On the Efficient Implementation of an Implicit Discrete-Time Differentiator

[1,3]J. E. CARVAJAL-RUBIO, [2]J. D. SANCHEZ-TORRES, [3]M. DEFOORT, [3]M. DJEMAI, [1]A. G. LOUKIANOV

[1]Dept. of Electrical Engineering, CINVESTAV-IPN Guadalajara, Zapopan, MEXICO.
[2]Dept. of Mathematics and Physics, ITESO, Tlaquepaque, MEXICO.
[3]LAMIH, CNRS UMR 8201, Polytechnic University of Hauts-de-France, Valenciennes, FRANCE.

Abstract—New methodologies are designed to reduce the time complexity of an implicit discrete-time differentiator and the simulation time to implement it. They rely on Horner's method and the Shaw-Traub algorithm. The algorithms are compared for differentiators of order 3, 7, and 10. The Half-Horner and Full-Horner methods showed the best performance and time complexity.

Key Words: Time complexity, Implicit Discretization, Differentiator, Sliding-Mode

Received: April 28, 2021. Revised: May 22, 2021. Accepted: May 25, 2021. Published: May 28, 2021.

1. Introduction

An online differentiator is useful for several applications, such as control laws based on derivatives of a signal, estimation of unmeasured states and parameters [1]–[3]. The well-known homogeneous differentiator [4] was proposed in continuous-time and allow to estimate the first n derivatives of a signal, if its n-th derivative has a known Lipschitz constant $L > 0$. As the standard differentiator is implemented in digital systems [5], a discrete-time version is implemented, for instance the explicit and implicit discrete-time realizations in [6]–[10]. Particularly, the implicit discrete-time differentiator, proposed in [10], has a remarkable reduction of the numerical chattering and preserves the main properties of the continuous-time differentiator [4], i.e., homogeneity property, asymptotic accuracy and convergence of its errors in finite-time to a vicinity of the origin. The main drawback of the implicit-discrete time differentiator is that the root of a polynomial has to be calculated almost each iteration for its implementation. The polynomial can not be calculated previous to its implementation because one of its parameters is updated each iteration. To implement this differentiator, in [10] was proposed the Halley's method [11], which converges to the unique positive root of the polynomial with 3-order for a non-restrictive set of initial conditions.

Although Halley's method reduces the time required to implement the implicit differentiator, Halley's method needs the evaluation of the function and its first two derivatives. Then Halley's algorithm has a quadratic time with respect to n. Several algorithm has been proposed to reduce the number of basic operations (time complexity [12]) required to evaluate a polynomial and its derivatives, for instance, Horner's method [13], Shaw-Traub algorithm [14], and the De Jong Van algorithm [15]. Other algorithms adapt or precondition parameters [16], they will not be considered because in this work the polynomials are updated each iteration. On the other hand, the evaluation of the implicit differentiator, after calculate the respective roots, presents a cubic time, but it can be reduced to a quadratic time or linearithmic time using Horner's and the discrete Fourier transform [17], respectively.

The first contribution of this paper is to introduce new methodologies designed to implement the implicit discrete-time differentiator [18] (HIDD), which allow to reduce the time complexity of the differentiator. The second one is a numeric comparative between them. This work is organized as follows. In Section II, implicit differentiator is defined. Additionally, Section III contains the proposed algorithms for the implementation of implicit differentiator and its time complexity is calculated. Section IV aims to compare the time complexity of the algorithms and the simulation time for its implementation. In Section V, the main results of the paper are summarized and the future work is presented.

2. Implicit Discrete-time Differentiator

Let τ be the sampling time constant and defined as $\tau = t_{k+1} - t_k$ for $k = 0, 1, 2, \cdots$. Then HIDD is defined by the following equations:

$$\begin{aligned} \boldsymbol{z}_{k+1} &= \boldsymbol{\Phi}(\tau)\, \boldsymbol{z}_k + \boldsymbol{B}^*(\tau)\, \boldsymbol{v}(\widetilde{\sigma}_{0,k+1}), \\ \boldsymbol{v}(\widetilde{\sigma}_{0,k+1}) &= \left[\Psi_{0,n}(\widetilde{\sigma}_{0,k+1}) \;\; \cdots \;\; \Psi_{n,n}(\widetilde{\sigma}_{0,k+1})\right]^T, \\ \Psi_{i,n}(\widetilde{\sigma}_{0,k+1}) &= -\lambda_{n-i} L^{\frac{i+1}{n+1}} \left|\widetilde{\sigma}_{0,k+1}\right|^{\frac{n-i}{n+1}} \xi_k. \end{aligned} \quad (1)$$

For HIDD, $z_{i,k+1}$ corresponds to the estimation of $f^i_{0,k}$. On the other hand, $\boldsymbol{\Phi}(\tau)$ and $\boldsymbol{B}^*(\tau)$ have the following representation:

$$\boldsymbol{\Phi}(\tau) = \begin{bmatrix} 1 & \tau & \frac{\tau^2}{2!} & \cdots & \frac{\tau^{n-1}}{(n-1)!} & \frac{\tau^n}{n!} \\ 0 & 1 & \tau & \cdots & \frac{\tau^{n-2}}{(n-2)!} & \frac{\tau^{n-1}}{(n-1)!} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & \tau \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix},$$

$$\boldsymbol{B}^*(\tau) = \begin{bmatrix} \tau & \frac{\tau^2}{2!} & \frac{\tau^3}{3!} & \cdots & \frac{\tau^n}{n!} & \frac{\tau^{n+1}}{(n+1)!} \\ 0 & \tau & \frac{\tau^2}{2!} & \cdots & \frac{\tau^{n-1}}{(n-1)!} & \frac{\tau^n}{n!} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \tau & \frac{\tau^2}{2!} \\ 0 & 0 & 0 & \cdots & 0 & \tau \end{bmatrix}.$$

Regarding $\widetilde{\sigma}_{0,k+1}$ and ξ_k, they are calculated according to the following lemma:

Lemma 1: ([10]) Let a_l and b_k be defined as:

$$a_l = \frac{\tau^{n-l+1}}{(n-l+1)!}\lambda_l L^{\frac{n-l+1}{n+1}}, \quad l = 0, \dots, n;$$
$$b_k = -\sigma_{0,k} - \sum_{l=1}^{n} \frac{\tau^l}{l!} z_{l,k}.$$

Then $\widetilde{\sigma}_{0,k+1} \in \mathbb{R}$ and ξ_k is the unique pair $(\widetilde{\sigma}_{0,k+1}, \xi_k)$ defined conforming to the following 3 cases:

- If $b_k > a_0$, then $\xi_k = \{-1\}$ and $\widetilde{\sigma}_{0,k+1} = -(r_0)^{n+1} \in \mathbb{R}^-$ where r_0 is the unique positive root of the following polynomial:

$$p(r) = r^{n+1} + a_n r^n + \cdots + a_1 r + (-b_k + a_0). \quad (2)$$

- If $b_k \in [-a_0, a_0]$, then $\widetilde{\sigma}_{0,k+1} = 0$ and $\xi_k = \left\{-\frac{b_k}{a_0}\right\}$.
- If $b_k < -a_0$, then $\xi_k = \{1\}$ and $\widetilde{\sigma}_{0,k+1} = r_0^{n+1} \in \mathbb{R}^+$ where r_0 is the unique positive root of the following polynomial:

$$p(r) = r^{n+1} + a_n r^n + \cdots + a_1 r + (b_k + a_0). \quad (3)$$

Note that under the assumption of a constant sampling time τ, a_l, and the elements of $\boldsymbol{\Phi}(\tau)$ and $\boldsymbol{B}^*(\tau)$ can be calculated previous to the implementation of HIDD. Opposite to the explicit discrete-time differentiators, HIDD requires estimating roots of different polynomials each iteration. They are estimated by using the Halley's method, which implies an evaluation of the polynomials (2), (3) and its first two derivatives multiple times.

2.1. Halley's Method

As it was mentioned previously, HIDD requires estimating r_0. In [10] the Halley's method was proposed as a good alternative, which experimentally needed 3 iteration to estimate r_0. It is applied as follows:

$$r_{0,j+1} = r_{0,j} - \frac{2\frac{dp(r)}{dr}|_{r=r_{0,j}} p(r_{0,j})}{2\left(\frac{dp(r)}{dr}|_{r=r_{0,j}}\right)^2 - \frac{d^2p(r)}{dr^2}|_{r=r_{0,j}} p(r_{0,j})} \quad (4)$$

As $p\left((b_k - a_0)^{\frac{1}{n+1}}\right) > 0$ and $p\left((-b_k - a_0)^{\frac{1}{n+1}}\right) > 0$ for the polynomials (2) and (3), then $r_0 \in \left[0, (b_k - a_0)^{\frac{1}{n+1}}\right]$ and $r_0 \in \left[0, (-b_k - a_0)^{\frac{1}{n+1}}\right]$, respectively. The above matches with the Cauchy's bound for the roots of a polynomial [13]. In [10] was demonstrated that the Halley's method converges monotonically to r_0 with an convergence order 3, [11], for any initial $r_{0,0}$ belonging to the previous sets of initial conditions. Hence, the following initial conditions were used:

$$r_{0,0} = \left(\frac{b_k - a_0}{2}\right)^{1/(n+1)}, \qquad \text{for } b_k > a_0,$$
$$r_{0,0} = \left(\frac{-b_k - a_0}{2}\right)^{1/(n+1)}, \qquad \text{for } b_k < -a_0.$$

3. Main Results

As it was mentioned previously, the objective of this work is to reduce the time complexity of HIDD. First, the variables ϕ_i and $\bar{b}^*$ are defined as:

$$\phi_i = \frac{\tau^{i-1}}{(i-1)!},$$
$$\bar{b}^*_{i,j} = \frac{\tau^{j+1-i}}{(j+1-i)!}\lambda_{n-j+1} L^{\frac{j}{n+1}}, \quad (5)$$

for $i = 1, 2, \cdots, n+1$ and $j = i, i+1, i+2, \cdots, n+1$. It allows to rewrite (1) as follows:

- If $b_k > a_0$,

$$z_{i,k+1} = \sum_{j=i}^{n} \phi_{j-i+1} z_{j,k} + \bar{b}^*_{i+1,j+1} r_0^{n-j}, \quad i = 0, 1, \cdots, n. \quad (6)$$

- If $b_k \in [-a_0, a_0]$,

$$z_{0,k+1} = b_k + \sum_{j=0}^{n} \frac{\tau^j}{j!} z_{j,k},$$
$$z_{i,k+1} = \bar{b}^*_{i+1,n+1}\left(\frac{b_k}{a_0}\right) + \sum_{j=i}^{n} \phi_{j-i+1} z_{j,k}, \quad i = 1, 2, \cdots, n. \quad (7)$$

- If $b_k < -a_0$,

$$z_{i,k+1} = \sum_{j=i}^{n} \phi_{j-i+1} z_{j,k} - \bar{b}^*_{i+1,j+1} r_0^{n-j}, \quad i = 0, 1, \cdots, n. \quad (8)$$

3.1. Direct Evaluation

The number of additions and subtraction, N_A and the number of multiplications and divisions, N_M, needed to evaluate $z_{i,k+1}$ directly, after obtain r_0, are calculated as:

$$N_{A1}(n) = (n+1)^2,$$
$$N_{M1}(n) = \frac{n^3}{6} + n^2 - \frac{1}{6}n - 1.$$

Therefore, taking into account the $(n+1)$ assignations of $z_{i,k+1}$, the time complexity is given as:

$$T_1(n) = \frac{n^3}{6} + 2n^2 + \frac{17}{6}n + 1,$$

which is a cubic time. On the other hand, the Halley's method is used recursively each iteration. To evaluate the derivatives and its derivatives, one could storage the following variables to reduce the number of operations.

$$\begin{aligned} c_{n+1} &= n+1, \\ c_i &= i a_i, \qquad \text{for } \ i=1,2,\cdots,n; \\ d_{n+1} &= n(n+1), \\ d_i &= i(i-1)a_i, \ \text{ for } \ i=2,3,\cdots,n. \end{aligned} \tag{9}$$

Additionally, $\bar{j}$ is defined as the number of iteration used to estimate r_0, then the number of additions, subtraction, multiplications and divisions used to evaluate the polynomials and its derivatives are given as:

$$\begin{aligned} N_{A2}(n) &= \bar{j}\,(3n+1)\,, \\ N_{M2}(n) &= \bar{j}\left(\frac{3}{2}n^2+\frac{3}{2}n\right). \end{aligned}$$

Since Halley's method is implemented 3 times each iteration for HIDD, $\bar{j} = 3$. Therefore, taking into account the assignation of values, the evaluation of (4), its initialization and comparatives:

$$T_2(n) = \frac{9}{2}n^2 + \frac{27}{2}n + 46. \tag{10}$$

where one of the operations is a $(n+1)$-th root and a for-loop was considered. Hence, the complexity of the algorithm is cubic and it is defined as:

$$T(n) = \frac{n^3}{6} + \frac{13}{2}n^2 + \frac{110}{6}n + 48.$$

where the multiplications and subtraction needed to evaluate b_k were taking into account.

3.2. Horner Method

Although the variables $\phi_\cdot$, $\bar{b}^*_{\cdot,\cdot}$, $c_\cdot$, $d_\cdot$ reduces the number of basic operations, it does not reduce time complexity of the realization (1) with respect to n. Based on the Horner methodology, one could calculate $z_{i,k}$ as follows:

- If $b_k > a_0$

$$\begin{aligned} & z_{i,k+1} = \sum_{j=i}^{n} \phi_{j-i+1} z_{j,k} + \dots \\ & \dots + (\cdots((\bar{b}^*_{i+1,i+1})r_0 + \bar{b}^*_{i+1,i+2})\cdots)r_0 + \bar{b}^*_{i+1,n+1}. \\ & i = 0,1,\cdots,n. \end{aligned} \tag{11}$$

- If $b_k < -a_0$

$$\begin{aligned} & z_{i,k+1} = \sum_{j=i}^{n} \phi_{j-i+1} z_{j,k} - \dots \\ & \dots - (\cdots((\bar{b}^*_{i+1,i+1})r_0 + \bar{b}^*_{i+1,i+2})\cdots)r_0 + \bar{b}^*_{i+1,n+1}. \\ & i = 0,1,\cdots,n. \end{aligned} \tag{12}$$

This methodology presents the following number of basic operations:

$$\begin{aligned} N_{A3}(n) &= (n+1)^2, \\ N_{M3}(n) &= n(n+1). \end{aligned}$$

As $\boldsymbol{\Phi}(\tau)$ and $\boldsymbol{B}(\tau)$ are Toeplitz matrix [19], the time complexity of evaluate $z_{i,k+1}$ could be reduce to a linearithmic time $(n\log n)$ using the discrete Fourier transform. This alternative will be analyzed in a future work. To evaluate the polynomials and its derivatives, n is considered greater than 1. Here two methodologies based on Horner's method are analyzed, the first one is evaluate the polynomials and derivatives as follows:

$$\begin{aligned} p\,(r) &= (\cdots((r+a_n)r+a_{n-1})\cdots)r + a_0 \pm b_k, \\ \frac{dp(r)}{dr} &= (\cdots((c_{n+1}r+c_n)r+c_{n-1})\cdots)r + c_1, \\ \frac{d^2p(r)}{dr^2} &= (\cdots((d_{n+1}r+d_n)r+d_{n-1})\cdots)r + d_2. \end{aligned} \tag{13}$$

The methodology (13) use the following number of basic operations:

$$\begin{aligned} N_{A4}(n) &= \bar{j}\,(3n+1)\,, \\ N_{M4}(n) &= \bar{j}\,(3n-1)\,. \end{aligned}$$

Similar to (10), one obtains:

$$T_4(n) = 18n + 43.$$

However, one could take advantage of the evaluation of $p(r)$ to evaluate $\frac{dp(r)}{dr}$ and $\frac{d^2p(r)}{dr^2}$ with the following methodology for $n \geq 2$:

$$\begin{aligned} F_{i+1} &= rF_i + a_{n-i-1}, \\ dF_{i+1} &= rdF_i + F_{i+1}, \\ ddF_{i+1} &= rddF_i + dF_{i+1}, \end{aligned}$$

for $i = 0,\cdots,n-3$, with $F_0 = r + a_n$, $dF_0 = r + F_0$, $ddF_0 = r + dF_0$, and the value of the evaluation is given as:

$$\begin{aligned} F_{n-1} &= rF_{n-2} + a_1, \\ p(r) &= rF_{n-1} + a_0 \pm b_k, \\ \frac{dp(r)}{dr} &= rdF_{n-2} + F_{n-1}, \\ \frac{d^2p(r)}{dr^2} &= 2ddF_{n-2}. \end{aligned} \tag{14}$$

It increases the number of assignations and additions but reduces the multiplications, therefore, one obtains the following time complexity:

$$T_5(n) = 36n + 55.$$

Using the methodologies (11), (12) and (13) a quadratic time is obtained, which is defined as:

$$T(n) = 2n^2 + 24n + 46.$$

If (14) is used instead of (13), the quadratic time is given as:

$$T(n) = 2n^2 + 42n + 58.$$

3.3. Shaw–Traub Algorithm

Similar to the methodology (14), an algorithm designed to calculate the normalized derivatives, $\frac{1}{i!}\frac{d^i p(r)}{dr^i}$, was proposed in [14]. Even, the Horner method used to evaluate a polynomial is a special case of this algorithm. It allows to reduce the number of multiplication but add divisions, assignations and additions. In this work a modified algorithm is used, which relies on Shaw–Traub algorithm:

$$\begin{aligned}
&t_1 = r, \quad t_i = t_{i-1}r, \quad \text{for} \quad i = 2, 3, \cdots, n; \\
&T_i^{-1} = a_{n-i}t_{n-i}, \quad \text{for} \quad i = 0, 1, \cdots, n-1; \\
&T_n^{-1} = a_0 \pm b_k, \\
&T_0^0 = t_n r, \quad T_1^1 = T_0^0, \quad T_2^2 = T_0^0; \\
&T_i^j = T_{i-1}^{j-1} + T_{i-1}^j, \quad \text{for} \quad j = 0, 1, 2; \ i = j+1, \cdots, n+1; \\
&p(r) = T_{n+1}^0, \quad \frac{dp(r)}{dr} = \frac{T_{n+1}^1}{t_1}, \quad \frac{d^2p(r)}{dr^2} = 2\frac{T_{n+1}^2}{t_2}.
\end{aligned} \tag{15}$$

Since $T_0^0 = T_1^1 = T_2^2$, two assignations could be avoided if T_0^0 is used instead of T_1^1 and T_2^2. Then the algorithm (15) presents an linear time given as:

$$T_6(n) = 30n + 70.$$

Applying the algorithms (11), (12) and (15), HIDD presents a quadratic time, which is given as:

$$T(n) = 2n^2 + 36n + 74. \tag{16}$$

It is important to mention that in [20], the best parameters for the family of algorithms presented in [14] were founded. Then the time complexity (16) could be reduced tuning its parameters but not its order. As Halley's method does not use higher-order derivatives, this work will not consider the algorithm presented in [15], which improve the algorithm proposed in [14] for the $n+1$ normalized derivatives.

4. Simulation

In this section, two comparatives are presented. The first one correspond to a graphical comparative of the required basic operations of the algorithms. The second one is a comparative of the time of simulation for the 4 methodologies proposed in this work. The methodology composed of (4), (6), (7), (8) and a direct evaluation of the polynomials is referenced as direct evaluation, the methodology composed of (4), (7) (11), (12) and (13) is referenced as half-Horner algorithm, the methodology composed of (4), (7), (11), (12) and (14) is referenced as Full-Horner algorithm and (4), (7), (11), (12) and (15) is referenced as Shaw-Traub algorithm.

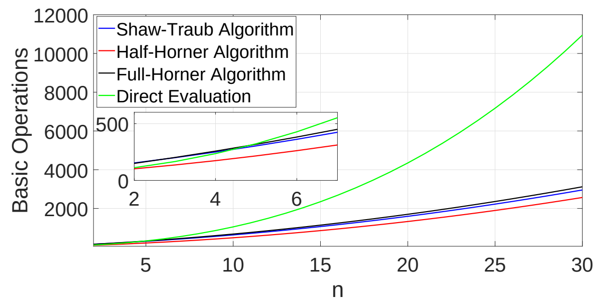

Fig. 1. Comparative between the time complexity of the methodologies.

4.1. Simulation I

In this simulation the time complexity of the methodologies are evaluated for $2 \leq n \leq 30$. The results are presented in Figure 1. One can see that the algorithm with less basic operations is Half-Horner algorithm. Furthermore Shaw-Traub and Full-Horner algorithm have less basic operations than direct evaluation for $n > 4$. It is important note the difference of Direct evaluation with respect to the other algorithms for a value of $n >= 7$, 240 or more basic operations with respect to Half-Horner algorithm.

4.2. Simulation II

Since the methodologies have a different proportion of basic operations, this simulation aims to compare the time require to simulate the algorithms. Additionally, an simulation without the parameters defined in Equations (5) and (9) is simulated, which is the same methodology than the direct evaluation without the parameters ϕ_i, $\bar{b}_{i,j}^*$, c_i and d_i. Here $n = 3$, $n = 7$ and $n = 10$ are considered, the values of the signal with noise and the constants defined in (5) and (9) are calculated previous to the simulation. The sampling time is selected as $\tau = 0.001\,\text{sec}$ and $t = 2000,\ 10000,\ 25000,\ 50000\,\text{sec}$. The results are presented in Tables I-III. The most efficient methods with respect to the simulation time were the Half-Horner and Full-Horner algorithms, both present a similar performance for $n = 3$, $n = 7$ and $n = 10$. For $n = 3$, direct evaluation has a better performance than Shaw-Traub, it contrasts to the results obtained for $n = 10$, where Shaw-Traub algorithm reduces the simulation time compared to direct evaluation. The above fact matches with its time complexity.

Remark 1: Half-Horner and Full-Horner algorithms reduced the simulation time more than 25 times for $n = 10$. It can be seen in Table III. It comes from the use of the variables $\bar{b}_{i,j}^*$, ϕ_i, c_i and d_i and a reduction of the time complexity.

	2000 sec	10000 sec	25000 sec	50000 sec
Evaluation without ϕ_i, $\bar{b}^*_{i,j}$, c_i and d_i	0.5963 sec	2.990 sec	7.470 sec	15.059 sec
Direct Evaluation.	0.3578 sec	1.783 sec	4.475 sec	9.027 sec
Half-Horner.	0.3494 sec	1.753 sec	4.411 sec	8.896 sec
Full-Horner.	0.3496 sec	1.756 sec	4.407 sec	8.908 sec
Shaw-Traub.	0.3852 sec	1.922 sec	4.813 sec	9.661 sec

TABLE I

SIMULATION TIME OF THE ALGORITHMS FOR $n = 3$ AND $\tau = 0.001$ sec.

	2000 sec	10000 sec	25000 sec	50000 sec
Evaluation without ϕ_i, $\bar{b}^*_{i,j}$, c_i and d_i	6.791 sec	33.808 sec	85.045 sec	169.95 sec
Direct Evaluation.	0.486 sec	2.414 sec	6.035 sec	12.51 sec
Half-Horner.	0.466 sec	2.286 sec	5.729 sec	11.61 sec
Full-Horner.	0.457 sec	2.293 sec	5.763 sec	11.51 sec
Shaw-Traub.	0.503 sec	2.46 sec	6.210 sec	12.718 sec

TABLE II

SIMULATION TIME OF THE ALGORITHMS FOR $n = 7$ AND $\tau = 0.001$ sec.

	2000 sec	10000 sec	25000 sec	50000 sec
Evaluation without ϕ_i, $\bar{b}^*_{i,j}$, c_i and d_i	14.312 sec	71.6 sec	179.37 sec	358.09 sec
Direct Evaluation.	0.831 sec	4.192 sec	10.33 sec	20.527 sec
Half-Horner.	0.5437 sec	2.75 sec	6.858 sec	13.692 sec
Full-Horner.	0.5631 sec	2.807 sec	6.997 sec	14.139 sec
Shaw-Traub.	0.6101 sec	3.097 sec	7.767 sec	15.464 sec

TABLE III

SIMULATION TIME OF THE ALGORITHMS FOR $n = 10$ AND $\tau = 0.001$ sec.

5. Conclusion

In this work four methodologies were designed to implement the implicit discrete-time differentiator HIDD, which rely on the Horner's method and the Shaw-Traub algorithm. 3 methodologies allow to reduce the time complexity of the implicit differentiator with respect to n, i.e., they present a quadratic time instead of a cubic time. The methodologies show a noticeable performance compared to an direct implementation without the parameters ϕ_i, $\bar{b}_{i,j}$, c_i and d_i. However, the simulations show that Half-Horner and Full-Horner algorithms reduce the number of basic operations and its simulation time. Even, they reduced the simulation time of the differentiator more than 25 times for $n = 10$. As future work, the discrete Fourier transform and the methodology proposed in [15] will be considered to reduce the time complexity.

References

[1] P. Kaveh and Y. B. Shtessel, "Blood glucose regulation using higher-order sliding mode control," *International Journal of Robust and Nonlinear Control*, vol. 18, no. 4-5, pp. 557–569, 2008.

[2] Y. B. Shtessel, I. A. Shkolnikov, and A. Levant, "Smooth second-order sliding modes: Missile guidance application," *Automatica*, vol. 43, no. 8, pp. 1470–1476, 2007.

[3] M. Iqbal, A. I. Bhatti, S. I. Ayubi, and Q. Khan, "Robust parameter estimation of nonlinear systems using sliding-mode differentiator observer," *IEEE Transactions on Industrial Electronics*, vol. 58, no. 2, pp. 680–689, Feb 2011.

[4] A. Levant, "Higher-order sliding modes, differentiation and output-feedback control," *International Journal of Control*, vol. 76, no. 9-10, pp. 924–941, 2003.

[5] V. I. Utkin, J. Guldner, and J. Shi, *Sliding Mode Control in Electro-Mechanical Systems*, 2nd ed. CRC Press, 2009.

[6] M. Livne and A. Levant, "Proper discretization of homogeneous differentiators," *Automatica*, vol. 50, no. 8, pp. 2007–2014, 2014.

[7] S. Koch and M. Reichhartinger, "Discrete-time equivalent homogeneous differentiators," in *2018 15th International Workshop on Variable Structure Systems (VSS)*, July 2018, pp. 354–359.

[8] S. Koch, M. Reichhartinger, M. Horn, and L. Fridman, "Discrete-time implementation of homogeneous differentiators," *IEEE Transactions on Automatic Control*, vol. 65, no. 2, pp. 757–762, Feb 2020.

[9] J.-P. Barbot, A. Levant, M. Livne, and D. Lunz, "Discrete differentiators based on sliding modes," *Automatica*, vol. 112, p. 108633, 2020.

[10] J. E. Carvajal-Rubio, J. D. Sánchez-Torres, M. Defoort, M. Djemai, and A. G. Loukianov, "Implicit and explicit discrete-time realizations of homogeneous differentiators," *International Journal of Robust and Nonlinear Control*, 2021, Special Issue on Homogeneous Sliding-Mode Control and Observation. [Online]. Available: https://doi.org/10.1002/rnc.5505

[11] J. M. McNamee and V. Pan, *Numerical Methods for Roots of Polynomials - Part II*, ser. Studies in Computational Mathematics. Amsterdam London: Elsevier Science, 2013, vol. 16.

[12] M. Sipser, *Introduction to the Theory of Computation*. Cengage learning, 2012.

[13] J. M. McNamee, *Numerical Methods for Roots of Polynomials - Part I*, ser. Studies in Computational Mathematics. Amsterdam London: Elsevier Science, 2007, vol. 14.

[14] M. Shaw and J. F. Traub, "On the number of multiplications for the evaluation of a polynomial and some of its derivatives," *Journal of the ACM (JACM)*, vol. 21, no. 1, pp. 161–167, 1974.

[15] L. De Jong and J. Van Leeuwen, "An improved bound on the number of multiplications and divisions necessary to evaluate a polynomial and all its derivatives," *ACM SIGACT News*, vol. 7, no. 3, pp. 32–34, 1975.

[16] D. E. Knuth, *Art of computer programming, volume 2: Seminumerical algorithms*. Addison-Wesley Professional, 2014.

[17] G. H. Golub and C. F. Van Loan, *Matrix computations*, 4th ed. The Johns Hopkins University Press, Baltimore, 2013.

[18] J. E. Carvajal-Rubio, A. G. Loukianov, J. D. Sánchez-Torres, and M. Defoort, "On the discretization of a class of homogeneous differentiators," in *2019 16th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, September 2019, pp. 1–6.

[19] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, *Templates for the solution of algebraic eigenvalue problems: a practical guide*. Society for Industrial and Applied Mathematics (SIAM), 2000.

[20] M. Shaw and J. F. Traub, "Analysis of a family of algorithms for the evaluation of a polynomial and some of its derivatives," *Department of Computer Science Carnegie Mellon University*, 1975.