function `TNSolve(C,b)` of this library, given the bidiagonal decomposition C of the inverse A^{-1} of a TP matrix A with high relative accuracy, returns the solution x of the linear system $Ax = b$ with high accuracy. So, we only need to obtain the bidiagonal decomposition of the inverse of a BV matrix with high relative accuracy in order to solve the above BV linear system of equations with high accuracy, obtaining in this way the corresponding interpolation polynomial in Bernstein form. In [Marco and Martínez 2007] a bidiagonal factorization of the inverse of a BV matrix is provided, as the following result recalls.

THEOREM 3.1. *Let B be a BV matrix. Then B^{-1} admits a factorization of the form $B^{-1} = G_1 G_2 \cdots G_n D^{-1} F_n F_{n-1} \cdots F_1$, where F_i (G_i^T, resp.) is the $(n+1)\times(n+1)$ lower triangular and bidiagonal matrix coinciding with the corresponding identity matrix, up to the entries $(i+1, i), (i+2, i+1), \ldots, (n+1, n)$, which are $-m_{i,i-1}, -m_{i+1,i-1}, \ldots, -m_{n,i-1}$ ($-\widetilde{m}_{i,i-1}, -\widetilde{m}_{i+1,i-1}, \ldots, -\widetilde{m}_{n,i-1}$, resp.) and $D = diag(p_{11}, p_{22}, \ldots, p_{n+1,n+1})$, with*

$$m_{ij} = \frac{(1-t_i)^{n-j}(1-t_{i-j-1})}{(1-t_{i-1})^{n-j+1}} \frac{\prod_{k=i-j}^{i-1}(t_i - t_k)}{\prod_{k=i-j-1}^{i-2}(t_{i-1} - t_k)}, \quad \textit{for } 0 \le j < i \le n,$$

$$\widetilde{m}_{ij} = \frac{n-i+1}{i} \frac{t_j}{1-t_j}, \quad \textit{for } 0 \le j < i \le n,$$

$$p_{ii} = \binom{n}{i} \frac{(1-t_i)^{n-i}}{\prod_{k=0}^{i-1}(1-t_k)} \prod_{k=0}^{i-1}(t_i - t_k), \quad \textit{for } 0 \le i \le n.$$

The elements $m_{ij}, \widetilde{m}_{ij}, p_{ii}$ can be computed with high relative accuracy, as was pointed out in [Marco and Martínez 2007], where the corresponding algorithm was proposed. The library `TNTool` contains the function `TNBDBV(t)` (contributed by J.-J. Martínez) implementing this algorithm, which provides the bidiagonal decomposition C of the BV matrix with nodes t with high relative accuracy. Then, the solution of the BV linear system can be computed by obtaining the bidiagonal decompositions of its coefficient matrix with `C=TNBDBV(t)` with $t = (t_0, \ldots, t_n)$ and then `c=TNSolve(C,q)`. The corresponding way of calling the constructor of our class for building up the interpolation polynomial is `polynomial(t,p)`. In this case, the constructor calls to the nonmember `interpolation(t,p)` function. `interpolation(t,p)` first calls the nonmember function `bd(t)`, which contains the implementation of `TNBDBV` and provides the bidiagonal decompositions of the correspondig BV matrix. Then, `interpolation` uses the code in `TNSolve` for the decomposition of the BV matrix to obtain the solution of the BV linear system.

3.2. A new method for converting a polynomial in monomial form to its Bernstein representation

The usual form for representing a polynomial uses the monomial basis. But, since the more accurate algorithms for the evaluation of polynomials use the Bernstein basis, it would be desirable to obtain a stable algorithm in order to construct a polynomial in monomial form to its Bernstein representation. It is well known (see [Farin 2002]) that

$$t^j = \sum_{k=j}^{n} \frac{\binom{k}{j}}{\binom{n}{j}} b_k^n(t), \quad \text{for } j = 0, 1, \ldots, n.$$

We can express the previous formula in matrix form as follows:

$$(1, t, \ldots, t^n) = (b_0^n(t) b_1^n(t), \ldots, b_n^n(t))A, \tag{6}$$

where $A = (a_{ij})_{1 \le i,j \le n+1}$ is the lower triangular matrix defined by $a_{ij} = \binom{i+j-2}{j-1}/\binom{n}{j-1}$ if $j \le i$ and by $a_{ij} = 0$ otherwise. Now we need an auxilliary result.

LEMMA 3.2. *The matrix A of (6) can be factorized in the following two ways:*

$$A = P_L D_2 \quad \textit{and} \quad A = D_1 \tilde{P}_L D_2,$$

where $D_1 = diag(2^0, 2^1, \ldots, 2^n)$, $D_2 = diag(1/\binom{n}{0}, 1/\binom{n}{1}, \ldots, 1/\binom{n}{n})$, $P_L = (\binom{i}{j})_{0 \le i,j \le n}$ is the lower triangular Pascal matrix and $\tilde{P}_L = (\binom{i}{j}/2^i)_{0 \le i,j \le n}$.

In Section 2 of [Alonso et al. 2013], a bidiagonal decomposition of the lower triangular Pascal matrix P_L was presented.

LEMMA 3.3. *(cf. Lemma 1 of [Alonso et al. 2013].) The lower triangular Pascal matrix $P_L = (\binom{i}{j})_{0 \le i,j \le n}$ admits the following bidiagonal matrix factorization:*

$$P_L = F_n \ldots F_1,$$

where F_i is the $(n+1) \times (n+1)$ matrix with 1's in the $(i+1, i), (i+2, i+1), \ldots, (n+1, n)$ and in the diagonal entries.

The following result presents the bidiagonal factorization of the matrix $\tilde{P}_L$ of Lemma 3.2.

LEMMA 3.4. *The triangular matrix $\tilde{P}_L = D_1^{-1} P_L = (\binom{i}{j}/2^i)_{0 \le i,j \le n}$, where $D_1 = diag(2^0, 2^1, \ldots, 2^n)$, admits the following bidiagonal matrix factorization:*

$$\tilde{P}_L = \tilde{F}_n \ldots \tilde{F}_1,$$

where $\tilde{F}_i$ is the $(n+1) \times (n+1)$ matrix with 1's in the diagonal entries $(1, 1), \ldots, (i, i)$ and 1/2's in $(i+1, i), (i+2, i+1), \ldots, (n+1, n)$ and $(i+1, i+1), (i+2, i+2), \ldots, (n+1, n+1)$ entries.

PROOF. Let us prove by induction on $i = 0, 1, \ldots, n-1$ that

$$D_1 \tilde{F}_n \tilde{F}_{n-1} \cdots \tilde{F}_{n-i} = F_n F_{n-1} \cdots F_{n-i} D_{1,i} \tag{7}$$

where $D_{1,i}$ is the $(n+1) \times (n+1)$ matrix diag$(2^0, 2^1, \ldots, 2^{n-2-i}, 2^{n-1-i}, \ldots, 2^{n-1-i})$ and the matrices F_i are defined in Lemma 3.3. For $i = 0$ we have that $D_1 \tilde{F}_n = F_n D_{1,0}$. Now let us assume that formula (7) holds for $i \in \{0, 1, \ldots, n-2\}$ and let us prove that it also holds for $i+1$. For this purpose, let us calculate $D_1 \tilde{F}_n \tilde{F}_{n-1} \cdots \tilde{F}_{n-1-i}$. By the induction hypothesis we have that

$$D_1 \tilde{F}_n \cdots \tilde{F}_{n-i} \tilde{F}_{n-1-i} = (D_1 \tilde{F}_n \cdots \tilde{F}_{n-i}) \tilde{F}_{n-1-i} = F_n F_{n-1} \cdots F_{n-i} D_{1,i} \tilde{F}_{n-1-i}. \tag{8}$$

So, now we only need to check that $D_{1,i} \tilde{F}_{n-1-i} = F_{n-1-i} D_{1,i+1}$ and the induction follows. By formula (7) for $i = n-1$, taking into account that $D_{1,n-1}$ is the identity matrix, and by Lemma 3.3, we have that $D_1\tilde{F}_n \cdots \tilde{F}_1 = F_n \cdots F_1 = P_L$. Hence, we conclude that $\tilde{F}_n \cdots \tilde{F}_1 = D_1^{-1}P_L = \tilde{P}_L$. $\square$

Taking into account the results in lemmas 3.2, 3.3 and 3.4, we can obtain bidiagonal factorizations of matrix A.

THEOREM 3.5. *The matrix* $A = (\binom{i}{j}/\binom{n}{j})_{0\le i,j\le n}$ *admits the following factorization:* $A = D_1\tilde{F}_n \ldots \tilde{F}_1 D_2$, *where the matrices* $\tilde{F}_i$ *are given in Theorem 3.4,* $D_1 = diag(2^0, 2^1, \ldots, 2^n)$ *and* $D_2 = diag(1/\binom{n}{0}, 1/\binom{n}{1}, \ldots, 1/\binom{n}{n})$.

We now include the conversion algorithm associated to the factorization of Theorem 3.5.

Algorithm 2 *MONOMIAL2BERNSTEIN algorithm* for the conversion of a polynomial in monomial form to its Bernstein representation

```
Require: $(d_i)_{i=0}^n$
Ensure: $(c_i)_{i=0}^n$ such that $\sum_{i=0}^n d_i m_i^n(t) \approx \sum_{i=0}^n c_i b_i^n(t)$
  for $i = 0$ to $n$ do
    $c_i = d_i \oslash \binom{n}{i}$
  end for
  for $r = 1$ to $n$ do
    for $i = 1$ to $n + 1 - r$ do
      $c_{n+1-i} = \frac{1}{2} \otimes c_{n-i} \oplus \frac{1}{2} \otimes c_{n+1-i}$
    end for
  end for
  for $i = 1$ to $n$ do
    $c_i = c_i \otimes 2^i$
  end for
```

Remark 3.6. Observe that the product $\tilde{F}_n \cdots \tilde{F}_1$ in Theorem 3.5 (corresponding to the steps of Algorithm 2 up to the first and last step) is a product of stochastic bidiagonal matrices. This implies that the associated operations are convex combinations and so they are very stable. Let us justify it. The growth factor of a numerical algorithm is usually defined as the quotient between the maximal absolute value of all the elements that occur during the execution of the algorithm and the maximal absolute value of all the initial data. It is well known that the growth factor is a stability indicator of the algorithm. Since each elementary step of the corner cutting algorithm is a convex combination of two numbers previously computed, the growth factor is optimal (that is, 1) and overflow is avoided.

The previous algorithm has been implemented as `monomial2Bernstein(d)` Matlab function, which is called from the constructor of the class when using the command `Polynomial(d,'m')`, where `'m'` means monomial, or shorter `Polynomial(d)`.

The next example compares the results obtained when evaluating a polynomial represented in its monomial form by the usual Horner algorithm versus evaluating it by de Casteljau and VS algorithms after converting the polynomial into its Bernstein form through Algorithm 2

EXAMPLE 3.7. *Let us consider the polynomial (see p. 753 of [Jiang et al. 2010])* $p(t) = (t - 3/4)^7 (t - 1)$ *We evaluate this polynomial by the usual Horner algorithm with double precision at* 400 *points equally distributed between* 0.70005 *and* 0.79995*, both included. In addition, we convert the polynomial above into its Bernstein form*

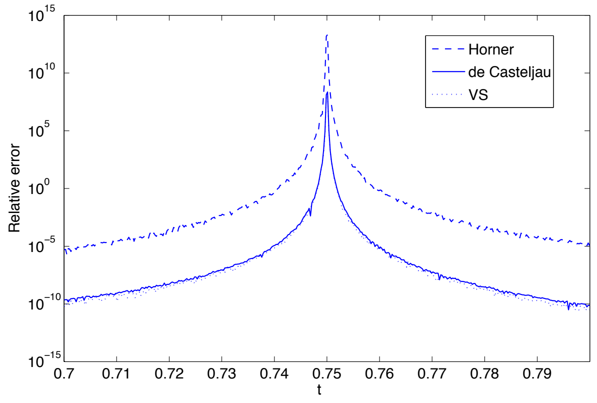

Fig. 1. Errors when evaluating $p(t)$

by Algorithm 2 and then we evaluate it by the de Casteljau and VS algorithms. We also obtain the exact value of the polynomial at those 400 *points by using a symbolic computation software and compute the relative errors of the three previous procedures for the evaluation. Figure 1 shows the relative errors.*

We can observe that, in spite of the conversion from monomial basis to Bernstein basis, de Casteljau and VS algorithms provide in general better aproximations of the values of the polynomial than the Horner algorithm.

3.3. Constructing the quotient polynomial of a division in Bernstein form

In the public member function `mrdivide(obj1,obj2)`, we overload the operator / in order to divide polynomials in Bernstein form taking into account the algorithm in [Tsai and Farouki 2001]. As we have mentioned in the introduction, that algorithm can present an undesired behaviour for polynomials whose true degree is lower than the degree of the Bernstein basis used in its representation. This algorithm provides the quotient and the remainder of the division by solving a linear system of equations. Given a polynomial in the monomial basis, its exact degree is obviously determined. However, in some occasions, a polynomial represented in the Bernstein basis of n degree can, in fact, be represented exactly in a Bernstein basis of degree lower than n. For example, $t = \sum_{i=0}^{n}(i/n)b_i^n(t)$ for any integer $n \geq 1$. In particular, if we perform the division $\sum_{i=0}^{4}(i/4)b_i^4(t)/\sum_{i=0}^{3}(i/3)b_i^3(t)$ with the algorithm in [Tsai and Farouki 2001] the coefficient matrix of the corresponding linear system is singular:

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 1/3 & 0 & 2/3 & 1/3 \\ 1/3 & 1/3 & 1/3 & 2/3 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

This fact can occur when the exact degree of some of the two polynomials involved in the division is lower than the degree of the basis used to represent it. In these cases the linear systems have infinite solutions. Depending on the version of Matlab we are using, we can either get one of these solutions arbitrarily or get no solution. In order to solve this problem, the degree of the Bernstein bases used in the representations has to be reduced to the true exact degree of the polynomial. So, we have included the member function `degreeReduction()`, which first checks the true exact degree of the polynomial and then returns the same polynomial expressed in the Bernstein basis of least degree possible. To check the true exact degree of a polynomial in Bernstein form $p(t)=\sum_{i=0}^{n} c_i b_i^n(t)$, we have used the first formula of Property 14 of Section 5.1 of [Farouki 2012] (page 392). So, the true exact degree of $p(t)$ is $n-r$ for some $r\geq 1$ if and only if $\Delta^{n-r}c_0\neq 0$ and $\Delta^k c_0=0$ for $k=n-r+1,\ldots,n$, where Δ^k is the k-th progressive difference. In the case of existing such an $r\geq 1$, using the second formula of Property 14 of Section 5.1 of [Farouki 2012], we can express $p(t)$ like $p(t)=\sum_{i=0}^{n-r} c_i^{n-r} b_i^{n-r}(t)$ where

$$c_i^{n-r}=\sum_{j=0}^{i}(-1)^{i-j}\frac{\binom{i-j+r-1}{r-1}\binom{n}{j}}{\binom{n-r}{i}}c_j.$$

Therefore, in `mrdivide(obj1,obj2)` we apply first the degree reduction function to the two polynomials `obj1` and `obj2` in order to avoid the previous problems and so, the remainder with true exact degree less than the degree of polynomial `obj2` and the correspondig quotient are obtained.

4. VS COMPENSATED ALGORITHM

This section presents the computationally efficient VS algorithm, shows how to compensate its rounding errors and includes a dynamic error estimate.

4.1. The Volk-Schumaker algorithm

In [Schumaker and Volk 1986] a nested type algorithm for the evaluation of bivariate polynomials of total degree n was presented. This algorithm has been adapted for the evaluation of polynomials with linear time complexity. The VS algorithm evaluates polynomials in $\mathcal{P}_n$ represented in the VS basis z_n with $\mathcal{O}(n)$ computational cost. Algorithm 3 shows the VS algorithm. This algorithm has been implemented as a usual Matlab function `[y,errBound]=Vs(coeff,x)`, with `coeff` a vector with the coefficients of the corresponding polynomial with respect to the Bernstein basis, `x` the point or vector of points where the polynomial is evaluated at. The function, in addition to the approximated values of the polynomial at the points in x, also returns realistic upper bounds on the corresponding relative errors through the relative running error bounds in Theorem 2.2.

4.2. Error-free transformations

In our algorithm we shall use the error-free trasformations `TwoSum` and `TwoProduct` (see [Ogita et al. 2005]) for computing sums and products. In [Knuth 1998] Knuth introduced the algorithm `TwoSum` for the summation, whereas in [Dekker 1971] Dekker presented the algorithm `TwoProduct`, due to G.W. Veltkamp, for the product. On the other hand, for computing quotients we shall use the error-free transformation `DivRem` introduced in [Pichat and Vignes 1993]. Algorithms 6, 4 and 7 show these three algorithms (`TwoSum`, `TwoProduct` and `DivRem`), which have been included in the software library as usual Matlab functions.

Error analyses of these three algorithms have been shown in Theorem 3.4 of [Ogita et al. 2005] and Théorème 3.14 of [Louvet 2007]. The following theorem summarizes these results.

THEOREM 4.1. *Let $\mathbb{F}$ be the set of standard floating point numbers corresponding to a certain floating point arithmetic. If $a,b\in\mathbb{F}$, then:*

i. $[x,y]=\mathtt{TwoSum}(a,b)$ *satisfies*

$$a+b=x+y,\quad x=a\oplus b,\quad |y|\leq u|x|,\quad |y|\leq u|a+b|.$$

Algorithm 3 *VS algorithm* for the evaluation of a polynomial p at a point t

```
Require: $t \in [0,1]$ and $(c_i)_{i=0}^n$
Ensure: $res \approx \sum_{i=0}^n c_i z_i^n(t)$
  if $t \geq 1/2$ then
    $q = (1 \ominus t) \oslash t$
    $p_0 = c_0$
    for $i = 1$ to $n$ do
      $p_i = p_{i-1} \otimes q \oplus c_i$
    end for
    $f_n = p_n$
    for $i = 1$ to $n$ do
      $f_{n-i} = f_{n+1-i} \otimes t$
    end for
    $res = f_0$
  else
    $q = t \oslash (1 \ominus t)$
    $p_n = c_n$
    for $i = 1$ to $n$ do
      $p_{n-i} = p_{n+1-i} \otimes q \oplus c_{n-i}$
    end for
    $f_0 = p_0$
    for $i = 1$ to $n$ do
      $f_i = f_{i-1} \otimes (1 - t)$
    end for
    $res = f_n$
  end if
```

Algorithm 4 *TwoProduct algorithm*

```
Require: $a, b$
Ensure: $[x, y]$ such that $x + y = a \cdot b$
  1: $x = a \otimes b$
  2: $[a_1, a_2] = Split(a)$
  3: $[b_1, b_2] = Split(b)$
  4: $y = a_2 \otimes b_2 \ominus (((x \ominus a_1 \otimes b_1) \ominus a_2 \otimes b_1) \ominus a_1 \otimes b_2)$
```

Algorithm 5 *Split algorithm*

```
Require: $a$
Ensure: $[x, y]$ such that $x + y = a$
  1: $c = \texttt{factor} \otimes a$    $\%\texttt{factor} = 2^{27} + 1$ in IEEE 754
  2: $x = c \ominus (c \ominus a)$
  3: $y = a \ominus x$
```

Algorithm 6 *TwoSum algorithm*

```
Require: $a, b$
Ensure: $[x, y]$ such that $x + y = a + b$
  $x = a \oplus b$
  $z = x \ominus a$
  $y = (a \ominus (x \ominus z)) \oplus (b \ominus z)$
```

Algorithm 7 *DivRem algorithm*

```
Require: $a, b$
Ensure: $[q, r]$ such that $a = b \cdot q + r$
  $q = a \oslash b$
  $[x, y] = \texttt{TwoProduct}(q, b)$
  $r = (a \ominus x) \ominus y$
```

ii. $[x, y] = \texttt{TwoProduct}(a, b)$ *satisfies, if not underflow occurs,*

$$a \cdot b = x + y, \quad x = a \otimes b, \quad |y| \leq u|x|, \quad |y| \leq u|a \cdot b|.$$

iii. $[q, r] = DivRem(a, b)$ *satisfies, if not underflow occurs,*

$$a = b \cdot q + r, \quad q = a \oslash b, \quad |r| = u|b \cdot q|, \quad |r| \leq u|a|.$$

4.3. Compensated VS algorithm with a dynamic error estimate

We now present the main result of this section: a compensated VS algorithm with a dynamic error estimate (see Algorithm 8). Algorithm 8 has been implemented in the library as the usual Matlab function `CompVs(coeff,x)`, where `coeff` is a vector with the coefficients of the polynomial to be evaluated with respect to the Bernstein basis and `x` is the point or vector of points where the polynomial is evaluated at.

Given $k \in \mathbf{N}_0$ such that $ku < 1$, let us define $\gamma_k := ku/(1 - ku) = ku + \mathcal{O}(u^2)$, where u is the unit roundoff.

THEOREM 4.2. *Let* $p(t) = \sum_{i=0}^{n} c_i \, z_i^n(t) \in \mathcal{P}_n$ *and* $t \in [0, 1]$. *Then, when Algorithm 8 is performed in floating point arithmetic with unit roundoff* u *we obtain*

i) if $t \geq 1/2$

$$p(t) = VS(t, (c_0, \ldots, c_n)) + \left[t \times p^M(t) + p_\alpha^M(t)\right],$$

where p^M *and* p_α^M *are* $n-1$ *degree polynomials given by*

$$p^M(t) = \sum_{i=0}^{n-1} l_{i+1}^M t^i (1-t)^{n-1-i} \quad \textit{and} \quad p_\alpha^M(t) = \sum_{i=0}^{n-1} \alpha_i^M \, t^i,$$

with $l_{i+1}^M = ((\rho + \beta^M)/t) \times p_i + \pi_{i+1}^M + \sigma_{i+1}^M$, *and if* $t < 1/2$

$$p(t) = VS(t, (c_0, \ldots, c_n)) + [(1-t) \times p^m(t) + p_\alpha^m(t)]$$

where p^m *and* p_α^m *are* $n-1$ *degree polynomials given by*

$$p^m(t) = \sum_{i=0}^{n-1} l_i^m t^i (1-t)^{n-1-i} \quad \textit{and} \quad p_\alpha^m(t) = \sum_{i=0}^{n-1} \alpha_i^m \, t^i,$$

with $l_i^m = ((\beta^m - \rho \times q)/(1-t))p_{i+1} + \pi_i^m + \sigma_i^m$.

ii) a floating point approximation res *to the exact value* $p(t)$ *where*

$$|p(t) - res| \leq \gamma_2 \, |p(t)| + 4\gamma_{4n}^2 \tilde{p}(t) \quad \textit{and} \quad \frac{|p(t) - res|}{|p(t)|} \leq \gamma_2 + 4\gamma_{4n}^2 S_{b_n}(p(t)).$$

if no underflow occurs.

The proof of the previous theorem can be seen in the accompanying electronic appendix.

Remark 4.3. Taking into account (3) and that $\gamma_k = k\,u + \mathcal{O}(u^2)$, we can deduce the following linear approximations of the bounds of the relative errors in Theorem 4.2 ii):

$$\frac{|p(t) - res|}{|p(t)|} \lesssim 2u + 64n^2u^2 \frac{fl(\tilde{p}(t))}{|res|}$$

if $|res| \gtrsim 2u\,|res| + 64n^2u^2 \, fl(\tilde{p}(t))$.

In the Matlab function implementing the compensated VS algorithm, the computation of the dynamic estimate has also been included. Although rounding errors have not been taken into account for the linear approximation to the dynamic estimate, we can observe that its computation is substraction-free and so it is computed with high relative accuracy. However, evaluation is affected by the truncation due to the linear approximation.

Algorithm 8 *VS compensated algorithm* for the evaluation of $p \in \mathcal{P}_n$ at t

```
Require: $t \in [0,1]$ and $(c_i)_{i=0}^n$
Ensure: $res \approx \sum_{i=0}^n c_i z_i^n(t)$
  $[r,\rho] = TwoSum(1,-t)$
  if $t \geq 1/2$ then
    $[q,\beta^M] = DivRem(r,t)$
    $p_0 = c_0$
    for $i = 1$ to $n$ do
      $[a_i,\pi_i^M] = TwoProd(q,p_{i-1})$
      $[p_i,\sigma_i^M] = TwoSum(a_i,c_i)$
      $l_i^M = ((\rho \oplus \beta^M) \oslash t) \otimes p_{i-1} \oplus \pi_i^M \oplus \sigma_i^M$
    end for
    $f_n = p_n$
    for $i = 1$ to $n$ do
      $[f_{n-i},\alpha_{n-i}^M] = TwoProd(f_{n+1-i},t)$
    end for
    $res = f_0 \oplus t \otimes VS(t,(l_1^M,\ldots,l_n^M)) \oplus Horner(t,(\alpha_0^M,\alpha_1^M,\ldots,\alpha_{n-1}^M))$
  else
    $[q,\beta^m] = DivRem(t,r)$
    $p_n = c_n$
    for $i = 1$ to $n$ do
      $[a_{n-i},\pi_{n-i}^m] = TwoProd(q,p_{n+1-i})$
      $[p_{n-i},\sigma_{n-i}^m] = TwoSum(a_{n-i},c_{n-i})$
      $l_{n-i}^m = ((\beta^m \ominus \rho \otimes q) \oslash (1 \ominus t)) \otimes p_{n+1-i} \oplus \pi_{n-i}^m \oplus \sigma_{n-i}^m$
    end for
    $f_0 = p_0$
    for $i = 1$ to $n$ do
      $[f_i,\alpha_i^m] = TwoProd(f_{i-1},1 \ominus t)$
    end for
    $res = f_n \oplus r \otimes VS(t,(l_0^m,\ldots,l_{n-1}^m)) \oplus Horner(t,(\alpha_n^m,\ldots,\alpha_1^m))$
  end if
```

5. NUMERICAL EXPERIMENTS AND THE ADAPTATIVE EVALUATION ALGORITHM

This section includes numerical experiments that motivate our polynomial adaptative evaluation algorithm presented at the end.

5.1. Numerical experiments

In Example 3.7 we have seen that the de Casteljau algorithm can provide more accurate evaluations than Horner algorithm even if the polynomial is given in its monomial form and it must be converted into its Bernstein form before being evaluated through the de Casteljau algorithm. In fact, if the polynomial is given in its Bernstein form, the evaluations provided by the de Casteljau algorithm are much more accurate than the ones provided by the Horner algorithm (see [Delgado and Peña 2009]). In order to propose an efficient evaluation algorithm we must study the performance of de Casteljau, compensated de Casteljau, VS and compensated VS algorithms. First, let us compare these four algorithms by evaluating the polynomials $p(t)$ in Example 3.7 and $q(t) = p(1-t)$.

EXAMPLE 5.1. *We have evaluated the polynomials* $p(t) = (t-3/4)^7\,(1-t)$ *and* $q(t) = (t-1/4)^7\,t$ *at* 400 *points equally distributed in* $[0.74995, 0.75005]$ *and* $[0.24995, 0.25005]$*, respectively, by the de Casteljau, VS algorithm and their compensated versions in dou-*

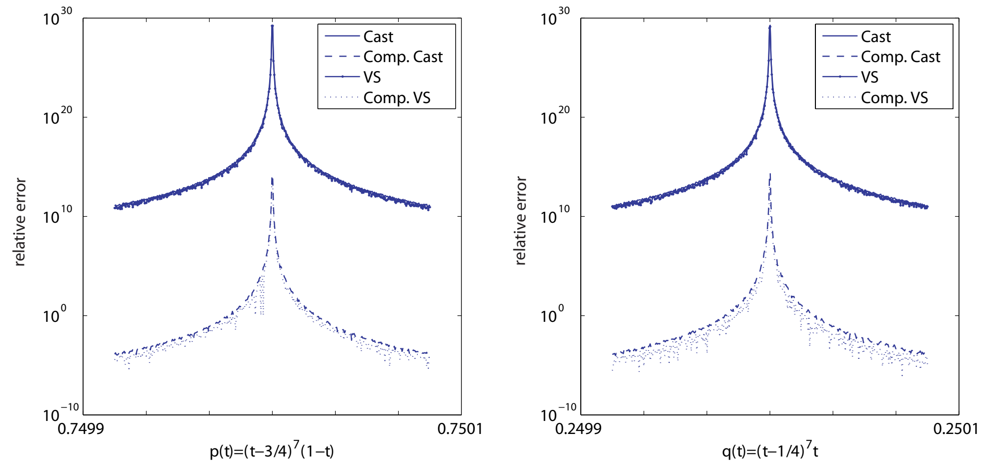

Fig. 2. Errors when evaluating $p(t)$ and $q(t)$

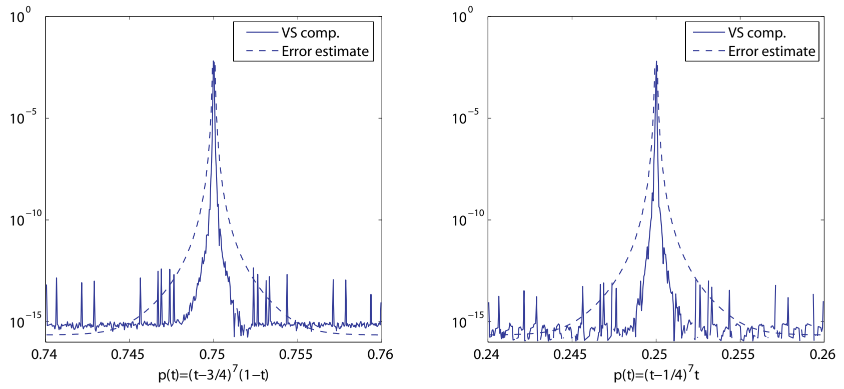

Fig. 3. Estimate of the relative errors when evaluating $p(t)$ and $q(t)$

ble precision. Then we have obtained the exact values of the polynomials at those points and we have computed the corresponding relative errors. Figure 2 shows the relative errors corresponding to the four algorithms. We can observe that the compensated versions of VS and de Casteljau algorithms behave much better than the usual VS and de Casteljau algorithms. In fact, both compensated versions of the VS and de Casteljau algorithms provide errors of the same magnitude as we can see in the figure. In turn, the curves of the errors for VS and de Casteljau algorithms overlap.

We have also evaluated both polynomials at 400 *points equally distributed in* $[0.74005, 0.75995]$ *and* $[0.24005, 0.25995]$*, respectively, by the compensated VS algorithm in double precision, obtaining the linear approximation of the bound presented in Remark 4.3, which can be considered as an estimate of the error because of the truncation and rounding errors. We have also obtained the exact values of the polynomials at those points and we have computed the corresponding relative errors. In Figure 3 we show the relative errors and the corresponding estimate for these errors. We can observe that the estimate of the errors present a better behaviour at the points where the polynomials are worse conditioned. In turn, at these points compensated algorithms also present the greatest improvements on the accuracy over their not compensated versions.*

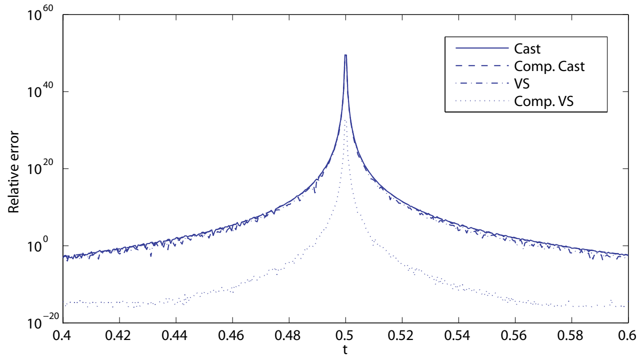

Fig. 4. Relative errors when evaluating $r(t)$

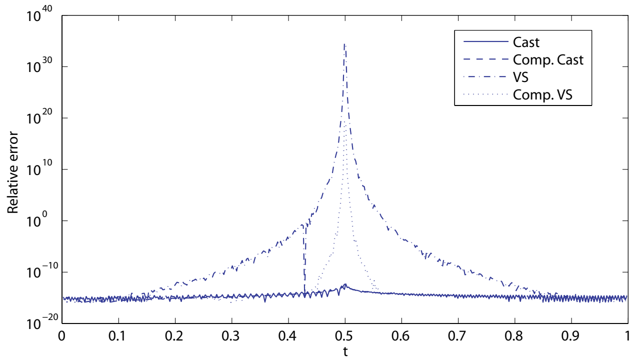

Fig. 5. Relative errors when evaluating $s(t)$

We have seen in the previous example that the compensated version of an algorithm provides more accurate results than the not compensated version with the drawback of a greater computational cost. The example also shows that both compensated VS algorithm and compensated de Casteljau algorithm have a very similar behaviour with respect to error. Although this is the most frequent behaviour, for some few polynomials this fact can vary. Let us see a couple of these examples.

EXAMPLE 5.2.

a) We have evaluated the polynomial $r(t) = \left(t - \frac{1}{2}\right)^{20} t$ *at* 400 *points equally distributed in* $[0.40005, 0.59995]$ *by the de Casteljau, VS algorithm and their compensated versions in double precision. Then we have obtained the exact values of the polynomials at those points and we have computed the corresponding relative errors. Figure 4 shows these errors. We can observe that de Casteljau, its compensated version and VS algorithms have a very similar behaviour with respect to the error, whereas the compensated VS algorithm provides the best approximations.*

b) We have evaluated the polynomial $s(t) = (t - 1/2)^{20}$ *at* 400 *points equally distributed in* $[0, 1]$ *by the de Casteljau, VS algorithm and their compensated versions in double precision. Then we have obtained the exact values of the polynomials at those points and we have computed the corresponding relative errors. Figure 5 shows the relative errors. We can observe that the de Cateljau algorithm and its compensated version have a good and similar behaviour with respect to error, whereas the compensated VS algorithm also provides very accurate approximations except at points very close to*

Table I. Errors and execution times for random polynomials

		Degree				
		10	20	30	40	50
Cast.	Time (sec)	4.4057e-03	1.5202e-02	3.3820e-02	5.6417e-02	8.8246e-02
	Mean rel. error	2.0558e-15	4.3251e-15	4.4042e-15	8.0022e-15	1.3028e-14
	Max. rel. error	7.8412e-15	2.3901e-14	1.0987e-14	2.4916e-14	8.2453e-14
comp. Cast.	Time (sec)	1.8609e-02	7.0303e-02	1.5848e-01	2.7072e-01	4.2927e-01
	Mean rel. error	5.4403e-16	8.2449e-16	6.4405e-16	5.2037e-16	8.3408e-16
	Max. rel. error	5.7845e-15	7.8514e-15	9.5099e-15	2.9006e-15	5.9944e-15
VS	Time (sec)	1.7140e-03	2.9853e-03	4.4099e-03	5.5108e-03	6.9082e-03
	Mean rel. error	1.2956e-15	1.7470e-15	3.4802e-15	3.0818e-15	4.6449e-15
	Max. rel. error	2.9565e-15	4.2721e-15	8.5307e-15	1.1587e-14	1.1329e-14
comp. VS	Time (sec)	1.2789e-02	2.1592e-02	3.1637e-02	3.9230e-02	4.9360e-02
	Mean rel. error	7.9047e-16	1.5601e-15	1.7146e-15	2.3832e-15	2.5049e-15
	Max. rel. error	5.0133e-15	9.6988e-15	7.2205e-15	6.1460e-15	7.1527e-15

0.5*, which is the root of multiplicity* 20 *of the polynomial. On the other hand, the VS algorithm has not an acceptable behaviour.*

The next example compares errors and running times for random polynomials.

EXAMPLE 5.3. *In this example we have considered* 100 *polynomials of degree* 10*,* 50 *polynomials of degree* 20*,* 40 *polynomials of degree* 30*,* 30 *polynomials of degree* 40 *and* 20 *polynomials of degree* 50*, all of them represented in the corresponding Bernstein basis. The coefficients of all the polynomials with respect to the Bernstein basis have been generated randomly as integers in the interval* $[-100, 100]$ *according to a uniform distribution. Then we have evaluated* 20 *times each one of the polynomials at* 21 *points uniformly distributed in* $[0, 1]$ *by the de Casteljau and VS algorithms, and their corresponding compensated versions. We have measured the execution time for each one of the algorithms when evaluating all the polynomials of a fixed degree at the* 21 *points considered for* 20 *times. Then, we have divided the time obtained by* 20 *and by the number of generated polynomials for the corresponding degree. We have also computed the corresponding relative errors. Table I shows the measured times, the mean of relative errors and the maximal relative error for each of the degrees considered.*

5.2. The adaptative evaluation algorithm

As a conclusion from the previous examples and many more, we can say that, in general, VS and de Casteljau algorithms provide similar approximations with respect to accuracy. The same occurs with its compensated versions. In general, the de Caslteljau and VS algorithms provide accurate enough approximations except for very ill-conditioned polynomials, where their compensated versions can be very useful. So, in order to decide how to evaluate a polynomial we must analyze the computational cost of the four evaluation algorithms. The de Casteljau algorithm evaluates a n degree polynomial using $n(n+1)/2+1$ sums and $n(n+1)$ products (quadratic complexity), whereas the VS algorithm uses $n+1$ sums, $2n$ products and 1 quotient (linear complexity). So, taking into account that routine `TwoSum` precises of 6 sums, routine `TwoProduct` of 6 products and 11 sums and, routine `DivRem` of 6 products, 13 sums and 1 quotient, the compensated de Casteljau algorithm performs $(33/2)n(n+1)+7$ sums and $(15/2)n(n+1)$ products, that is a total of $24n(n+1)+7$ elementary operations. Analogously, the compensated VS algorithm consists of $33n+20$ sums, $16n+4$ products and $n+2$ quotient, that is, a total of $50n+26$ operations, for the evaluation of a n degree polynomial.

Although we have observed in the previous examples that VS and de Casteljau algorithms can have a bad behavior with respect to the error, this phenomenon is not usual, but only bounded to very ill-conditioned polynomials. So, first we will evaluate polynomials by the VS algorithm since it has a lower computational cost than the de

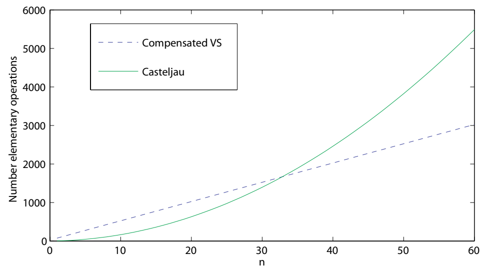

Fig. 6. Computational cost of compensated VS and de Casteljau algorithms

Casteljau algorithm. With the error bound we will decide whether the obtained approximations are accurate enough or not. For the points where the error test is not satisfactory we have three alternatives: de Casteljau algorithm or some compensated version of VS and de Casteljau algorithms. The compensated version of the de Casteljau algorithm is very expensive computationally and, taking into account that in most of the cases it provides results with an accuracy very similar to those provided by the compensated VS algorithm, we rule out. In the previous example we have seen that VS and de Calteljau have a very similar behaviour and probably we would expect it since both algorithms use representations sharing the same condition number. But for a few polynomials the de Casteljau algorithm can provide more accurate evaluations since it is formed by linear convex combinations and has a optimal growth factor of 1 (see [Delgado and Peña 2009] for more details). So we must decide between the de Casteljau and the compensated VS algorithm. In examples 5.1 and 5.2, we have seen that the compensated VS algorithm can outperform the de Casteljau algorithm. In general, for ill-conditioned polynomials, we can expect more accurate results from the compensated VS algorithm, but for low degrees VS compensated algorithm has a greater computational cost than de Casteljau algorithm. We can check that for degrees $n \leq 32$ de Casteljau algorithm has a lower computational cost than the corresponding to the compensated VS algorithm, whereas for other degrees the compensated VS algorithm is more efficient (in Figure 6 we have drawn the degree of the polynomial versus the number of elementary operations for both algorithms). Execution times of Table I confirms this theoretical observation on the computational costs. Taking into account that most of the polynomials are not ill-conditioned, if $n \geq 33$ we evaluate the polynomials by the compensated VS algorithm, and otherwise, first we evaluate them by the de Casteljau algorithm and only the points where the corresponding test error is not satisfactory are evaluated by the compensated VS algorithm. Algorithm 9 has implemented this idea with an adaptative algorithm.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

ACKNOWLEDGMENTS

Research partially supported the Spanish Research Grant MTM2012-31544 and by Gobierno de Aragón and Fondo Social Europeo. The authors thank the valuable suggestions provided by anonymous referees, which have helped to improve the article.

REFERENCES

P. Alonso, J. Delgado, R. Gallego, and J. M. Peña. 2013. Conditioning and accurate computations with Pascal matrices. *J. Comput. Appl. Math.* 252 (2013), 21–26.

Algorithm 9 *Eval algorithm* for the adaptative evaluation of $p \in \mathcal{P}_n$ at t

```
Require: $t \in [0,1]$, $\mathbf{c} = (c_i)_{i=0}^n$ and $Tol$
Ensure: $res \approx \sum_{i=0}^n c_i b_i^n(t)$
  $[res, errBound] = VS(\mathbf{c}, t)$
  if $errBound > Tol$ then
    if $n \leq 32$ then
      $[res, errBound] = Cast(\mathbf{c}, t)$
      if $errBound > Tol$ then
        $[res, errBound] = CompVs(\mathbf{c}, t)$
      end if
    else
      $[res, errBound] = CompVs(\mathbf{c}, t)$
    end if
  end if
```

T. J. Dekker. 1971. A floating-point technique for extending the available precision. *Numer. Math.* 18 (1971), 224–242.

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

J. Delgado and J. M. Peña. 2006. Error Analysis of an Effient Alternative to the De Casteljau Algorithm. *International Journal of Computer Graphics and CAD/CAM* 1 (2006), 1–15.

J. Delgado and J. M. Peña. 2009. Running Relative Error for the Evaluation of Polynomials. *SIAM Journal on Scientific Computing* 31 (2009), 3905–3921.

J. Demmel, M. Gu, S. Eisenstat, I. Slapničar, K. Veselić, and Z. Drmač. 1999. Computing the singular value decomposition with high relative accuracy. *Linear Algebra Appl.* 299, 1-3 (1999), 21–80. DOI:http://dx.doi.org/10.1016/S0024-3795(99)00134-2

G. Farin. 2002. *Curves and Surfaces for Computer Aided Geometric Design* (fifth ed.). Academic Press, Inc., San Diego, CA.

R. T. Farouki. 2012. The Bernstein polynomial basis: A centennial retrospective. *Computer Aided Geometric Design* 29 (2012), 379–419.

R. T. Farouki and T. N. T. Goodman. 1996. On the optimal stability of the Bernstein basis. *Math. Comp.* 65 (1996), 1553–1566.

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

M. Gasca and J. M. Peña. 1996. *Total Positivity and Its Applications*. Kluver Academic Publishers, Dordrecht, The Netherlands, Chapter On factorizations of totally positive matrices, 109–130.

N. J. Higham. 2002. *Accuracy and Stability of Numerical Algorithms* (second ed.). SIAM, Philadelphia.

H. Jiang, S. Li, L. Cheng, and F. Su. 2010. Accurate evaluation of a polynomial and its derivative in Bernstein form. *Computers & Mathematics with Applications* 60 (2010), 744–755.

D. E. Knuth. 1998. *The Art of Computer Programming: Seminumerical Algorithms, volume 2* (third ed.). Addison-Wesley, Reading, MA, USA.

P. Koev. 2005. Accurate Eigenvalues and SVDs of Totally Nonnegative Matrices. *SIAM J. Matrix Anal. Appl.* 27 (2005), 1–23.

P. Koev. 2007. Accurate Computations with totally nonnegative matrices. *SIAM J. Matrix Anal. Appl.* 29 (2007), 731–751.

P. Koev. 2013. (2013). http://math.mit.edu/~plamen/software/TNTool.html

PH. Langlois and N. Louvet. 2007. How to Ensure a Faithful Polynomial Evaluation with the Compensated Horner Algorithm. In *ARITH '07 Proceedings of the 18th IEEE Symposium on Computer Arithmetic*. 141–149.

PH. Langlois, N. Louvet, and S. Graillat. 2005. *Compensated Horner Scheme*. Technical Report RR2005-04. Université de Perpignan Via Domitia.

N. Louvet. 2007. *Algorithmes compensés en arithmétique flottante: précision, validation, performances*. Ph.D. Dissertation.

T. Lyche and J. M. Peña. 2004. Optimally stable multivariate bases. *Advances in Computational Mathematics* 20 (2004), 149–159.

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

A. Marco and J. J. Martínez. 2007. A fast and accurate algorithm for solving Bernstein-Vandermonde linear systems. *Linear Algebra Appl.* 422 (2007), 616–628.

T. Ogita, S. M. Rump, and S. Oishi. 2005. Accurate Sum and Dot Product. *SIAM Journal on Scientific Computing* 26 (2005), 1955–1988.

J. M. Peña. 2002. On the optimal stability of bases of univariate functions. *Numer. Math.* 91 (2002), 305–318.

J. M. Peña. 2006. A note on the optimal stability of bases of univariate functions. *Numer. Math.* 103 (2006), 151–154.

M. Pichat and J. Vignes. 1993. *Ingénierie du contrôle de la précision des calculs sur ordinateur*. Editions Technip.

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

Y. F. Tsai and R. T. Farouki. 2001. Algorithm 812: BPOLY: An Object-Oriented Library of Numerical Algorithms for Polynomials in Bernstein Form. *ACM Trans. Math. Software* 27 (2001), 267–296.

Online Appendix to: Algorithm ???: POLYNOMIAL: An object-oriented Matlab library of fast and efficient algorithms for polynomials

JORGE DELGADO, Universidad de Zaragoza
and JUAN MANUEL PEÑA, Universidad de Zaragoza

A. SKETCH OF THE PROOF OF THEOREM 4.2

Sketch of the proof of i) in Theorem 4.2 Applying `TwoSum` error-free transformation for the substraction $r = 1 \ominus t$ we have $[r, \rho] = \texttt{TwoSum}(1, -t)$, where

$$r = 1 - t - \rho, \quad |\rho| \leq u|1 \ominus t|, \quad |\rho| \leq u|1 - t| \tag{1}$$

by i) of Theorem 4.1. In the case that $t \geq 1/2$, VS algorithm computes $q = (1 - \ominus t) \oslash t$. If we compute the quotient in the previous expression by the error-free transformation `DivRem` one has, by iii) of Theorem 4.1, that $[q, \beta^M] = \texttt{DivRem}(r, t) = \texttt{DivRem}(1 \ominus t, t)$, where $q = (1 \ominus t) \oslash t$ and

$$1 \ominus t = t \times q + \beta^M, \quad |\beta^M| \leq u|t \times q|, \quad |\beta^M| \leq u\,|r|. \tag{2}$$

At the ith iteration we have that $p_i = p_{i-1} \otimes q \oplus c_i$ with $p_0 = c_0$. Applying in the previous expression error-free transformations `TwoProd` and `TwoSum` we obtain by i) and ii) of Theorem 4.1

$$[a_i, \pi_i^M] = \texttt{TwoProd}(q, p_{i-1}) \quad \text{and} \quad [p_i, \sigma_i^M] = \texttt{TwoSum}(a_i, c_i),$$

where

$$\begin{aligned} a_i &= q \otimes p_{i-1} \quad \text{and} \quad a_i = q \times p_{i-1} - \pi_i^M, \\ p_i &= a_i \oplus c_i \quad \text{and} \quad p_i = a_i + c_i - \sigma_i^M. \end{aligned}$$

By the previous formulas we have that $p_i = q \times p_{i-1} + c_i - \pi_i^M - \sigma_i^M$. By this formula and taking into account that from (1) and (2) we deduce that $q = (1-t)/t - (\rho + \beta^M)/t$, we can deduce that

$$\begin{aligned} \sum_{j=0}^{i} c_j \left(\frac{1-t}{t}\right)^{i-j} - p_i = \frac{1-t}{t} \times \left[\sum_{j=0}^{i-1} c_j \left(\frac{1-t}{t}\right)^{i-1-j} - p_{i-1}\right] \\ + \left(\frac{\rho + \beta^M}{t} \times p_{i-1} + \pi_i^M + \sigma_i^M\right) \end{aligned}$$

Denoting $l_i^M = \frac{\rho+\beta^M}{t} \times p_{i-1} + \pi_i^M + \sigma_i^M$ and iterating the previous formula for $i = 1, \ldots, n$, taking into account that $p_0 = c_0$, we have

$$\sum_{i=0}^{n} c_i \left(\frac{1-t}{t}\right)^{n-i} - p_n = \sum_{i=0}^{n-1} l_{i+1}^M \left(\frac{1-t}{t}\right)^{n-1-i}. \tag{3}$$

Finally, the last step of the VS algorithm is $res = p_n \otimes fl(t^n)$. In order to calculate res we have to perform n products as pointed out by the second loop: $f_{n-i} = f_{n+1-i} \otimes t$. Applying the error-free transformation for this product, that is `TwoProd`, we obtain

© 13 ACM. 0098-3500/13/01-ARTA $15.00
DOI: 0000001.0000001

$[f_{n-i}, \alpha^M_{n-i}] = \texttt{TwoProd}(f_{n+1-i}, t)$, where, by ii) of Theorem 4.1, $f_{n-i} = f_{n+1-i} \times t - \alpha^M_{n-i}$. Iterating the previous formula for $i = 1, \dots, n$ we obtain

$$f_0 = p_n \times t^n - \sum_{i=0}^{n-1} \alpha^M_i \, t^i$$

Multiplying formula (3) by t^n and taking into account the previous formula and that by the last step of the algorithm $res = f_0$ we derive

$$\begin{aligned} p(t) = \sum_{i=0}^{n} c_i \, t^i (1-t)^{n-i} &= p_n \times t^n + t \times \sum_{i=0}^{n-1} l^M_{i+1} \, t^i (1-t)^{n-1-i} \\ &= res + t \times \sum_{i=0}^{n-1} l^M_{i+1} \, t^i (1-t)^{n-1-i} + \sum_{i=0}^{n-1} \alpha^M_i \, t^i . \end{aligned}$$

The case where $t < 1/2$ is symmetric and analogous to the previous case. □

In order to prove ii) of Theorem 4.2 we need some auxilliary results. Let us denote the polynomials $\sum_{i=0}^{n-1} l^M_{i+1} t^i (1-t)^{n-1-i}$ and $\sum_{i=0}^{n-1} l^m_i t^i (1-t)^{n-1-i}$ by $p_M(t)$ and $p_m(t)$, respectively. In the proof we shall deal with quantities satisfying that their absolute value is bounded above by γ_k. Following [Higham 2002] we denote by θ_k such quantities, and let us take into account that, by Lemma 3.3 of [Higham 2002], the following property holds: $(1+\theta_k)(1+\theta_j) = 1 + \theta_{k+j}$. A straightforward, but tedious application of these properties allows us to prove the following Lemma A.1.

LEMMA A.1. *Under the hypotheses in Theorem 4.2 we have that*

$$\tilde{p}_M(t) \le 2\gamma_{4n} \, \tilde{p}(t), \quad \sum_{j=0}^{n-1} |\alpha^M_j| \cdot |t^j| \le \gamma_{4n} \tilde{p}(t),$$

if $t \ge 1/2$*, and*

$$\tilde{p}_m(t) \le 2\gamma_{4n} \, \tilde{p}(t), \quad \sum_{j=0}^{n-1} |\alpha^m_{n-j}| \cdot |t^j| \le \gamma_{4n} \tilde{p}(t),$$

if $t < 1/2$.

LEMMA A.2. *Under the hypotheses in Theorem 4.2 we have that*

$$\left| t \otimes VS(t, (l^M_1, \dots, l^M_n)) - t \times \tilde{p}_M(t) \right| \le \gamma_{4n-3} |t| \tilde{p}_M(t) \tag{4}$$

and

$$\left| Horner(t, (\alpha^M_0, \alpha^M_1, \dots, \alpha^M_{n-1})) - \sum_{j=0}^{n-1} \alpha^M_j \cdot t^j \right| \le \gamma_{2n-2} \sum_{j=0}^{n-1} |\alpha^M_j| \cdot |t^j| \tag{5}$$

if $t \ge 1/2$*, and*

$$\left| (1 \ominus t) \otimes VS(t, (l^m_0, \dots, l^m_{n-1})) - (1-t) \times p_m(t) \right| \le \gamma_{4n-2} |1-t| \tilde{p}_m(t)$$

and

$$\left| Horner(t, (\alpha^m_n, \alpha^m_{n-1}, \dots, \alpha^m_1)) - \sum_{j=0}^{n-1} \alpha^m_{n-j} \cdot t^j \right| \le \gamma_{2n-2} \sum_{j=0}^{n-1} |\alpha^m_{n-j}| \cdot |t^j|$$

if $t < 1/2$.

PROOF. Both formulas follow straightforwardly from the error analysis fo VS and Horner algorithms in Section 2. □

Sketch of the proof of ii) in Theorem 4.2 If $t \geq 1/2$, the absolute forward error corresponding to the approximation of $p(t)$ given by Algorithm 8 is

$$|res - p(t)| = |(1+\theta_2)(f_0 + t \otimes VS(t,(l_1^M,\ldots,l_n^M)) + Horner(t,(\alpha_0^M,\ldots,\alpha_{n-1}^M))) - p(t)|,$$

where the expressions $l_1^M,\ldots,l_n^M$ are given in the proof of i) of Theorem 4.2. From the previous formula and the two formulas in Lemma A.2, taking into account that $f_0 = VS(t,(c_0,...,c_n))$ and that $p(t) = VS(t,(c_0,...,c_n)) + \sum_{j=0}^{n-1} l_{j+1}^M t^{j+1}(1-t)^{n-1-j} + \sum_{j=0}^{n-1} \alpha_j^M \cdot t^j$, we deduce that

$$|res - p(t)| \leq \gamma_2\,|p(t)| + (1+\gamma_2)\gamma_{4n-3}|t|\tilde{p}_M(t) + (1+\gamma_2)\gamma_{2n-2}\sum_{j=0}^{n-1}|\alpha_j|\cdot|t^j|.$$

By Lemma A.1 and taking into account that $|t| \leq 1$, we have

$$\begin{aligned}|res - p(t)| &\leq \gamma_2\,|p(t)| + 2(1+\gamma_2)\gamma_{4n-3}\gamma_{4n}\tilde{p}(t) + (1+\gamma_2)\gamma_{2n-2}\gamma_{4n}\tilde{p}(t)\\ &= \gamma_2\,|p(t)| + [2(1+\gamma_2)\gamma_{4n-3} + (1+\gamma_2)\gamma_{2n-2}]\gamma_{4n}\tilde{p}(t).\end{aligned}$$

Taking into account that $\gamma_{4n-3}, \gamma_{2n-2} \leq \gamma_{4n}$ and that in every practical floating point arithmetic system $1+\gamma_2 \leq 4/3$ we conclude from the previous expression that $|res - p(t)| \leq \gamma_2\,|p(t)| + 4\gamma_{4n}^2\tilde{p}(t)$ and so

$$\frac{|res - p(t)|}{|p(t)|} \leq \gamma_2 + 4\gamma_{4n}^2 S_{b_n}(p(t)). \quad \square$$