Worst Cases for $\sin(\text{BIG})$

Paul Zimmermann

(joint work with G. Hanrot, V. Lefèvre, D. Stehlé)

IEEE 754 Revision Committee, Apple Cupertino, 2nd November 2006

Open Problems (last year talk . . .)

- Compute WC for decimal formats: not a big deal!

Open Problems (last year talk . . .)

- Compute WC for decimal formats: not a big deal!
- trig(BIG) is still expensive: $\sin(\mathrm{MAXDBL})$ requires an argument reduction on $1024 + 128 = 1152$ bits!

$\Longrightarrow$ error bound $(\frac{1}{2} + \epsilon)$ ulp

$\Longrightarrow$ call arbitrary precision library?

Open Problems (last year talk . . .)

- Compute WC for decimal formats: not a big deal!

- trig(BIG) is still expensive: $\sin(\mathrm{MAXDBL})$ requires an argument reduction on $1024 + 128 = 1152$ bits!

$\Longrightarrow$ error bound $(\frac{1}{2} + \epsilon)$ ulp

$\Longrightarrow$ call arbitrary precision library?

- Two-variable functions like $x^y, \mathrm{atan}(y/x)$ are still out-of-reach for exhaustive worst-case search: 2^{128} cases!

$\Longrightarrow$ 1st step is still CR

$\Longrightarrow$ add rounding test in 2nd step (raise exception?)

$\Longrightarrow$ or call arbitrary precision library. . .

The Problem (1/2)

Consider $x_i = 1/2 + i \cdot \mathrm{ulp}(1/2)$:

$\sin x_0 \approx 0.4794255386042030\textcolor{red}{0}$

$\sin x_1 \approx 0.4794255386042031\textcolor{red}{0}$

$\sin x_2 \approx 0.4794255386042032\textcolor{red}{0}$

The Problem (1/2)

Consider $x_i = 1/2 + i \cdot \mathrm{ulp}(1/2)$:

$\sin x_0 \approx 0.47942553860420300$

$\sin x_1 \approx 0.47942553860420310$

$\sin x_2 \approx 0.47942553860420320$

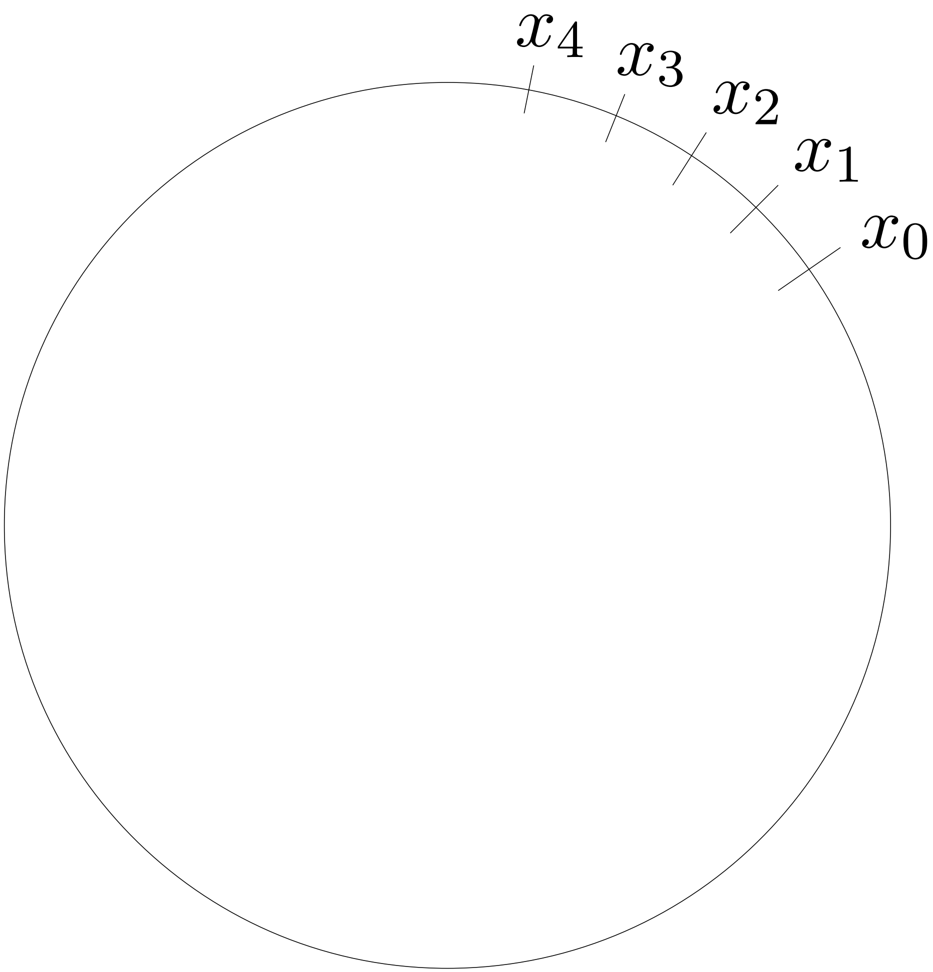
$$f(x_i) \approx a + i \cdot b + \cdots$$

where $a = \sin x_0 \approx 0.47942553860420300$,

and $b = 2^{-53} \cdot \cos x_0 \approx 0.97431236622022320 \cdot 10^{-16}$.

A graphical view

Modulo the period Π:



The Problem (2/2)

Consider $x_i = 2^{1023} + i \cdot \mathrm{ulp}(2^{1023})$:

The Problem (2/2)

Consider $x_i = 2^{1023} + i \cdot \mathrm{ulp}(2^{1023})$:

$\sin x_0 \approx 0.56312777985088401$

The Problem (2/2)

Consider $x_i = 2^{1023} + i \cdot \mathrm{ulp}(2^{1023})$:

$$\sin x_0 \approx 0.56312777985088401$$

$$\sin x_1 \approx -0.97617177188664794$$

The Problem (2/2)

Consider $x_i = 2^{1023} + i \cdot \mathrm{ulp}(2^{1023})$:

$\sin x_0 \approx 0.56312777985088401$

$\sin x_1 \approx -0.97617177188664794$

$\sin x_2 \approx 0.15999647656281522$

The Problem (2/2)

Consider $x_i = 2^{1023} + i \cdot \mathrm{ulp}(2^{1023})$:

$\sin x_0 \approx 0.5631277798508840$

$\sin x_1 \approx -0.9761717718866479$

$\sin x_2 \approx 0.1599964765628152$

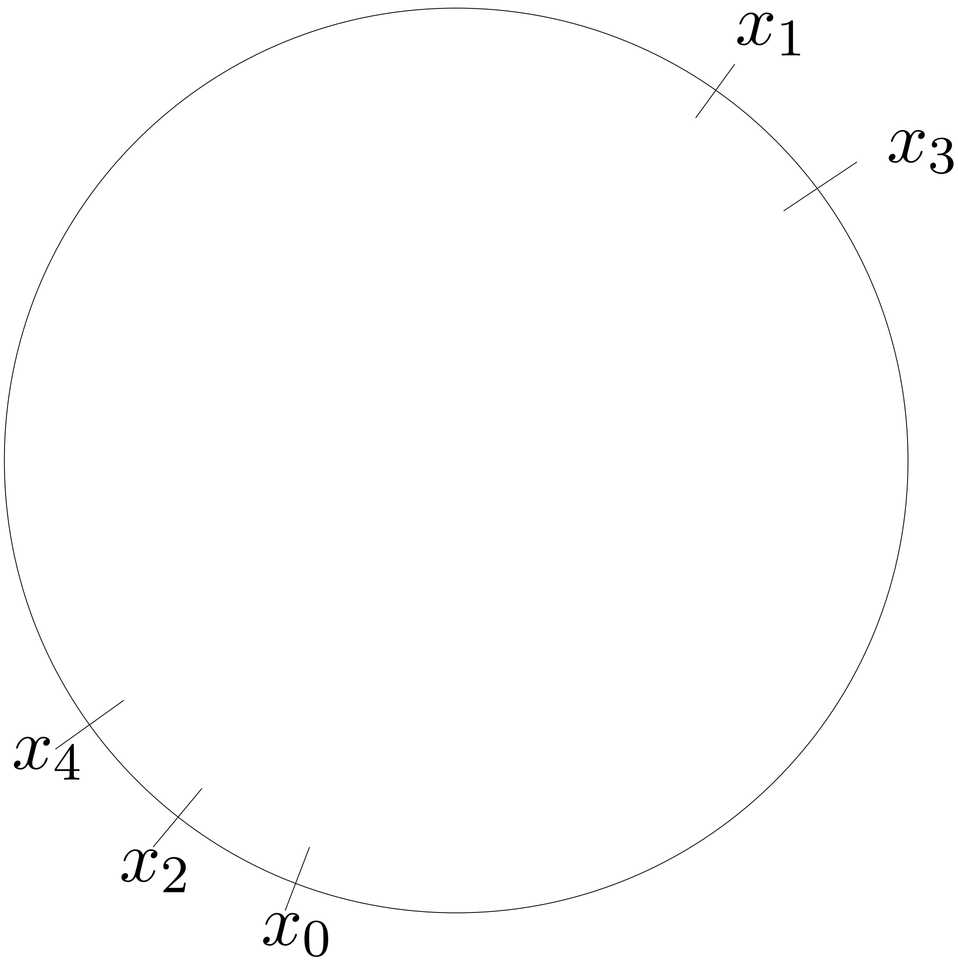
$$f(x_i) \approx a + i \cdot b + \cdots$$

where $a = \sin x_0 \approx 0.5631277798508840$,

and $b = 2^{971} \cdot \cos x_0 \approx -0.1649302226506456 \cdot 10^{293}$.

A graphical view

Modulo the period Π:



The main result

Theorem 1 *Given a periodic* $\mathcal{C}^\infty$ *function* f*, a precision* p*, an exponent* e *and a bad case bound* $\varepsilon = 2^{-p+O(1)}$*, all* p*-bit numbers in* $[2^{e-1}, 2^e]$ *such that*

$$|2^p \cdot \mathrm{mantissa}(f(x)) \,\mathrm{cmod}\, 1| \leq \varepsilon,$$

can be found in heuristic time $O(2^{(7-2\sqrt{10})p})$*, after a precomputation on* $e + O(p)$ *bits.*

Remark 1: $7 - 2\sqrt{10} \approx 0.675$.

Remark 2: the complexity does not depend on e!

Lefèvre and Muller's method

References: Lefèvre PhD (2000), Lefèvre/Muller Arith'15 paper *Worst Cases for Correct Rounding of the Elementary Functions in Double Precision*

Approximate $f(x_0 + t \cdot \mathrm{ulp}(x_0))$ by a linear polynomial

$$p(t) := a + b \cdot t$$

on small intervals (a, b real numbers, $|t| < T$ integer)

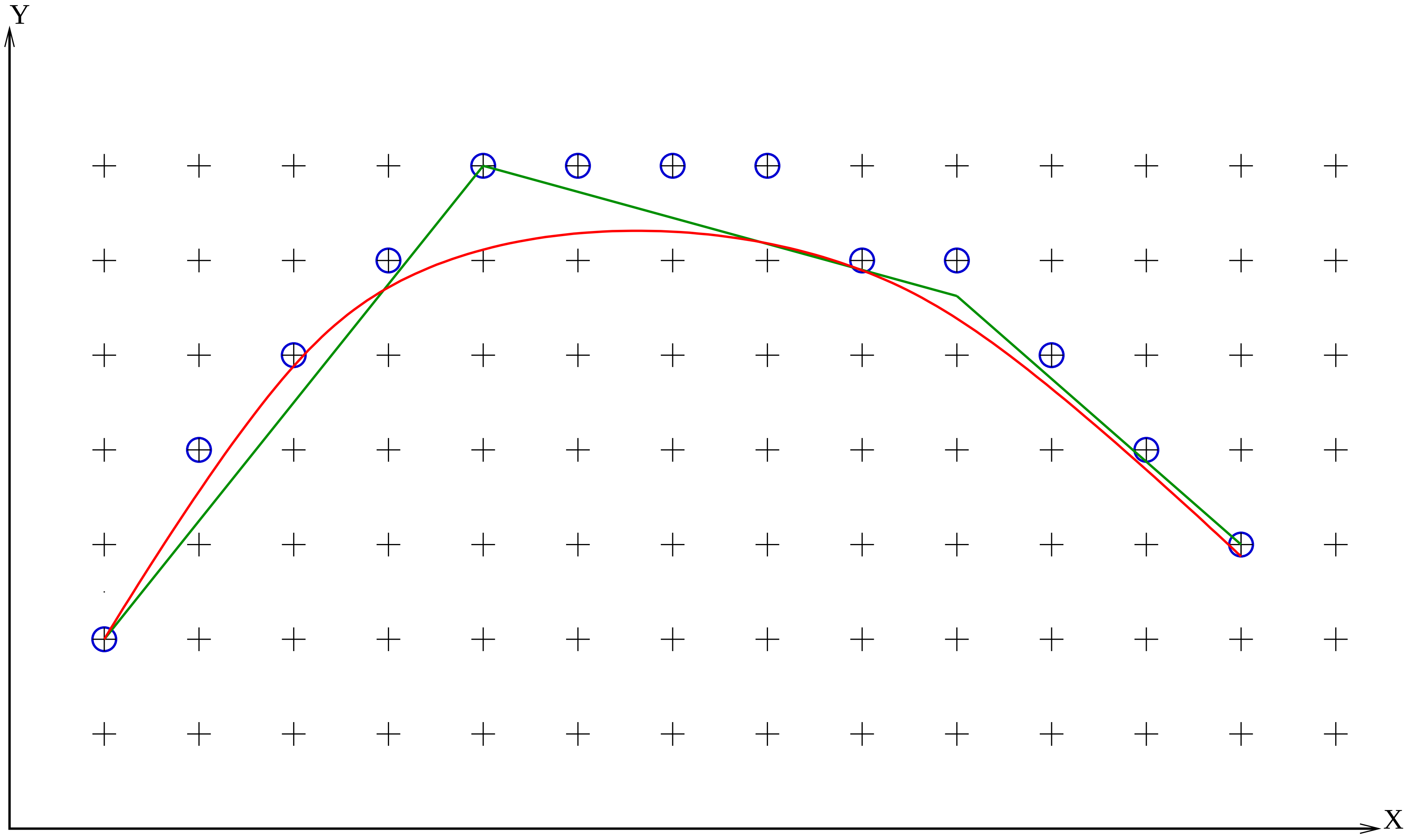
On each interval, find the values of t such that

$$p(t) \,\mathrm{cmod}\, \mathrm{ulp}(x_0)$$

is small with a variant of the extended gcd algorithm.

Largest possible value of T is $2^{p/3}$ for precision p: complexity $2^{2p/3}$.

Expensive but feasible in double precision.


Y
X

The SLZ algorithm (1/2)

References: Stehlé, Lefèvre, Z., *Worst Cases and Lattice Reduction*, Arith'16 (2003) and *Searching Worst Cases of a One-Variable Function Using Lattice Reduction*, IEEE TC (2005).

Approximate $f(x_0 + t \cdot \mathrm{ulp}(x_0))$ by a degree-d polynomial

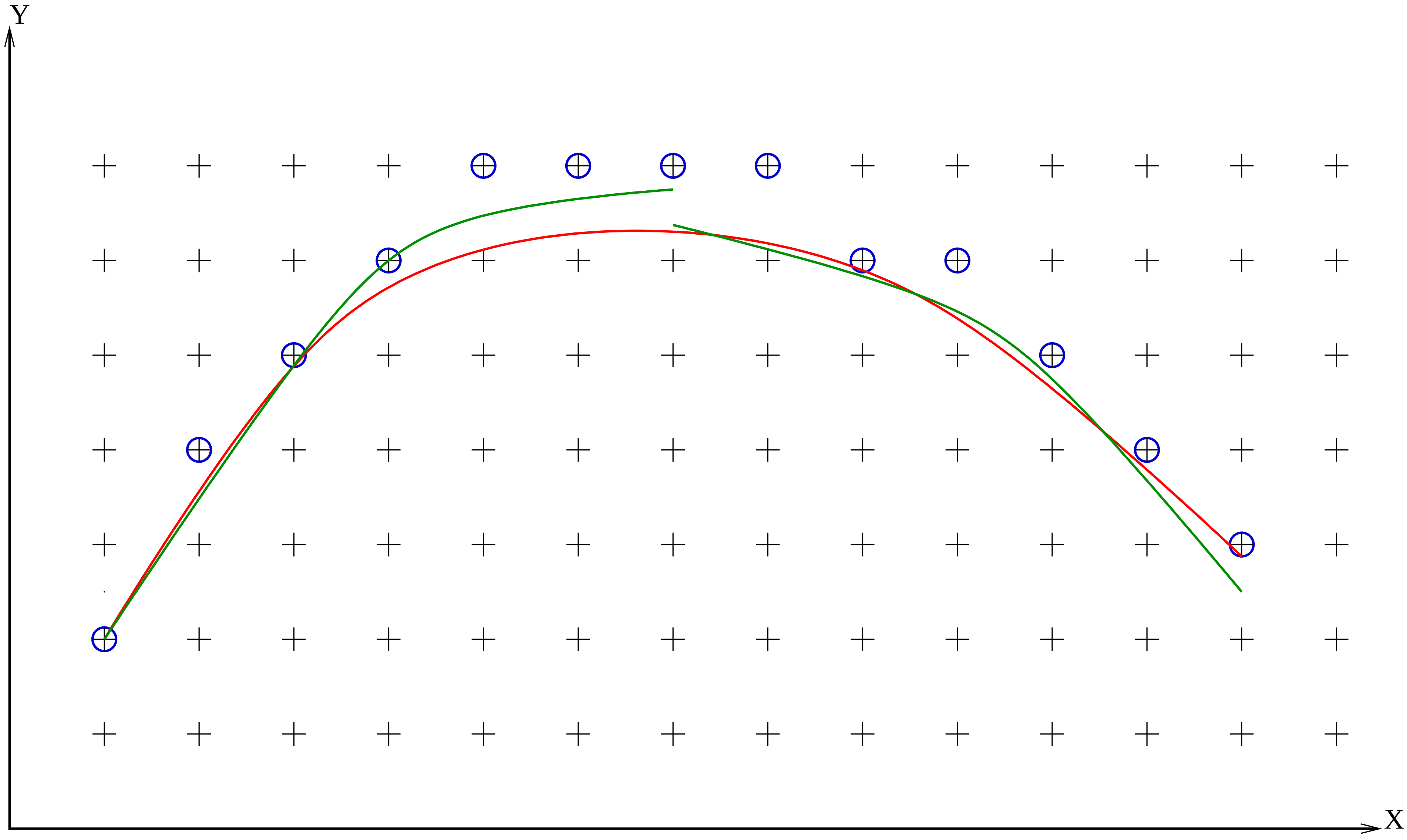
$$p(t) = a_0 + a_1 t + \cdots + a_d t^d$$

on small intervals (a_i real numbers, $|t| < T$ integer)

On each interval, find the values of t such that

$$p(t) \operatorname{cmod} \mathrm{ulp}(a_0)$$

is small using a variant of Coppersmith's algorithm to find the small roots of a modular equation, which uses lattice reduction.


Y
X

The SLZ algorithm (2/2)

Largest possible value of T is $2^{3p/7}$ for precision p: complexity $2^{4p/7}$.

Comparable to Lefèvre's method in double precision, slightly faster in double-extended precision (complete check for 2^x).

In practice: $d = 2$ or $d = 3$.

Complete check of $\exp(x)$ for `decimal64` (with V. Lefèvre and D. Stehlé).

Worst case for $|x| \geq 3 \cdot 10^{-11}$:

$$\exp(\underbrace{9.407822313572878}_{16} \cdot 10^{-2}) = \underbrace{1.098645682066338}_{16}\,5\underbrace{0000000000000000}_{16}278$$

The search for bad cases can be written:

$$|\lambda \cdot f(x_0 + \mu t) \operatorname{cmod} 1| \leq \varepsilon \tag{1}$$

with x_0 a p-bit number, $t \in [0, T]$ integer.

The search for bad cases can be written:

$$|\lambda \cdot f(x_0 + \mu t) \operatorname{cmod} 1| \leq \varepsilon \tag{2}$$

with x_0 a p-bit number, $t \in [0, T]$ integer.

Classical setting: $1/2 \leq x, f(x) < 1$

$$\lambda = 2^p, \mu = 2^{-p}$$

The search for bad cases can be written:

$$|\lambda \cdot f(x_0 + \mu t) \operatorname{cmod} 1| \leq \varepsilon \tag{3}$$

with x_0 a p-bit number, $t \in [0, T]$ integer.

Classical setting: $1/2 \leq x, f(x) < 1$

$$\lambda = 2^p, \mu = 2^{-p}$$

$\sin(\text{BIG})$ setting: $2^{e-1} \leq x < 2^e$

$$\lambda = 2^p, \mu = 2^{e-p}$$

The problem is that μ is large.

First Solution

$$|\lambda \cdot f(x_0 + \mu t) \operatorname{cmod} 1| \leq \varepsilon$$

Reduce the argument!

Write $x_0 \equiv x_0' \bmod \Pi$, $\mu \equiv \mu' \bmod \Pi$ with $0 < x_0', \mu' < \Pi$

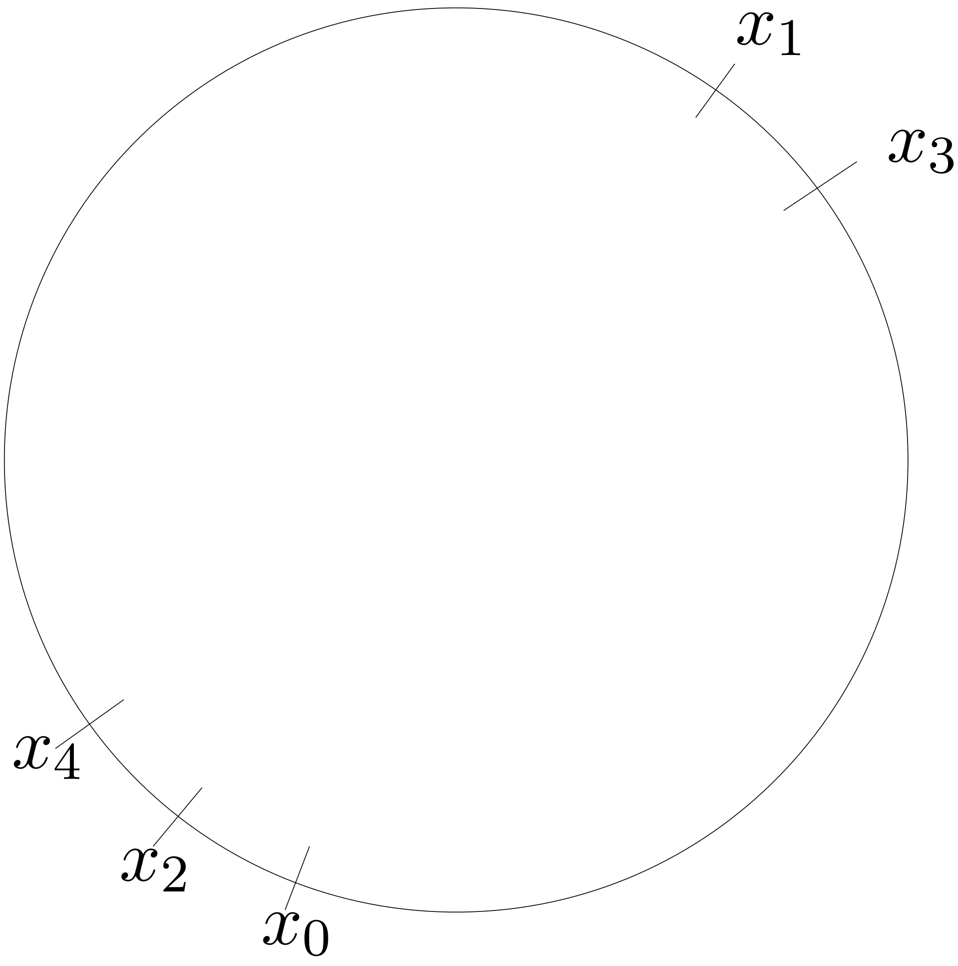
$$|\lambda \cdot f(x_0' + \mu' t) \operatorname{cmod} 1| \leq \varepsilon$$

We are back to Eq. (3), with x_0' and μ' real now.

The big exponent of x_0 does not play any role any more!

But μ' is still not small ...

A graphical view



The main idea

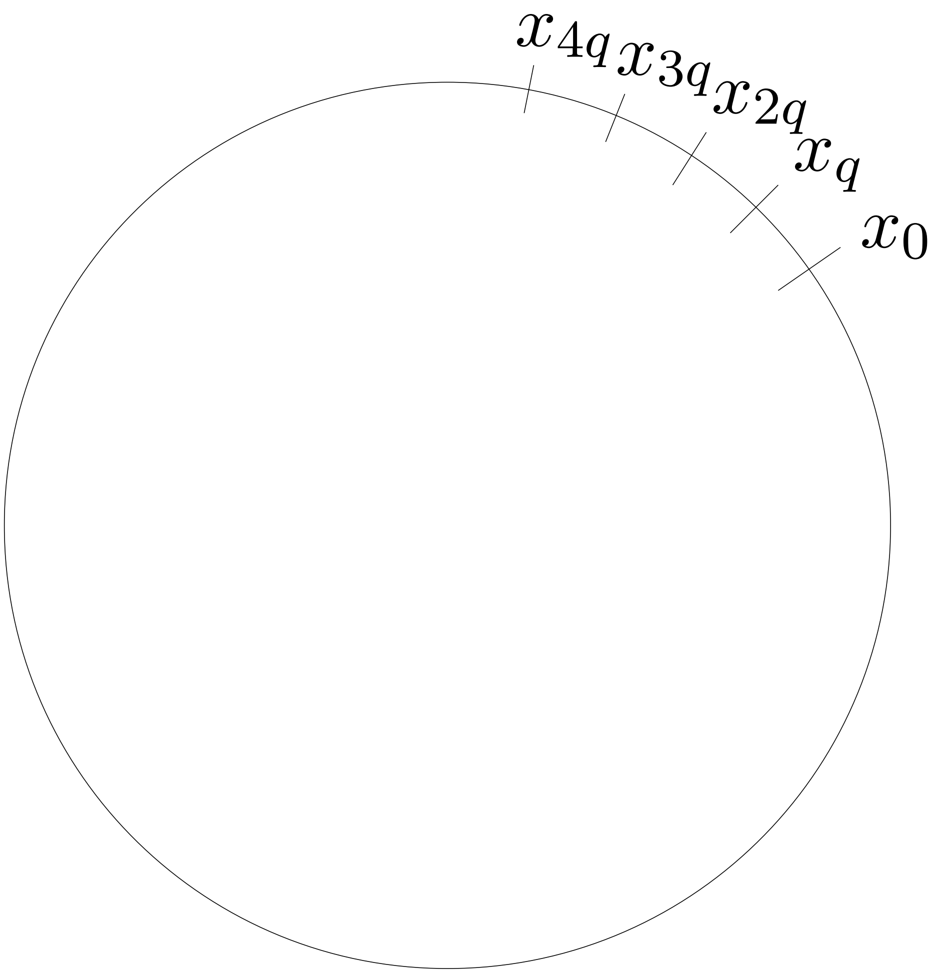
$$|\lambda \cdot f(x_0 + \mu t) \operatorname{cmod} 1| \leq \varepsilon$$

with real numbers $0 < x_0, \mu < \Pi$.

Find an integer q such that $\tau := q\mu \operatorname{cmod} \Pi$ is small.

We are back to the classical case, with $0 \leq t \leq T/q$.

A graphical view



The algorithm

0. Argument reduce x_0 and μ

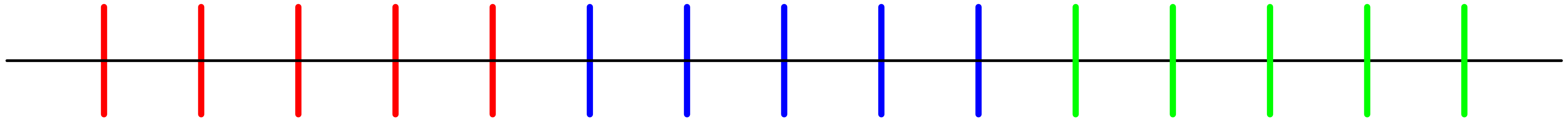
1. Find q such that $\tau := q\mu \operatorname{cmod} \Pi$ is small

2. For $0 \leq i < q$ where $x_i = x_0 + i\mu$ do

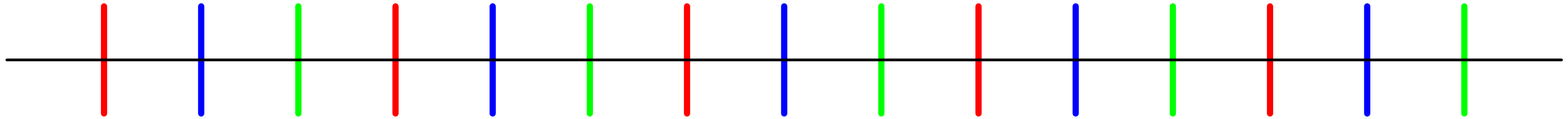
   Run Lefèvre's algorithm or SLZ on $x_i + t\tau$, $t < T/q$.

A graphical view

Classical search:



New algorithm:



How to find q?

Take a convergent $\frac{p}{q}$ from $\frac{\mu}{\Pi}$ ($\mu = \mathrm{ulp}(x_0)$).

$$\left| \frac{p}{q} - \frac{\mu}{\Pi} \right| \leq \frac{1}{q^2}$$

thus $|q\mu \,\mathrm{cmod}\, \Pi| \leq \Pi/q$.

How to find q?

Take a convergent $\frac{p}{q}$ from $\frac{\mu}{\Pi}$ ($\mu = \mathrm{ulp}(x_0)$).

$$\left|\frac{p}{q} - \frac{\mu}{\Pi}\right| \leq \frac{1}{q^2}$$

thus $|q\mu \,\mathrm{cmod}\, \Pi| \leq \Pi/q$.

Example for $\mu = 2^{971}$, $\Pi = 2\pi$:

$q = 15106909301$, $\tau := q\mu \,\mathrm{cmod}\, (2\pi) \approx 0.441 \cdot 10^{-12}$

Numerical Results

Implemented the algorithm in BACSEL, using FPLLL, MPFR and GMP.

Numerical Results

Implemented the algorithm in BACSEL, using FPLLL, MPFR and GMP.

Considered $\sin x$ on $[2^{1023}, 2^{1024})$ (largest `binary64` exponent).

Numerical Results

Implemented the algorithm in BACSEL, using FPLLL, MPFR and GMP.

Considered $\sin x$ on $[2^{1023}, 2^{1024})$ (largest `binary64` exponent).

Used $q = 15106909301$, with degree $d = 3$.

Numerical Results

Implemented the algorithm in BACSEL, using FPLLL, MPFR and GMP.

Considered $\sin x$ on $[2^{1023}, 2^{1024})$ (largest `binary64` exponent).

Used $q = 15106909301$, with degree $d = 3$.

Each arithmetic progression $x_i + t \cdot \tau$ has ≤ 298116 elements.

Numerical Results

Implemented the algorithm in BACSEL, using FPLLL, MPFR and GMP.

Considered $\sin x$ on $[2^{1023}, 2^{1024})$ (largest `binary64` exponent).

Used $q = 15106909301$, with degree $d = 3$.

Each arithmetic progression $x_i + t \cdot \tau$ has ≤ 298116 elements.

$$\sin(4621478864517314 \cdot 2^{971}) =$$

$$-0.0\,\underbrace{10010110001110101110010000111100111100010000000100000}_{53}\,1^{43}\,0001\ldots$$

Numerical Results

Implemented the algorithm in BACSEL, using FPLLL, MPFR and GMP.

Considered $\sin x$ on $[2^{1023}, 2^{1024})$ (largest `binary64` exponent).

Used $q = 15106909301$, with degree $d = 3$.

Each arithmetic progression $x_i + t \cdot \tau$ has ≤ 298116 elements.

$$\sin(4621478864517314 \cdot 2^{971}) =$$

$$-0.0\,\underbrace{10010110001110101110010000111100111100010000000100000}_{53}\,1^{43}\,0001\ldots$$

$$\sin(5501214608935005 \cdot 2^{971}) =$$

$$0.00\,\underbrace{10011000110011100101110100111100011011010010001111111}_{53}\,0^{45}\,1011\ldots$$

Time Estimates

Machine used: 2.4 Ghz Opteron under Linux.

Time Estimates

Machine used: 2.4 Ghz Opteron under Linux.

Naive implementation: after a single argument reduction, about 6 seconds to check one million values, 850 years to check the whole binade.

Time Estimates

Machine used: 2.4 Ghz Opteron under Linux.

Naive implementation: after a single argument reduction, about 6 seconds to check one million values, 850 years to check the whole binade.

Our algorithm: 0.008s to check one arithmetic progression (298116 values), about 4 years to check the whole binade.

Conclusion and Open Questions

A very first step to find bad cases of $\sin(\text{BIG})$.

Conclusion and Open Questions

A very first step to find bad cases of $\sin(\text{BIG})$.

The algorithm and/or the implementation can probably be improved.

Conclusion and Open Questions

A very first step to find bad cases of $\sin(\text{BIG})$.

The algorithm and/or the implementation can probably be improved.

$\sin$, $\cos$, $\tan$ for the whole `binary64` format are now within reach (at most 4000 cpu years each)

Conclusion and Open Questions

A very first step to find bad cases of $\sin(\text{BIG})$.

The algorithm and/or the implementation can probably be improved.

$\sin$, $\cos$, $\tan$ for the whole `binary64` format are now within reach (at most 4000 cpu years each)

Moore's law:

$$w = \int_0^T 2^{2t/3} dt$$

$w = 1000$ gives $T \approx 13.3$ years

Conclusion and Open Questions

A very first step to find bad cases of $\sin(\text{BIG})$.

The algorithm and/or the implementation can probably be improved.

$\sin$, $\cos$, $\tan$ for the whole `binary64` format are now within reach (at most 4000 cpu years each)

Moore's law:

$$w = \int_0^T 2^{2t/3} dt$$

$w = 1000$ gives $T \approx 13.3$ years

If we start now, we'll be ready well before 754R'...