Bivariate Function Polynomial Approximation with Dynamic Degree, Real-Time Software Implementation

Mathieu Percelay[1]*

Abstract
The approximation of mathematical functions is an emerging field of research. The needs in embedded systems has never been as much considerable. Many efforts have been made for univariate function approximation. However, regarding multivariate function approximation, only hardware solutions have been proposed. This paper presents a software-oriented polynomial approximation for bivariate functions with a non-uniform segmentation strategy and a dynamic polynomial degree.

Keywords
Approximate computing — Embedded Systems — Minimum Mean Square Error Method — Non-uniform Segmentation

[1] *Department of Electronics and Computer Science, INSA Rennes, Rennes, France*
***Corresponding author**: Mathieu.Percelay@insa-rennes.fr

Contents

1. Introduction

Improvements in micro-electronics and the development of the Internet of Things make the amount of Embedded Systems rising sharply. In domains such as telecommunications, signal processing or robotics, mathematical function evaluation is widely used. Even if the computer performances has risen over the past decades, exact computation is done at the expense of hardware costs, execution time, energy consumption or memory footprint. Such costs can be unsuitable for Real-Time Embedded applications. It is then possible to reduce the precision of computation, i.e. approximate the function in order to decrease these costs. The challenge is to find the best tradeoff for the application and the hardware target. Approximate-computing is today a very important paradigm [1] in computer science and micro-electronics fields. Some methods have been previously used as the CORDIC iterative algorithm for trigonometric functions [2], multipartite tables [3] or polynomial approximation [4]. These methods where developed for univariate function approximation. Some additional issues appear when these methods are extended to multivariate.

To this day, the majorities of multivariate function approximation deals with linear approximation and hardware implementation, by J.Rust et al [5] or D. Linaro et al [6]. The subject of this paper is the study of a software polynomial approximation technique with adaptive methods such as a non-uniform segmentation and a dynamic degree in the idea of being suitable for most applications while maintaining performances.

The paper is organized as follows. In Section 2 related works on both univariate and multivariate are introduced. Section 3 explains in details the approximation model creation with MATLAB and the C implementation of the evaluation program. Results about the approximation model and the software implementation are discussed in Section 4 before the conclusion in Section 5.

2. Related Works

2.1 Univariate Polynomial Function Approximation

The starting point of the works presented in this paper is the method proposed by J. Bonnot in [4]. The paper presents a smart univariate polynomial approximation method. It is combined with a non-uniform segmentation method and a fixed point coefficient encoding. The approximating polynomial is found through the Remez algorithm which minimizes the maximum error between the polynomial and the approximated function. An error criterion has to be observed and if it is not, a segmentation of the function domain is performed. A new polynomial is computed on each of the two resulting sub-segments. If the criterion is still unsatisfied, the process is repeated recursively.

The degree is fixed for all polynomials and given at the begin of the process.

That segmentation is then represented in a binary tree which is then reduced by a memory saving strategy. This way of segmentation allows an efficient most-significant-bit indexing when evaluating the approximating polynomial and a very concise representation.

2.2 Hardware-Oriented Bivariate Linear Approximation

In [5] are presented interesting methods about the segmentation with two variables function. The function is linearly approximated, i.e. approximated by a bivariate polynomial of degree 1. The coefficients of this polynomial are computed through the Minimum Mean Square Error (MMSE) Method. A bivariate non-uniform segmentation similar to the univariate case in [4] is performed. If the error criterion is not observed, the original segment is split into four equal squared subsegments.

When the recursive process is terminated, a segment merging step is considered. A merging between two segments is performed when they are neighbors, i.e. they have a complete side in common, and the approximation created by the mean of their coefficients observes the error criterion on the whole segment. An example of such a segmentation is shown in Figure 1.

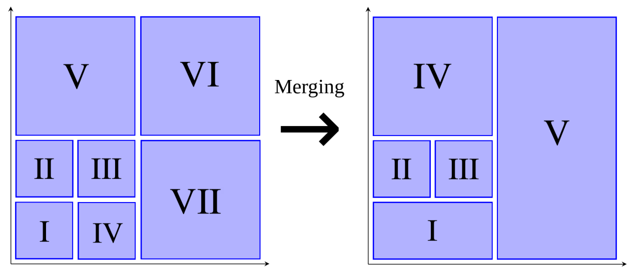

Figure 1. An example of a non uniform segmentation with merging. (I, IV) and (VI, VII) have been merged

3. Proposed Solution

In order to overcome the lack of software-oriented solutions of polynomial bivariate-function approximation, the proposed method aims to offer an efficient and adaptive technique for the higher number of applications. This technique is based on the Mean Square Minimization Method which is appropriate for the majority of approximation problems. This solution uses a smart non-uniform segmentation in order to better fit irregularities of the approximated function. Furthermore, for best performances, the degree of the bivariate approximating polynomial is dynamic which avoid data and time waste. The proposed method is split into two step. The first step is the Approximation Model Creation which compute the polynomial coefficients and space segmentation for the desired approximated function with MATLAB. The second step is the use of this approximation model in a Real-Time-oriented embedded system with an optimized C-software.

3.1 Approximation Model Creation with MATLAB

3.1.1 Mean Square Minimization Method Adaptation

Let consider a bivariate function f defined on a space E.

$$\begin{array}{cccc} f & : & E \subset \mathbb{R}^2 & \rightarrow & \mathbb{R} \\ & & p=(x,y) & \mapsto & f(p) \end{array}$$

The Mean Square Minimization method will be used as follow. An uniformly segmented rectangular space is considered which will be given by n points $\{p_i = (x_i, y_i) \in E, i = 1..n\}$. Let be m bivariate monomials

$$\left\{ \begin{array}{cccccc} M_i & : & E & \rightarrow & \mathbb{R} & i = 1..m \\ & & p=(x,y) & \mapsto & x^{a_i} \times y^{b_i} & a_i, b_i \in \mathbb{N} \end{array} \right\}$$

and m coefficients $\{\alpha_i \in \mathbb{R}, i = 1..m\}$. The corresponding bivariate polynomial P is defined as follow :

$$\begin{array}{ccccc} P & : & E & \rightarrow & \mathbb{R} \\ & & p=(x,y) & \mapsto & \sum_{i=1}^{m} \alpha_i \times M_i(p) \end{array}$$

The goal is to find the right $\alpha_i s$ in order to get a statisfying approximation if f by P. Such a solution can be find by the *Minimum Mean Square Error Method* (MMSE). Let be

$$A \in \mathbb{R}^{n \times m} = \begin{pmatrix} M_1(p_1) & \cdots & M_m(p_1) \\ \vdots & \ddots & \vdots \\ M_1(p_n) & \cdots & M_m(p_n) \end{pmatrix}$$

$$\alpha \in \mathbb{R}^m = \begin{pmatrix} \alpha_i \\ \vdots \\ \alpha_m \end{pmatrix} \text{ and } F \in \mathbb{R}^n = \begin{pmatrix} f(p_1) \\ \vdots \\ f(p_n) \end{pmatrix}$$

The Mean Square Error (MSE) of the approximation of f by P problem is : $MSE = \|(A\alpha - F)^2\|$. The $MMSE$ Method gives the solution $\tilde{\alpha}$ of the MSE minimum by solving the equation $A^T A \tilde{\alpha} = A^T F$.

The degree of a polynomial is defined as : $d = max(a_i + b_i), 1_i)$ [7]. For a a given degree d, there is $m = \frac{(d+1)d}{2}$ different monomials of degree $\leq d$. The corresponding polynomial P is of degree d. The proposed technique uses polynomials characterized by their degree, which can be dynamic, in other words, different for each polynomial used. The monomials

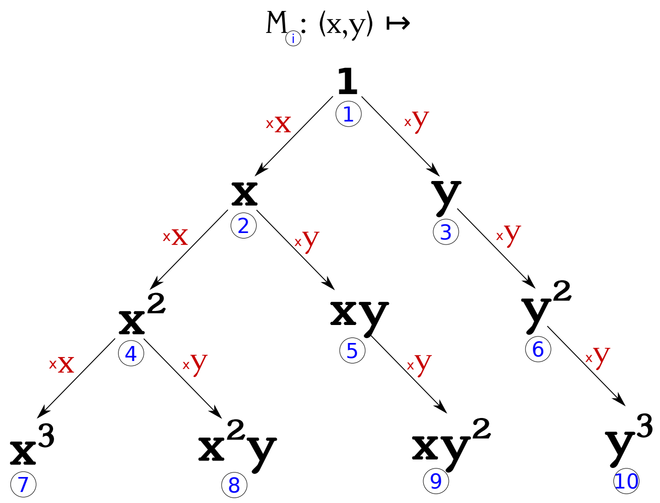

Figure 2. The way of computation of the monomials for a polynomial of degree 3. Each *i*th monomial is deduced from the previous one according to the nodes ordering in the tree.

M_i have to be memorized in a known order as described in Figure 2, in order to rebuild the polynomial P well.

The MSE is then compared with a criterion ε. If it is not respected (ie. $MSE \geq \varepsilon$), a segmentation is done.

3.1.2 Sufficient Degree Segmentation Method

To match irregularities of f, a segmentation of E can be useful. Segmenting E means splitting it in several subsets Es_i and the same MMSE Method is applied on these subsets to solve the minimization problem. The solution to the global problem on E is equal to the union of the solution on each Es_i. For the proposed technique, the segmentation is implemented as a four regular quadrant splitting for performances and implementation ease. For $E = [a,b] \times [c,d]$:

$$Es = \begin{cases} Es_1 = [a, \frac{a+b}{2}] \times [c, \frac{c+d}{2}] \\ Es_2 = [\frac{a+b}{2}, b] \times [c, \frac{c+d}{2}] \\ Es_3 = [a, \frac{a+b}{2}] \times [\frac{c+d}{2}, d] \\ Es_4 = [\frac{a+b}{2}, b] \times [\frac{c+d}{2}, d] \end{cases} \quad (1)$$

The segmentation can be applied recursively on one or several subsets. The segmentation scheme becomes then a non-uniform segmentation which is described with a quadtree as shown in Figure 3. Each node of the tree is containing the coefficients vector α computed by the MMSE Method on a given segment. The degree of P can be different, so the length of α is dynamic according to the considered segment.

The Sufficient Degree Segmentation Method as for input

- a function f and its domain of definition E
- a MSE criterion ε
- a degree limit d_{max}

Then the method uses the MMSE Method as described in the previous section according to the recursive algorithm Alg.1

Algorithm 1 Sufficient Degree Segmentation (SDS) Algorithm

```
1:  function $SDS(f, E, \varepsilon)$
2:      $d \leftarrow 0$
3:      do
4:          $\alpha \leftarrow polynomial\_least\_square(f, E, d)$
5:          $MSE \leftarrow get\_MSE(f, E, \alpha)$
6:          $d \leftarrow d + 1$
7:      while $MSE > \varepsilon$ and $d \leq d_{max}$
8:      $t \leftarrow tree(\alpha)$
9:      if $MSE > \varepsilon$ then
10:         for $e \in segmentation(E)$ do
11:             $t \leftarrow add\_subtree(t, SDS(f, e, \varepsilon))$
12:         end for
13:     end if
14: return $t$
15: end function
```

The final approximating polynomial model is given by the segmentation tree and the corresponding polynomial coefficients given by the MMSE Method for each segment.

3.1.3 Best Tradeoff Segmentation Method

The previous method is a naive sufficient solution search. This solution might not be satisfying because the memory footprint is too big or the evaluation latency is too long. These two important data in embedded systems are considered in the Best Tradeoff Segmentation Method.

The memory footprint is computed from the data needed for the segmentation tree representation and from the polynomial coefficients storage. The latency is estimated in number of elementary operations. It takes in account the segmentation tree traversing and the evaluation using the polynomial coefficients. As each algorithmic operation (addition, multiplication, comparison, memory access) have different latencies according to the hardware target so these parameters have to be estimated previously. Given the limits m_{max}, t_{max} and ε which are respectively the maximum footprint, the maximum latency and the maximum MSE desired.

The Best Tradeoff Method looks for a satisfying solution respecting these limits through an exhaustive research in all the configurations of the segmentation tree and the polynomial degrees. This exhaustive research is done until a maximum tree depth s_{max} and a maximum polynomial degree d_{max} for performances and in case of no satisfying solution.

3.2 Real-Time-oriented Software Evaluation

Once the adapted approximation model has been found through the previously detailed methods, it needs to be translated in a Embedded-Oriented software language. For best performances, the C language has been chosen.

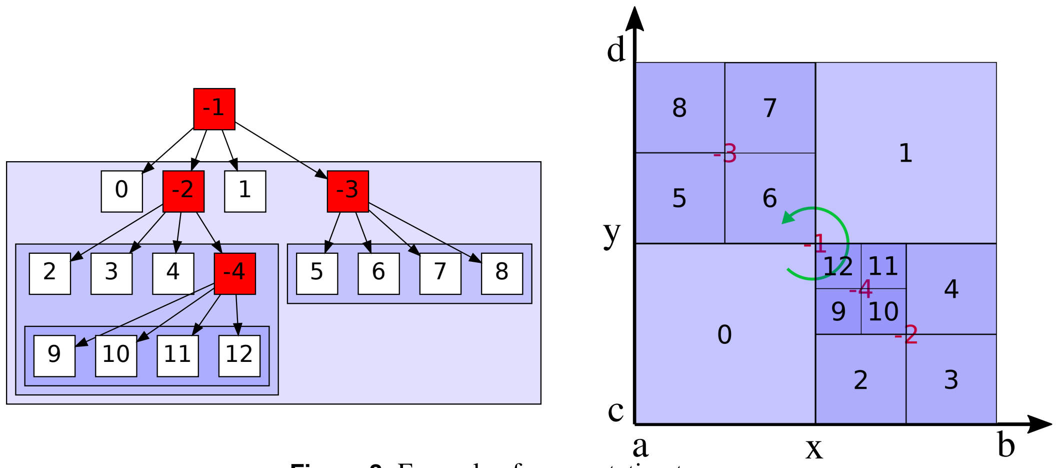

Figure 3. Example of segmentation tree

3.2.1 Efficient Model Description in C with floating point

The first item to depict is the segmentation tree. A Breadth First Search (BFS) is performed in the segmentation tree in order to number the nodes. The numbering is different according to the node nature : leaf or father. A father is a node where the MMSE solution α is not satisfying and the segment has been divided. A leaf is a final node where α describes a polynomial function P that respects the MSE criterion ε with f for the sub-segment. For each depth, the four neighboring nodes are counterclockwise numbered starting with the first one as described in equation 1. One table T_{leaf} is created with this numbering and is indexed by the order given by the BFS. Given x and y the function arguments expressed in floating point representation, the leaf corresponding to the segment which includes x and y can be found with Algorithm 2 with $index = 0$, the root of the tree.

Algorithm 2 Segmentation Tree Traversal Algorithm

```
1: function $leaf = FindLeaf(T_{leaf}, index, x, y)$
2:     if $T_{leaf}[index] < 0$ then
3:         $firstQuadrant \leftarrow (T_{leaf}[index] - 1) * 4 + 1$
4:         $q = FindQuadrant(T_{leaf}, index, x, y)$
5:         return $FindLeaf(T_{leaf}, firstQuadrant + q, x, y)$
6:     else
7:         return $T_{leaf}[index]$
8:     end if
9: end function
```

$leaf$ indexes two other tables T_{first_α} and T_d which indicate respectively the index of the first coefficient of the segment in the table T_α and the degree of the approximating polynomial P. T_α contains the concatenation of all the coefficients in floating point representation. α is then reconstructed.

$$\begin{cases} first\_\alpha = T_{first\_\alpha}[leaf] \\ d = T_d[leaf] \\ \alpha = T_\alpha[first\_\alpha : first\_\alpha + NB\_\alpha(d)] \end{cases}$$

The monomials M are computed as it was done in Figure 1 according to the degree d. The monomials are then multiplied by there respective coefficient in α and added up.

$$P(x,y) = \sum_{i=1}^{NB\_\alpha(d)} \alpha_i \times M_i \approx f(x,y)$$

4. Results and Discussion

4.1 peaks MATLAB example function

The *peaks* MATLAB bivariate example function is very useful for segmentation visualization with their irregularities. It is also a function hard to evaluate with classical mathematical function toolboxes as *libm* due to the presence of exponential and square functions.

$$peaks(x,y) = 3 \times (1-x)^2 \times e^{-x^2-(y+1)^2} - 10 \times (\frac{x}{5} - x^3 - y^5) \times e^{-x^2-y^2} - \frac{1}{3} \times e^{-(x+1)^2-y^2}$$

A visual example of the approximation model and the original function is presented in Figure 4.

A series of tests has been performed on a ARM Cortex A53 processor with the C implementation presented and the Best Tradeoff Method. The Figure 5 shows them. When d_{max} if small, in order to to respect the error criterion, the segmentation is very fine and needs a lot of memory to be represented. The evaluation time is on the contrary small because there is few coefficient then few floating point operations to process. When d_{max} is increasing, the memory footprint is decreasing but the evaluation is longer. Once $d_{max} \geq 5$, the model does not change because the Best Tradeoff Method found that the minimum of memory footprint is reached.

5. Conclusion

In this paper, an adaptative technique for efficient bivariate function polynomial approximation has been presented thanks to a non-uniform segmentation strategy and a dynamic degree solution. A software implementation shows the efficiency

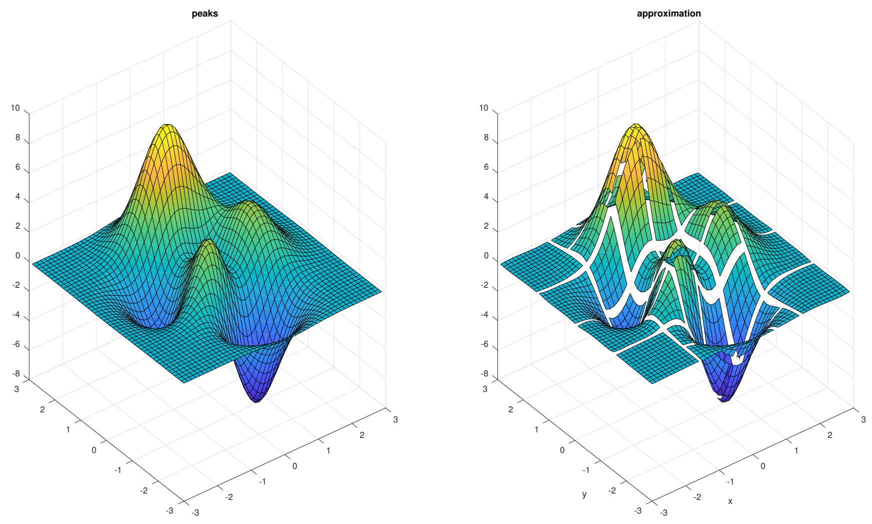

Figure 4. peaks approximation function for $\varepsilon = 0.001$, $d_{max} = 4$ and $s_{max} = 6$

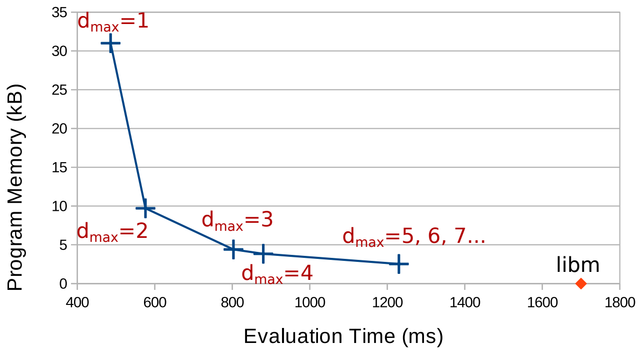

Figure 5. Results for evaluation of 1 million points uniformly distributed with $f = peaks$, $E = [-3,3] \times [-3,3]$, $\varepsilon = 0.001$ and $s_{max} = 6$

of the proposed technique with a bivariate function example on an ARM Cortex A53 processor, designed for very low consumption applications. This implementation uses floating point arithmetic for both segmentation tree traversal and coefficient encoding. The floating point processing is often inefficient in embedded systems due to calculation costs. A fixed point solution must be considered for future work.

Acknowledgments

The author wishes to thank J. Bonnot and D. Ménard for their previous works and their helpful comments and discussions.

References

[1] J. Han and M. Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *2013 18th IEEE European Test Symposium (ETS)*, pages 1–6, May 2013.

[2] J. E. Volder. The cordic trigonometric computing technique. *IRE Transactions on Electronic Computers*, EC-8(3):330–334, Sep. 1959.

[3] F. de Dinechin and A. Tisserand. Multipartite table methods. *IEEE Transactions on Computers*, 54(3):319–330, March 2005.

[4] Justine Bonnot, Daniel Menard, and Erwan Nogues. Segmentation non-uniforme pour l'approximation polynomiale de fonctions pour processeurs embarqués. In *Conférence d'informatique en Parallélisme, Architecture et Système*, Lorient, France, July 2016.

[5] J. Rust and S. Paul. Bivariate function approximation with encoded gradients. In *2016 IEEE Nordic Circuits and Systems Conference (NORCAS)*, pages 1–6, Nov 2016.

[6] D. Linaro and M. Storace. A method based on a genetic algorithm to find pwl approximations of multivariate nonlinear functions. In *2008 IEEE International Symposium on Circuits and Systems*, pages 336–339, May 2008.

[7] Jörg Hörner Klaus Höllig. *Approximation and Modeling with B-Splines*. SIAM, 2015.