Finding Worst Cases for Correct Rounding of Numerically Regular Math Functions in Fixed Precision

Vincent LEFÈVRE

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

2009-10-20

History

Main facts about the search for worst cases itself:

1996	First tests on exp between 1/2 and 1 in double precision (binary64) using finite differences on degree-2 polynomials (several months on ~ 100 machines).
October 1996	First ideas, which will give my algorithm that computes a lower bound on the distance between a segment and $\mathbb{Z}^2$ (published in June 1997).
1996–1998	First implementations.
January 1999	Last rewrite from scratch, with pen-and-paper proof. The datatypes have not changed since!
October 2002	SLZ (not used in my binary64 tests).
January 2003	Switch from Sparc assembly to C with 64-bit types (algo with divisions) + mpn layer of GMP (hierarchical approximations).
May 2004	Variant of my algorithm implemented.
June 2005	Arith-17 paper (new proof of my algorithm, variants).

History [2]

Reliability:

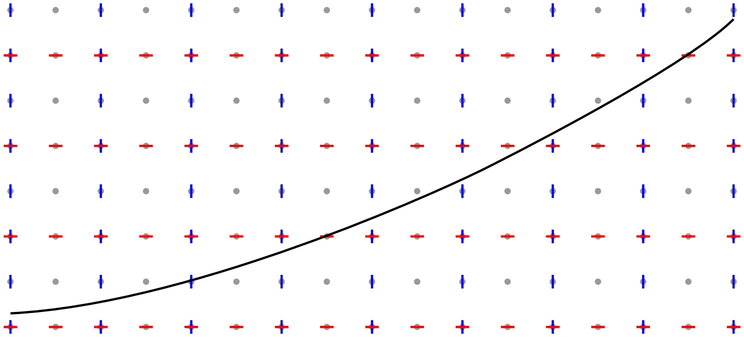
January 2001	Important bug fix (worst cases may be missed).
January 2003	Important bug fix (same bug).
February 2003	Important bug fix (same bug). Testcase.
March 2003	Important bug fix (C version from January).
March 2003	Detect bad GMP installations.
June 2004	More data loss detection (communications with Maple).
February 2005	Important bug fix (in variant from May 2004).
May 2009	Detect some bugs (in my code, the compiler, the Linux kernel, etc.) from the first-step results: check that all results look like worst cases and that the number of potential worst cases is not smaller than some bound (probabilistic hypotheses). To be tested/completed.

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.

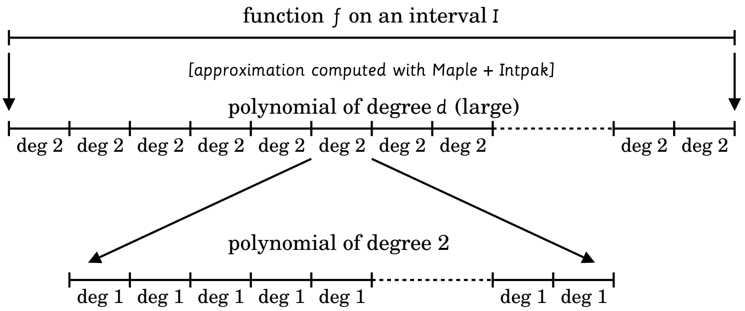
$\rightarrow$ Worst cases for f and the inverse function f^{-1}.



[tmd2009.tex 32597 2009-10-20 13:53:22Z vinc17/vin]

Hierarchical Approximations by Polynomials

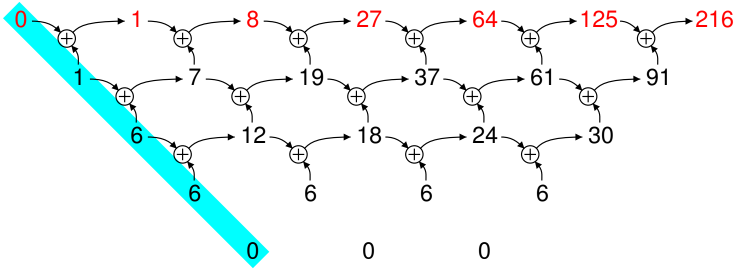
Current implementation (but one could have more than 3 levels):


function f on an interval I
[approximation computed with Maple + Intpak]
polynomial of degree d (large)
deg 2
deg 2
deg 2
deg 2
deg 2
deg 2
deg 2
deg 2
deg 2
deg 2
polynomial of degree 2
deg 1
deg 1
deg 1
deg 1
deg 1
deg 1
deg 1

- 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 in the basis $\left\{1, X, \frac{X(X-1)}{2}, \frac{X(X-1)(X-2)}{3!}, \ldots\right\}$.

Can be done modulo some constant (very useful here).

Hierarchical approximations based on this method (regularly spaced intervals).

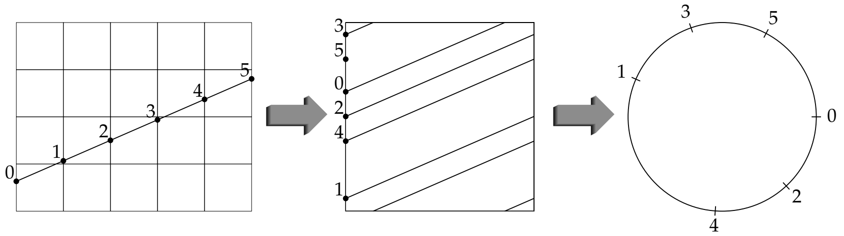
The Problem With a Degree-1 Polynomial

In each interval:

- f is approximated by a polynomial of degree 1 $\longrightarrow$ segment $y = b - ax$.
- Multiplication of the coordinates by powers of 2 $\longrightarrow$ grid $= \mathbb{Z}^2$.

One searches for the values n such that $\{b - n.a\} < d_0$, where a, b and d_0 are real numbers and $n \in [\![0, N-1]\!]$.

$\{x\}$ denotes the positive fractional part of x.



The Problem With a Degree-1 Polynomial [2]

- We chose a positive fractional part instead of centered.

  $\rightarrow$ An upward shift is taken into account in b and d_0.

- If a is rational, then the sequence $0.a$, $1.a$, $2.a$, $3.a$, … (modulo 1) is periodical.

  $\rightarrow$ This makes the theoretical analysis more difficult.

  $\rightarrow$ In the proof, one assumes that a is irrational, or equivalently, that a is a rational number + an arbitrary small irrational number.

  But in the implementation, a is rational.

  $\rightarrow$ Extension to rational numbers by continuity.

  $\rightarrow$ Care has to be taken with the inequality tests since

  - they are not continuous functions;
  - problems can occur when the period has been reached: endless loops…

Notations / Properties of $k.a$ mod 1 ($0 \leq k < n$)

Properties of the two-length configurations $C_n = \{k.a \in \mathbb{R}/\mathbb{Z} : k \in \mathbb{N},\ k < n\}$, to be proved by induction:

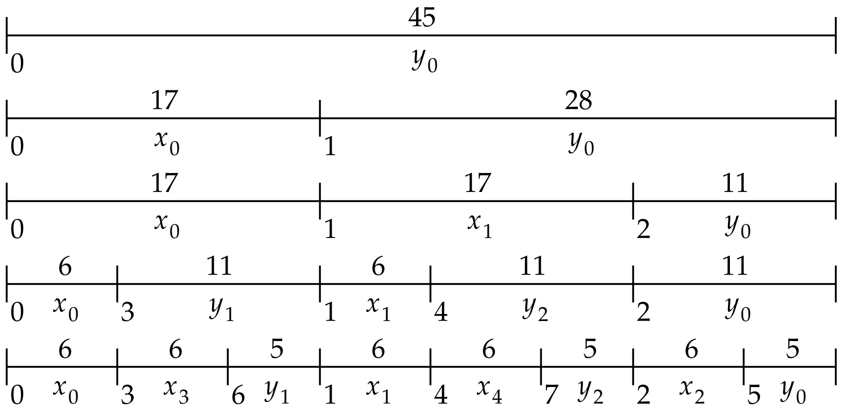
- Intervals $x_0, x_1, \ldots, x_{u-1}$ of length x, where x_0 is the left-most interval and $x_r = x_0 + r.a$ (translation by $r.a$ modulo 1).
- Intervals $y_0, y_1, \ldots, y_{v-1}$ of length y, where y_0 is the right-most interval and $y_r = y_0 + r.a$ (translation by $r.a$ modulo 1).
- Total number of points (or intervals): $n = u + v$ (determined by induction).

In short: **2 primary intervals $\boldsymbol{x_0}$ (left) and $\boldsymbol{y_0}$ (right) + images.**

Initial configuration: $n = 2$, $u = v = 1$.

Example: The First Configurations

with $a = 17/45$.



Note: scaling by 45 on the figure.

From a Configuration to the Next One

The main idea: when adding new points, one of the primary intervals (no inverse image) is affected first, then all its images are affected in the same way.
For instance, see both intervals of length 17 on the figure.

- Since a is irrational, $n.a$ is strictly between two points of smaller indices, one of which, denoted r is non zero.
- Therefore the points of indices $r - 1$ and $n - 1$ (obtained by a translation) are adjacent, and their distance ℓ is either x or y.
  $\rightarrow$ Same distance ℓ between the points of indices r and n.
- Thus the new point n splits an interval of length $h = \max(x, y)$ into two intervals of respective lengths $\ell = \min(x, y)$ and $h - \ell$.
- The length $h - \ell$ is new, therefore the corresponding interval does not have an inverse image (i.e. by adding $-a$).
- Therefore this interval has as a boundary point of index 0.

$\rightarrow$ As a consequence, the point of index n is completely determined.

From a Configuration to the Next One [2]

The other intervals of length h will be split in the same way, one after the other with increasing indices (translations by a).

- Indices of the intervals of length $h - \ell$: these are the indices of the corresponding intervals of length h.
- Indices of the intervals of length ℓ: assume that $\ell = x$ (same reasoning for $\ell = y$); the first interval of length x is obtained by a translation of an old interval of length x (as shown in previous slide), necessarily x_{u-1} (the last one) since the image of x_{i-1} is x_i for all $i < u$. Thus this interval is x_u and we have $x_u = x_0 + u.a$. The next intervals: x_{u+1}, x_{u+2}, etc.

For the algorithm(s):

- We only need to focus on what occurs in the primary intervals.
- At the same time, we track the position of the point b:
  - whether it is in an interval x_k or in an interval y_k;
  - its distance to the left endpoint of the interval.

The Algorithms

Basic algorithm (1997): returns a lower bound d on $\{b - n.a\}$ for $n \in [\![0, N-1]\!]$ (in fact, d is the *exact* distance for $n \in [\![0, N'-1]\!]$, where $N \leq N' < 2N$).

Here: parameters chosen so that $d \geq d_0$ in most intervals, allowing to immediately conclude that there are no worst cases in the interval.

New algorithm (mentioned in 1998): returns the index $n < N$ of the first point such that $\{b - n.a\} < d_0$, otherwise any value $\geq N$ if there are no such points.

Gives the information we need, but uses an additional variable, so that it is slower.

Good replacement for the naive algorithm.

Another improvement: test with a shift (fast!) if it is interesting to replace a sequence of iterations by a single one with a division.

The Algorithms [2]

The necessary data:

- the lengths x and y, and the numbers u and v of these intervals;
- a binary value saying whether the point b is in an interval of length x or y;
- the index r of this interval (new algorithm only);
- the distance d between b and the left endpoint of this interval.

Immediate consequence of the properties:

- the left endpoint of an interval x_r has index r;
- the left endpoint of an interval y_r has index $u + r$.

Subtractive Version of the Algorithms

In red: additional statements for the new algorithm.

```
Initialization: $x = \{a\}$ ; $y = 1 - \{a\}$ ; $d = \{b\}$ ; $u = v = 1$ ; $r = 0$ ;
if ($d < d_0$) return 0
Unconditional loop:
  if ($d < x$)
      while ($x < y$)
          if ($u + v \geq N$) return $N$
          $y = y - x$; $u = u + v$;
      if ($u + v \geq N$) return $N$
      $x = x - y$;
      if ($d \geq x$) $r = r + v$;
      $v = v + u$;
  else
      $d = d - x$;
      if ($d < d_0$) return $r + u$
      while ($y < x$)
          if ($u + v \geq N$) return $N$
          $x = x - y$; $v = v + u$;
      if ($u + v \geq N$) return $N$
      $y = y - x$;
      if ($d < x$) $r = r + u$;
      $u = u + v$;
```

Example of Domain Splitting

Input interval $[1, 2[$ decomposed into $2^{13} = 8192$ sub-intervals I.

For each sub-interval I of size 2^{40}:

- Function f is approximated by a degree-d polynomial.
- Code (C with the mpn layer of GMP) is generated: my algorithm is applied on sub-intervals J of $2^{15} = 32768$ points (64-bit integer arithmetic), and in case of failure, $2^{12} = 4096$ (or $2^{11} = 2048$) points, and if this still fails, the naive method (difference table). Note: this can probably be improved, e.g. larger intervals J (with 128-bit arithmetic?), variant instead of the naive method...
- If GCC is used, the code is compiled using `-fprofile-generate` and tested on the first $2^8 = 256$ sub-intervals (for up to 22% speed-up on Opteron).
- The code is recompiled using `-fprofile-use` and run.

The accuracy (chosen for efficiency) is not sufficient to determine the worst cases. A second filter step is necessary: conventional algorithm (much slower but run on much fewer inputs) on each potential worst case.

Polynomial Degree and Coefficient Size

Examples with a 54-bit significand and splitting into intervals of size 2^{40}.

For some functions and left endpoints of the interval, the table gives the degree of the polynomial and the size (in bits) of the coefficient of highest degree.

function	x_0	degree	size
$\exp x$	1	6	320
$\exp x$	8	7	352
$\exp x$	64	9	416
$\log x$	2	6	320
$\log x$	2^{1000}	6	320
x^4	1	4	224
x^{17}	1	8	384
x^{345}	1	12	544
x^{2065}	1	18	736
x^{2065}	$2 - \varepsilon$	15	640