sensors

MDPI

Communication

Optimal Piecewise Polynomial Approximation for Minimum Computing Cost by Using Constrained Least Squares

Jieun Song and Bumjoo Lee *

Department of Electronic Engineering, Myongji University, Yongin 17058, Republic of Korea; song12069595@gmail.com
* Correspondence: bjlee@mju.ac.kr

Abstract: In this paper, the optimal approximation algorithm is proposed to simplify non-linear functions and/or discrete data as piecewise polynomials by using the constrained least squares. In time-sensitive applications or in embedded systems with limited resources, the runtime of the approximate function is as crucial as its accuracy. The proposed algorithm searches for the optimal piecewise polynomial (OPP) with the minimum computational cost while ensuring that the error is below a specified threshold. This was accomplished by using smooth piecewise polynomials with optimal order and numbers of intervals. The computational cost only depended on polynomial complexity, i.e., the order and the number of intervals at runtime function call. In previous studies, the user had to decide one or all of the orders and the number of intervals. In contrast, the OPP approximation algorithm determines both of them. For the optimal approximation, computational costs for all the possible combinations of piecewise polynomials were calculated and tabulated in ascending order for the specific target CPU off-line. Each combination was optimized through constrained least squares and the random selection method for the given sample points. Afterward, whether the approximation error was below the predetermined value was examined. When the error was permissible, the combination was selected as the optimal approximation, or the next combination was examined. To verify the performance, several representative functions were examined and analyzed.

Keywords: piecewise polynomial; function approximation; regression; constrained least squares

Citation: Song, J.; Lee, B. Optimal Piecewise Polynomial Approximation for Minimum Computing Cost by Using Constrained Least Squares. *Sensors* **2024**, *24*, 3991. https://doi.org/10.3390/s24123991

Academic Editor: Zhengguo Li

Received: 29 May 2024
Revised: 17 June 2024
Accepted: 17 June 2024
Published: 20 June 2024

1. Introduction

1.1. Importance of Approximation and Previous Studies

In many applications in the scientific and engineering fields, approximation for a complex function or data set is important, such as compression of ECG signals [1]; various voice processing applications such as speech recognition, synthesis, and conversion [2,3]; and correction of sensor data [4,5]. Numerous methods have been studied for approximation, with the least squares method being the preferred choice. The least squares method offers high accuracy by minimizing the residuals and is also simple to implement in a computer program. Formulating the optimal coefficients as least squares problems dates back to Gendre (1805) and Gauss (1809). The first application was performed by Gergonne in 1815 [6]. Therefore, in this paper, the least squares method is employed for approximation.

The approximation error at the sample points generally tends to decrease as high-order polynomials are used, but this causes overfitting, which leads to significant oscillation between sample points. To resolve the overfitting, it is recommended to use lower-order polynomials by splitting the sections. In the case of using piecewise polynomials, it takes less computation time than using a single high-order polynomial with the same approximation error. Since the maximum number of intervals and orders depends on the number of samples and the complexity, several studies have been proposed to find the appropriate order and number of intervals. In [7–10], piecewise polynomial fitting was proposed. Additionally, ref. [7] proposed least squares piece fitting using a cubic

polynomial, ref. [9] used a method of adjusting the boundary of the segments to increase the operation speed, and ref. [10] used piecewise polynomial modeling to lower the error rate. However, in [8–10], there was a limitation that the order of the polynomial must be determined by the user. In [4], the whole domain was evenly divided into several intervals, and each interval was approximated by a cubic polynomial using the least squares method with the constraint that it had continuity in all boundaries. The studies cited in [11–15] proposed an approximation method using piecewise linear approximation (PLA), while ref. [16] proposed a method consisting of using several linear functions and then moving the endpoints of the interval appropriately to reduce the approximation error of the interval.

There are two considerations when approximating with piece-wise polynomials: the order of polynomials and the number of intervals. Higher-order polynomials require higher computational costs. Further, this may cause overfitting. Therefore, it is necessary to determine the appropriate order of the polynomials. When using approximated function, i.e., piece-wise function, it requires additional steps to determine the subdomain corresponding to the input value. If there are many intervals, the approximation precision increases, but more time is also needed to determine the subdomain. Thus, the order and the number of polynomials should be balanced and optimized. In order to accomplish this, an optimization scheme is proposed in contrast with [4–16], which utilized a predetermined polynomial order and number of polynomials.

Several studies have adopted optimization methods for curve fitting. Among the optimization methods, the hp-adaptive pseudo-spectral method, proposed in [17], is most similar to the algorithm proposed in this paper. The method determines the locations of segment breaks and the order of the polynomial in each segment. To be more specific, when the error within a segment displays a consistent pattern, the number of collocation points is increased. On the other hand, if there are isolated points with significantly higher errors than the rest of the segments, then the segment is split at those points. This produces solutions with better accuracy than global pseudo-spectral collocation while utilizing less computational time and resources. The hp-adaptive pseudo-spectral method and the optimal piecewise polynomial (OPP) approximation algorithm proposed in this paper are similar in that they increase accuracy by increasing the number of orders and intervals. The OPP approximation algorithm compares computational costs to obtain an approximation of the minimum cost while satisfying the error norm constraint for the approximation.

1.2. OPP Approximation Algorithm

As mentioned in the introduction, polynomial approximation methods are necessary for sensor approximation and other approximation tasks. Through a literature review concerning such methods, we propose the optimization of piecewise polynomials. As an improvement over previous approximation methods, the proposed OPP approximation algorithm automatically determines both the order and number of intervals while focusing on computational efficiency to obtain the approximate polynomial with the smallest computational cost within the specified error norm.

As seen in Figure 1, approximation is processed by finding the optimized order and number of intervals that satisfy the error norm constraint with the minimum computational cost. To achieve this, the algorithm adopts two cost functions. One is for the approximation error that is used by the least squares. The calculated approximation error is compared with the error norm constraint for approximation. The other is computational cost function. It indicates the program run time needed to calculate the value of the function for an input value, x, and is composed of two runtime costs: the cost of polynomial function call and the cost of a binary search tree to determine the relevant interval. In other words, the computational cost is calculated given the order of the polynomial and the number of intervals. The costs for all possible combinations are calculated and sorted in ascending order offline. This is used to search for the order and number of intervals satisfying the error constraint with the minimum cost. Therefore, the OPP approximation algorithm is useful in systems that must be efficient to compute, such as real-time systems and

embedded systems. When approximating, if the approximation functions of each interval are placed independently, discontinuity occurs over the entire interval. To address this, a constraint is introduced using the Lagrange multiplier method to smoothly connect the approximation functions of each interval. Applying the approximation function obtained using the OPP approximation algorithm to the systems can minimize the computational cost while satisfying the error norm constraint. In this way, obtaining an approximation function before applying the system can reduce computational time and memory.

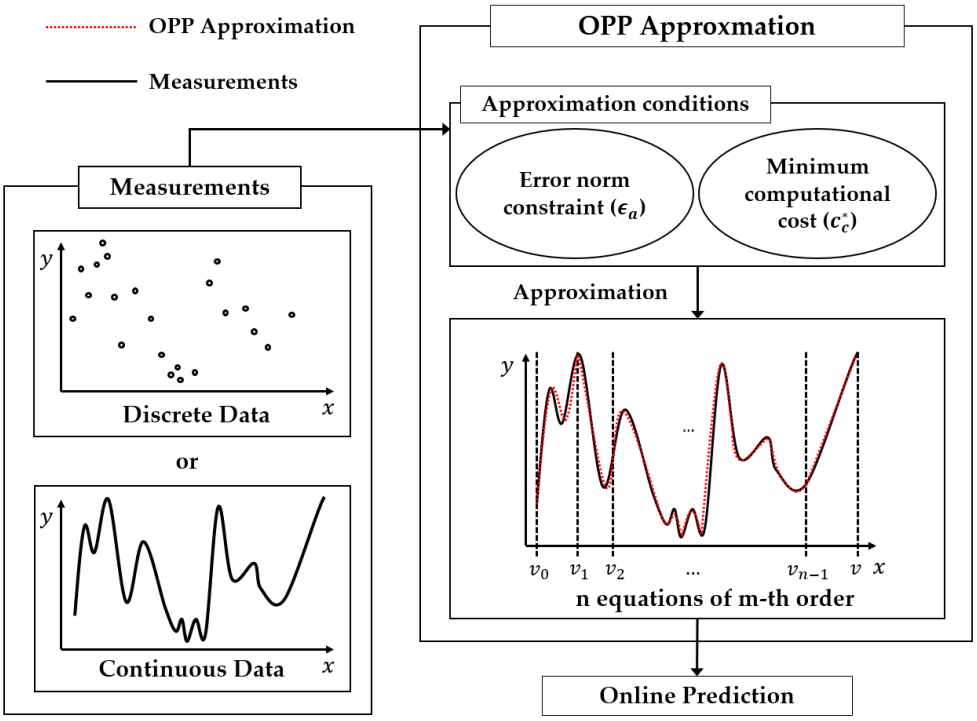

Figure 1. OPP approximation algorithm overview.

This paper is organized as follows. In Section 2, previous knowledge related to the algorithms proposed Section 3 is described. In Section 4, the algorithm is examined with representative test functions. Subsequently, the OPP approximation algorithm is discussed, and further work to improve the performance is proposed in Section 5.

2. Preliminaries

2.1. Nomenclature

The algorithm and all formulas in this study use the notation in Table 1. The symbols necessary to understand the algorithm are as follows: $a_{k,i}$ means the coefficient of the i^{th} order term of the k^{th} polynomial. $\boldsymbol{q}_k$ represents the coefficients of the approximation polynomial in the k^{th} interval as a vector, and $\boldsymbol{q}$ is a $(m+1)n$ vector concatenated as $\boldsymbol{q}_k$. c_a is the approximation cost and c_c is the computational cost at runtime. m^* and n^* indicate the optimal polynomial order and the number of intervals, respectively. $*$ describes the optimized value for c_a within ϵ_a and the minimum c_c. Scalars, vectors, and matrices are written in lowercase, lowercase boldface, and uppercase boldface, respectively.

Table 1. Meanings of symbols.

Symbol	Signification
$f(x)$	Function to approximate
$x_{k,j}$	The j^{th} sample in the k^{th} interval
$y_{k,j}$	The j^{th} function value in the k^{th} interval
$\eta(k)$	Number of samples for the k^{th} interval

Table 1. *Cont.*

Symbol	Signification
m	Polynomial order
n	Number of intervals
$\boldsymbol{q}_k$	$(m+1) \times 1$ vector of polynomial coeffs in the k^{th} interval
$\boldsymbol{q}$	$(m+1)n \times 1$ vector concatenated all $\boldsymbol{q}_k$
$a_{k,i}$	Coefficient of the i^{th} order term for the k^{th} polynomial
$\boldsymbol{v}$	$(n+1) \times 1$ vector of boundary values
c_a	Average sum of error squares at sample points
ϵ_a	Error norm constraint for approximation
$c_c(m,n)$	Computational cost according to (m,n) polynomial expressed by CPU instruction cycles
(m^*, n^*)	(m,n) for the minimal computational cost with $\lvert c_c(m,n)\rvert \leq \epsilon_a$
p_k	Approximation polynomial for the k^{th} interval

2.2. Formulation of Constrained Least Squares

The least squares method is a representative approach in regression analysis to approximate functions or discrete samples by minimizing the sum of the squares of the residuals in the results of each individual equation. The residual is the difference between the given sample and the approximate value, represented by weighted squares, as shown in (1). The matrix form of the least squares method is represented as follows:

$$e = \frac{1}{2}\sum_{k=1}^{n} \left((\boldsymbol{y}_k - \boldsymbol{F}_k \boldsymbol{q}_k)^T \boldsymbol{W}_k (\boldsymbol{y}_k - \boldsymbol{F}_k \boldsymbol{q}_k) \right) \tag{1}$$

where $\boldsymbol{F}_k = \begin{bmatrix} 1 & x_{k,1} & x_{k,1}^2 & \cdots & x_{k,1}^m \\ 1 & x_{k,2} & x_{k,2}^2 & \cdots & x_{k,2}^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{k,\eta(k)} & x_{k,\eta(k)}^2 & \cdots & x_{k,\eta(k)}^m \end{bmatrix}$, $\mathbf{y}_k = \begin{bmatrix} y_{k,1} & \cdots & y_{k,\eta(k)} \end{bmatrix}^T$, and $\boldsymbol{q}_k = [a_{k,0} a_{k,1} \cdots a_{k,m}]^T$. The least squares method is accomplished by finding $\boldsymbol{q}$, which makes the partial differential of error, i.e., the gradient, 0. Consequently, this leads to optimal polynomials with the minimum average of the residual sum of squares at the sample points. For the sake of simplicity, Equation (1) can be shortened into a single monomial expression using the sparse diagonal matrix $\boldsymbol{F}$ as follows:

$$e = \frac{1}{2}(\boldsymbol{y} - \boldsymbol{F}\boldsymbol{q})^T \boldsymbol{W} (\boldsymbol{y} - \boldsymbol{F}\boldsymbol{q}), \tag{2}$$

where $\boldsymbol{F} = \mathrm{diag}(\boldsymbol{F}_k)$, $\boldsymbol{y} = \begin{bmatrix} \boldsymbol{y}_1^T & \cdots & \boldsymbol{y}_n^T \end{bmatrix}^T$, and $\boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}_1^T & \cdots & \boldsymbol{q}_n^T \end{bmatrix}^T$. The polynomial, p_k, should be smooth at the boundaries, v_{k-1} and v_k, with adjacent polynomials p_{k-1} and p_{k+1}, respectively. Without a loss of generality, in the proposed algorithm, the 1st-order differentiability was appended as a constraint, $\boldsymbol{G}\boldsymbol{q} = \boldsymbol{0}$, for the smoothness. Note that if necessary, it is possible to increase the order for differentiability at the intersection of the adjacent intervals. Consequently, Equation (2) was modified with the Lagrange Multiplier, $\boldsymbol{\lambda}$, as follows:

$$e = \frac{1}{2}(\boldsymbol{y} - \boldsymbol{F}\boldsymbol{q})^T \boldsymbol{W} (\boldsymbol{y} - \boldsymbol{F}\boldsymbol{q}) + \boldsymbol{\lambda}^T \boldsymbol{G}, \tag{3}$$

where $\boldsymbol{G} = \mathrm{diag}(\boldsymbol{G}_k)$ and $\boldsymbol{G}_l = \begin{bmatrix} 1 & \boldsymbol{v}_k & \cdots & \boldsymbol{v}_k^m & -1 & -\boldsymbol{v}_k & \cdots & -\boldsymbol{v}_k^m \\ 0 & 1 & \cdots & m\boldsymbol{v}_k^{m-1} & 0 & -1 & \cdots & -m\boldsymbol{v}_k^{m-1} \end{bmatrix}$. By partial differentiation of (3) with respect to $\boldsymbol{q}$ and $\boldsymbol{\lambda}$, two equations are obtained, i.e., $\frac{\partial}{\partial q} e = \boldsymbol{0}$ and $\boldsymbol{G}\boldsymbol{q} = \boldsymbol{0}$. By solving the above equation, the optimal coefficients, which minimize c_a, are obtained.

$$\boldsymbol{q} = \boldsymbol{H}(-\boldsymbol{G}^T(\boldsymbol{G}\boldsymbol{H}\boldsymbol{G}^T)^{-1}\boldsymbol{G}\boldsymbol{H}\boldsymbol{F}^T\boldsymbol{W}^T\boldsymbol{y} + \boldsymbol{F}^T\boldsymbol{W}^T\boldsymbol{y}), \tag{4}$$

where $\boldsymbol{H} = \left(\boldsymbol{F}^T\boldsymbol{W}^T\boldsymbol{F}\right)^{-1}$.

3. OPP Approximation Algorithm

3.1. Overall Algorithm Flow

The proposed algorithm is intended for use in systems where computing time is critical, such as real-time systems and/or embedded systems. With the algorithm's runtime, it is particularly important to reduce function call times. Since the approximate function is composed of several polynomials, it must be known how much time is required to execute the function at runtime. To accomplish this, the computational cost function was defined from the four arithmetic operations and binary search tree. This is explained in more detail in the next subsection. Offline, based on the computational cost function, an approximation polynomial with the smallest calculation time was obtained, then used in the system.

The overall flow for the OPP approximation is described in Figure 2. First, the maximum values of polynomial order, m_{max}, and the number of intervals, n_{max}, to explore were set. Afterward, the computational costs for all possible combinations of the number of intervals and the order were calculated and tabulated in ascending order ($m \in [2, m_{max}]$, $n \in [1, n_{max}]$). Subsequently, $\boldsymbol{q}$ and c_a were calculated using the constrained least squares for the k^{th} pair $(m, n)_k$, the sample points (x, y), and the polynomial intervals $\boldsymbol{v}$. This was repeated with randomly selected $\boldsymbol{v}$ N times, and the case with the smallest c_a was selected. If c_a was greater than ϵ_a, the next pair $(m, n)_{k+1}$ was examined and $\boldsymbol{q}$ and c_a were calculated again. The loop was repeated until c_a was less than ϵ_a. When c_a became smaller than ϵ_a, the piecewise polynomials with the order and the number of intervals were determined as the optimized approximate. If the c_a of the pair $(m, n)_{max}$ was greater than ϵ_a for the pair, it was considered that finding the piecewise polynomial approximation function failed in the specified range. In this case, it would be possible to continue searching by using larger m_{max} and/or n_{max} values, or by relaxing ϵ_a.

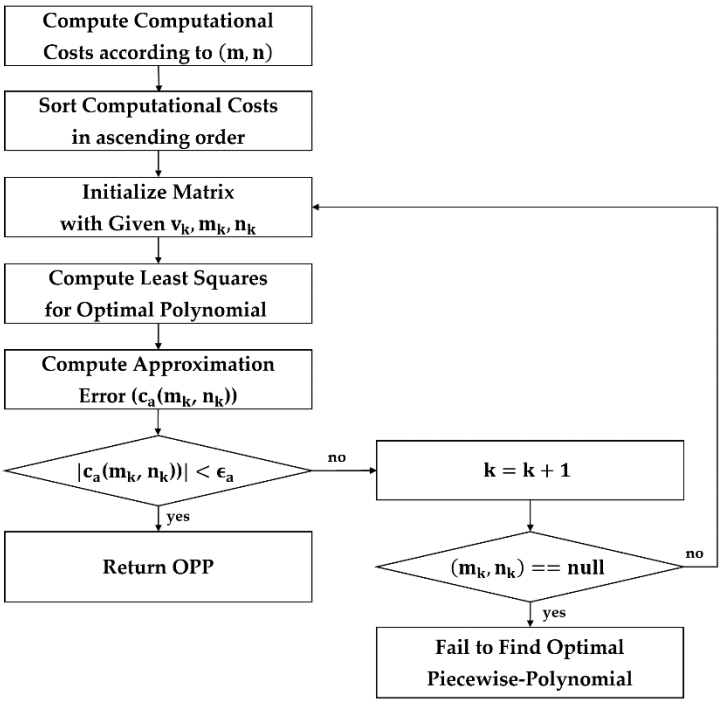

Figure 2. Overall algorithm flow for OPP approximation.

3.2. Computational Cost

In this section, the computational cost is described. Since the approximated function was piecewise with several polynomials, at runtime, it took two computation times: the

time taken for a single polynomial computation and the time taken for selecting a specific polynomial. Therefore, the computational cost function is defined as follows:

$$c_c(m, n) = c_p + c_b. \tag{5}$$

The computing time for a single polynomial depends on the order of polynomial. This is defined as $c_p(m)$, as follows:

$$c_p(m) = n_a r_a + n_r r_r m + n_f r_f m, \tag{6}$$

where n_a, n_r, and n_f are the number of assignments, arithmetic, and 'for' instructions, respectively. In addition, r_a, r_r, and r_f are cycle counts of assignment, arithmetic, and 'for' instructions. m, the order of polynomial, indicates the repetition count.

Since the approximate function is implemented with several polynomials, the clock cycles are required for the binary search, which is intended to find the suitable intervals for a certain input x. This is defined as c_b and determined as the average of the short path case and long path case in a binary search tree. Two examples are illustrated in Figure 3.

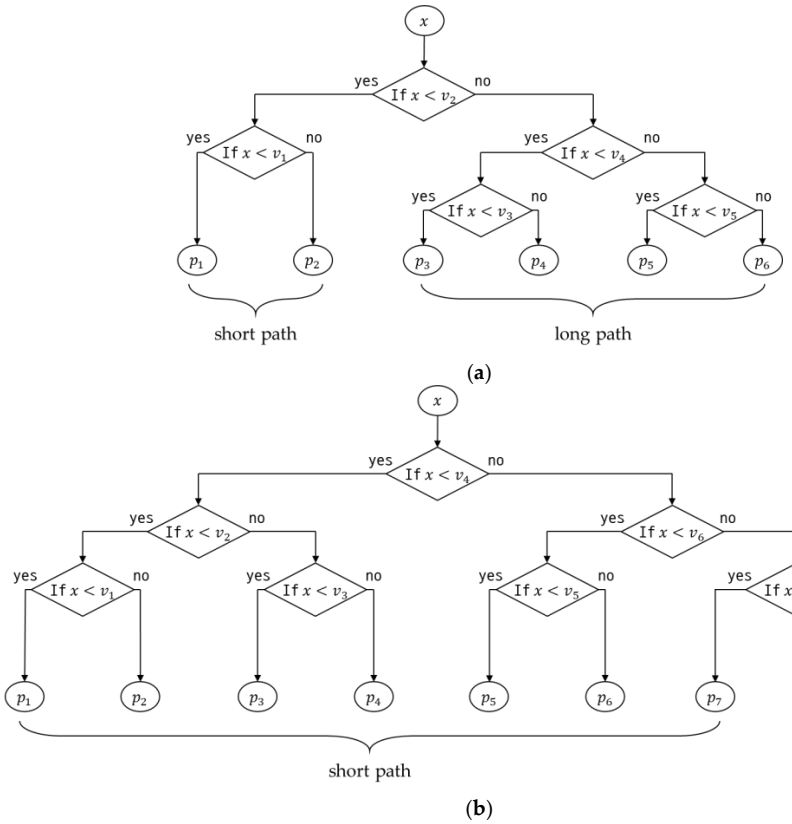

Figure 3. Binary search tree to select a suitable polynomial corresponding to input value. (**a**) Example 1: $n = 6$. (**b**) Example 2: $n = 9$.

Since the average clock cycles to determine the corresponding polynomial depend on the probability that the input is in a certain interval, c_b is a probability distribution function for the input value. For the sake of simplicity, the probability that an input is in a particular interval is the same for all intervals. Consequently, the costs associated with the number of intervals are as follows:

$$c_b = c_{bs} p_s + c_{bl} p_l, \tag{7}$$

where $c_{bs} = (d_m - 1)(n_a r_a + n_d r_d + n_r r_r + n_w r_w + n_i r_i)$, $p_s = n_s/n$, $c_{bl} = d_m(n_a r_a + n_d r_d + n_r r_r + n_w r_w + n_i r_i)$, and $p_l = n_l/n$. c_{bs} and c_{bl} represent the binary search tree costs for the short and the long path, respectively. Similarly, n_s and n_l are the numbers of the short

path and the long path, respectively. Note that $n_s + n_l = n$. d_m indicates the depth of the tree and is an integer value satisfying $2^{d_m-1} \leq n < 2^{d_m}$. In addition, n_d, n_h, and n_i are the numbers of divisions, 'while', and 'if' instructions, respectively. Finally, r_d, r_h, and r_i are cycle counts of division, 'while', and 'if' instruction for the ARM Cortex-M7 core (see [18]), respectively. The calculated $c_c(m, n)$ is organized into a table by m and n (see Figure 4) and sorted in ascending order. In this paper, it is a priority to increase m when the costs are the same.

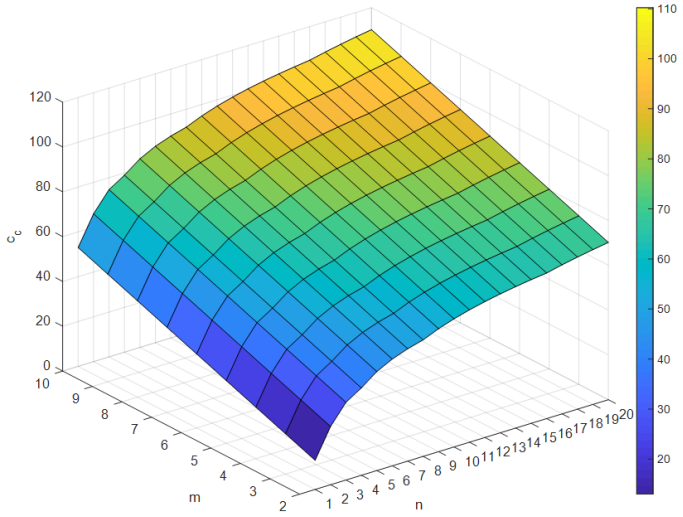

Figure 4. Computational cost according to the order of polynomial (m) and the number of intervals (n) $(m_{max} = 10,\ n_{max} = 20)$.

3.3. Application of the OPP Approximation Algorithm

The approximation function determined through the OPP algorithm is applied to the system and used. Implementing the OPP algorithm has several advantages. First, the OPP algorithm improves the calculation process by minimizing the resources required for real-time calculations. Second, this speeds up the execution time and improves the overall performance of the system. It makes resource management easier by reducing memory and computational load in the system. This is especially important for efficiently using resources in systems with limited hardware capabilities. Third, the algorithm can process pre-computations depending on the size and complexity of the data, allowing for efficient expansion. Thanks to this scalability, the OPP algorithm is suitable for a wide range of applications across various domains. Including the OPP approximation algorithm in the system design not only simplifies the computational process, but also improves the system's ability to efficiently manage resources and handle complex datasets.

4. Experimental Results

In order to verify the performance of the proposed OPP algorithm, it was implemented using MATLAB with several nonlinear functions. In this experiment, we set the number of samples to 100, $m_{max} = 10$, $n_{max} = 20$, and $\epsilon_a = 0.0001$. And the boundary values were determined as the values when the approximate errors were the smallest after randomly changing 100 times. Figure 5 is the result of approximation for the logarithmic function. The logarithmic function was chosen as the test function because there are many sensors whose output results are logarithmic, such as temperature sensors and distance sensors. Figure 6 shows the approximation results for the sine function that is frequently used in many applications. Figures 7 and 8 show the approximate results for the triangular and square functions, respectively. The triangular function is adopted to examine an approximation of the continuous nonlinear function with undifferentiable points. In Figure 7, the approximation is shown with a smooth curve at a sharp point. And the square function is used for curve fitting of the function, which has discontinuous points. Figure 8

shows that overfitting occurs in the section where the value changes rapidly, resulting in oscillation. Figure 9 is the result of approximation of the sigmoid function. The sigmoid function, also known as the logistic function, is commonly used in various fields such as neural networks, machine learning, and statistics. Figure 10 is the result of approximation for the ReLU function, which is one of the most commonly used activation functions in deep learning models. Figure 11 is the result of the user-specified function for testing a nonlinear function that is more complex than the previous differential functions across all intervals. The function is represented as follows:

$$y = \frac{1}{1+e^{-x}} + \sin(x) - \ln(x). \tag{8}$$

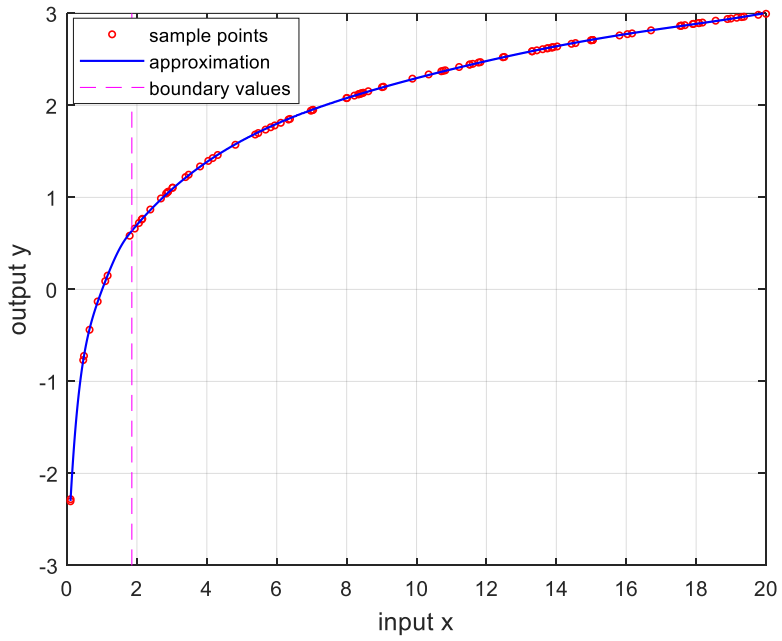

Figure 5. Approximation for $ln(x)$, $\begin{pmatrix} 0.1 \leq x \leq \\ 20 \end{pmatrix}$. $(m^*, n^*) = (5, 2)$, and the error is 3.3249×10^{-5}.

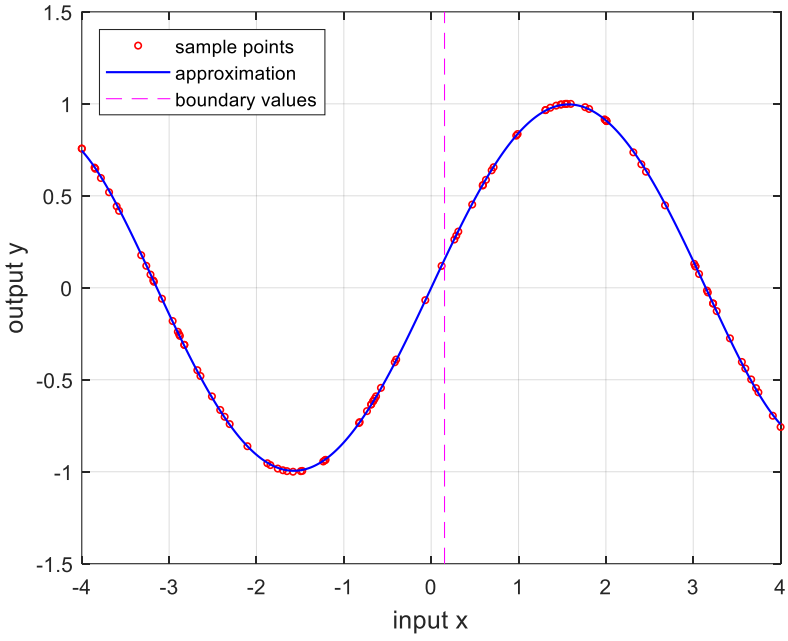

Figure 6. Approximation for $\sin(x)$, $(-4 \leq x \leq 4)$. $(m^*, n^*) = (4, 2)$, and the error is 2.0771×10^{-5}.

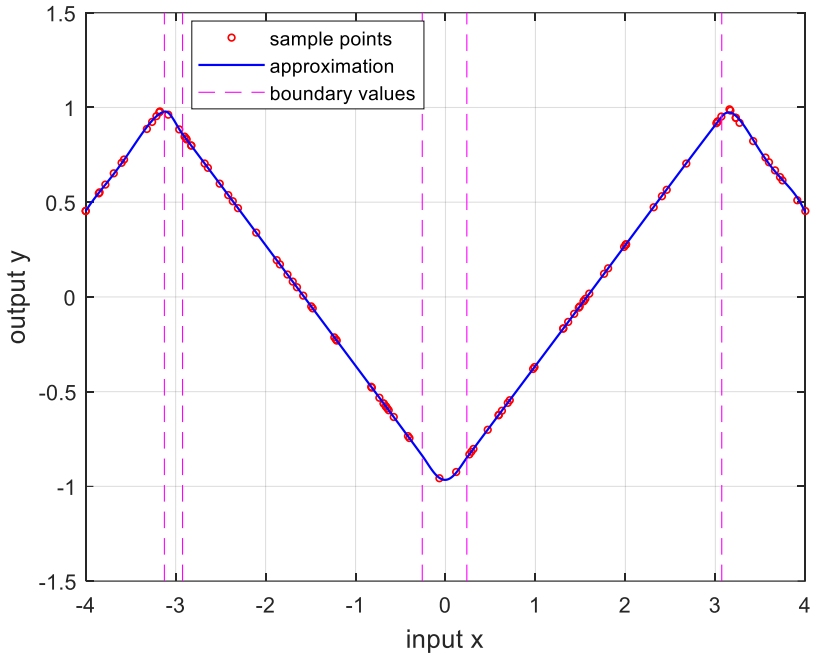

Figure 7. Approximation for sawtooth(x), $(-4 \leq x \leq 4)$. $(m^*, n^*) = (4, 6)$, and the error is 1.8994×10^{-5}.

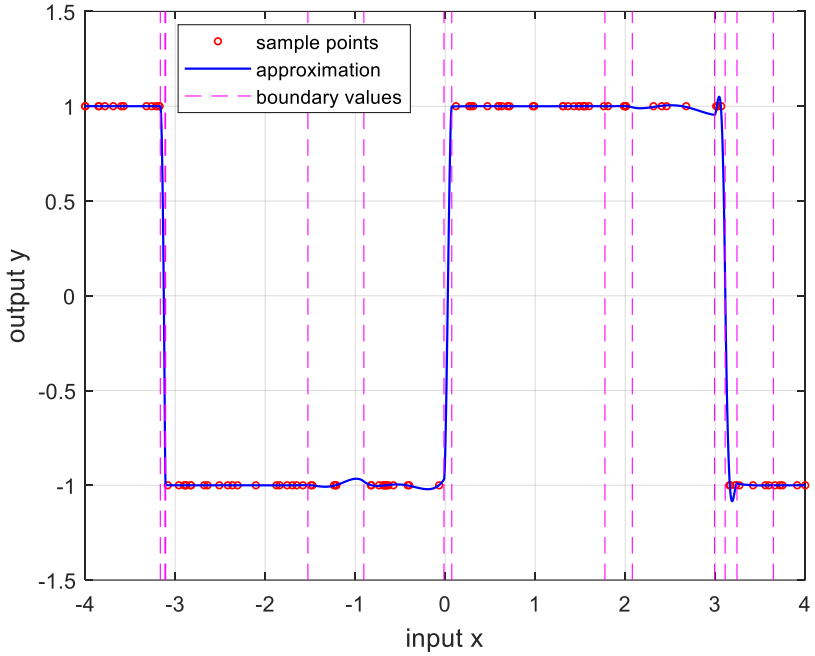

Figure 8. Approximation for square(x), $(-4 \leq x \leq 4)$. $(m^*, n^*) = (4, 14)$, and the error is 5.4552×10^{-5}.

Overall, the approximation for continuous and differentiable functions over all intervals is a satisfactory result. However, in functions with continuous, but undifferentiable, points and functions with discontinuous points, they have lower accuracy and higher $c_c(m, n)$ than continuous functions. This can be solved by increasing the sample point. Another method is to vibrate the boundary value, which will be studied later.

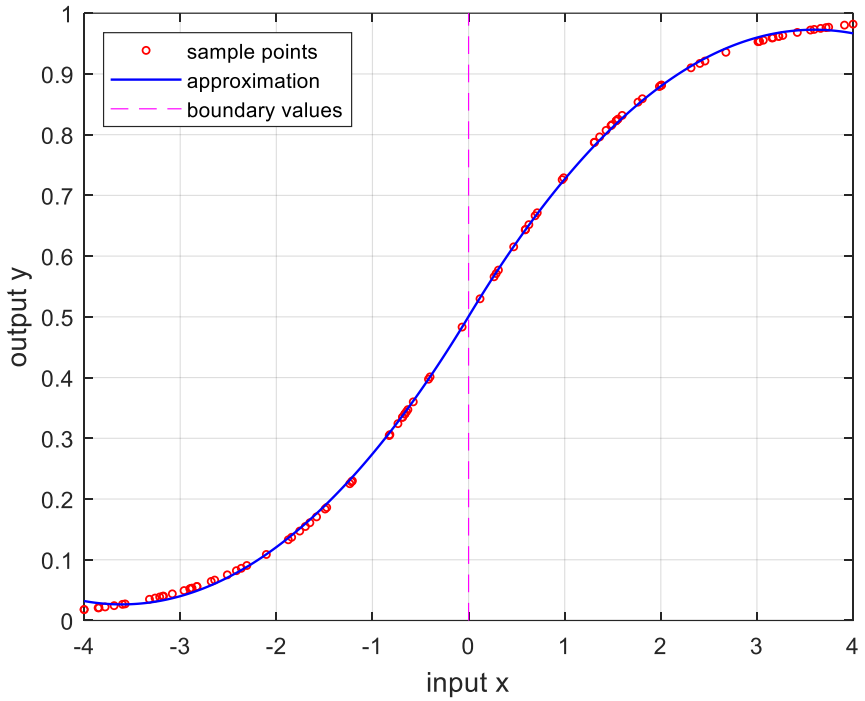

Figure 9. Approximation for $\frac{1}{1+e^{-x}}$, $(-4 \leq x \leq 4)$. $(m^*, n^*) = (2, 2)$, and the error is 2.8652×10^{-5}.

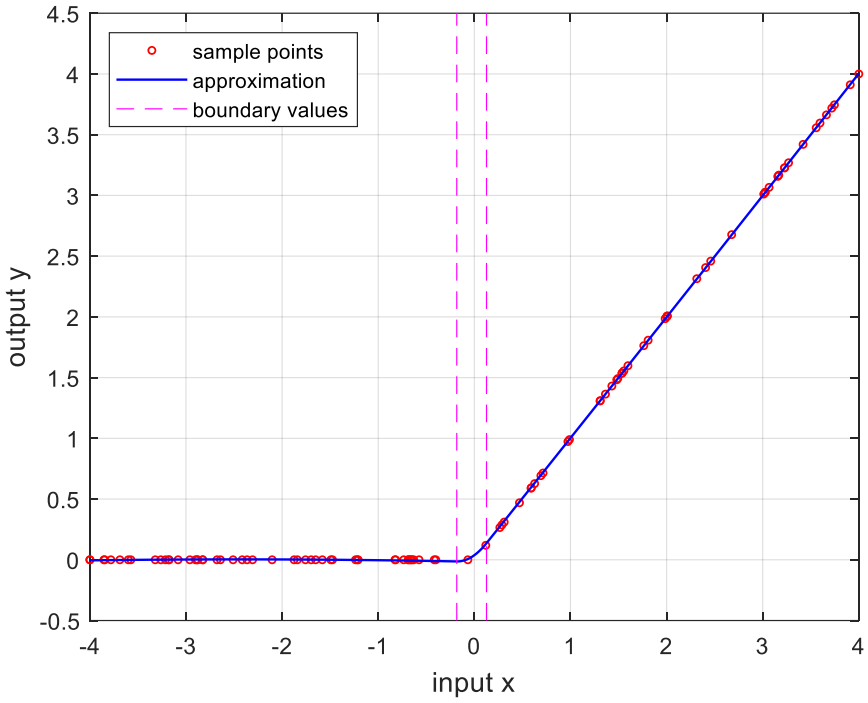

Figure 10. Approximation for $\max(0, x)$, $(-4 \leq x \leq 4)$. $(m^*, n^*) = (2, 3)$, and the error is 1.5845×10^{-5}.

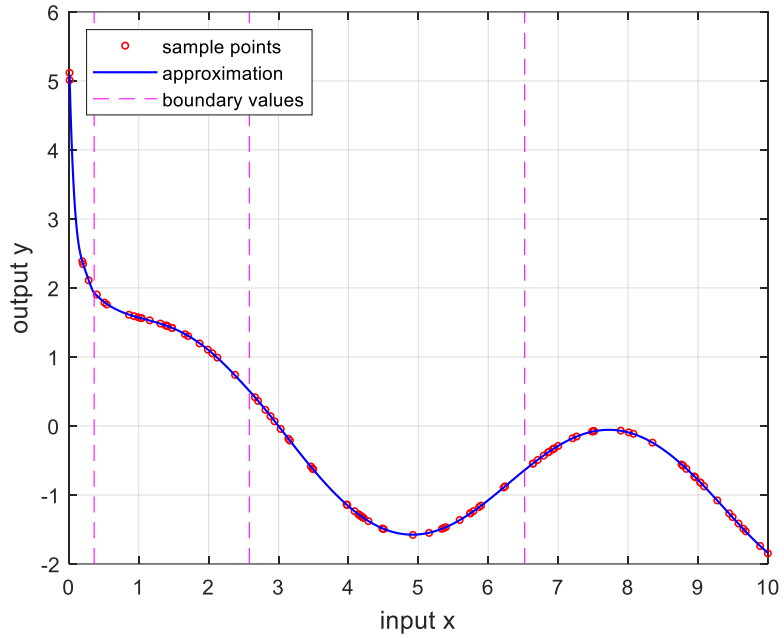

Figure 11. Approximation for (8), $(0.1 \leq x \leq 10)$. $(m^*, n^*) = (4, 4)$, and the error is 2.9888×10^{-5}.

5. Conclusions

To approximate the complex function or discrete sample points, the optimal piecewise polynomial (OPP) approximation algorithm was proposed. The maximum values of polynomial order and the number of intervals to explore were preset, and the computational cost was calculated to determine the order and number of intervals for the approximation function. Then, the combination of the order and the number of intervals was sorted in ascending order based on the computational cost offline. Subsequently, the coefficient of the polynomial and the approximation error at the sample points were determined using constrained least squares. If the error was greater than the given error bound, the next combination was examined until the error was less than the bound. Ultimately, the OPP approximation algorithm determined the fastest approximation function at a runtime within a permissible error margin. The performance of the proposed algorithm was demonstrated through several nonlinear functions.

There are some further works needed to obtain a more accurate approximation function. In this paper, the optimal intervals were obtaioned by random sampling, i.e., the Monte Carlo method. To improve the performance, the gradient-based method will also be applied to determine the optimal boundaries, which will minimize the approximation error. Moreover, the performance will be verified and analyzed in experiments with actual embedded systems.

Author Contributions: Conceptualization, B.L.; Methodology, B.L.; Software, J.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by a National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIP) (No. NRF-2020R1A2C1010891).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Nygaard, R.; Haugland, D. Compressing ECG signals by piecewise polynomial approximation. In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, Seattle, WA, USA, 15 May 1998; Volume 3, pp. 1809–1812. [CrossRef]
2. Ghosh, P.K.; Narayanan, S.S. Pitch contour stylization using an optimal piecewise polynomial approximation. *IEEE Signal Process. Lett.* **2009**, *16*, 810–813. [CrossRef] [PubMed]
3. Ravuri, S.; Ellis, D.P. Stylization of pitch with syllable-based linear segments. In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, Las Vegas, NV, USA, 31 March–4 April 2008; pp. 3985–3988. [CrossRef]
4. Kim, J.B.; Kim, B.K. The Calibration for Error of Sensing using Smooth Least Square Fit with Regional Split (SLSFRS). In Proceedings of the Korea Automatic Control Conference, Jeju Island, Republic of Korea, 10–12 December 2009; pp. 735–739.
5. Dong, N.; Roychowdhury, J. Piecewise polynomial nonlinear model reduction. In Proceedings of the 40th Annual Design Automation Conference, Anaheim, CA, USA, 2–6 June 2003; pp. 484–489. [CrossRef]
6. Stigler, S.M. Gergonne's 1815 paper on the design and analysis of polynomial regression experiments. *Hist. Math.* **1974**, *1*, 431–439. [CrossRef]
7. Ferguson, J.; Staley, P.A. Least squares piecewise cubic curve fitting. *Commun. ACM* **1973**, *16*, 380–382. [CrossRef]
8. Pavlidis, T.; Horowitz, S.L. Segmentation of plane curves. *IEEE Trans. Comput.* **1974**, *100*, 860–870. [CrossRef]
9. Gao, J.; Ji, W.; Zhang, L.; Shao, S.; Wang, Y.; Shi, F. Fast piecewise polynomial fitting of time-series data for streaming computing. *IEEE Access* **2020**, *8*, 43764–43775. [CrossRef]
10. Cunis, T.; Burlion, L.; Condomines, J.-P. Piecewise polynomial modeling for control and analysis of aircraft dynamics beyond stall. *J. Guid. Control Dyn.* **2019**, *42*, 949–957. [CrossRef]
11. Eduardo, C.; Luiz, F.N. Models and Algorithms for Optimal Piecewise-Linear Function Approximation. *Math. Probl. Eng.* **2015**, *2015*, 876862. [CrossRef]
12. Grützmacher, F.; Beichler, B.; Hein, A.; Kirste, T.; Haubelt, C. Time and Memory Efficient Online Piecewise Linear Approximation of Sensor Signals. *Sensors* **2018**, *18*, 1672. [CrossRef] [PubMed]
13. Marinov, M.B.; Nikolov, N.; Dimitrov, S.; Todorov, T.; Stoyanova, Y.; Nikolov, G.T. Linear Interval Approximation for Smart Sensors and IoT Devices. *Sensors* **2022**, *22*, 949. [CrossRef] [PubMed]
14. Warwicker, J.A.; Rebennack, S. Efficient continuous piecewise linear regression for linearising univariate non-linear functions. *IISE Trans.* **2024**. [CrossRef]
15. Ploussard, Q. Piecewise linear approximation with minimum number of linear segments and minimum error: A fast approach to tighten and warm start the hierarchical mixed integer formulation. *Eur. J. Oper. Res.* **2024**, *315*, 50–62. [CrossRef]
16. Liu, B.; Liang, Y. Optimal function approximation with ReLU neural networks. *Neurocomputing* **2021**, *435*, 216–227. [CrossRef]
17. Darby, C.L.; Hager, W.W.; Rao, A.V. An hp-adaptive pseudospectral method for solving optimal control problems. *Optim. Control Appl. Methods* **2011**, *32*, 476–502. [CrossRef]
18. M.OWEN. Cortex-M7 Instruction Cycle Counts, Timings, and Dual-Issue Combinations. Available online: https://www.quinapalus.com/cm7cycles.html (accessed on 20 March 2024).