Journal of Computational and Applied Mathematics 235 (2011) 4076–4082

Contents lists available at ScienceDirect

Journal of Computational and Applied Mathematics

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

A fast segmentation algorithm for piecewise polynomial numeric function generators

Jon T. Butler [a,*], C.L. Frenzen [b], Njuguna Macaria [a], Tsutomu Sasao [c]

[a] Department of Electrical and Computer Engineering, Naval Postgraduate School, Monterey, CA, 93943-5121, USA

[b] Department of Applied Mathematics, Naval Postgraduate School, Monterey, CA, 93943-5216, USA

[c] Department of Computer Science and Electronics, Kyushu Institute of Technology, Iizuka, 820-8502, Japan

ARTICLE INFO

Article history:
Received 15 June 2010

MSC:
68U07

Keywords:
Numerical approximation
Piecewise polynomial approximation
Numeric function generators
Segmentation algorithm
Piecewise linear approximation

ABSTRACT

We give an efficient algorithm for partitioning the domain of a numeric function f into segments. The function f is realized as a polynomial in each segment, and a lookup table stores the coefficients of the polynomial. Such an algorithm is an essential part of the design of lookup table methods Ercepovac et al. (2000) [5], Lee et al. (2003) [7], Nagayama et al. (2007) [12], Paul et al. (2007) [6] and Sasao et al. (2004) [8] for realizing numeric functions, such as $\sin(\pi x)$, $\ln(x)$, and $\sqrt{-\ln(x)}$. Our algorithm requires many fewer steps than a previous algorithm given in Frenzen et al. (2010) [10] and makes tractable the design of numeric function generators based on table lookup for *high-accuracy* applications. We show that an estimate of segment width based on local derivatives greatly reduces the search needed to determine the exact segment width. We apply the new algorithm to a suite of 15 numeric functions and show that the estimates are sufficiently accurate to produce a minimum or near-minimum number of computational steps.

Published by Elsevier B.V.

1. Introduction

The existence of large logic circuits has led to increased interest in an old problem — the realization of numeric functions. More than 150 years ago, Babbage designed the difference engine to automatically compute logarithmic and trigonometric functions [1]. This was intended to replace hand computation which was prone to error.

The availability of circuits to compute quickly functions like $\sin(x)$ and $\log(x)$ offers real-time execution of algorithms that can be used in applications such as the rendering of graphics or digital signal processing.

In this paper we give an efficient algorithm for partitioning the domain of a numeric function f into segments. Within each segment, the function f is realized as a polynomial with a lookup table storing the coefficients of the polynomial. We use an estimate of segment width based on local derivatives to greatly reduce the search needed to determine the exact optimal segment width. We apply the algorithm to a suite of 15 numeric functions, showing that the estimates are sufficiently accurate to produce a minimum or near-minimum number of steps in the computation.

Lookup tables have been used previously to implement a truncated series expression approximation of the given function. In [2], the function is realized by a converging series in which a single large memory is replaced by two or more smaller lookup tables. In [3,4], a Taylor series expansion is used. The first two terms of the expansion are realized by small lookup tables. The reciprocal, square root, inverse square root, and certain elementary functions were realized in [5] using a Taylor expansion and tables. Lookup tables have been used in the implementation of logarithm and antilogarithm computations in [6].

* Corresponding address: Department of Electrical and Computer Engineering, Naval Postgraduate School, Code EC/Bu, Monterey, CA, 93943-5121, USA. Tel.: +1 831 656 3299; fax: +1 831 656 2760.

E-mail addresses: jon_butler@msn.com (J.T. Butler), cfrenzen@nps.edu (C.L. Frenzen), nmacaria@hotmail.com (N. Macaria), sasao@cse.kyutech.ac.jp (T. Sasao).

0377-0427/$ – see front matter. Published by Elsevier B.V.
doi:10.1016/j.cam.2011.02.033

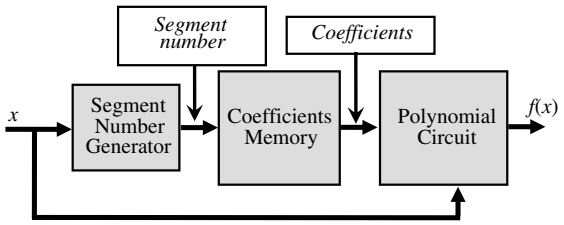

Fig. 1. Architecture of a numerical function generator using a piecewise polynomial approximation.

In [7], trigonometric and logarithmic functions are realized by table lookup using a non-uniform segmentation method. In this algorithm, narrow segments are used where the change in the function is large, and wide segments are used where the change in the function is small. The choice of segmentation, however, is determined by an a priori restriction on the circuit, and is non-optimum.

In [8], the Douglas–Peucker [9] algorithm is used to partition a given function into segments that are realized by a linear approximation. It is shown that a circuit producing an *optimum* non-uniform segmentation has a tractable realization for common numeric functions. Unfortunately, the Douglas–Peucker algorithm does not produce optimum segmentations [10].

Fig. 1 shows the architecture of the numeric function generator (NFG) that realizes a given function as a piecewise polynomial approximation [11]. It consists of three blocks. The Segment Number Generator uses the value of x to generate a segment number that is applied to the address input of the Coefficients Memory. The Coefficients Memory produces the coefficients in the polynomial expression for the given function. The piecewise polynomial approximation $f(x) \approx c_m x^m + \cdots + c_1 x + c_0$ is computed by the Polynomial Circuit in Fig. 1 using the coefficients produced by the Coefficients Memory.

Each segment in the function domain corresponds to a word in the memory that stores the polynomial coefficients for the function approximation in that segment. For a given approximation error, we seek a segmentation of the domain that has the fewest segments possible. This minimizes the memory required for the lookup table.

The algorithm given in this paper efficiently divides the domain of f into segments so that the error in polynomial approximation in each segment is no greater than a specified error. An algorithm that produces segmentations with the fewest segments is presented in [10]. However, it is computationally intensive, with a computation time sometimes measured in days or weeks. Although applied only once in the synthesis of a numeric function generator, the previous algorithm can make high accuracy applications impractical. Our main result, the new algorithm presented here, is orders of magnitude faster and still yields the fewest segments.

While the proposed segmentation algorithm applies to any order approximating polynomial, our examples focus on linear and quadratic approximations. In [12], it is shown that presently available field programmable gate arrays (FPGAs) have insufficient arithmetic elements, such as multipliers, to efficiently implement third or higher order polynomials. As FPGA technology improves, this may change.

2. Background

Because the variable x and the function's value $f(x)$ are represented as binary numbers with a fixed number of bits, a numeric function generator's output is inherently an approximation of the exact function value. While we may view the value of x as exact, it may not be possible to view $f(x)$ as exact. For example, consider the function $f(x) = \sqrt{x}$. If $x = 2$, we can realize 2 exactly. However, the irrationality of $\sqrt{2}$ means that its exact value cannot be realized in finitely many bits.

3. Estimating the segment width

An essential part of the new segmentation algorithm is the derivation of an estimate of the segment width. An accurate estimate is essential, because subsequently a search must be performed for the exact segment width. Later, we analyze the estimate's accuracy and show that, in many cases, it is as accurate as it can possibly be. First, we focus on deriving the estimate.

Let the segment over which we seek an nth-order polynomial approximation span $[e, s]$. The maximum approximation error ε of a Chebyshev approximation [13] is

$$\varepsilon = \frac{2(e-s)^{n+1}}{4^{n+1}(n+1)!} \max_{s \le x \le e} |f^{(n+1)}(x)|. \tag{1}$$

Solving (1) for the segment width, $e - s$ yields

$$e - s = 4 \sqrt[n+1]{\frac{(n+1)!\varepsilon}{2 \max_{s \le x \le e} |f^{(n+1)}(x)|}}. \tag{2}$$

For the two special cases of linear and quadratic approximating polynomials, we have

$$e - s|_{\text{linear}} = 4\sqrt{\frac{\varepsilon}{\max\limits_{s\le x\le e}|f''(x)|}} = 4\sqrt{\frac{\varepsilon}{|f''_{e-s}(x*)|}} \tag{3}$$

and

$$e - s|_{\text{quadratic}} = 4\sqrt[3]{\frac{3\varepsilon}{\max\limits_{s\le x\le e}|f'''(x)|}} = 4\sqrt[3]{\frac{3\varepsilon}{|f'''_{e-s}(x*)|}}, \tag{4}$$

where $e - s|_{\text{linear}}$ and $e - s|_{\text{quadratic}}$ are the segment widths for linear and quadratic approximations, respectively. We have chosen to replace $\max_{s\le x\le e}|f^{(n+1)}(x)|$ and $\max_{s\le x\le e}|f'''(x)|$ by the abbreviations $|f''_{e-s}(x^*)|$ and $|f'''_{e-s}(x^*)|$, respectively, recognizing that if the appropriate derivative is continuous on a closed interval, then the maxima above will each be attained at some point x^* within that interval.

4. The segmentation algorithm

4.1. Introduction

The algorithm is shown in Table 1. It applies to polynomial approximations of any order. We assume that the function domain is represented by a vector of N discrete points. For example, if the interval is [0, 1) and the accuracy is 8 bits, then we may choose N to be 256, and the points, in binary to be $0.0000\ 0000_2$, $0.0000\ 0001_2$, $0.0000\ 0010_2$, . . ., and $0.1111\ 1111_2$. That is, the domain in this example is the vector [0.000, 0.0039, 0.0078, . . . , 0.9961]. This assumption is consistent with the algorithm's implementation in MATLAB [14]. In MATLAB, we associate this vector with variable x. $f(x)$, in MATLAB, is then a vector of elements corresponding to the function evaluated at each of the elements in x. Therefore, $f(x)$ also has N elements.

Definition 1. For a given function, a *step* in a segmentation algorithm is a computation of the maximum absolute error between the function and its approximating polynomial on the proposed segment.

Because so much computation time occurs in the calculation of the maximum absolute error, a step, as defined in Definition 1, is an appropriate measure of the execution time. We compare the number of steps needed in the proposed algorithm with the number of steps needed in a brute force method.

In the brute force segmentation, the beginning point of the first segment is chosen to be the leftmost point in the interval of approximation; i.e. x_{low}. Then, the second point and successive points are chosen as prospective end points, and, for each choice, the error between the function and its approximating polynomial is computed. When this error exceeds the approximation error ε, the exact segment width has been found; it corresponds to the end point just before the end point that resulted in an error that exceeded ε. In the brute force method, all but the leftmost point in the interval is a prospective end point at which the error between the function and its approximating polynomial is computed. Therefore, approximately N steps are needed, where N is the number of points to represent the function in the whole interval of approximation.

The algorithm proceeds from the smallest value in the domain x_{low} to the largest x_{high}. It establishes the largest segment, starting at x_{low}, such that the maximum approximation error is ε. It repeats this process starting at e_1, the end point of the first segment, until it reaches x_{high}. Often, the last segment is truncated because x_{high} is reached before a segment end occurs (where the approximation error is ε). As a result, it is not unusual for the last segment to have a maximum approximation error strictly less than ε.

Fig. 2 shows an example segmentation. The vertical axis plots the function value $f(x)$, while the horizontal axis plots x. x_{low} is the left-hand end of the interval over which $f(x)$ is realized, and x_{high} is the right-hand end of the interval.

4.2. Three parts to the algorithm

There are three parts to the algorithm.

In the first part, ESTIMATE, the segment width is estimated. The process of estimation is discussed in the next section. Using the estimated segment width, an end point is found and the approximation error for the proposed segment is computed. This counts as one step.

In the second part, LOCATE, two points in the domain are located such that one point yields a segment whose approximation error is just below (or equal to) ε, and the other point yields a segment whose approximation error is just above.

This is accomplished as follows: from the estimated segment width computed in ESTIMATE, it is known whether the corresponding point is above the optimum segment width or below (or equal). If above, LOCATE proceeds towards lower values of x searching for two points that straddle the optimum segment width. If below, LOCATE proceeds towards higher values. Assume the point is below. The algorithm proceeds toward the optimum segment width by one point initially. It

Table 1
Algorithm to segment a given function based on estimates of the segment length.

Algorithm: Segment the domain $[x_{\text{low}}, x_{\text{high}}]$ of a given function $f(x)$, where $f(x)$ is approximated in each segment by a polynomial $c_n x^n + \cdots + c_1 x + c_0$.
Input: Function $f(x)$, domain $[x_{\text{low}}, x_{\text{high}}]$, approximation error ε, and order of the approximating polynomial, n. **Output:** Optimum segmentation, in which the i-th segment is specified as $[s_i, e_i]$, where s_i and e_i are the beginning and end point, respectively.
1. $i \leftarrow 1$. $s_1 \leftarrow x_{\text{low}}$.
ESTIMATE 2. Estimate the current segment width determining end point e_{est} and approximation error ε_{est}. If $e_{\text{est}} > x_{\text{high}}$, then $e_{\text{est}} \leftarrow x_{\text{high}}$. If $e_{\text{est}} = x_{\text{high}}$ and $\varepsilon_{\text{est}} \leq \varepsilon$, then STOP with $e_i \leftarrow e_{\text{est}}$. Set $L = e_{\text{est}}$ and $H = e_{\text{est}}$.
LOCATE 4. If $\varepsilon_{\text{est}} < \varepsilon$, then increase H to find upper and lower bounds H and L on the segment end point with the property (a) $\varepsilon_L \leq \varepsilon < \varepsilon_H$, where ε_H and ε_L are the approximation errors for the segments $[s_i, H]$ and $[s_i, L]$, respectively. Go to Step 5. (b) $\varepsilon_H \leq \varepsilon$ and $H = x_{\text{high}}$. STOP with $e_i \leftarrow e_{\text{est}}$. If $\varepsilon_{\text{est}} \geq \varepsilon$, then decrease L to find upper and lower bounds H and L on the segment end point.
PINPOINT 5. Using H and L, produce Hpp and Lpp with the property $\varepsilon_{Lpp} \leq \varepsilon < \varepsilon_{Hpp}$, where ε_{Hpp} and ε_{Lpp} are the approximation errors for the segments $[s_i, Hpp]$ and $[s_i, Lpp]$, respectively that are *adjacent* points above and below the optimum segment width. Choose the segment end point e_i to be Lpp.
6. $s_{i+1} \leftarrow$ point above e_i. $i \leftarrow i + 1$. Go to Step 2.

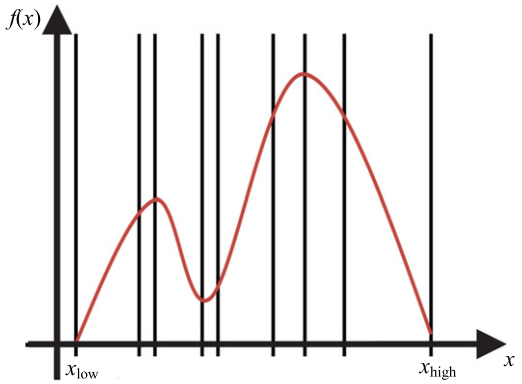

Fig. 2. Example segmentation.

computes the approximation error of the new segment, adding 1 to the number of steps. If the approximation error exceeds ε, ESTIMATE stops. It has found two points on each side of the optimum segment width. Indeed, they are adjacent and the algorithm stops; there is no need to proceed to the next step, PINPOINT.

However, if the approximation error is still less than (or equal to) ε, the algorithm advances *two* points. Again, it computes the approximation error, adding 1 to the number of steps, and repeats the process above. This is repeated except that the algorithm advances four, eight, etc. points, until two points are found that are on each side of the optimum segment width. If a total of m steps are taken, the algorithm has advanced $1+2+4+\cdots+2^{m-1} = 2^m - 1$ points. At the end of ESTIMATE, the last two points considered, H and L, correspond to end points of segments that straddle the *exact* end point of the segment Specifically, H is the end point of a segment in which the error achieved is either greater than ε, and L is the end point of a segment in which the error achieved is less than or equal to ε.

In the third part, PINPOINT, a bisection method is applied to H and L. That is, the midpoint $A = \frac{H+L}{2}$ is computed. Then, a new segment whose end point is A is created, and its approximation error is computed. If this exceeds ε, then H is replaced by A and the process is repeated. If this is less than or equal to ε, then L is replaced by A and the process is repeated. Each time a new approximation error is computed, the number of steps is increased by 1. The process stops when the H and L are adjacent. The segment end point is chosen to be L, since the maximum error in the segment ending in L is less than or equal to ε, while the maximum error in the segment ending in H exceeds ε.

4.3. Number of steps

Because of the way the Algorithm is constructed, the difference between H and L is 2^m. We have the following lemma.

Lemma 1. *Let the number of points between the high and low point $H - L$ be a power of 2, 2^m. For all but the last segment, the average and the worst case number of steps $N_{\text{PINPOINT}}(m)$ required by PINPOINT is m. No steps are required by PINPOINT to compute the last segment.*

Proof. The proof is by induction on m. For $m = 0$, H and L are adjacent, and no further steps are needed. Assume the hypothesis is true for all $m < m'$, and consider $H - L = 2^{m'}$. There is one step required to compute the approximation error for a segment that ends at $P = \frac{H+L}{2}$. Either H or L is replaced by P, and the problem is one of determining the number of steps needed to compute the segment end point between (the new) H and L. Since $H - L = 2^{m'-1}$, from the assumption, $m' - 1$ steps are needed, for a total of m' steps.

No steps are required by PINPOINT to compute the last segment because H is x_{high} and the error associated with a segment end point of H is equal to or less than ε. □

Similarly, the number of steps required by LOCATE can be calculated, as shown in Lemma 2. We begin with a definition.

Definition 2. A *truncated segment* is a segment whose estimated end point is greater than x_{high}.

For each segment, the algorithm in Table 1 provides an estimated segment end point that is used to start the search for the exact end point. A truncated segment has the property that its estimated end point is greater than x_{high}. Often, a truncated segment occurs as the last (rightmost) segment in a segmentation.

Interestingly, a truncated segment is not necessarily the last segment. For example, suppose that in a linear approximation of the function, the function is nearly linear throughout most of the interval, except near the end. In this case, the segment proposed end point may reach the end point of the interval of approximation (especially, if the segment is near the interval end point). Thus, it may be a truncated segment. However, when PINPOINT is applied, the exact end point may be found to be an internal point. Since the next segment might be in a highly non-linear part of the domain, the segment is necessarily narrow, and its end point may not reach the interval's end point. Therefore, subsequently constructed segments may be *non*-truncated. In the course of generating the experimental data, our proposed algorithm encountered this phenomena.

Lemma 2. *The number of steps required to construct a non-truncated segment in LOCATE, $N_{\text{LOCATE}}(m)$, is $N_{\text{PINPOINT}}(m) + 2$.*

Proof. The proof is by induction on m. For $m = 0$, H and L are adjacent, and PINPOINT requires no steps. The approximation error associated with segments whose end points are H and L require a total of two steps. Therefore, $N_{\text{PINPOINT}}(0) = 0$ and $N_{\text{LOCATE}}(0) = 2$. Assume the hypothesis is true for all $m < m'$, and consider m'. It follows that $N_{\text{LOCATE}}(m') = N_{\text{LOCATE}}(m' - 1) + 1$. The hypothesis follows. □

For the last segment, LOCATE requires some number of steps before it is determined that $H = x_{\text{high}}$. At this point, if the approximation error with H as the segment end point is equal to or less than ε, then it is established that, indeed, this is the last segment. No steps are needed by PINPOINT.

In the best case, ESTIMATE produces a segment end point that is no more than one step away from the optimum segment end point. This requires one step. To verify this and thus terminate the segment construction, another step is required, for a total of two steps per segment. From the discussion above, a truncated segment may, in the best case, require only one step. Therefore, we have

Lemma 3. *At least $2s - 1$ steps are needed to segment a domain, where s is the number of segments in the segmentation.*

Lemma 3 assumes that the estimates of segment length are as accurate as possible. As N, the number of points in the domain, becomes large, then the percentage of steps needed compared to the brute force method approaches 0. The program shows a clear tendency to lower percentage of steps as N increases.

5. Artifacts associated with the use of different accuracies

5.1. A Conundrum

Intuition suggests that using many points (e.g. 10,000,000) to represent an interval of approximation $[x_{\text{low}}, x_{\text{high}}]$ yields a more accurate segmentation than when fewer points are used (e.g. 256). Thus, one expects the segments to be narrower (or the same) when fewer points are used. On the contrary, if the segments are wider, the approximation error is greater than ε. Therefore, one expects *more* segments (or the same) are needed when there are fewer points to represent the interval of approximation.

However, this is not the case. Table 2 shows the number of segments needed to realize three functions, $\sqrt{-\ln(x)}$, $-(x \log_2 x + (1 - x) \log_2(1 - x))$, and $\sin(e^x)$, using 8-bits of precision and a linear approximation[1]. There are two cases, $N = 256$ and $N = 10{,}000{,}000$. For all three functions, the number of segments for $N = 10{,}000{,}000$ is *larger* than for $N = 256$.

5.2. Resolution

In the algorithm shown in Table 1, the beginning point of a segment is the next point after the end point of the previous segment (*not* the same point at which the last segment ends). This recognizes that each point belongs to exactly one

[1] $\sqrt{-\ln(x)}$, $-(x \log_2 x + (1 - x) \log_2(1 - x))$, and $\sin(e^x)$ were considered in [7,10,11], respectively.

Table 2
Three functions that require fewer segments when $N = 256$ than when $N = 10{,}000{,}000$ for a linear piecewise approximation.

Function $f(x)$	Interval x	No. of Segs.	
		$N = 256$	$N = 10^7$
$\sqrt{-\ln(x)}$	$\left[\frac{1}{256}, \frac{1}{4}\right)$	12	14
$-(x\log_2 x + (1-x)\log_2(1-x))$	$(0, 1)$	19	20
$\sin(e^x)$	$[0, 2]$	27	28

Table 3
Percentage of steps (compared to brute force) required to segment functions approximated by linear polynomials using different estimates of segment width for $N = 2^{16}$ and $\varepsilon = 2^{-17}$.

Function $f(x)$	Interval x	% of Steps vs. Brute Force				Min. %	# of Segs
		0	1	2	3		
2^x	$[0, 1)$	2.28	0.46	[a]0.23	[a]0.23	0.23	75
$1/x$	$[1, 2)$	2.34	0.75	[a]0.23	[a]0.23	0.23	75
$\sqrt{x}$	$[1, 2)$	1.19	0.46	[a]0.11	[a]0.11	0.11	35
$1/\sqrt{x}$	$[1, 2)$	1.62	0.62	[a]0.15	[a]0.15	0.15	50
$\log_2(x)$	$[1, 2)$	2.35	0.67	[a]0.23	[a]0.23	0.23	76
$\ln x$	$[1, 2)$	2.00	0.60	[a]0.19	[a]0.19	0.19	63
$\sin(\pi x)$	$\left[0, \frac{1}{2}\right)$	3.16	0.71	0.38	0.35	0.33	109
$\cos(\pi x)$	$\left[0, \frac{1}{2}\right)$	3.15	0.70	0.35	[a]0.33	0.33	109
$\tan(\pi x)$	$\left[0, \frac{1}{4}\right)$	2.25	0.83	0.27	0.25	0.22	73
$\sqrt{-\ln(x)}$	$\left[\frac{1}{256}, \frac{1}{4}\right]$	4.87	1.36	[a]0.63	[a]0.63	0.63	207
$\tan^2(\pi x) + 1$	$\left[0, \frac{1}{4}\right)$	4.25	0.82	[a]0.46	[a]0.46	0.46	152
$-(x\log_2 x + (1-x)\log_2(1-x))$	$\left[\frac{1}{256}, \frac{255}{256}\right]$	7.74	1.38	[a]0.96	[a]0.96	0.96	314
$\frac{1}{1+e^{-x}}$	$[0, 1)$	0.72	0.37	0.14	0.10	0.06	20
$\frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$	$[0, \sqrt{2}]$	2.32	0.84	0.38	0.30	0.23	53
$\sin(e^x)$	$[0, 2)$	10.19	2.05	1.43	1.40	1.35	449

[a] *All* segments require the fewest steps.

segment. Thus, there is "space" between segments that need not be realized by a polynomial. This is benign; there are no input combinations that correspond to values in this space. For example, in the case of $\sin(e^x)$, there are 27 segments, and thus, 26 spaces between segments. Since only 256 points represent the segment, more than 26/256 of the interval is *not* realized in the approximation. This effectively shortens the interval by about 10%. In the case of $N = 10{,}000{,}000$, the space between segments is a much smaller fraction of the total interval width. This effect dominates and is the reason that fewer points yields fewer segments in Table 1.

6. Experimental results

To analyze the benefit of estimates in the proposed algorithm, we configured a MATLAB program to apply only LOCATE and PINPOINT in constructing each segment. That is, estimates were *not* used in specifying a prospective end point of the next segment. Instead, the initial end point was chosen to be just beyond the beginning point of the newly constructed segment. In this case, LOCATE and PINPOINT must search over the full segment. Table 3 shows how this compares to the brute force method when applied to a suite of 15 functions for $\varepsilon = 2^{-17}$. Each entry represents the ratio of the number of steps needed to compute the segmentation using the proposed algorithm to the number of steps needed by the brute force method. This is expressed as a percentage. The values, shown in the column labeled % *of Steps vs. Brute Force* 0, range from 0.72% for $\frac{1}{1+e^{-x}}$ to 10.19% for $\sin(e^x)$. This shows that LOCATE and PINPOINT realize a significant reduction over the brute force method. For 13 of the 15 functions, the ratios are less than 5.0%, which is a significant reduction in the number of steps.

However, estimates provide still further improvement. Table 3 shows the benefits of 1, 2, and 3 estimates. The column labeled % *of Steps vs. Brute Force* 1 shows that, when one estimate is used, the number of steps is reduced by as much as one-fifth that needed in the case of no estimate. For example, in the case of the entropy function $-(x\log_2 x + (1-x)\log_2(1-x))$ no estimate yields a percentage of 7.74%, while one estimate achieves a percentage of 1.38%, which is 1/5.6 of the number of steps.

In the case of one estimate, the beginning point of the segment is used to determine an estimate for the segment width. For example, when linear approximation is used, the second derivative of the new segment beginning point is computed and substituted into (3) to derive an estimate for the segment width. Then, a proposed end point is obtained by adding the estimated segment width to the beginning point. The approximation error is computed and used to determine in which direction from the estimated end point LOCATE should search.

The next column labeled % *of Steps vs. Brute Force* 2 shows the benefit of two estimates. In this case, the estimate of the segment width computed with the first step in the segment (discussed in the previous paragraph) is averaged with the

estimate of the segment width computed with the segment end point, as estimated from the first step. This approach is based on the assumption that the average of two estimates, one at the beginning and one near the end of the proposed segment provides a better estimate of the actual segment width than one estimate alone. As can be seen in Table 3, two estimates provides substantial reduction in the number of steps. Indeed, for 9 of the 15 functions, the minimum number of steps is achieved (where the minimum was *not* achieved for any of the 15 function in the case of one estimate). An asterisk indicates that this percentage is the best that can be obtained, as shown in Lemma 3. The reduction in the number of steps achieved by using two estimates instead of one ranges from 1/1.4 to 1/4.4.

The next column labeled *% of Steps vs. Brute Force* 3 shows the benefit of three estimates. In this case, the final estimate is the average of three estimates, one from the beginning, one from the end, and one from the middle of the segment whose width is estimated from the first point in the segment. Now, 10 of the 15 functions achieve the minimum number of steps.

The column labeled *Min %* shows a percentage that represents the minimum number of steps required if the estimates were perfect, as specified by Lemma 3. Comparing this with the column labeled *% of Steps vs. Brute Force* 3 shows that, even for the five functions that did not achieve a minimum number of steps, the number of steps is close to minimum. Four of the five functions are within 30%, while one $\frac{1}{1+e^{-x}}$ is within 72%.

7. Concluding remarks

We give a segmentation algorithm that efficiently segments a given numeric function, such as $\sin(\pi x)$, in such a way that the polynomial approximation error is less than or equal to some given value. The algorithm is optimum and requires many fewer steps than a previous algorithm [10]. Experimental results show that, in some instances, only the absolute minimum number of steps is needed. In most instances, it requires close to the minimum number of steps. For more implementation information, see [15].

Acknowledgements

This research is supported in part by an NSA Contract, Grants in Aid for Scientific Research of JSPS, and MEXT, and a grant of the Kitakyushu Innovative Cluster Project.

References

[1] A.G. Bromley, The evolution of Babbage's calculating engines, Ann. Hist. Comput. 9 (1987) 113–136.
[2] H. Hassler, N. Takagi, Function evaluation by table lookup and addition, in: Proc. of the 12th IEEE Symp. on Computer Arithmetic, ARITH'95, Bath, England, July, 1995, pp. 10–16.
[3] M.J. Schulte, J.E. Stine, Approximating elementary functions with symmetric bipartite tables, IEEE Trans. Comput. 48 (8) (1999) 842–847.
[4] J.E. Stine, M.J. Schulte, The symmetric table addition method for accurate function approximation, J. VLSI Signal Process. 21 (2) (1999) 167–177.
[5] M.D. Ercepovac, T. Lang, J.-M. Muller, A. Tisserand, Reciprocation, square root, inverse square root, and some elementary functions using small multipliers, IEEE Trans. Comput. 49 (7) (2000) 628–637.
[6] S. Paul, N. Jayakumar, S.P. Khatri, A hardware approach for approximate, efficient logarithm and antilogarithm computations, IWLS-2007, San Diego, CA, May 30–June 1, 2007, pp. 260–265.
[7] D.U. Lee, Wayne Luk, J. Villasenor, P.Y.K. Cheung, Non-uniform segmentation for hardware function evaluation, in: Proc. Inter. Conf. on Field Programmable Logic and Applications, Lisbon, Portugal, September, 2003, pp. 796–807.
[8] T. Sasao, J.T. Butler, M.D. Riedel, Application of LUT cascades to numerical function generators, in: The 12th Workshop on Synthesis and System Integration of Mixed Information technologies, SASIMI2004, Kanazawa, Japan, October 18–19, 2004, pp. 422–429.
[9] D.H. Douglas, T.K. Peucker, Algorithms for the reduction of the number of points required to represent a line or its caricature, Canadian Cartog. 10 (2) (1973) 112–122.
[10] C.L. Frenzen, T. Sasao, J.T. Butler, On the number of segments needed in a piecewise linear approximation, J. Comput. Appl. Math. 234 (2010) 437–466.
[11] J. Muller, Elementary Functions—Algorithms and Implementation, Birkhäuser, Boston, 1997.
[12] S. Nagayama, T. Sasao, J.T. Butler, Design method of numerical function generators based on polynomial approximation for FPGA implementation, in: 10th Euromicro Conference on Digital System Design, Architecture, Methods, and Tools, DSD 2007, Lübeck, Germany, August 27–31, 2007.
[13] J.H. Mathews, Numerical Methods for Computer Science, Engineering, and Mathematics, Prentice-Hall, Inc, Englewood Cliffs, NJ, 1987.
[14] S.J. Chapman, MATLAB Programming for Engineers, 2nd ed., Brooks/Cole Thomson Learning, 2002, p. 52.
[15] N. Macaria, High-speed numeric function generator using piecewise quadratic approximations, Naval Postgraduate School, Master's Thesis, Monterey, CA, September, 2007.