Journal of Computational and Applied Mathematics 243 (2013) 28–47

Contents lists available at SciVerse ScienceDirect

Journal of Computational and Applied Mathematics

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

Accurate evaluation of the *k*-th derivative of a polynomial and its application☆

Hao Jiang [a,b,c,*], Stef Graillat [c], Canbin Hu [d], Shengguo Li [a], Xiangke Liao [e], Lizhi Cheng [a,b], Fang Su [a]

[a] School of Science, National University of Defense Technology, Changsha, 410073, China
[b] The State Key Laboratory for High Performance Computation, National University of Defense Technology, Changsha, 410073, China
[c] PEQUAN, LIP6, Université Pierre et Marie Curie, CNRS, Paris, France
[d] College of Electronic Science and Engineering, National University of Defense Technology, Changsha, 410073, China
[e] School of Computer, National University of Defense Technology, Changsha, 410073, China

ARTICLE INFO

Article history:
Received 7 July 2010
Received in revised form 5 November 2012

Keywords:
Derivative evaluation
Rounding error
Compensated algorithm
Floating-point arithmetic
Error-free transformation

ABSTRACT

This paper presents a compensated algorithm for the evaluation of the *k*-th derivative of a polynomial in power basis. The proposed algorithm makes it possible the direct evaluation without obtaining the *k*-th derivative expression of the polynomial itself, with a very accurate result to all but the most ill-conditioned evaluation. Forward error analysis and running error analysis are performed by an approach based on the data dependency graph. Numerical experiments illustrate the accuracy and efficiency of the algorithm.

1. Introduction

The need to evaluate the derivatives of a polynomial at a specific point arises in many fields of engineering and mathematics. Horner's rule is the classic algorithm for evaluating polynomials. It can also efficiently evaluate various order derivatives of a polynomial. We denote this Horner scheme with derivatives by HD algorithm (Horner derivative algorithm). The HD algorithm carries out the process of synthetic division without determining the derivative expression of the polynomial. This algorithm and its error analysis have been studied by Müller [1], Olver [2] and Burrus [3], and summarized by Higham in [4]. However, when performed in floating-point arithmetic, the computed result by the HD algorithm may be less accurate than expected due to cancellations. Therefore, some high accurate algorithms are required. Recently, Graillat, Langlois and Louvet [5–7] proposed a compensated Horner algorithm to evaluate the polynomial in power basis. The algorithm, using error-free transformations (EFTs) [8–10], yields a full precision accuracy for not too ill-conditioned polynomial. Motivated by this previous research, we propose the compensated Horner derivative algorithm to evaluate the *k*-th derivative of a polynomial.

The rest of the paper is organized as follows. In Section 2 we introduce some basic notations and results about floating-point arithmetic and error-free transformations. In Section 3, we recall the HD algorithm and present its forward

☆ The research was supported by National Natural Science Foundation of China (60921062, 61072118, 11101430), the Foundation for Innovative Research Groups of the National Natural Science Foundation of China (Grant No. 60626003), Science Research Project of National University of Defense Technology JC 12-02-01, JC 11-02-06 and China Scholarship Council 2011611057 and 2011611060.

* Corresponding author at: School of Science, National University of Defense Technology, Changsha, 410073, China. Tel.: +86 15802680698.
E-mail addresses: jhnudt@yahoo.cn (H. Jiang), stef.graillat@lip6.fr (S. Graillat), clzcheng@nudt.edu.cn (L.Z. Cheng).

0377-0427/$ – see front matter
doi:10.1016/j.cam.2012.11.008

error bound using a new approach which is different from that in [4]. In Section 4 the compensated Horner derivative algorithm is derived with EFT. In Section 5 the forward error analysis is performed. Section 6 is devoted to the running error analysis. In Section 7 numerical experiments illustrate the accuracy and efficiency of the proposed algorithm. Finally, we conclude by giving an application of the proposed algorithm.

2. Preliminaries

In this paper we assume all the floating-point computation is performed in double precision, with the round to the nearest rounding mode and no underflow occurring. We also assume that the computation in floating-point arithmetic obeys the model

$$a \,\mathrm{op}\, b = \mathrm{fl}(a \circ b) = (a \circ b)(1+\varepsilon_1) = (a \circ b)/(1+\varepsilon_2), \tag{1}$$

where $\mathrm{op} \in \{\oplus, \ominus, \otimes\}$, $\circ \in \{+, -, \times\}$ and $|\varepsilon_1|, |\varepsilon_2| \le u$. The symbol u is the round-off unit and "op" represents the floating-point computation, e.g. $a \oplus b = \mathrm{fl}(a+b)$. We also assume that all the operations are done with rounding to the nearest and that the computed result of $a \in \mathbb{R}$ in floating-point arithmetic is denoted by $\hat{a}$ or $\mathrm{fl}(a)$ and $\mathbb{F}$ denotes the set of all floating-point numbers (see [4] for more details). Following [4], we will use the following classic properties in error analysis (we always assume that $nu < 1$).

(1) if $\delta_i \le u$, $\rho_i = \pm 1$, then $\prod_{i=1}^{n}(1+\delta_i)^{\rho_i} = 1+\theta_n = \langle n \rangle$,
(2) $|\theta_n| \le \gamma_n := nu/(1-nu)$,
(3) $(1+\theta_k)(1+\theta_j) \le (1+\theta_{k+j})$ and $\langle k \rangle \langle j \rangle = \langle k+j \rangle$,
(4) $\gamma_k + \gamma_j + \gamma_k\gamma_j \le \gamma_{k+j}$ and $\gamma_k < \gamma_{k+1}$.

To derive the running error bound, we need the next relations (see details in [5,7]).

$$\widehat{\gamma}_k = (ku) \oslash (1 \ominus ku), \qquad \gamma_k \le (1+u)\widehat{\gamma}_k, \qquad (1+u)^n|x| \le \mathrm{fl}\left(\frac{|x|}{1-(n+1)u}\right). \tag{2}$$

Next let us introduce some results concerning error-free transformations (EFTs). For a pair of floating-point numbers $a, b \in \mathbb{F}$, when no underflow occurs, there exists a floating-point number y satisfying $a \circ b = x + y$, where $x = \mathrm{fl}(a \circ b)$ and $\circ \in \{+, -, \times\}$. The transformation $(a, b) \longrightarrow (x, y)$ is regarded as an error-free transformation. The error-free transformation algorithms of the sum and product of two floating-point numbers used later in this paper are the `TwoSum` algorithm by Knuth [11] and the `TwoProd` algorithm by Dekker [12], respectively (see Appendix B). The following theorem exhibits the important properties of the `TwoSum` and `TwoProd` algorithms (see [10]).

Theorem 1 ([10]). *For $a, b \in \mathbb{F}$ and $x, y \in \mathbb{F}$, `TwoSum` and `TwoProd` verify*

$$[x, y] = \mathtt{TwoSum}(a, b), \quad x = \mathrm{fl}(a+b),\ x+y = a+b,\ y \le u|x|,\ y \le u|a+b|,$$
$$[x, y] = \mathtt{TwoProd}(a, b), \quad x = \mathrm{fl}(a \times b),\ x+y = a \times b,\ y \le u|x|,\ y \le u|a \times b|.$$

3. Horner derivative algorithm

The Horner derivative (`HD`) algorithm is the classic method for the evaluation of the k-derivative of a polynomial $p(x) = \sum_{i=0}^{n} a_i x^i$. This algorithm permits the direct evaluation without obtaining the k-th derivative of the polynomial itself at any point x (see more details in [4]). Using a data dependency graph between the computed values, we present a new method to obtain the error bound of `HD` algorithm here.

First of all, we will introduce the data dependency graph as a convenient technique in the error analysis, which is similar to those in the literature, such as [13,14]. Fig. 1 shows the data dependency graphs for `HD` algorithm in Section 3 and `CompHD` algorithm in Section 5. There exist the following relations:

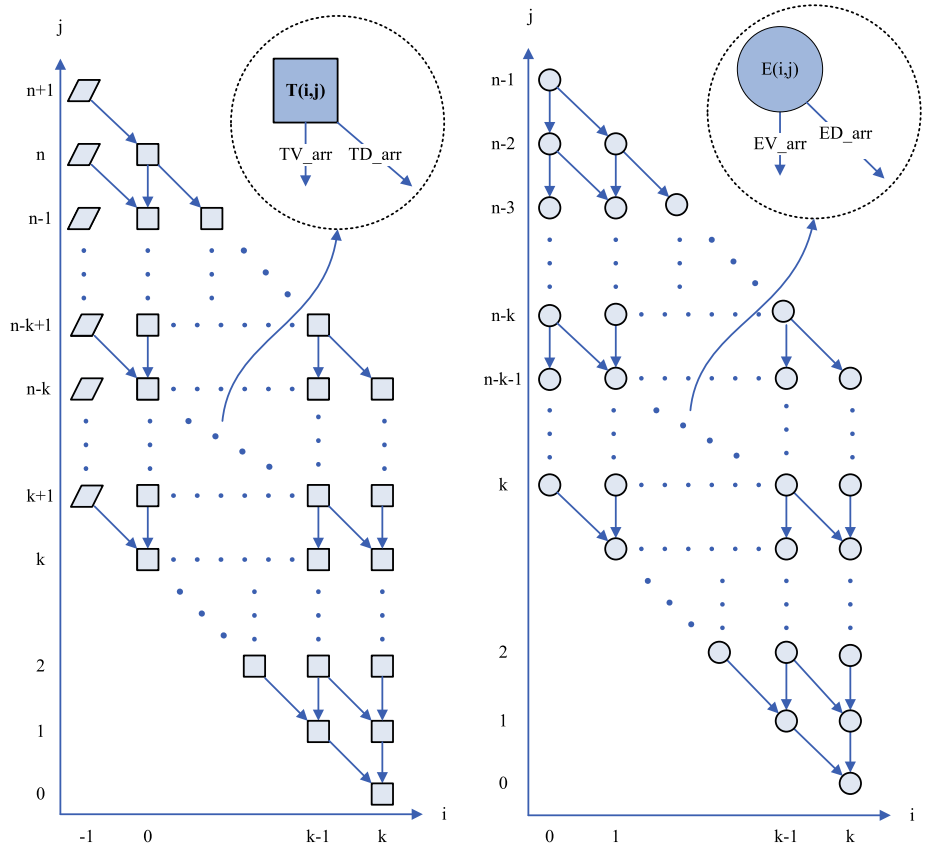
$$T(i,j) = T(i,j+1) \times \mathrm{TV\_arr} + T(i-1,j+1) \times \mathrm{TD\_arr}, \tag{3}$$
$$E(i,j) = E(i,j+1) \times \mathrm{EV\_arr} + E(i-1,j+1) \times \mathrm{ED\_arr} + \mathrm{CS}(i,j), \tag{4}$$

where $T(i,j)$ and $E(i,j)$ are computed values, the vertical arrows TV_arr, EV_arr and the diagonal arrows TD_arr, ED_arr are coefficients, and $\mathrm{CS}(i,j)$ is a constant.

For further error analysis, the condition number for the k-th derivative evaluation of a polynomial $p(x) = \sum_{i=0}^{n} a_i x^i$ at entry x is given

$$\mathtt{cond}(p, x, k) = \frac{k! \sum_{m=k}^{n} C_m^k |x|^{m-k}|a_m|}{\left|k! \sum_{m=k}^{n} C_m^k x^{m-k} a_m\right|} = \frac{\widetilde{p}^{(k)}(x)}{|p^{(k)}(x)|}, \quad C_m^k = \binom{k}{m}. \tag{5}$$

Next, we recall the classic `HD` algorithm and prove its accuracy with data dependency graph technique. The analogous result is obtained by expressing Algorithm 1 in matrix form in [4], however, which only deal with the first derivative.



(a) Data dependency graph for y_i^j and $\widehat{y}_i^j$. (b) Data dependency graph for ϵy_i^j and $\widehat{\epsilon y}_i^j$.

Fig. 1. Data dependency graphs for HD and CompHD algorithms.

Algorithm 1. Horner derivative algorithm

function res $=$ HD(p, x, k)
$y_i^j = 0$, for $i = 0:1:k$, and $j = n+1:-1:0$
$y_{-1}^{j+1} = a_j$, for $j = n:-1:0$
for $j = n:-1:0$
  for $i = \min(k, n-j):-1:\max(0, k-j)$
    $y_i^j = x \times y_i^{j+1} + y_{i-1}^{j+1}$
  end
end
res $= k! \times y_k^0$

The data dependency graph for the HD algorithm in theoretical computation can be expressed with Fig. 1(a), where the parallelograms represent the initial values, that is $T(-1, j+1) = y_{-1}^{j+1} = a_j$ and the squares represent the iterative values, that is $T(i, j) = y_i^j$. The vertical arrow (TV_arr) and the diagonal arrow (TD_arr) represent x and 1, respectively. Then, (3) embodies the following Eq. (6).

Before proving the main theorem, we display two recursive formulas as follows, which correspond to the theoretical and numerical computation, respectively,

$$y_i^j = x \times y_i^{j+1} + y_{i-1}^{j+1}, \tag{6}$$

$$\widehat{y}_i^j = x \otimes \widehat{y}_i^{j+1} \oplus \widehat{y}_{i-1}^{j+1}. \tag{7}$$

The following lemma gives the bound of the floating-point evaluation $\widehat{y}_i^j$, which is used in the accuracy analysis of the HD algorithm and the further proposed CompHD algorithm.

Lemma 1. *Let $p(x) = \sum_{i=0}^{n} a_i x^i$ be a polynomial of degree n with floating-point coefficients, and let x be a floating-point value. $\widehat{y}_i^j$ is the floating-point evaluation of y_i^j in Algorithm 1, and it satisfies*

$$|\widehat{y}_i^j| \le (1+\gamma_{2n-k}) \sum_{m=i+j}^{n} C_{m-j}^{i} |x|^{m-(i+j)} |a_m|. \tag{8}$$

Proof. From (7) and the standard model (1) we can deduce

$$\widehat{y}_i^j = x \times \widehat{y}_i^{j+1}\langle 2\rangle + \widehat{y}_{i-1}^{j+1}\langle 1\rangle. \tag{9}$$

Actually, when $i+j = n, \widehat{y}_i^j = \widehat{y}_{i-1}^{j+1}$, but for the error analysis of convenience, here we let $\widehat{y}_i^j = \widehat{y}_{i-1}^{j+1}\langle 1\rangle$ when $i > 0$ and $\widehat{y}_0^n = \widehat{y}_{-1}^{n+1} = a_n$, which is consistent with the following $\zeta_n = 0$ in Eq. (11). Then with the same initial value $\widehat{y}_{-1}^t = y_{-1}^t = a_{t-1}$, $k+1 \le t \le n+1$, we can obtain

$$\widehat{y}_i^j = \sum_{m=i+j}^{n} C_{m-j}^{i} x^{m-(i+j)} a_m (1+\xi_{ijm}), \tag{10}$$

with

$$1+\xi_{ijm} = 1+\theta_{2(m-(i+j))+i+\zeta_m} = \langle 2(m-(i+j))+i+\zeta_m\rangle, \quad \zeta_m = \begin{cases} 0, & \text{if } m = n, \\ 1, & \text{else.} \end{cases} \tag{11}$$

Since $n \ge m \ge i+j \ge k \ge i$, we obtain $\max\{|\xi_{ijm}|\} \le \gamma_{2n-k}$.

Therefore, from (10) we can easily obtain (8). □

In the case of the iterative relation (9), in contrast to (3), the vertical arrow TV_arr and the diagonal arrow TD_arr represent $x \times \langle 2\rangle$ and $\langle 1\rangle$, respectively, $T(i,j) = \widehat{y}_i^j$ and the initial values $T(-1,t) = \widehat{y}_{-1}^t = a_{t-1}$. Fig. 1(a) shows that each path from the node $(0,m)$ to the node (i,j) consists of $m-(i+j)$ vertical arrows and i diagonal arrows, with the number of these paths be C_{m-j}^i. And there is only one path of the diagonal arrow from the node $(0,m)$ to $(-1,m+1)$. Considering $\widehat{y}_0^n = \widehat{y}_{-1}^{n+1} = a_n$ and $\widehat{y}_0^m = x \times \widehat{y}_0^{m+1}\langle 2\rangle + \widehat{y}_{-1}^{m+1}\langle 1\rangle$ for $m \ge j$, we have the definition of ζ_m in (11).

When computing $\widehat{y}_i^j$, we find that the node (i,j) is in the line $x+y = i+j$ and the value $\widehat{y}_i^j$ in the node (i,j) only depends on the values in the nodes $(0,i+j),\ldots,(0,n)$. Then from Fig. 1(a) it is easy to obtain that $n \ge m \ge i+j \ge k \ge i$.

The above discussion about the data dependency graph of Algorithm 1 helps explaining how to obtain Eqs. (10), (11) and finally (8).

Theorem 2. *Let $p(x) = \sum_{i=0}^{n} a_i x^i$ be a polynomial of degree n with floating-point coefficients, and x a floating-point value. The relative forward error bound in Algorithm 1 verifies:*

$$\frac{|\mathtt{HD}(p,x,k) - p^{(k)}(x)|}{|p^{(k)}(x)|} \le \gamma_{2n}\mathtt{cond}(p,x,k). \tag{12}$$

Proof. According to (10) and (11) in Lemma 1, let $i = k$ and $j = 0$, then we have

$$\widehat{y}_k^0 = \sum_{m=k}^{n} C_m^k x^{m-k} a_m (1+\theta_{2m-k+\zeta_m}). \tag{13}$$

From Algorithm 1, we have

$$\mathtt{HD}(p,x,k) = \widehat{p}^{(k)}(x) = k! \otimes \widehat{y}_k^0 = k!\widehat{y}_k^0(1+\theta_1), \quad |\theta_1| \le u. \tag{14}$$

Here, we deem that the process $k!$ does not generate rounding error.

Thus, from (13) and the property (3) in Section 2 we have

$$\mathtt{HD}(p,x,k) = k! \sum_{m=k}^{n} C_m^k x^{m-k} a_m (1+\theta_{2m-k+\zeta_m+1}). \tag{15}$$

Since $k \ge 1$, we have $|\theta_{2m-k+\zeta_m+1}| \le |\theta_{2n-k+\zeta_n+1}| \le \gamma_{2n-k+1} \le \gamma_{2n}$. Using the condition number (5), we finally write the error bound (12). □

Theorem 2 tells us that when $\mathrm{cond}(p,x,k) > \gamma_{2n}^{-1}$, the computed result $\mathtt{HD}(p,x,k)$ does not contain any correct digit.

4. Compensated Horner derivative algorithm

In this section, the rounding errors generated by Algorithm 1 in each step are exhibited with EFT algorithm. We give the expression of $y_k^0 - \widehat{y}_k^0$, which consists of these rounding errors. On the basis of this expression we propose a compensated Horner derivative (CompHD) algorithm.

Applying error-free transformation to formula (7), we obtain

$$[s, \pi_i^j] = \texttt{TwoProd}(x, \widehat{y}_i^{j+1}), \tag{16}$$

$$[\widehat{y}_i^j, \sigma_i^j] = \texttt{TwoSum}(s, \widehat{y}_{i-1}^{j+1}). \tag{17}$$

Theorem 3. *Let $p(x) = \sum_{i=0}^{n} a_i x^i$ be a polynomial of degree n with floating-point coefficients, and let y_k^0 and $\widehat{y}_k^0$ be the theoretical result and the numerical result, respectively, by Algorithm 1. Then we have*

$$y_k^0 = \widehat{y}_k^0 + \sum_{i=0}^{k} \sum_{j=k-i}^{n-1-i} C_j^{k-i} x^{i+j-k} (\pi_i^j + \sigma_i^j), \quad C_j^{k-i} = \binom{j}{k-i}, \tag{18}$$

where π_i^j and σ_i^j are rounding errors generated in the floating-point evaluation and obtained with EFT.

Proof. Since Eqs. (16) and (17) are EFT, from Theorem 1, we obtain

$$\widehat{y}_i^j = x \times \widehat{y}_i^{j+1} + \widehat{y}_{i-1}^{j+1} - \pi_i^j - \sigma_i^j \tag{19}$$

Let

$$\epsilon y_i^j = y_i^j - \widehat{y}_i^j, \quad \text{for } i = 0 : k \text{ and } j = k - i : n - i \tag{20}$$

then from (6) and (19), we have

$$\epsilon y_i^j = x \times \epsilon y_i^{j+1} + \epsilon y_{i-1}^{j+1} + \pi_i^j + \sigma_i^j. \tag{21}$$

Since $y_{-1}^{j+1} = \widehat{y}_{-1}^{j+1} = a_j$ is a constant, we have $\epsilon y_{-1}^j = 0$, for $j = n : -1 : 0$. Also, it is obvious that if $i+j = n$, $y_i^j = \widehat{y}_i^j = a_n$, and so $\epsilon y_i^j = 0$. Then we have the initial bound values.

According to the recurrence relation (21), it is easy to verify that at the end of loop we have

$$\epsilon y_k^0 = \sum_{i=0}^{k} \sum_{j=k-i}^{n-1-i} C_j^{k-i} x^{i+j-k} (\pi_i^j + \sigma_i^j). \tag{22}$$

Due to $\epsilon y_k^0 = y_k^0 - \widehat{y}_k^0$, we obtain (18). □

The recurrence relation (21) is shown in Fig. 1(b), where the roundness represents the iterative values, that is $E(i, j) = \epsilon y_i^j$. The constant value CS(i, j) in (4) is $\pi_i^j + \sigma_i^j$ here. The vertical arrow (EV_arr) and the diagonal arrow (ED_arr) represent x and 1, respectively. We find that each of the paths from the node (i, j) to the node $(k, 0)$ consists of $i + j - k$ vertical arrows and $k - i$ diagonal arrows, with the number of these paths be C_j^{k-i}. Considering (21) and the bound initial values, we find that ϵy_k^0 should consist of all the constants CS(i, j) for $i = 0 : k$ and $j = k - i : n - 1 - i$. Thus, based on the data dependency analysis above, we deduce that the coefficient of $(\pi_i^j + \sigma_i^j)$ contained in the final formulation of ϵy_k^0 should be $C_j^{k-i} x^{i+j-k}$.

From Theorem 3, we have

$$p^{(k)}(x) = y_k^0 \times k! = (\widehat{y}_k^0 + \epsilon y_k^0) \times k!. \tag{23}$$

Therefore, the key of the new algorithm proposed in this section is to compute an approximate $\widehat{\epsilon y}_k^0$ of ϵy_k^0 in the working precision, with a corrected result

$$\bar{p}^{(k)}(x) = (\widehat{y}_k^0 \oplus \widehat{\epsilon y}_k^0) \otimes k!. \tag{24}$$

The corrected result $\bar{p}^{(k)}(x)$ is expected to be more accurate than the floating-point result (14) by the traditional HD algorithm.

The previous discussion leads to the following compensated Horner derivative algorithm (CompHD algorithm)

Algorithm 2. Compensated Horner derivative algorithm

function res = CompHD(p, x, k)
$y_i^j = 0, \widehat{\epsilon y}_i^j = 0$, for $i = 0 : 1 : k$, and $j = n+1 : -1 : 0$
$y_{-1}^{j+1} = a_j, \widehat{\epsilon y}_{-1}^{j+1} = 0$, for $j = n : -1 : 0$
for $j = n : -1 : 0$
  for $i = \min(k, n-j) : -1 : \max(0, k-j)$
    $[s, \pi_i^j] = \mathtt{TwoProd}(x, \widehat{y}_i^{j+1})$
    $[\widehat{y}_i^j, \sigma_i^j] = \mathtt{TwoSum}(s, \widehat{y}_{i-1}^{j+1})$
    $\widehat{\epsilon y}_i^j = x \otimes \widehat{\epsilon y}_i^{j+1} \oplus \widehat{\epsilon y}_{i-1}^{j+1} \oplus (\pi_i^j \oplus \sigma_i^j)$
  end
end
res $= (\widehat{y}_k^0 \oplus \widehat{\epsilon y}_k^0) \otimes k!$

Algorithm 2 consists of two parts: the computations of $\widehat{y}_k^0$ and $\widehat{\epsilon y}_k^0$, the iterative relations of which are exhibited in Fig. 1(a) and (b), respectively. It must be noticed that the iterative value $\widehat{y}_i^j$ obtained by Algorithm 1 is as the same as that obtained by Algorithm 2.

5. Accuracy of the compensated Horner derivative algorithm

In this section we prove that the result of the compensated Horner derivative algorithm is nearly as accurate as the one computed by the original Horner derivative algorithm with twice working precision and then roughly rounded to working precision.

Lemma 2. *Let $p(x) = \sum_{i=0}^n a_i x^i$ be a polynomial of degree n with floating-point coefficients. Algorithm 2 computes the floating-point evaluation of ϵy_k^0 defined in Theorem 3. Then the computed result $\widehat{\epsilon y}_k^0$ satisfies the following forward error bound.*

$$|\epsilon y_k^0 - \widehat{\epsilon y}_k^0| \le \gamma_{3n-k-1} \sum_{i=0}^{k} \sum_{j=k-i}^{n-1-i} C_j^{k-i} |x|^{i+j-k} (|\pi_i^j| + |\sigma_i^j|). \tag{25}$$

Proof. By Algorithm 2, we have

$$\begin{aligned} \widehat{\epsilon y}_i^j &= x \otimes \widehat{\epsilon y}_i^{j+1} \oplus \widehat{\epsilon y}_{i-1}^{j+1} \oplus (\pi_i^j \oplus \sigma_i^j) \\ &= x \times \widehat{\epsilon y}_i^{j+1}(1+\theta_3) + \widehat{\epsilon y}_{i-1}^{j+1}(1+\theta_2) + (\pi_i^j + \sigma_i^j)(1+\theta_2) \\ &= x \times \widehat{\epsilon y}_i^{j+1}\langle 3\rangle + \widehat{\epsilon y}_{i-1}^{j+1}\langle 2\rangle + (\pi_i^j + \sigma_i^j)\langle 2\rangle. \end{aligned} \tag{26}$$

Actually, when $i+j = n-1$, $\widehat{\epsilon y}_i^{n-1-j} = \widehat{\epsilon y}_{i-1}^{n-j}\langle 2\rangle + (\pi_i^{n-1-j} + \sigma_i^{n-1-j})\langle 1\rangle$, and when $i = 0$, $\widehat{\epsilon y}_0^t = x \times \widehat{\epsilon y}_0^{t+1}\langle 2\rangle + (\pi_0^t + \sigma_0^t)\langle 1\rangle$. However, for the sake of convenience of error analysis, it is feasible and reasonable that we assume that

$$\begin{aligned} \widehat{\epsilon y}_i^{n-1-j} &= \widehat{\epsilon y}_{i-1}^{n-j}\langle 2\rangle + (\pi_i^{n-1-j} + \sigma_i^{n-1-j})\langle 2\rangle, \\ \widehat{\epsilon y}_0^t &= x \times \widehat{\epsilon y}_0^{t+1}\langle 3\rangle + (\pi_0^t + \sigma_0^t)\langle 2\rangle, \end{aligned}$$

at the bound computation of Algorithm 2. Then at the end of loop we will have

$$\widehat{\epsilon y}_k^0 = \sum_{i=0}^{k} \sum_{j=k-i}^{n-1-i} C_j^{k-i} x^{i+j-k} (\pi_i^j + \sigma_i^j)(1+\xi_{ij}), \tag{27}$$

where $(1+\xi_{ij}) = \langle 3(i+j-k) + 2(k-i) + 2\rangle$. Since $n-1 \ge i+j \ge k \ge i \ge 0$, it is obvious that $\max\{|\xi_{ij}|\} \le \gamma_{3n-k-1}$.
Thus, from (22) and (27), we obtain (25). □

Concerning the iterative relation (26), we can explain (27) with the data dependency graph in Fig. 1(b). Here, EV_arr and ED_arr represent the multiplication by $x \times \langle 3\rangle$ and that by $\langle 2\rangle$, respectively, with $E(i,j) = \widehat{\epsilon y}_i^j$ and $\mathrm{CS}(i,j) = (\pi_i^j + \sigma_i^j)\langle 2\rangle$. The scheme is analogous to the previous one about ϵy_i^j.

Lemma 3. *Let $p(x) = \sum_{i=0}^n a_i x^i$ be a polynomial of degree n with floating-point coefficients, and let x be a floating-point value. Then if no underflow occurs, Algorithm 2 satisfies*

$$|\pi_i^j| + |\sigma_i^j| \le 2u(1+\gamma_{2n-k+1}) \sum_{m=i+j}^{n} C_{m-j}^i |x|^{m-(i+j)} |a_m|. \tag{28}$$

Proof. Since $[s, \pi_i^j] = \texttt{TwoProd}(x, \widehat{y}_i^{j+1})$ and $[\widehat{y}_i^j, \sigma_i^j] = \texttt{TwoSum}(s, \widehat{y}_{i-1}^{j+1})$ in Algorithm 2, according to Theorem 1, we obtain

$$|\pi_i^j| \le u|s| \tag{29}$$

$$|\sigma_i^j| \le u|s + \widehat{y}_{i-1}^{j+1}| \le u(|s| + |\widehat{y}_{i-1}^{j+1}|). \tag{30}$$

Taking into account that $s = x \otimes \widehat{y}_i^{j+1} = (1+\delta)(x \times \widehat{y}_i^{j+1})$, we have $|s| \le (1+u)(|x| \times |\widehat{y}_i^{j+1}|)$.

Hence, by (29) and (30), we derive

$$|\pi_i^j| + |\sigma_i^j| \le 2u((1+u)(|x| \times |\widehat{y}_i^{j+1}|) + |\widehat{y}_{i-1}^{j+1}|). \tag{31}$$

Considering that the values of $\widehat{y}_i^j$ obtained by Algorithm 1 and by Algorithm 2 are the same, from Lemma 1, we have

$$|x| \times |\widehat{y}_i^{j+1}| \le (1+\gamma_{2n-k}) \sum_{m=i+j+1}^{n} C_{m-j-1}^i |x|^{m-(i+j)} |a_m|, \tag{32}$$

$$|\widehat{y}_{i-1}^{j+1}| \le (1+\gamma_{2n-k}) \sum_{m=i+j}^{n} C_{m-j-1}^{i-1} |x|^{m-(i+j)} |a_m|. \tag{33}$$

Using the classic property (4) in error analysis in Section 2, we have $(1+u)(1+\gamma_{2n-k}) \le (1+\gamma_{2n-k+1})$ and $\gamma_{2n-k} \le \gamma_{2n-k+1}$.

Meanwhile according to $C_{m-j-1}^{i-1} + C_{m-j-1}^i = C_{m-j}^i$ and $C_{i-1}^{i-1} = C_i^i = 1$, we can obtain the expected bound (28) from (31)–(33). □

Lemma 4. *Let $p(x) = \sum_{i=0}^n a_i x^i$ be a polynomial of degree n with floating-point coefficients. The forward error bound between the theoretical result ϵy_k^0 and the computed result $\widehat{\epsilon y}_k^0$ in Lemma 2 can be refined as follows:*

$$|\epsilon y_k^0 - \widehat{\epsilon y}_k^0| \le (k+1)\gamma_{3n-k-1}\gamma_{2n} \sum_{m=k}^{n} C_m^k |x|^{m-k} |a_m|. \tag{34}$$

Proof. From Lemma 3, we obtain

$$C_j^{k-i} |x|^{i+j-k} (|\pi_i^j| + |\sigma_i^j|) \le 2u(1+\gamma_{2n-k+1}) \sum_{m=i+j}^{n} C_j^{k-i} C_{m-j}^i |x|^{m-k} |a_m|. \tag{35}$$

Taking into account that $C_j^{k-i} C_{m-j}^i \le C_m^k$, we have

$$\sum_{j=k-i}^{n-1-i} C_j^{k-i} |x|^{i+j-k} (|\pi_i^j| + |\sigma_i^j|) \le 2u(1+\gamma_{2n-k+1})(n-k) \sum_{m=k}^{n} C_m^k |x|^{m-k} |a_m|.$$

Since $k \ge 1$, we obtain

$$2u(n-k)(1+\gamma_{2n-k+1}) \le 2un(1+\gamma_{2n}) = \gamma_{2n},$$

then

$$\sum_{j=k-i}^{n-1-i} C_j^{k-i} |x|^{i+j-k} (|\pi_i^j| + |\sigma_i^j|) \le \gamma_{2n} \sum_{m=k}^{n} C_m^k |x|^{m-k} |a_m|. \tag{36}$$

The right-hand side of this inequality is independent of i. Thus, from (25) in Lemma 2 and (36), we obtain (34). □

Now we can give the accuracy of the CompHD algorithm for the evaluation the k-derivative of a polynomial in power basis.

Theorem 4. *Let $p(x) = \sum_{i=0}^n a_i x^i$ be a polynomial of degree n with floating-point coefficients, and x a floating-point value. The relative forward error bound in Algorithm 2 is such that*

$$\frac{|\texttt{CompHD}(p, x, k) - p^{(k)}(x)|}{|p^{(k)}(x)|} \le 2u + (k+1)\gamma_{2n}\gamma_{3n}\texttt{cond}(p, x, k). \tag{37}$$

Proof. First, we deem that the procedure of computing $k!$ generates no rounding error.

From (24), we obtain

$$\tilde{p}^{(k)}(x) = (\widehat{y}_k^0 + \widehat{\epsilon y}_k^0)(1+\varepsilon_1) \times k! \times (1+\varepsilon_2) \tag{38}$$

$$= k!(\widehat{y}_k^0 + \epsilon y_k^0 - \epsilon y_k^0 + \widehat{\epsilon y}_k^0)(1+\theta_2), \tag{39}$$

where $|\varepsilon_1|, |\varepsilon_2| \le u$ and $(1+\theta_2) = (1+\varepsilon_1)(1+\varepsilon_2)$ with $|\theta_2| \le \gamma_2$. Then by (23), we deduce

$$\tilde{p}^{(k)}(x) = p(x)(1+\theta_2) + k!(\widehat{\epsilon y}_k^0 - \epsilon y_k^0)(1+\theta_2). \tag{40}$$

Thus the forward error bound in Algorithm 2 is

$$|p^{(k)}(x) - \tilde{p}^{(k)}(x)| \le \gamma_2 |p^{(k)}(x)| + (1+\gamma_2)k!|\widehat{\epsilon y}_k^0 - \epsilon y_k^0|. \tag{41}$$

Then from Lemma 4, we have

$$(1+\gamma_2)k!|\widehat{\epsilon y}_k^0 - \epsilon y_k^0| \le (k+1)(1+\gamma_2)\gamma_{3n-k-1}\gamma_{2n}k! \sum_{m=k}^{n} C_m^k |x|^{m-k}|a_m|. \tag{42}$$

Since $(1+\gamma_2)\gamma_{3n-k-1} \le \gamma_{3n-k+1} \le \gamma_{3n}$ and $k! \sum_{m=k}^{n} C_m^k |x|^{m-k}|a_m| = \tilde{p}^{(k)}(x)$, we have

$$(1+\gamma_2)k!|\widehat{\epsilon y}_k^0 - \epsilon y_k^0| \le (k+1)\gamma_{2n}\gamma_{3n}\widetilde{p}^{(k)}(x). \tag{43}$$

From (41) and (43), we obtain

$$|p^{(k)}(x) - \tilde{p}^{(k)}(x)| \le \gamma_2 |p^{(k)}(x)| + (k+1)\gamma_{2n}\gamma_{3n}\widetilde{p}^{(k)}(x). \tag{44}$$

Using the condition number (5), and considering that $\gamma_2 \simeq 2u$ which are almost the same to each other in working precision, we obtain the expected relative error bound (37) roughly. □

From Theorem 4, we can observe that, if $(k+1)\gamma_{2n}\gamma_{3n}\mathrm{cond}(p,x,k) < 2u$, the relative error of the result computed by Algorithm 2 is bounded by the constant value $2u$.

Compared with Theorem 2, the computed value by the CompHD algorithm is nearly as accurate as the result computed by the HD algorithm with twice working precision and then roughly rounded to working precision.

6. Running error bound for compensated Horner derivative algorithm

In the practical computation, we usually wish to compute a corresponding error bound along with the result. The *a priori* error bound (44) in Theorem 4 is entirely adequate for theoretical purposes, but lacks sharpness. This section is devoted to perform running error analysis of the compensated Horner derivative algorithm, which provides a sharper and *a posteriori* error bound.

Lemma 5. *Let $p(x) = \sum_{i=0}^{n} a_i x^i$ be a polynomial of degree n with floating-point coefficients. We use the notations of Algorithm 2. Then the error bound between the theoretical result ϵy_k^0 and the computed result $\widehat{\epsilon y}_k^0$ is given by*

$$|\epsilon y_k^0 - \widehat{\epsilon y}_k^0| \le \mathrm{fl}\left(\frac{\widehat{\gamma}_{3n-k-1} \otimes \widehat{w}_k^0}{1-(3n+1)u}\right) := \widehat{\alpha} \tag{45}$$

where $\widehat{w}_k^0$ is the computed result of

$$\sum_{i=0}^{k}\sum_{j=k-i}^{n-1-i} C_j^{k-i}|x|^{i+j-k}(|\pi_i^j| + |\sigma_i^j|) \tag{46}$$

in Lemma 2 and it is derived from

$$\widehat{w}_i^j = |x| \otimes \widehat{w}_i^{j+1} \oplus \widehat{w}_{i-1}^{j+1} \oplus (|\pi_i^j| \oplus |\sigma_i^j|) \tag{47}$$

with the initial values $\widehat{w}_i^j = 0$ for $i = -1:k, j = 0:n$.

Proof. From Lemma 2 we have

$$|\epsilon y_k^0 - \widehat{\epsilon y}_k^0| \le \gamma_{3n-k-1} w_k^0, \tag{48}$$

where w_k^0 is the theoretical result of (46) with the iterative relation

$$w_i^j = |x| \times w_i^{j+1} + w_{i-1}^{j+1} + (|\pi_i^j| + |\sigma_i^j|) \tag{49}$$

and the initial values $w_i^j = 0$ for $i = 0:k, j = 0:n$.

By (47) and the floating-point arithmetic model (1), we can deduce

$$\widehat{w}_i^j = \left\{ \left[(|x| \times \widehat{w}_i^{j+1}) \frac{1}{1+\varepsilon_1} + \widehat{w}_{i-1}^{j+1} \right] \frac{1}{1+\varepsilon_2} + (|\pi_i^j| + |\sigma_i^j|) \frac{1}{1+\varepsilon_3} \right\} \frac{1}{1+\varepsilon_4},$$

with $|\varepsilon_t| \le u$, for $t = 1, 2, 3, 4$.

Then taking into account that $\widehat{w}_i^j \ge 0$, we have

$$(1+u)^3 \widehat{w}_i^j \ge |x| \times \widehat{w}_i^{j+1} + \widehat{w}_{i-1}^{j+1} + (|\pi_i^j| + |\sigma_i^j|). \tag{50}$$

Considering $w_0^{n-1} = |\pi_0^{n-1}| + |\sigma_0^{n-1}|$ and $\widehat{w}_0^{n-1} = |\pi_0^{n-1}| \oplus |\sigma_0^{n-1}|$, we obtain

$$w_0^{n-1} \le (1+u)\widehat{w}_0^{n-1}. \tag{51}$$

From (50) and (51), we can prove by induction that, for $j = n-1 : -1 : 0$,

$$w_i^j \le (1+u)^{3(n-1-j)+1} \widehat{w}_i^j, \tag{52}$$

which in turn is, for $j = 0$,

$$w_k^0 \le (1+u)^{3(n-1)+1} \widehat{w}_k^0. \tag{53}$$

By (48) and (53), we deduce

$$|\epsilon y_k^0 - \widehat{\epsilon y}_k^0| \le (1+u)^{3(n-1)+1} \gamma_{3n-k-1} \widehat{w}_k^0. \tag{54}$$

From the second relation in (2) and the model (1) it follows that

$$\begin{aligned} |\epsilon y_k^0 - \widehat{\epsilon y}_k^0| &\le (1+u)^{3(n-1)+1} [(1+u)\widehat{\gamma}_{3n-k-1} \otimes w_k^0](1+u), \\ &\le (1+u)^{3n} (\widehat{\gamma}_{3n-k-1} \otimes w_k^0). \end{aligned}$$

Using the third relation in (2), we obtain the expected error bound (45). □

We can explain the relationship between (46) and (49) with Fig. 1(b). Here, EV_arr and ED_arr represent $|x|$ and 1, respectively, with $E(i,j) = w_i^j$ and $\mathrm{CS}(i,j) = |\pi_i^j| + |\sigma_i^j|$.

Now let us consider the forward error bound in Algorithm 2. By (23) and (24) we can write

$$\begin{aligned} |p^{(k)}(x) - \bar{p}^{(k)}(x)| &\le |(\widehat{y}_k^0 + \epsilon y_k^0) \times k! - (\widehat{y}_k^0 \oplus \widehat{\epsilon y}_k^0) \otimes k!| \\ &\le |(\widehat{y}_k^0 + \widehat{\epsilon y}_k^0) \times k! - (\widehat{y}_k^0 \oplus \widehat{\epsilon y}_k^0) \otimes k!| + |\widehat{\epsilon y}_k^0 - \epsilon y_k^0| \times k!. \end{aligned} \tag{55}$$

The first term of (55) is the absolute error which comes from the evaluation of $\mathrm{res} = (\widehat{y}_k^0 \oplus \widehat{\epsilon y}_k^0) \otimes k!$ in Algorithm 2. We can apply algorithm `TwoSum` and `TwoProd` to obtain this actual error exactly. Meanwhile Lemma 5 gives the bound of the second term. Hence, we can obtain a running error bound of Algorithm 2.

Theorem 5. *A running error bound of `CompHD` algorithm (Algorithm 2) is given by*

$$|\mathtt{CompHD}(p, x, k) - p^{(k)}(x)| \le \mathrm{fl}\left(\frac{\widehat{\alpha} \otimes k! \oplus \widehat{\beta}}{1-4u} \right), \tag{56}$$

where $\widehat{\alpha}$ is the error bound defined by (48) in Lemma 5 and $\widehat{\beta}$ is obtained from the following equality

$$\widehat{\beta} = |c \otimes k! \oplus e| \tag{57}$$

with $[s, c] = \mathtt{TwoSum}(\widehat{y}_k^0, \widehat{\epsilon y}_k^0)$ and $[\bar{p}^{(k)}(x), e] = \mathtt{TwoProd}(s, k!)$.

Proof. From Theorem 1, we have

$$\begin{aligned} &\widehat{y}_k^0 + \widehat{\epsilon y}_k^0 = s + c, \\ &s \times k! = \bar{p}^{(k)}(x) + e. \end{aligned}$$

Then from (24) it follows that

$$(\widehat{y}_k^0 + \widehat{\epsilon y}_k^0) \times k! - (\widehat{y}_k^0 \oplus \widehat{\epsilon y}_k^0) \otimes k! = (c \times k! + e). \tag{58}$$

From (55), (58) and (45), we can obtain

$$|p^{(k)}(x) - \bar{p}^{(k)}(x)| \le |c \times k! + e| + |\epsilon y_k^0 - \widehat{\epsilon y}_k^0| \times k! \le (1+u)^3 (\widehat{\alpha} \otimes k! \oplus \widehat{\beta}).$$

Finally from the third relation in (2) and $\bar{p}^{(k)}(x) = \mathtt{CompHD}(p, x, k)$, we obtain the expected error bound (56). □

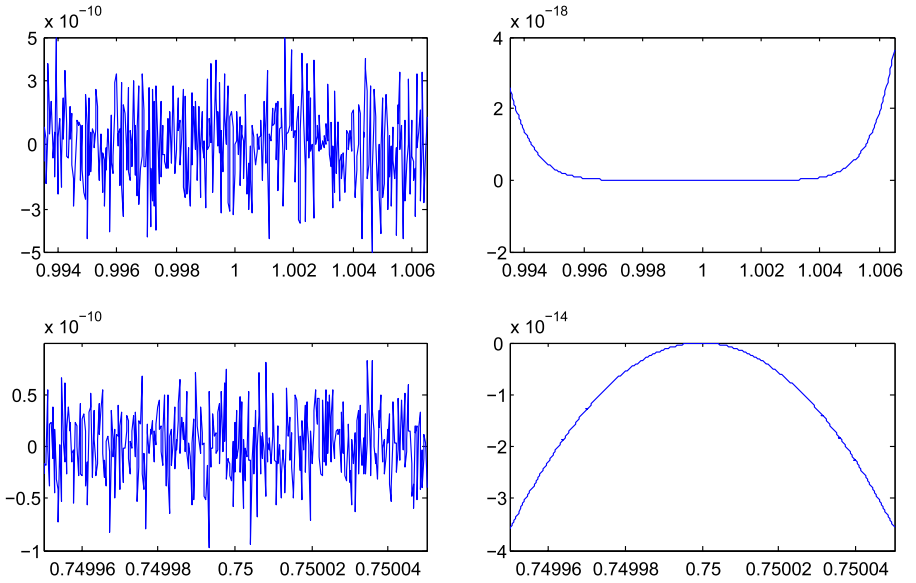


Fig. 2. Evaluation the 3rd derivative of the polynomial $p(x) = (x - 0.75)^5(x - 1)^{11}$ in the neighborhood of its multiple roots, using HD (left) and CompHD (right).

From Theorem 5 we deduce the following algorithm for the computation of the compensated result $\bar{p}^{(k)}(x)$ together with the bound μ of the running error.

Algorithm 3. Compensated Horner derivative algorithm with running error bound

$$
\begin{array}{l}
\text{function } [\text{res}, \mu] = \texttt{CompHDWErr}(p, x, k) \\
y_i^j = 0, \widehat{\epsilon y}_i^j = 0, \widehat{w}_i^j = 0, \text{ for } i = 0:1:k, j = n+1:-1:0 \\
y_{-1}^{j+1} = a_j, \widehat{\epsilon y}_{-1}^{j+1} = 0, \widehat{w}_{-1}^{j+1} = 0, \text{ for } j = n:-1:0 \\
\text{for } j = n:-1:0 \\
\quad \text{for } i = \min(k, n-j):-1:\max(0, k-j) \\
\qquad [s, \pi_i^j] = \texttt{TwoProd}(x, \widehat{y}_i^{j+1}) \\
\qquad [\widehat{y}_i^j, \sigma_i^j] = \texttt{TwoSum}(s, \widehat{y}_{i-1}^{j+1}) \\
\qquad \widehat{\epsilon y}_i^j = x \otimes \widehat{\epsilon y}_i^{j+1} \oplus \widehat{\epsilon y}_{i-1}^{j+1} \oplus (\pi_i^j \oplus \sigma_i^j) \\
\qquad \widehat{w}_i^j = |x| \otimes \widehat{w}_i^{j+1} \oplus \widehat{w}_{i-1}^{j+1} \oplus (|\pi_i^j| \oplus |\sigma_i^j|) \\
\quad \text{end} \\
\text{end} \\
[s, c] = \texttt{FastTwoSum}(\widehat{y}_k^0, \widehat{\epsilon y}_k^0) \\
[\text{res}, e] = \texttt{TwoProd}(s, k!). \\
\widehat{\alpha} = (\widehat{\gamma}_{3n-k-1} \otimes \widehat{w}_k^0) \oslash (1 \ominus (3n+1)u) \\
\widehat{\beta} = |c \otimes k! \oplus e| \\
\mu = (\widehat{\alpha} \otimes k! \oplus \widehat{\beta}) \oslash (1 \ominus 4u)
\end{array}
$$

Algorithm 3 includes an estimation of the error bound at the same time as the evaluation without increasing significantly its computational cost.

7. Numerical experiments

All our experiments are performed in IEEE-754 double precision as working precision corresponding to about 16 decimal digits. Here, we consider the polynomials with floating-point coefficients and floating-point entry x. In this section we present accuracy and timing results. All the programs about accuracy measurements have been written in Matlab 7.0, and the ones about timing measurements are written in C code.

7.1. Accuracy of the compensated Horner derivative algorithm

In the first experiment, we evaluate the 3rd derivative of the polynomial $p(x) = (x - 0.75)^5(x - 1)^{11}$, written in its expanded form, in the neighborhood of its multiple roots 0.75 and 1 (the tested polynomial is from [5]). Fig. 2 presents the

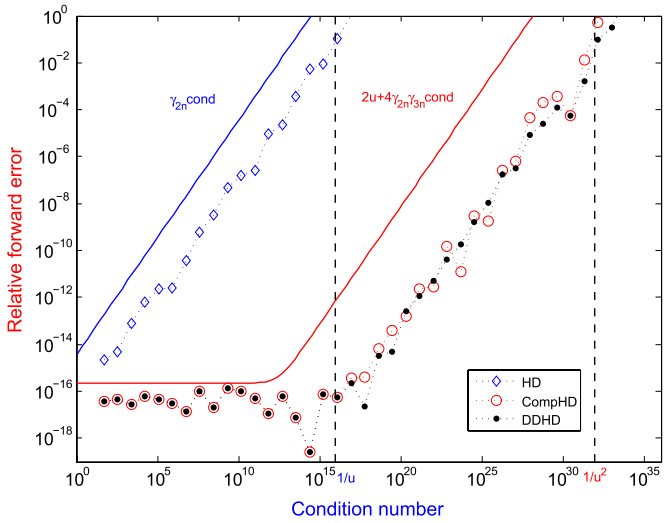


Fig. 3. Accuracy of the 3rd derivative evaluation of $p(x) = (x-1)^n$ for $n = 5, \ldots, 45$ in expanded form at the entry $x = \mathrm{fl}(1.333)$ with respect to the condition number.

evaluation for 400 equally spaced points in the intervals [0.74995, 0.75005] and [0.9935, 1.0065]. It is clear that the CompHD algorithm (Algorithm 2) gives much more smooth drawing than the original HD algorithm (Algorithm 1). The results show that the compensated algorithm is an effective method of accurate evaluation to recover the expected curve.

In the second experiment, we focus on the forward error bound of our compensated Horner derivative algorithm. As problem we consider the 3rd derivative evaluation of $p(x) = (x-1)^n$ for $n = 5, \ldots, 45$ in expanded form at the entry $x = \mathrm{fl}(1.333)$. The results of the tests performed with the CompHD and HD algorithm are reported in Fig. 3. As expected, when the condition number is smaller than $1/u$, the relative error of the result by CompHD algorithm (Algorithm 2) is equal to or smaller than $2u$. This relative error increases nearly linearly for the condition number between $1/u$ and $1/u^2$.

It is interesting to compare the compensated Horner derivative algorithm with other approaches to obtain high-precision. A standard way is by using multiple precision libraries, but if we just want to double the IEEE-754 double precision, the most efficient way is by using the double–double arithmetic [15,16]. Thus we compare CompHD algorithm with the Horner derivative algorithm written in double–double arithmetic (DDHD algorithm, see Algorithm 13), here the result of DDHD should be rounded to the working precision (double precision). As we see in Fig. 3, CompHD algorithm has nearly the same accuracy as DDHD algorithm.

In the next experiment, we illustrate the advantage of the running error bound (56) over the *a priori* one (44) in the accuracy of the error bound. We evaluate the 3rd derivative of $p(x) = (x-1)^8$ in expanded form for 400 equally spaced points in the interval [0.9935, 1.0065]. The results are reported in Fig. 4. The running error bound is more significant than the *a priori* one especially near $x = 1$.

7.2. Computational complexity and running time performances

In this subsection, we will pay attention to the computational complexity of all the algorithms HD, CompHD, CompHDwErr and DDHD, and then show the practical efficiency of our algorithm in terms of running time. Algorithms 1–3 and 13 are convenient to the error analysis, however they should be modified in the practical program. Hence in this subsection, it must be emphasized that the HD, CompHD, CompHDwErr and DDHD algorithms represent the corresponding modified algorithms written in C code after modification, such as taking the procedure Split(x) out of inner loop (referring to the technique in [17]). Similarly, we also take the procedure abs(x) out of recurrence in Algorithm 3.

We assume that one counts the process of abs(x), min(x, y) or max(x, y) as well as every addition or multiplication as one flop. And we let $y_i^j = a_n$, for $i + j = n$, as the initial values. Just as shown in Appendix, TwoSum, TwoProd, Split and FastTwoSum algorithms require 6, 17, 4 and 3 flops respectively. Then it is easy to obtain the computational cost of Algorithms 1–3 and 13:

- HD: $2(n-k)(k+1) + k - 1 + 4n$ flops,
- CompHD: $23(n-k)(k+1) + k + 4 + 4n$ flops,
- CompHDwErr: $29(n-k)(k+1) + k + 43 + 4n$ flops,
- DDHD: $38(n-k)(k+1) + k + 2 + 4n$ flops.

We measured the flop count ratios among HD, CompHD, CompHDwErr and DDHD and display the average ratios for $n = 50 : 5 : 1000$ and $k = 1 : 1 : 8$ in Table 1. We can observe that CompHD is as accurate as DDHD but only requires

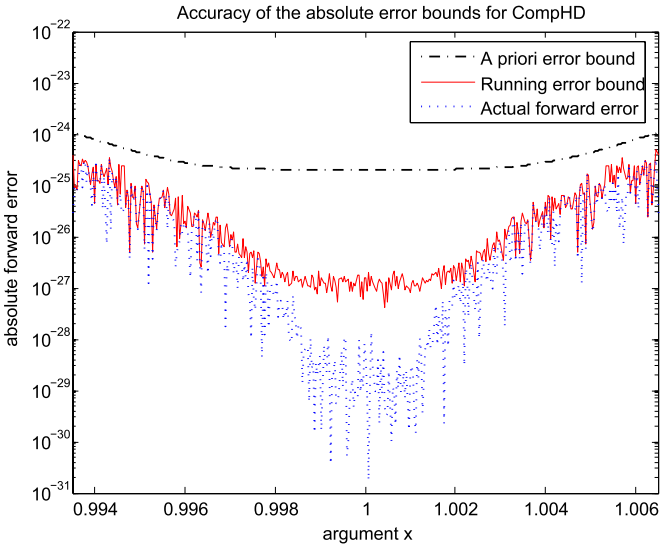


Fig. 4. Significance of the running error bound.

Table 1
Average ratios of the floating-point operations.

$\frac{\texttt{CompHD}}{\texttt{HD}}$	$\frac{\texttt{CompHDwErr}}{\texttt{HD}}$	$\frac{\texttt{DDHD}}{\texttt{HD}}$	$\frac{\texttt{CompHD}}{\texttt{CompHDwErr}}$	$\frac{\texttt{CompHD}}{\texttt{DDHD}}$
8.35	10.46	13.60	79.87%	61.47%

on the average only about 61% of flop count of that one. We also see that the over-cost due to the running error analysis for `CompHDwErr` is quite reasonable.

We also compared `HD`, `CompHD`, `CompHDwErr` and `DDHD` in terms of measured computing time. The experiments are performed on a laptop with a Intel(R) Core(TM) i5-2520M processor, with two cores each at 2.50 GHz. We used gcc version 4.4.5 with the compiler option "−o3" on Linux Ubuntu 11.04 and Microsoft Visual studio C++ 9.0 with the default compiler option "/od" on Windows 7 as two testing environments. Here, "/od" suppresses code movement, it simplifies the debugging process. Just like the statement in [5], we also deem that the computing time of these algorithms does not depend on the coefficients of the polynomials, nor the argument x, but on the degree n and the derivative order k. Thus, we generate the tested polynomials with random coefficients and arguments in the interval $[-1, 1]$, whose degree vary from 50 to 1000 by the step of 5 and derivative order vary from 1 to 8 by the step of 1 (In accordance with the case in flop count comparison in Table 1.)

The average time ratio are reported in Table 2. In contrast with the data in Table 1, we see that the measured computing time ratio of `CompHD/DDHD` is obviously better than the theoretical flop count one. Thanks to the analysis in terms of instruction level parallelism (ILP) (see details in [17,18]), this phenomenon is surprising, but reasonable. Briefly speaking, avoiding the renormalization step needed for double–double computations, the `CompHD` algorithm presents more ILP than its counterpart `DDHD` algorithm. Also note that `CompHD`, `CompHDwErr` and `DDHD` have much more instructions than `HD`, then they will introduce some more instruction-level parallelism. This partly explains the phenomenon that the first three ratios of running time are smaller than those of theoretical flop count. Considering that `CompHDwErr` has some procedures for computing the running error bound than `CompHDwErr`, which limits exploiting ILP. Then it is reasonable that the measured running time ratio between `CompHD` and `CompHDwErr` is smaller than the theoretical flop count one. Due to page limitation, the data-flow graphs for these algorithms to evaluate their ILP are not presented here, which are similar to the ones in [18]. In order to let readers appreciate the running time properties of the proposed algorithms in their own environments, the proposed C and Matlab codes in this paper could be downloaded on the website.[1]

We also consider how n and k modify the flop count ratios and the measured computing time ratios. We found that, for $n \gg k$ and for both floating point operation ratios and measured computing time ratios, the first three kinds of ratios `CompHD/HD`, `CompHDwErr/HD` and `DDHD/HD` increase with the increasing of k for some constant n. In the contrary, n has no significant impact on the ratios for some constant k. Hence, using the average ratios can partly well exhibit their properties, such as the phenomenon discussion above.

[1] https://sites.google.com/site/haojiangaccuratecomputing/clients/resources.

Table 2
Measured running time ratios.

	$\frac{\texttt{CompHD}}{\texttt{HD}}$	$\frac{\texttt{CompHDwErr}}{\texttt{HD}}$	$\frac{\texttt{DDHD}}{\texttt{HD}}$	$\frac{\texttt{CompHD}}{\texttt{CompHDwErr}}$ (%)	$\frac{\texttt{CompHD}}{\texttt{DDHD}}$ (%)
Linux gcc 4.4.5	3.85	6.44	8.14	61.76	47.42
Windows Vc++9.0	4.58	7.54	9.79	61.98	47.06

8. Simple application

In this section, we present an application to show the effectiveness of the proposed compensated Horner derivative algorithm. We consider the Newton's method in floating-point arithmetic [19] for solving the equation of the univariate polynomial $p(x)$ with a simple root. In that case, we improved the accurate Newton's method proposed in [20], by using CompHD algorithm to accurately compute the derivative. The classic Newton's method and the accurate Newton's method are presented as follows:

Algorithm 4. The classic Newton's method

$$x_0 = \xi$$
$$x_{i+1} = x_i - \frac{\texttt{Horner}(p,x_i)}{\texttt{HD}(p,x_i,1)}$$

Algorithm 5 ([20])**.** The accurate Newton's method

$$x_0 = \xi$$
$$x_{i+1} = x_i - \frac{\texttt{CompHorner}(x_i)}{\texttt{HD}(p,x_i,1)}$$

For a polynomial $p(x)$ with a simple zero x, x is not a zero of p', however, sometimes the evaluation of p' near x can be still ill-conditioned. Hence, in this case it is necessary to accurately evaluate $p'(x_i)$, then we choose the CompHD algorithm to modify Algorithm 5 and obtain the following algorithm.

Algorithm 6. The new accurate Newton's method

$$x_0 = \xi$$
$$x_{i+1} = x_i - \frac{\texttt{CompHorner}(p,x_i)}{\texttt{CompHD}(p,x_i,1)}$$

The following error analysis and numerical example proves and illustrates, respectively, that the convergence of iteration strongly depends on the accuracy of the derivative's evaluation when the problem of finding simple root is too ill-conditioned, and that the accuracy of the final iteration result depends on the accuracy with which the residual is computed.

8.1. Forward error

In floating-point arithmetic, Newton's method for general function f can be expressed as follows:

$$\widehat{v}_{i+1} = \widehat{v}_i \ominus \widehat{f}(\widehat{v}_i) \oslash \widehat{f}'(\widehat{v}_i) = \widehat{v}_i - \frac{\widehat{f}(\widehat{v}_i)}{\widehat{f}'(\widehat{v}_i)} + E_i \tag{59}$$

where

$$E_i = \frac{\widehat{f}(\widehat{v}_i)}{\widehat{f}'(\widehat{v}_i)}\delta_i + \varepsilon_i \tag{60}$$

with $|\varepsilon_i| < u|\widehat{v}_{i+1}|$ and $|\delta_i| < u$.

For notational convenience, we write $\bar{v} = \widehat{v}_{i+1}$, $v = \widehat{v}_i$ and $E = E_i$. We obtain

$$\bar{v} = v - \frac{\widehat{f}(v)}{\widehat{f}'(v)} + E. \tag{61}$$

The condition number of finding simple root of a univariate polynomial used in this paper is given as follows (see [20])

$$\texttt{cond}_{\texttt{root}}(p,x) = \frac{\widetilde{p}(|x|)}{|x|\,|p'(x)|}. \tag{62}$$

Theorem 6. *Assume that the simple root is α such that $f(\alpha) = 0, f'(\alpha) \neq 0$ with f is continuously differentiable in a neighborhood of the root, and in floating point arithmetic the computation of the derivative satisfies*

$$\textit{Assumption } 1: \quad \left|\frac{\widehat{f}'(v) - f'(v)}{f'(v)}\right| < \omega < \frac{1}{2}, \tag{63}$$

where ω is an upper bound given when v is closed to α, which shows the relative error bound of $f'(v)$. Assume also that for any v, obtained from the iteration from the initial value v_0 sufficiently close to the root α, satisfies

$$\text{Assumption 2}: \ 0 < \frac{f(v)}{f'(v)(v-\alpha)} < \mu_1. \tag{64}$$

Here, μ_1 is an upper bound which partly shows the ratio bound between the secant and the tangent. In the iterative process, $f'(v) \neq 0$ and $\widehat{f}'(v) \neq 0$, meanwhile ω and μ_1 satisfy

$$\text{Assumption 3}: \ \mu_1 + 2\omega \leq 2. \tag{65}$$

Newton's method (Algorithm 4) or its improved versions (Algorithms 5 and 6) in floating point arithmetic generates a sequence $\{\widehat{v}_i\}$ converging to v_*. Then assume that, when the iteration converges, there is

$$\text{Assumption 4}: \ 0 < \mu_2 < \frac{f(v_*)}{f'(v_*)(v_*-\alpha)}. \tag{66}$$

μ_2 means a lower bound of equation in (64) for the final iterated result v_*. The parameters ω, μ_1 and μ_2 used in Assumptions 1–4 will help to obtain the accuracies guaranteed by the algorithms as follows.

In case of Algorithm 4:

$$\left|\frac{\alpha - v_*}{v_*}\right| < C\gamma_{2n}\mathtt{cond}_{\mathtt{root}}(p, v_*). \tag{67}$$

In case of Algorithms 5 and 6:

$$\left|\frac{\alpha - v_*}{v_*}\right| < Ku + D\gamma_{2n}^2\mathtt{cond}_{\mathtt{root}}(p, v_*) \tag{68}$$

where C, K and D are the constants that consist of ω and μ_2.

Proof. First, expand f in a Taylor series around the simple root α,

$$0 = f(\alpha) = f(v) + (\alpha - v)f'(r) \tag{69}$$

where r is between v and α. From (61) and (69), we deduce

$$(\alpha - \bar{v}) = (\alpha - v)\left(1 - \frac{f'(r)}{\widehat{f}'(v)}\right) + \frac{\widehat{f}(v) - f(v)}{\widehat{f}'(v)} - E. \tag{70}$$

For convenience, let

$$G = 1 - \frac{f'(r)}{\widehat{f}'(v)} \quad \text{and} \quad g = \frac{\widehat{f}(v) - f(v)}{\widehat{f}'(v)} - E. \tag{71}$$

Second, we will prove that $|G| < 1$. From Assumption 1 (63), we can obtain

$$\left|\frac{f'(v)}{\widehat{f}'(v)}\right| < \frac{1}{1-\omega} \quad \text{and} \quad \left|\frac{\widehat{f}'(v)}{f'(v)}\right| < 1 + \omega. \tag{72}$$

And from (69), the first part of (72), Assumption 2 (64) and Assumption 3 (65) we have

$$\left|\frac{f'(r)}{\widehat{f}'(v)}\right| = \left|\frac{f(v)}{\widehat{f}'(v)(v-a)}\right| < \frac{1}{1-\omega}\left|\frac{f(v)}{f'(v)(v-a)}\right| < \frac{\mu_1}{1-\omega} < 2. \tag{73}$$

Meanwhile, from Assumption 1 (63), it is easy to obtain that $f'(v)$ and $\widehat{f}'(v)$ have the same sign. Hence, by (73) and the first inequality in Assumption 2 (64), we have

$$0 < \frac{f(v)}{\widehat{f}'(v)(v-a)} < 2. \tag{74}$$

Therefore, it follows that

$$|G| = \left|1 - \frac{f'(r)}{\widehat{f}'(v)}\right| = \left|1 - \frac{f(v)}{\widehat{f}'(v)(v-a)}\right| < 1. \tag{75}$$

As a consequence, the iterative process converges with $v \to v_*$.

Next, from (70) and the denotations in (71), it follows that

$$|\alpha - v_*| < \frac{|g_*|}{|1 - G_*|}. \tag{76}$$

Using the second inequality of (72) and Assumption 4 (66), we can deduce

$$\frac{1}{|1 - G_*|} = \left|\frac{\widehat{f}'(v_*)}{f'(r_*)}\right| = \left|\frac{\widehat{f}'(v_*)}{f'(v_*)}\right| \cdot \left|\frac{f'(v_*)}{f'(r_*)}\right| < (1+\omega)\left|\frac{f'(v_*)(v_* - \alpha)}{f(v_*)}\right| < \frac{1+\omega}{\mu_2} \tag{77}$$

where r_* is between v_* and α. From the definitions of g in (71) and E in (60), with the first inequality in (72), we have

$$|g_*| < \left|\frac{\widehat{f}(v_*) - f(v_*)}{\widehat{f}'(v_*)}\right| + |E_*| < \frac{1}{1-\omega}\left|\frac{\widehat{f}(v_*) - f(v_*)}{f'(v_*)}\right| + |E_*|, \tag{78}$$

where

$$|E_*| < u\left|\frac{\widehat{f}(v_*)}{\widehat{f}'(v_*)}\right| + u|v_*| < \frac{u}{1-\omega}\left(\left|\frac{f(v_*)}{f'(v_*)}\right| + \left|\frac{\widehat{f}(v_*) - f(v_*)}{f'(v_*)}\right|\right) + u|v_*|. \tag{79}$$

Here, by Assumption 2 (64) we have

$$\left|\frac{f(v_*)}{f'(v_*)}\right| < \mu_1|(\alpha - v_*)|. \tag{80}$$

Finally, we will show the forward error bounds of Algorithms 4–6. Here the general function f is reduce to the polynomial function p.

In case of Algorithm 4, the error bound of the polynomial evaluation is

$$|\widehat{p}(x) - p(x)| = |\mathtt{Horner}(p, x) - p(x)| < \gamma_{2n}\widetilde{p}(|x|). \tag{81}$$

Then from (76)–(81), using the definition of condition number (62) we have

$$\left[1 - \frac{(1+\omega)\mu_1}{(1-\omega)\mu_2}u\right]\left|\frac{\alpha - v_*}{v_*}\right| < \frac{1+\omega}{(1-\omega)\mu_2}(1+u)\gamma_{2n}\mathtt{cond}_{\mathtt{root}}(p, v_*) + u\frac{1+\omega}{\mu_2}.$$

Considering that $\mathtt{cond}_{\mathtt{root}} > 1$ and $u < \gamma_{2n}$, we simplify the inequality and approximately obtain

$$\left|\frac{\alpha - v_*}{v_*}\right| < \frac{(1+\omega)(2-\omega)}{(1-\omega)\mu_2}\gamma_{2n}\mathtt{cond}_{\mathtt{root}}(p, v_*). \tag{82}$$

In case of Algorithms 5 and 6, `CompHorner` is used for the evaluation of the polynomial, with the error bound

$$|\widehat{p}(x) - p(x)| = |\mathtt{CompHorner}(p, x) - p(x)| < u|p(x)| + \gamma_{2n}^2\widetilde{p}(|x|). \tag{83}$$

Similar to the discussion above, it follows that

$$\left[1 - \frac{(1+\omega)\mu_1}{(1-\omega)\mu_2}u(2+u)\right]\left|\frac{\alpha - v_*}{v_*}\right| < \frac{1+\omega}{(1-\omega)\mu_2}(1+u)\gamma_{2n}^2\mathtt{cond}_{\mathtt{root}}(p, v_*) + u\frac{1+\omega}{\mu_2},$$

and its approximate simplification

$$\left|\frac{\alpha - v_*}{v_*}\right| < \frac{1+\omega}{\mu_2}u + \frac{1+\omega}{(1-\omega)\mu_2}\gamma_{2n}^2\mathtt{cond}_{\mathtt{root}}(p, v_*). \quad \square \tag{84}$$

Assumption 1 (63) is necessary and reasonable. When the relative error is larger than 1/2, the computed result maintains nearly no more than one bit precision, which means there is nearly no useful information left. Assumption 2 (64) and Assumption 3 (65) will guarantee the convergence of the iteration. These assumptions are not strong that even when the derivative evaluation is too ill-conditioned they can still hold. From the expression G in (75), we deem that the convergence depends on the accuracy of the function's derivative but not that of the function itself. When the evaluation of the derivative is too ill-conditioned, such that $u\mathtt{cond}(p, x, 1) > 1$, Algorithm 6 still converges but Algorithm 5 does not.

In the proof of Theorem 6, we expand f in a Taylor series around simple root α and truncate the expansion at the first order, and then consider the convergence based on the hypothesis that the iteration converges linearly. Actually, the convergence factor of Newton method in floating point arithmetic is between 1 and 2. Here, assume that $f \in C^2$, we have

$$0 = f(\alpha) = f(v) + (\alpha - v)f'(v) + \frac{1}{2}(\alpha - v)^2 f''(c), \tag{85}$$

where c is between α and v. Then we obtain

$$(\alpha - \overline{v}) = (\alpha - v)\left(1 - \frac{f'(v)}{\widehat{f}'(v)}\right) - \frac{1}{2}(\alpha - v)^2\frac{f''(c)}{\widehat{f}'(v)} + \frac{\widehat{f}(v) - f(v)}{\widehat{f}'(v)}. \tag{86}$$

In exact arithmetic we have $\widehat{f}(v) = f(v)$ and $\widehat{f}'(v) = f'(v)$, then it shows that Newton's method is quadratic convergent. Note that for $\omega \ll 1$ in Assumption 1 (63), $1 - \frac{f'(v)}{\widehat{f}'(v)}$ will be close to zero, then the rate of convergence of iteration is nearly quadratic. It can be predicted that the adoption of accurate computing derivative does not obviously reduce the number of iterations, when the problem is not too ill-conditioned.

When the sequence converges, we present the error bounds of the three algorithms under Assumption 4 (66). It is required that μ_2 should not be too small, which is not a strong assumption and usually can be fulfilled. From (76)–(79), considering that E_* has a constant multiplicative factor u, we can assert that the accuracy guaranteed by the algorithm depends on the accuracy with which the residual is computed, that is $\widehat{f}(v_*) - f(v_*)$. On the contrary, due to Assumption 1 (63), the computation of the derivative in floating point arithmetic only results in the largest perturbation $\frac{1}{1-\omega}$ in (78), then we deem that the accuracy, with which the derivative is computed, has little effect on the final accuracy. It is also interesting to note that in the error bound (84) with adopting the accurate residual computation, if $\gamma_{2n}^2\mathtt{cond}_{\mathtt{root}}(p, v_*) \approx Ru$ for some constant R, it is expected to obtain a relative error of order $\mathcal{O}(u)$.

8.2. Example

To illustrate the effectiveness and accuracy of Algorithm 6, we compare Algorithms 4–6 by computing the simple real zero of the expanded form of the polynomial $p_n(x) = (x-1)^n - 2^{-31}$, for $n = 2 : 55$, the condition number of which varies roughly from 10^4 to 10^{32} at the real zero. Note that, if n is even, there are two real roots: $1 \pm 2^{-M/n}$; if n is odd, there is only one real root $1 + 2^{-M/n}$. We set the initial value $v_0 = 2$, then considering the local convergence property of the Newton method, we deem that the iteration sequence will converge to the real root $\alpha = 1 + 2^{-M/n}$.

There are two possible stopping criteria for terminating the iterative process: a stopping test based on the residual $f(x_k) < \varepsilon$ and on the increment $|x_{k+1} - x_k| < \mathrm{tol}$. If n is large, we will obtain $p_n'(\alpha) = n(\alpha - 1)^{n-1} \ll 1$, then the first stopping test is not reliable (see details on pp. 237 in [21]). Hence, in our example, we choose the stopping criterion $|\widehat{v}_{k+1} - \widehat{v}_k| < \mathrm{tol} = 10^{-15}$ and set the maximum admissible number of steps for the iterative process as $\mathrm{Num} = 100$.

We carry out some numerical experiments in MATLAB 7.0, for which the unit roundoff error is $u = 2^{-53} \approx 1.12 \times 10^{-16}$. First, we consider the evaluation for the values in the assumptions in Theorem 6 and the partial results are reported in Fig. 5. Here, we use symbolic toolbox in MATLAB to compute the values of Assumptions 1–4. We can observe that all the four assumptions are fulfilled to Algorithm 6 for condition number varying from 10^4 to 10^{30} corresponding to $n = 2 : 53$, to Algorithms 4 and 5 only for condition number roughly from 10^4 to 10^{14} and 10^{15}, respectively, corresponding to $n = 2 : 19$ and $n = 2 : 22$.

Clearly for $\omega = \frac{1}{3}, \mu_1 = \frac{4}{3}, \mu_2 = \frac{1}{3}$, we have the following two forward error bounds on the condition that the assumptions are met.

$$\text{Algorithm 4}: \quad \left|\frac{\alpha - v_*}{v_*}\right| < 10\gamma_{2n}\mathtt{cond}_{\mathtt{root}}(p, v_*),$$

$$\text{Algorithms 5 and 6}: \quad \left|\frac{\alpha - v_*}{v_*}\right| < 4u + 6\gamma_{2n}^2\mathtt{cond}_{\mathtt{root}}(p, v_*).$$

Fig. 6 shows the relative accuracy $|v_* - \alpha|/|v_*|$, where α is the exact root and v_* is the computed value by three algorithms. We also plot the *a priori* error estimations.

As we can see, Algorithm 6 is as accurate as Algorithm 5, if the condition number is less than 10^{16}, and yields nearly a full precision. However, when the condition number is equal to or greater than 10^{16}, the inaccurate evaluation for the derivative causes the convergence failure of Algorithm 5 as predicted by the second one in Fig. 5. In contrast, Algorithm 6 still converges and presents the accuracy which decreases almost linearly. Considering that the first three assumptions in Theorem 6 are sufficient but not necessary for the iteration's convergence, it is reasonable that even though in Algorithm 5 the assumptions is not fulfilled, with respect to the condition numbers between 10^{15} and 10^{16}, the iterative process still converges. Here, "converge" means that iteration stops with satisfying the increment stopping criteria tol and then the iteration number is smaller than the maximum admissible number Num.

In the numerical tests we find a phenomenon, that is some iterations stop for the maximum admissible number of iteration steps without satisfying the increment stopping criteria. There are two reasons. One is that the iteration does not converge. The other is that the increment stopping criteria tol may be too small for some iteration.

Most of the cases with this phenomenon are due to the first reason. For Algorithm 4, it usually happens when the condition number is larger than 10^{10} with the percentage about 87.04%. For Algorithm 5, it usually happens when the condition number is larger than 10^{16} with the percentage 61.11%. We perform the numerical tests for $\mathrm{tol} = 10^{-8}, \ldots, 10^{-15}$ with the largest maximum iteration number $\mathrm{Num} = 100$. Then we find that, for Algorithms 4 and 5, the percentages of the phenomenon

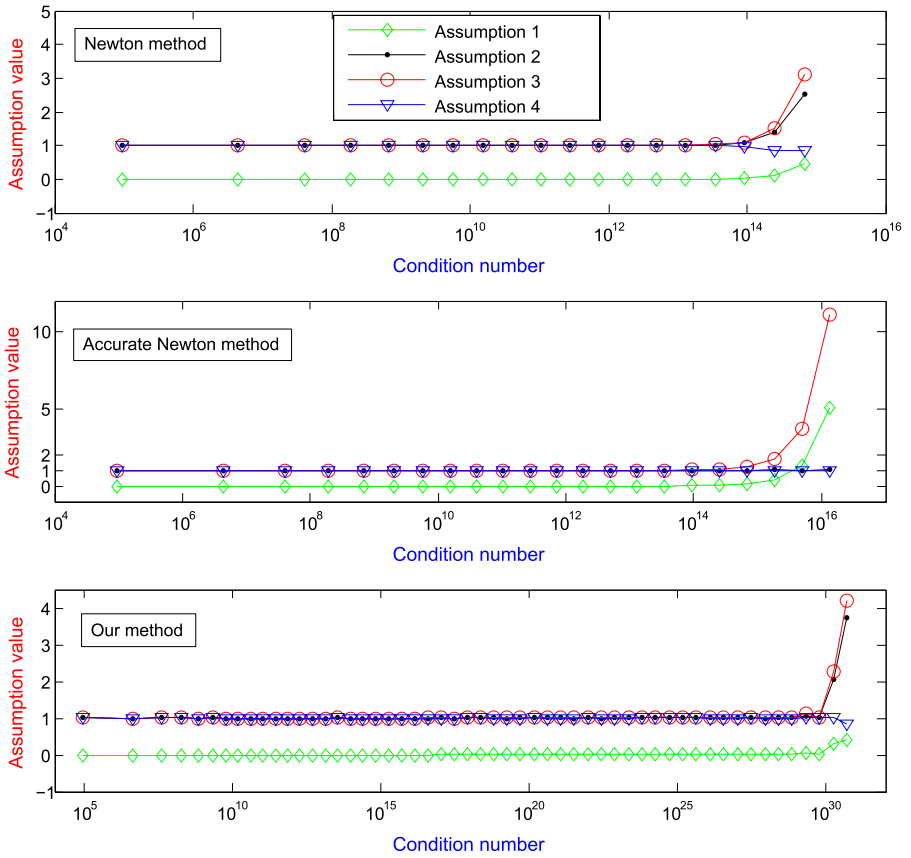


Fig. 5. Assumption values of three algorithms with respect to condition number. Here, Assumptions 1 and 2 represent the largest $|\widehat{f}'(v) - f'(v)/f'(v)|$ and the largest $f(v)/f'(v)(v-\alpha)$ for all of the iterates v with respect to some condition number, respectively; Assumption 3 represents the summation of Assumption 2 and double Assumption 1; Assumption 4 represents the smallest $f(v_*)/f'(v_*)(v_*-\alpha)$ with respect to some condition number.

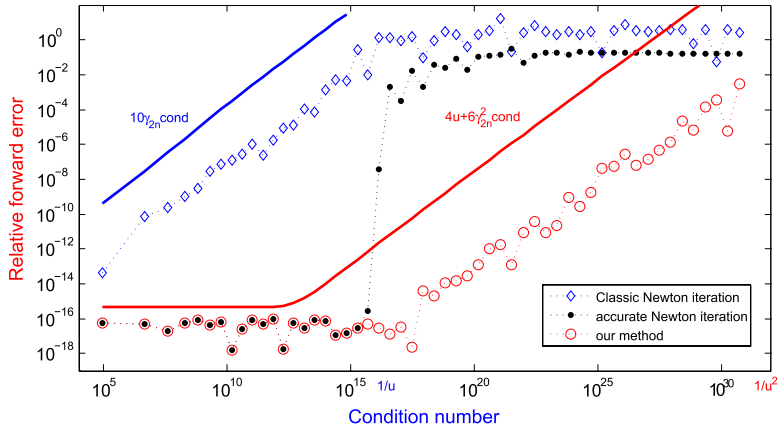


Fig. 6. Accuracy of the three algorithms with respect to the condition number.

happening are nearly the same. Hence, we affirm that the reason is that the iteration does not converge. Finally, we deem that Algorithms 4 and 5 do not converge in the case of too-ill conditioned problem.

However, for Algorithm 6, this phenomenon happens on only a few point (about 7.41%) with tol $= 10^{-15}$ and Num $= 100$. If we reset tol $= 10^{-8}$, we found that no iteration stop for the maximum admissible number. Hence, we deduce in this case the phenomenon is due to the setting of the increment stopping criteria tol. Comparing with the exact Newton method, our method (Algorithm 6) still has the rounding errors, so sometimes the numerical iteration of our method has an oscillation near the convergence result. If tol is not small enough, the iteration seems to converge. But if tol is small enough, the

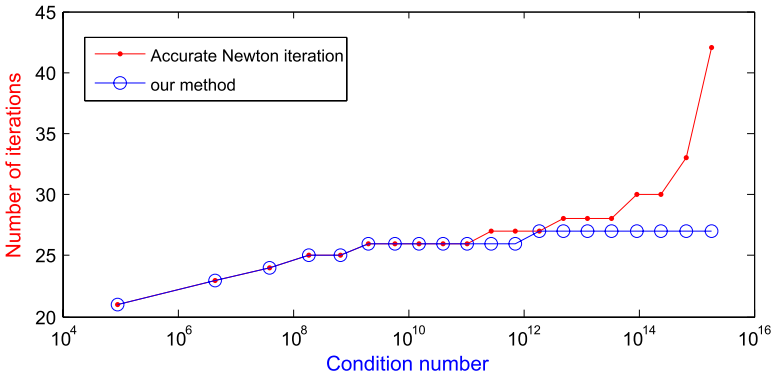


Fig. 7. Accurate computing derivative's effect on the rate of convergence of Newton's method.

effectiveness of oscillation will be significant, then it seems that the iteration does not converge. Whatever we can deem that Algorithm 6 converges in the case of too-ill conditioned problem.

Obviously, for Algorithms 4 and 5, when the iteration does not converge, the larger maximum admissible number of iteration is useless. Also for Algorithm 6, if tol is small enough, it is useless using larger maximum admissible number. The discussion above also shows that it is a good option to choose the maximum admissible number of iteration Num $= 100$ and the increment stopping criteria tol $= 10^{-15}$ in Fig. 6.

We also present the behavior of the convergence rate of Algorithms 5 and 6 when the problem is not too ill-conditioned. The results in Fig. 7 illustrate, as predicted by our analysis before, that the accurate computing the derivative has no obvious effect on the rate of convergence for the iteration. Hence, if the problem is not too ill-conditioned (condition number is smaller than 10^{15}), it is not required to accurately compute the derivative. In contrast, it is necessary to compute the derivative accurately to guarantee the convergence of iterative process when the problem is too ill-conditioned.

In conclusion, the adoption of accurate evaluation for the derivative guarantees the convergence of Newton's method when the problem is ill-conditioned, and that of accurate evaluation for the residual improves the accuracy of final iterative result.

9. Conclusion and future work

In this paper we have provided and analyzed the compensated Horner derivative algorithm for accurate evaluation derivatives of a polynomial in power basis. The results confirm that the algorithm can yield an evaluation as accurate as if computed with twice working precision and rounded back to working precision. Numerical tests illustrate the proposed algorithm performs more well than the other competitive algorithm, such as the original algorithm in the double–double format. A simple application of the proposed is performed showing its effectiveness.

As the future work, we will consider using our compensated Horner derivative algorithm to accurately compute $p'(x)$ or $p''(x)$, when the root has multiplicity, then by which, we can modify Newton and other higher-order iterations. For the evaluation of the derivative of the polynomials in other forms, we will also propose new accurate algorithms. The compensated de Casteljau algorithm to accurately evaluate the derivative of a polynomial in Bernstein form has been proposed in [22]. For the evaluation of pth derivative of Jacobi series, which is studied in [23], the corresponding accurate compensated algorithm will be considered based on [24].

Acknowledgments

The authors are thankful to the reviewers and editor for their valuable comments and suggestions.

Appendix. Double–double library

The QD package [16,25], which provides double–double and quad-double arithmetic, is based on the following algorithms for the accurate addition and multiplication of two IEEE 64-bit operands using rounded arithmetic, due to Knuth [11] and Dekker [12]:

Algorithm 7 ([11]). Error-free transformation of the sum of two floating-point numbers

$$\begin{aligned}&\text{function } [x, y] = \texttt{TwoSum}(a, b)\\ &\quad x = a \oplus b\\ &\quad z = x \ominus a\\ &\quad y = (a \ominus (x \ominus z)) \oplus (b \ominus z)\end{aligned}$$

Algorithm 7 requires 6 flops.

Algorithm 8 ([12]). Error-free transformation of the sum of two floating-point numbers ($|a| \geq |b|$)

$$\begin{aligned}&\text{function } [x, y] = \texttt{FastTwoSum}(a, b)\\ &\quad x = a \oplus b\\ &\quad y = (a \ominus x) \oplus b\end{aligned}$$

Algorithm 7 requires 3 flops.

Algorithm 9 ([12]). Error-free split of a floating-point numbers into two parts

$$\begin{aligned}&\text{function } [x, y] = \texttt{Split}(a)\\ &\quad c = \text{factor} \otimes a \text{ (in double precision factor } = 2^{27} + 1)\\ &\quad x = c \ominus (c \ominus a)\\ &\quad y = a \ominus x\end{aligned}$$

Algorithm 9 requires 4 flops.

Algorithm 10 ([12]). Error-free transformation of the product of two floating-point numbers

$$\begin{aligned}&\text{function } [x, y] = \texttt{TwoProd}(a; b)\\ &\quad x = a \otimes b\\ &\quad [a1, a2] = \texttt{Split}(a)\\ &\quad [b1, b2] = \texttt{Split}(b)\\ &\quad y = a2 \otimes b2 \ominus (((x \ominus a1 \otimes b1) \ominus a2 \otimes b1) \ominus a1 \otimes b2)\end{aligned}$$

Algorithm 10 requires 17 flops.

Algorithm 11 ([15]). Addition of a double–double number and a double–double number

$$\begin{aligned}&\text{function } [rh, rl] = \texttt{add\_dd\_dd}(ah, al, bh, bl)\\ &\quad [sh, sl] = \texttt{TwoSum}(ah, bh)\\ &\quad [th, tl] = \texttt{TwoSum}(al, bl)\\ &\quad sl = sl \oplus th\\ &\quad th = sh \oplus sl\\ &\quad sl = sl \ominus (th \ominus sh)\\ &\quad tl = tl \oplus sl\\ &\quad [rh, rl] = \texttt{FastTwoSum}(th, tl)\end{aligned}$$

Algorithm 11 requires 20 flops.

Algorithm 12 ([15,17]). Multiplication of a double–double number by a double number

$$\begin{aligned}&\text{function } [rh, rl] = \texttt{prod\_dd\_d}(ah, al, b)\\ &\quad [th, tl] = \texttt{TwoProd}(ah, b)\\ &\quad tl = al \otimes b \oplus tl\\ &\quad [rh, rl] = \texttt{FastTwoSum}(th, tl)\end{aligned}$$

Algorithm 12 requires 22 flops.

Algorithm 13. Horner derivative algorithm in double–double arithmetic

$$\begin{aligned}&\text{function } [\text{resh}, \text{resl}] = \texttt{DDHD}(p, x, k)\\ &yh_i^j = 0, \text{for } i = 0 : 1 : k, \text{and } j = n + 1 : -1 : 0\\ &yh_{-1}^{j+1} = a_j, \text{for } j = n : -1 : 0\\ &yl_i^j = 0, \text{for } i = 0 : 1 : k, \text{and } j = n + 1 : -1 : 0\\ &\text{for } j = n : -1 : 0\\ &\quad \text{for } i = \min(k, n - j) : -1 : \max(0, k - j)\\ &\quad\quad [rh1, rl1] = \texttt{prod\_dd\_d}(yh_i^{j+1}, yl_i^{j+1}, x);\\ &\quad\quad [yh_i^j, yl_i^j] = \texttt{add\_dd\_dd}(rh1, rl1, yh_{i-1}^{j+1}, yl_{i-1}^{j+1})\\ &\quad \text{end}\\ &\text{end}\\ &\text{resh} = k! \times yh_k^0\\ &\text{resl} = k! \times yl_k^0\end{aligned}$$

References

[1] K.H. Müller, Rounding error analysis of Horner's scheme, Computing 30 (1983) 285–303.
[2] F.W.J. Olver, Error bounds for polynomial evaluation and complex arithmetic, IMA J. Numer. Anal. 6 (1986) 373–379.
[3] C.S. Burrus, Horner's method for evaluation and deflating polynomials, connexions, November 28, 2007. http://cnx.org/content/m15099/1.6/.
[4] N.J. Higham, Accuracy and Stability of Numerical Algorithm, second ed., SIAM, Philadelphia, 2002.
[5] S. Graillat, P. Langlois, N. Louvet, Algorithms for accurate, validated and fast polynomial evaluation, in: Verified Numerical Computation, Japan J. Indust. Appl. Math. 26 (2009) 191–214.
[6] S. Graillat, P. Langlois, N. Louvet, Compensated Horner scheme, Research Report RR2005-04, LP2A, University of Perpignan, France, July 2005.
[7] P. Langlois, N. Louvet, How to ensure a faithful polynomial evaluation with the compensated Horner algorithm, in: P. Kornerup, J.M. Muller (Eds.), 18th IEEE International Symposium on Computer Arithmetic, IEEE Computer Society, 2007, pp. 141–149.
[8] S.M. Rump, T. Ogita, S. Oishi, Accurate floating-point summation part I: faithful rounding, SIAM J. Sci. Comput. 31 (2008) 189–224.
[9] S.M. Rump, T. Ogita, S. Oishi, Accurate floating-point summation part II: sign, K-fold faithful and rounding to nearest, SIAM J. Sci. Comput. 31 (2008) 1269–1302.
[10] T. Ogita, S.M. Rump, S. Oishi, Accurate sum and dot product, SIAM J. Sci. Comput. 26 (2005) 1955–1988.
[11] D.E. Knuth, The Art of Computer Programming: Seminumerical Algorithms, third ed., Addison-Wesley, 1998.
[12] T.J. Dekker, A floating-point technique for extending the available precision, Numer. Math. 18 (1971) 224–242.
[13] W. Miller, Graph transformations for Roundoff analysis, SIAM J. Comput. 5 (1976) 204–216.
[14] O. Caprani, Roundoff errors in floating-point summation, BIT 15 (1975) 5–9.
[15] X.S. Li, J.W. Demmel, D.H. Bailey, G. Henry, Y. Hida, J. Iskandar, W. Kahan, S.Y. Kang, A. Kapur, M.C. Martin, B.J. Thompson, T. Tung, D.J. Yoo, Design, implementation and testing of extended and mixed precision BLAS, ACM Trans. Math. Softw. 28 (2) (2002) 152–205.
[16] D.H. Bailey, Library for double–double and quad-double arithmetric, QD Library. Available from: http://www.nersc.gov/~dhbailey/mpdist/mpdist.html.
[17] N. Louvet, Compensated algorithms in floating-point arithmetic: accuracy, validation, performances, Ph.D. Thesis, Université de Perpignan Via Domitia, November, 2007.
[18] P. Langlois, N. Louvet, More instruction level parallelism explains the actual efficiency of compensated algorithms, Technical Report, hal-00165020, DALI Research Team, University of Perpignan, France, 2007.
[19] F. Tisseur, Newton's method in floating point arithmetic and iterative refinement of generalized eigenvalue problems, SIAM J. Matrix Anal. Appl. 22 (4) (2001) 1038–1057.
[20] S. Graillat, Accurate simple zeros of polynomials in floating point arithmetic, Comput. Math. Appl. 56 (2008) 1114–1120.
[21] A. Quarteroni, R. Sacco, F. Saleri, Numerical Mathematics, second ed., Springer Verlag, 2007.
[22] H. Jiang, S.G. Li, L.Z. Cheng, F. Su, Accurate evaluation of a polynomial and its derivative in Bernstein form, Comput. Math. Appl. 60 (2010) 744–755.
[23] R. Barrio, J.M. Peña, Numerical evaluation of the pth derivative of Jacobi series, Appl. Numer. Math. 43 (2002) 335–357.
[24] H. Jiang, R. Barrio, H.S. Li, X.K. Liao, L.Z. Cheng, F. Su, Accurate evaluation of a polynomial in Chebyshev form, Appl. Math. Comput. 217 (2011) 9702–9716.
[25] Y. Hida, X.Y. Li, D.H. Bailey, Algorithms for quad-double precision floating point arithmetic, in: 15th IEEE Symposium on Computer Arithmetic, IEEE Computer Society, 2001, pp. 155–162.