Hierarchical Approximations of a Function by Polynomials in LEMA

Vincent LEFÈVRE

Arénaire, INRIA Grenoble – Rhône-Alpes / LIP, ENS-Lyon

2010-02-26
The Problem

**Goal:** the exhaustive test of the elementary functions for the TMD in a fixed precision (e.g., in binary64), i.e. “find all the breakpoint numbers $x$ such that $f(x)$ is very close to a breakpoint number”.

Breakpoint number: machine number or midpoint number.

→ Worst cases for $f$ and the inverse function $f^{-1}$. 

![Diagram](https://via.placeholder.com/150)
Hierarchical Approximations by Polynomials

**Current implementation** (but one could have more than 3 levels):

---

- Finding approximations must be very fast: from the previous one.
- Degree-1 polynomials: fast algorithm that computes a lower bound on the distance between a segment and $\mathbb{Z}^2$ (in fact, this distance, but on a larger domain) [filter] + slower algorithms when needed.
Computing the Successive Values of a Polynomial

Example: $P(X) = X^3$. Difference table:

On the left: coefficients of the polynomial in the basis

$$\left\{1, X, \frac{X(X - 1)}{2}, \frac{X(X - 1)(X - 2)}{3!}, \ldots \right\}$$
Representation in the LEMA Tree

Computations can (and will) be done modulo some constant (much faster).
→ The corresponding arithmetic must be supported by LEMA.

In practice, some coefficients will be close to 0 (either from above or from below).
→ In the LEMA tree, notion of magnitude (like with real numbers).

How can this be expressed in LEMA?

- With a list (tuple) containing the coefficients? (But the degree $d$ is not necessarily a constant parameter.)
- With a function taking two arguments $i$ and $n$ returning the coefficient $a_i(n)$ of $P(X + n)$ in the basis

$$\left\{1, X, \frac{X(X - 1)}{2}, \frac{X(X - 1)(X - 2)}{3!}, \ldots \right\}?$$

The polynomial object is less visible, but this should be easier.
An Example of Coefficient Values

An example of coefficient values from the current implementation:

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_{0_0}$</td>
<td>A6ABF7160809CF4F 3C762E7160F38B4E</td>
</tr>
<tr>
<td>$a_{0_1}$</td>
<td>5458A4173B436123 9CA0E833FEB6CB85 ABFC8C9</td>
</tr>
<tr>
<td>$a_{0_2}$</td>
<td>000002B7E1516295 CCAFBO49B66C0BEA 354A25BAAB8404F</td>
</tr>
<tr>
<td>$a_{0_3}$</td>
<td>0000000000000000 0000000000000000 0000000000DAF85458A986FD E62637A70A321BD8 4F1A4229E540A478</td>
</tr>
<tr>
<td>$a_{0_4}$</td>
<td>0000000000000000 00000000000000000002B7E1516 51628AED2A6ABF71 58809CF4F3C762E7</td>
</tr>
<tr>
<td>$a_{1_0}$</td>
<td>5BF0A8B145769AA5 225B715628DDCEBF</td>
</tr>
<tr>
<td>$a_{1_1}$</td>
<td>0000000000000000 0000000000000000000000000000002B7E1516 51628AED2A6ABF71 58809CF4F3C762E7</td>
</tr>
<tr>
<td>$a_{1_2}$</td>
<td>0000000000000000 0000000000000000000000000000002B7E1516 51628AED2A6ABF71 58809CF4F3C762E7</td>
</tr>
<tr>
<td>$a_{1_3}$</td>
<td>0000000000000000 0000000000000000000000000000002B7E1516 51628AED2A6ABF71 58809CF4F3C762E7</td>
</tr>
<tr>
<td>$a_{1_4}$</td>
<td>0000000000000000 0000000000000000000000000000002B7E1516 51628AED2A6ABF71 58809CF4F3C762E7</td>
</tr>
<tr>
<td>$a_{1_5}$</td>
<td>0000000000000000 0000000000000000000000000000002B7E1516 51628AED2A6ABF71 58809CF4F3C762E7</td>
</tr>
<tr>
<td>$a_{2_0}$</td>
<td>0000000000000000 0000000000000000000000000000002B7E1516 51628AED2A6ABF71 58809CF4F3C762E7</td>
</tr>
<tr>
<td>$a_{2_1}$</td>
<td>0000000000000000 0000000000000000000000000000002B7E1516 51628AED2A6ABF71 58809CF4F3C762E7</td>
</tr>
<tr>
<td>$a_{2_2}$</td>
<td>0000000000000000 0000000000000000000000000000002B7E1516 51628AED2A6ABF71 58809CF4F3C762E7</td>
</tr>
<tr>
<td>$a_{2_3}$</td>
<td>0000000000000000 0000000000000000000000000000002B7E1516 51628AED2A6ABF71 58809CF4F3C762E7</td>
</tr>
<tr>
<td>$a_{2_4}$</td>
<td>0000000000000000 0000000000000000000000000000002B7E1516 51628AED2A6ABF71 58809CF4F3C762E7</td>
</tr>
</tbody>
</table>
From an Interval to the Next One

The problem (to be considered recursively):

- Input: a polynomial $P$ of degree $d$ on an interval $I$.
- The interval $I$ is split into subintervals $J_n$ of the same length.
- The polynomial $P$ will be approximated by polynomials $P_n$ of degree $d'$ on the intervals $J_n$ (sequentially).
- Goal: generate code to compute the (initial) coefficients of $P_n$ very quickly (from the work done for $P_{n-1}$ on $J_{n-1}$).
- All errors need to be bounded formally: an acceptable error bound will be part of the input, and various parameters (the precision of the coefficients, etc.) will be determined from it.
Two methods:

1. Take into account the computations that haven’t been done, i.e. those involving the coefficients of degrees $> d'$.
   → Linear combinations of coefficients: additions and multiplications of coefficients by integer constants (constant in the generated code).

2. Use the fact that the intervals $J_n$ have the same length: each (initial) degree-$i$ coefficient of $P_n$ can be seen as the value of a polynomial $a_i(n)$.
   → The difference table method can be used: only additions.
Error Bounds

Three kinds of errors:

- Error due to the approximation of function $f$ by a polynomial.
- Approximation errors: coefficients of degree $d > d'$ are ignored.

\[ \rightarrow \text{Error bound of the form: } \sum_{i=d'+1}^{d} U_i \cdot |a_i(0)| \]

where $U_i$ depends on $i$ and the size of the intervals $I$ and $J_n$.

- Rounding errors on the coefficients $a_i$ (due to their representation with an absolute precision $n_i$): initial errors and after each computation.

\[ \rightarrow \text{Error bound on } a_i(m) \text{ of the form: } \sum_{j=i}^{d'-1} V_{i,j,m} \cdot 2^{-n_j}. \]

Formally determining $U_i$ and $V_{i,j,m}$ can be to difficult to be done automatically. But it should be possible to verify (prove) them with LEMA with conventional error analysis:

- with help of computer algebra software for generic formulas;
- numerically, after instantiation.
Distribution of the Jobs on Different CPU’s

2 possibilities:

- Completely independent jobs (as with the current implementation): the domain is split into intervals $I$, on which $f$ is approximated by a polynomial $P$ and so on. If need be, the code generation can be done on a different machine.

- On some machine (regarded as a server), $f$ is approximated by $P$ on an interval $I$, which is split into $N$ subintervals $J_n$; the coefficients of $P_n$ are computed directly. The corresponding $N$ jobs are distributed on different machines.

Note: the input parameters can be chosen to control the size thus the estimated average execution time of a job (actually the order of magnitude).
LEMA Features That Will Be Needed

- Fast automatic generation of correct (in fact, proved) code, possibly with annotations (for provers, but this is currently limited because of the specific arithmetic).
- Possibility to test various parameters.
- Code instrumentation (was forgotten in most discussions), e.g. to count the number of word additions. For instance, transform the LEMA tree to replace a result $x$ by a pair $(x, c_x)$, and $x + y$ by $(x + y, c_x + c_y + 1)$?
- Checking that the LEMA tree is correct, e.g. that formulas written by the human are correct.