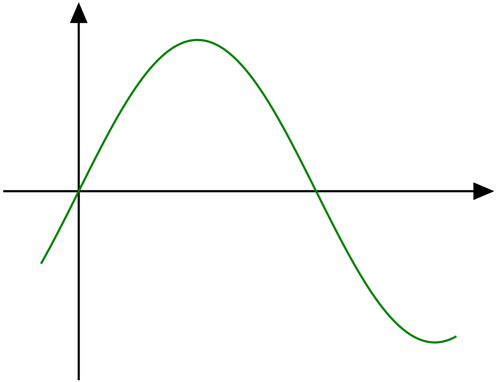
REPRISES F2F meeting
CC IN2P3, Lyon, June 21, 2018

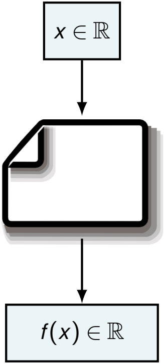
Exact Lookup Tables for the Evaluation of Trigonometric and Hyperbolic Functions

Hugues de Lassus Saint-Geniès David Defour Guillaume Revy

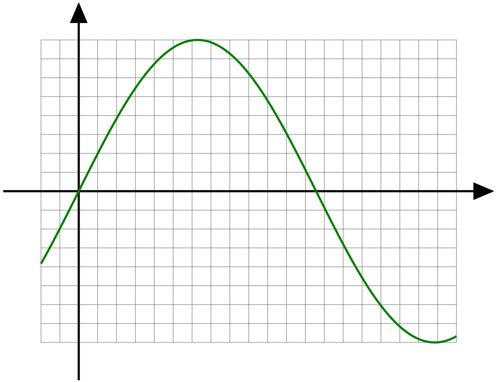
Univ. Perpignan Via Domitia, DALI, Perpignan
LIRMM, Univ. Montpellier, CNRS (UMR 5506), Montpellier

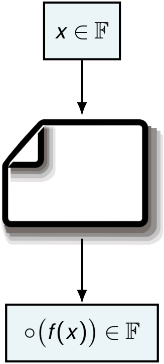
From a function to its implementation





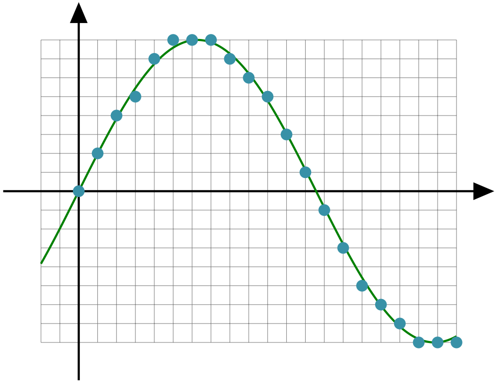
From a function to its implementation

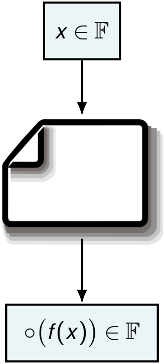




- Input and output are represented using finite precision arithmetic

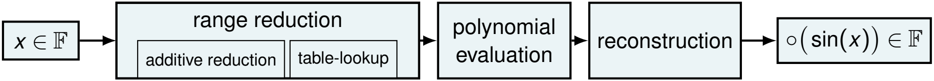
From a function to its implementation

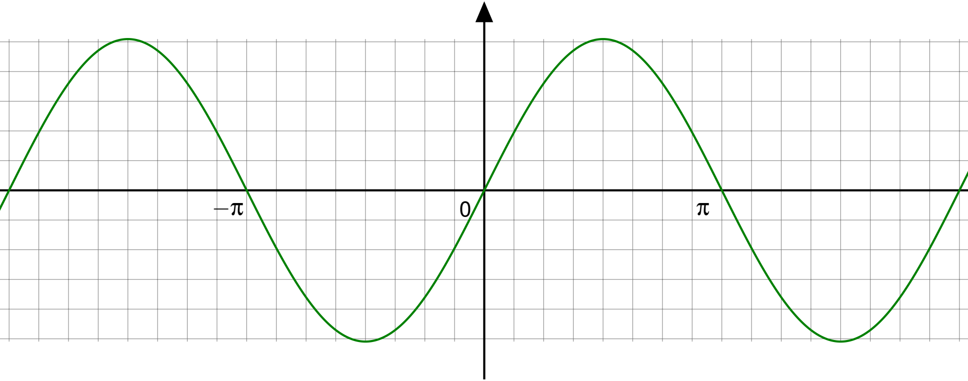




- Input and output are represented using finite precision arithmetic
  - the computed value is an approximation of the function $f(x)$

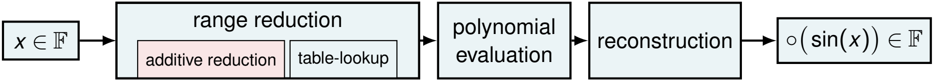
Example of the sine function

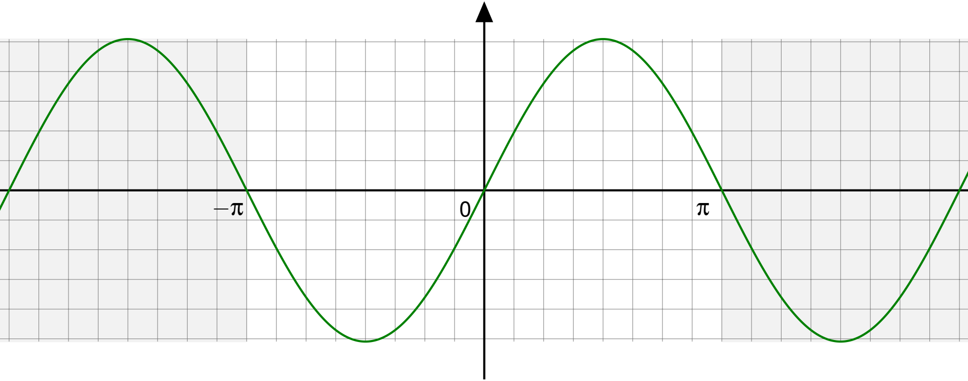




- Use of trigonometric symmetries and periodicities

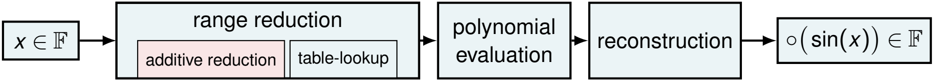
Example of the sine function

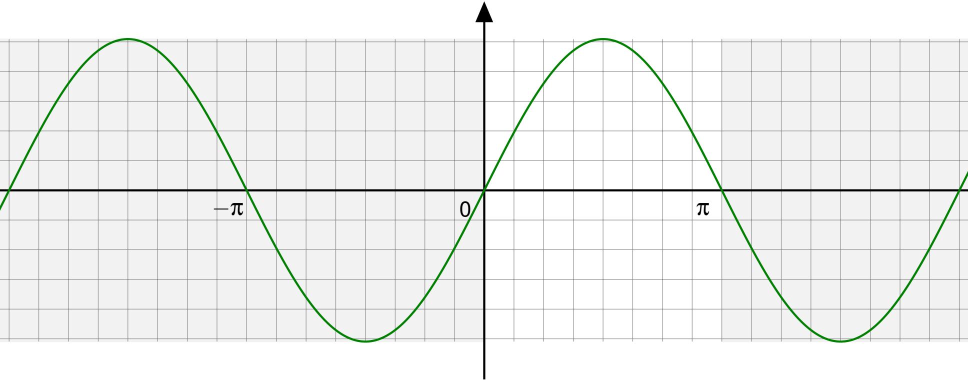




- Use of trigonometric symmetries and periodicities

Example of the sine function

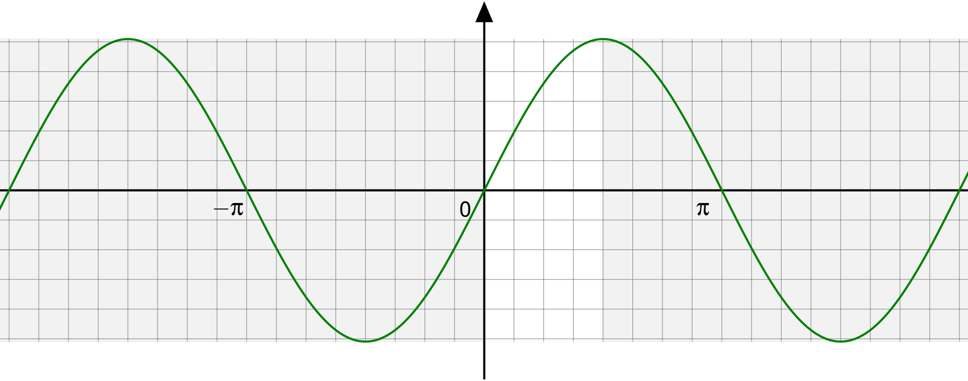




- Use of trigonometric symmetries and periodicities

Example of the sine function

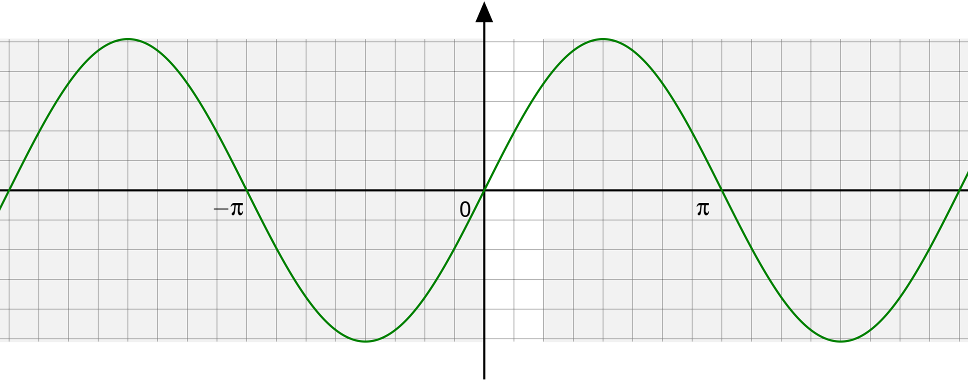
$x \in \mathbb{F}$ → range reduction (additive reduction | table-lookup) → polynomial evaluation → reconstruction → $\circ(\sin(x)) \in \mathbb{F}$



- Use of trigonometric symmetries and periodicities

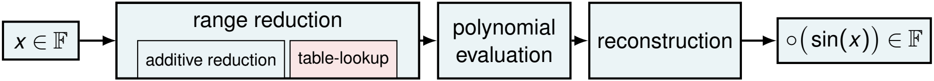
Example of the sine function

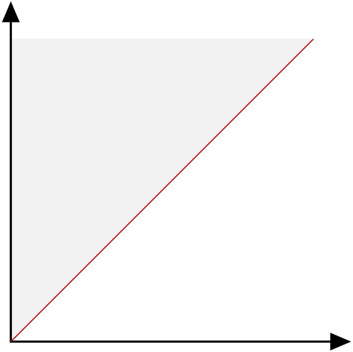
$x \in \mathbb{F}$ → range reduction [additive reduction | table-lookup] → polynomial evaluation → reconstruction → $\circ(\sin(x)) \in \mathbb{F}$



- Use of trigonometric symmetries and periodicities

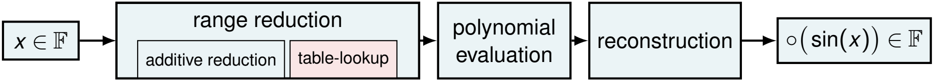
Example of the sine function (Tang's method)

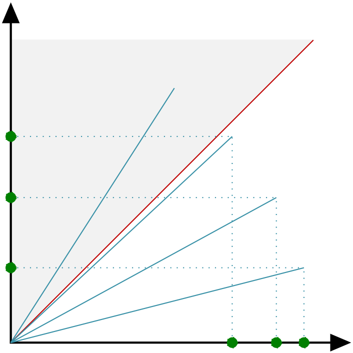




- After first range reduction:
  $x \in \mathbb{F} \mapsto x^* \in [0, \pi/4]$

Example of the sine function (Tang's method)

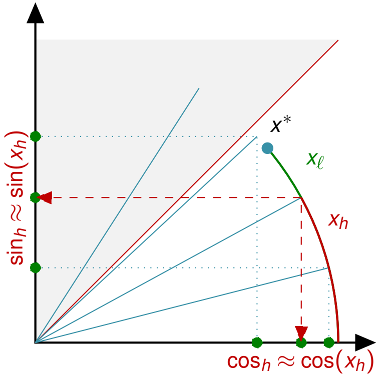




- After first range reduction:
  $x \in \mathbb{F} \mapsto x^* \in [0, \pi/4]$
- Tabulate $\cos(\cdot)/\sin(\cdot)$ of regularly spaced angles $x_h(i) = i \cdot 2^{-p}$

Example of the sine function (Tang's method)

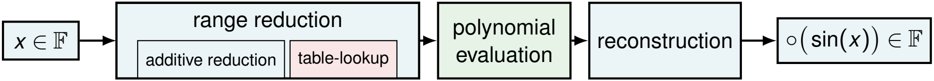
$x \in \mathbb{F}$ → range reduction [additive reduction | table-lookup] → polynomial evaluation → reconstruction → $\circ(\sin(x)) \in \mathbb{F}$

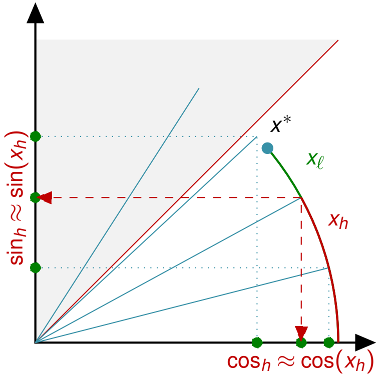


- After first range reduction:
  $x \in \mathbb{F} \mapsto x^* \in [0, \pi/4]$
- Tabulate $\cos(\cdot)/\sin(\cdot)$ of regularly spaced angles $x_h(i) = i \cdot 2^{-p}$
- Split x^* into $(x_h, x_\ell) \rightsquigarrow x^* = x_h + x_\ell$

$$x^* = \underbrace{x.xx\cdots xx}_{p \text{ bits: } x_h}\underbrace{xx\cdots xx}_{x_\ell}$$

Example of the sine function (Tang's method)



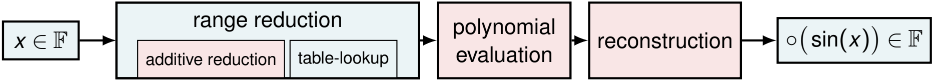


- After first range reduction:
  $x \in \mathbb{F} \mapsto x^* \in [0, \pi/4]$
- Tabulate $\cos(\cdot)/\sin(\cdot)$ of regularly spaced angles $x_h(i) = i \cdot 2^{-p}$
- Split x^* into $(x_h, x_\ell) \rightsquigarrow x^* = x_h + x_\ell$

$$x^* = \underbrace{x.xx\cdots xx}_{p \text{ bits: } x_h}\underbrace{xx\cdots xx}_{x_\ell}$$

- Retrieve sine result using: $\sin(x^*) \approx \sin_h \otimes \cos(x_\ell) \oplus \cos_h \otimes \sin(x_\ell)$

Sources of error in the classical method

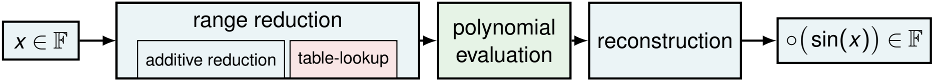


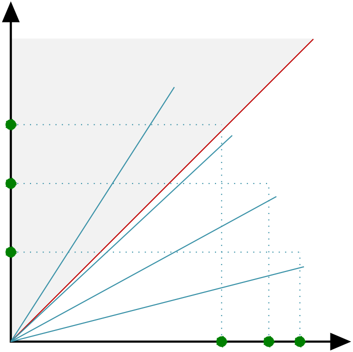
In this process, errors appear:

- in x^*, because of the additive range reduction
- in $\sin_h$ and $\cos_h$, which are rounded approximations
- in $\sin(x_\ell)$ and $\cos(x_\ell)$, computed by polynomial evaluations
- during the reconstruction process

Drawback: additional bits in computation steps for accuracy purposes

Reducing the error on the tabulated values (Gal's method)

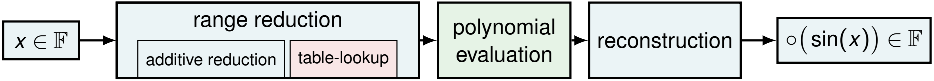


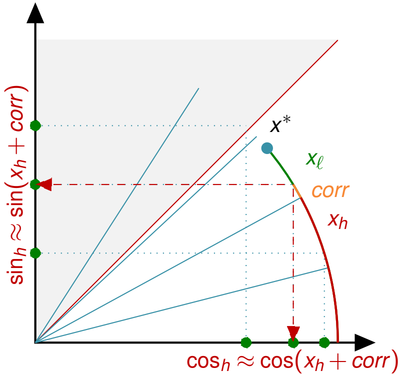


- Add small perturbations to the table inputs: correction term (*corr*)
- Make $\sin_h$ and $\cos_h$ closer to machine numbers (10 up to 21 bits)
- Values $\sin(x_h)$ and $\cos(x_h)$ look like

$$x.xx \cdots xx \underbrace{00000 \cdots 00000}_{\text{10 up to 21 bits}} 10101 \cdots$$

Reducing the error on the tabulated values (Gal's method)



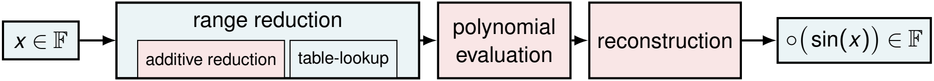


- Add small perturbations to the table inputs: correction term (*corr*)
- Make $\sin_h$ and $\cos_h$ closer to machine numbers (10 up to 21 bits)
- Values $\sin(x_h)$ and $\cos(x_h)$ look like

$$x.xx\cdots xx\underbrace{00000\cdots00000}_{\text{10 up to 21 bits}}10101\cdots$$

- Retrieve sine result using: $\sin(x^*) \approx \sin_h \otimes \cos(x_\ell - corr) \oplus \cos_h \otimes \sin(x_\ell - corr)$

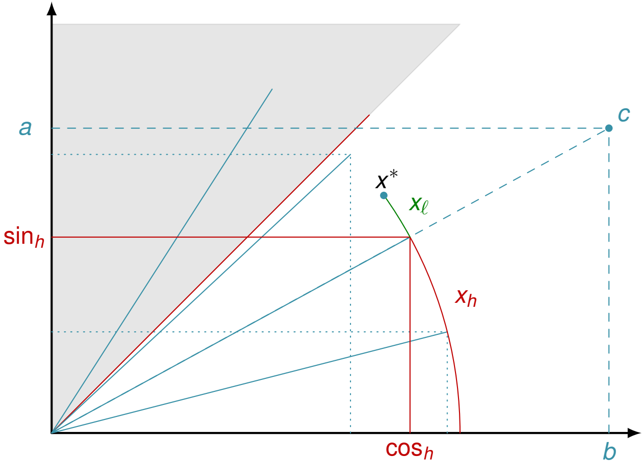
Main objective: build exact lookup tables



Key idea: remove the error on tabulated values

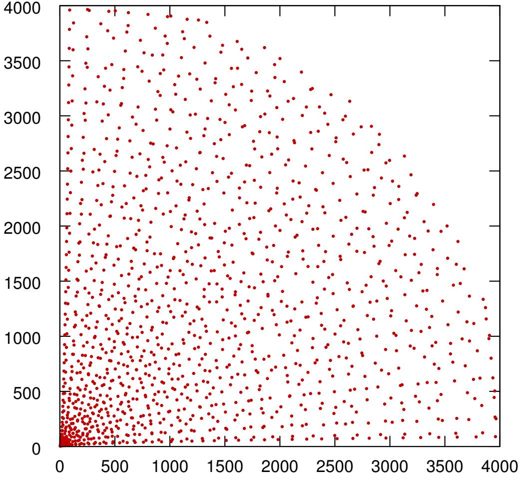
- tables should store in exact values on machine numbers
- it saves bits
- it concentrates the error on subsequent steps
- it makes the reconstruction step easier

What if we had rational numbers to tabulate?



$$\sin(x^*) \approx (a \otimes \cos(x_\ell) \oplus b \otimes \sin(x_\ell)) \oslash c$$

We've been having them since Euclid ($\approx$ 300 BC)



Each dot represents a primitive Pythagorean triple (PPT):

$$\exists(\mathbf{a},\mathbf{b},\mathbf{c}) \in \mathbb{N}^{\mathbf{3}} \text{ coprime } \mid$$

$$a^2 + b^2 = c^2$$

$$\Rightarrow \exists x \in \left[0, \frac{\pi}{2}\right] \mid$$

$$\sin(\mathbf{x}) = \frac{\mathbf{a}}{\mathbf{c}} \quad \text{and} \quad \cos(\mathbf{x}) = \frac{\mathbf{b}}{\mathbf{c}}$$

Primitive Pythagorean triples (PPT) generation

One easy way: ternary-trees with root $(3, 4, 5)$

- 3 linear relationships $\rightarrow$ 3 children/node, e.g:

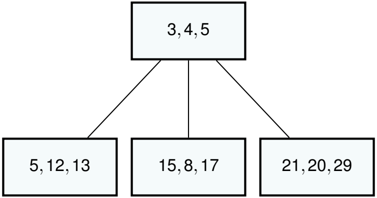
$$\begin{pmatrix} 1 & -2 & 2 \\ 2 & -1 & 2 \\ 2 & -2 & 3 \end{pmatrix}, \begin{pmatrix} -1 & 2 & 2 \\ 2 & -1 & 2 \\ -2 & 2 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 3 \end{pmatrix}$$

Primitive Pythagorean triples (PPT) generation

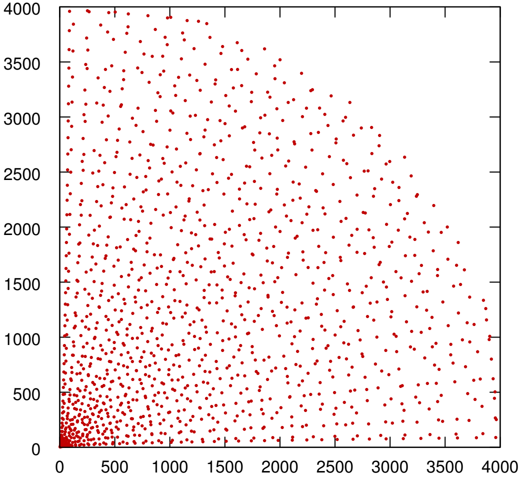
One easy way: ternary-trees with root $(3, 4, 5)$

- 3 linear relationships $\rightarrow$ 3 children/node, e.g:

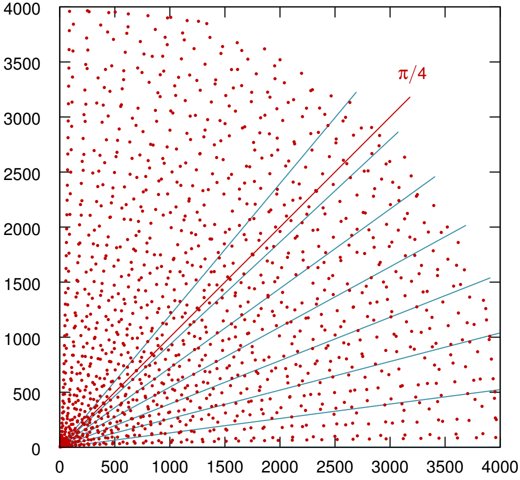
$$\begin{pmatrix} 1 & -2 & 2 \\ 2 & -1 & 2 \\ 2 & -2 & 3 \end{pmatrix}, \begin{pmatrix} -1 & 2 & 2 \\ 2 & -1 & 2 \\ -2 & 2 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 3 \end{pmatrix}$$



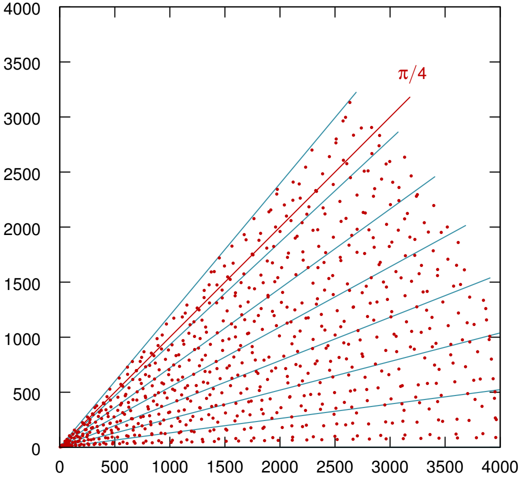
Primitive Pythagorean triples with $c \leq 2^{12}$



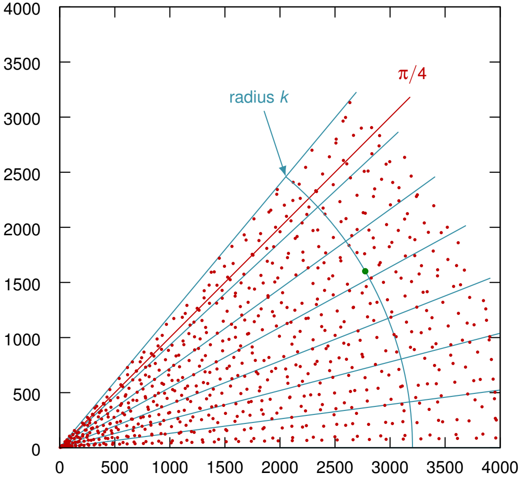
Primitive Pythagorean triples with $c \leq 2^{12}$



Primitive Pythagorean triples with $c \leq 2^{12}$

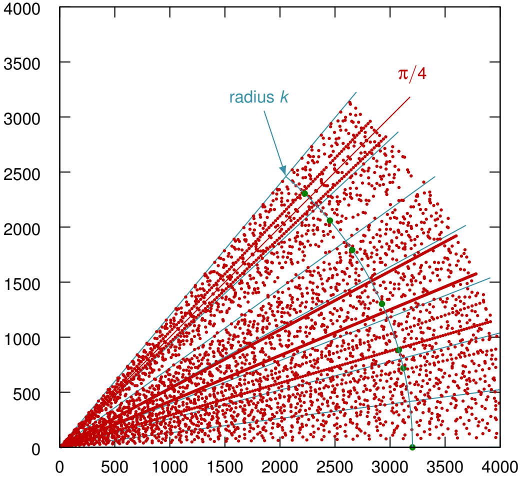


What are "good" Pythagorean triples?



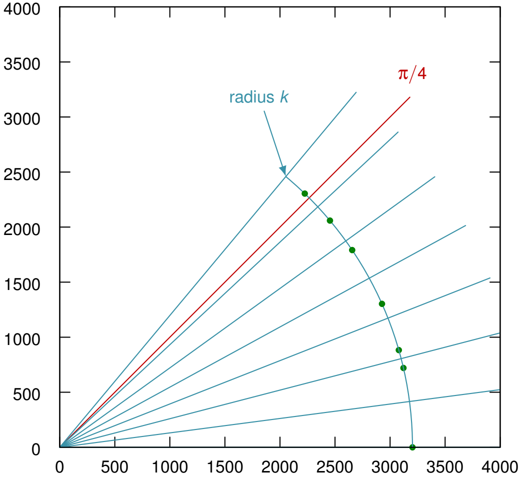
$$\sin(x^*) \approx (a_i \otimes \cos(x_\ell) \oplus b_i \otimes \sin(x_\ell)) \oslash c_i$$

What are “good” Pythagorean triples?



$$\sin(x^*) \approx \left(\frac{a_i \cdot k}{c_i} \otimes \cos(x_\ell) \oplus \frac{b_i \cdot k}{c_i} \otimes \sin(x_\ell) \right) \oslash k$$

What are "good" Pythagorean triples?



$$\sin(x^*) \approx A_i \otimes [\cos(x_\ell)/k] \oplus B_i \otimes [\sin(x_\ell)/k]$$

Actual use of Pythagorean triples

- For each table entry i:
  - store integers $A_i = \frac{a_i}{c_i} \cdot k$ and $B_i = \frac{b_i}{c_i} \cdot k$ $(A_i, B_i \in \mathbb{F})$

Actual use of Pythagorean triples

- For each table entry i:
  - store integers $A_i = \dfrac{a_i}{c_i} \cdot k$ and $B_i = \dfrac{b_i}{c_i} \cdot k$ $(A_i, B_i \in \mathbb{F})$
- Incorporate $\frac{1}{k}$ into polynomial approximants:

$$\frac{\sin(x_\ell)}{k} \quad \text{and} \quad \frac{\cos(x_\ell)}{k}$$

- Remark: k should be *a small* least common multiple (LCM) of any combination of one hypotenuse c per table entry
  - small $\rightarrow$ each A_i and B_i fits in a machine word

Table generation algorithm

- <u>Input</u>
  - p: table index size

- <u>Algorithm steps</u>

$n \leftarrow 4$
repeat
    Generate all PPTs (a, b, c) such that $c \leq 2^n$.
    Search for the LCM k among all generated hypotenuses c.
    $n \leftarrow n + 1$
until such a k is found
Build tabulated values $(A, B, corr)$ for every entry.

Table example for $p = 4$

Index	S_h (A_i)	C_h (B_i)
0	0	5525
1	235	5520
2	612	5491
3	1036	5427
4	1360	5355
5	1547	5304
6	2044	5133

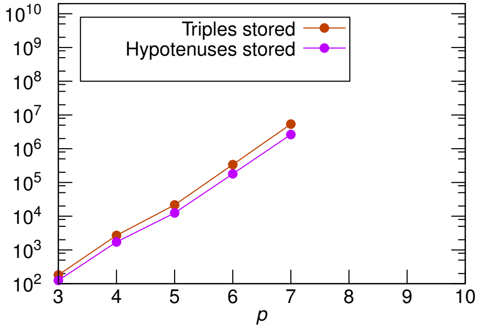
Index	S_h (A_i)	C_h (B_i)
7	2340	5005
8	2600	4875
9	2880	4715
10	3315	4420
11	3500	4275
12	3720	4085
13	3952	3861

- Computed value k = 5525
  - $612/5525 = 0.1107 \cdots \approx \cos(2 \cdot 2^{-4})$
  - $5491/5525 = 0.9938 \cdots \approx \sin(2 \cdot 2^{-4})$

Exhaustive search

Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60 GHz (32 cores) 125 GB RAM

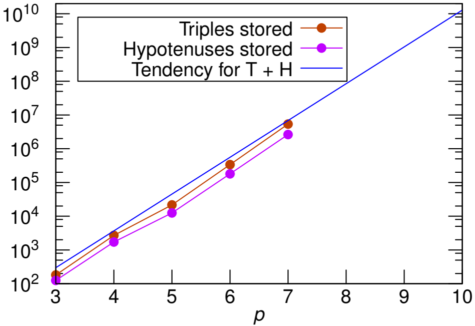
p	k_{min}	n	Time (s)
3	725	10	$\ll 1$
4	10625	14	0.01
5	130645	17	0.14
6	1676285	21	6
7	32846125	25	1000



Exhaustive search

Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60 GHz (32 cores) 125 GB RAM

p	k_{min}	n	Time (s)
3	725	10	$\ll 1$
4	10625	14	0.01
5	130645	17	0.14
6	1676285	21	6
7	32846125	25	1000



- Bottlenecks = searching the LCM and memory
- Finding k for a 10-bit addressed table is desperate with this exhaustive technique
  - our estimations show that $n = \lceil \log_2(k_{min}) \rceil \approx 37$

Heuristic search

p	k_{min}	Prime Factorization	Triples
5	130645	$5 \cdot 17 \cdot 29 \cdot 53$	21588
6	1676285	$5 \cdot 13 \cdot 17 \cdot 37 \cdot 41$	338660
7	32846125	$5^3 \cdot 13 \cdot 17 \cdot 29 \cdot 41$	5365290

Observation: k is always a product of *small* Pythagorean primes

- Pythagorean primes less than 70: {5, 13, 17, 29, 37, 41, 53, 61}

Heuristic search

p	k_{min}	Prime Factorization	Triples
5	130645	$5 \cdot 17 \cdot 29 \cdot 53$	21588
6	1676285	$5 \cdot 13 \cdot 17 \cdot 37 \cdot 41$	338660
7	32846125	$5^3 \cdot 13 \cdot 17 \cdot 29 \cdot 41$	5365290

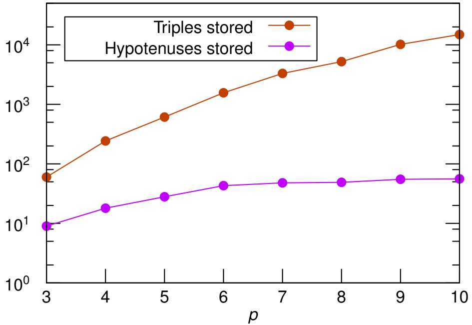
Observation: k is always a product of *small* Pythagorean primes

- Pythagorean primes less than 70: {5, 13, 17, 29, 37, 41, 53, 61}
- During the generation, only keep triples with hypotenuse

$$c = \prod_i p[i]^{r_i} \quad \text{with} \quad \begin{cases} r_i = 0 \textit{ or } 1 & \text{if } p[i] \neq 5 \\ r_i \in \mathbb{N} & \text{if } p[i] = 5 \end{cases}$$

Heuristic search results

p	k_{min}	n	Time (s)
6	1676285	21	0.12
7	32846125	25	0.88
8	243061325	28	4.63
9	12882250225	34	269
10	370668520625	39	8563



- Same results as the exhaustive search for $p \leq 7$
- Exponentially better than exhaustive search with respect to p
- New bottleneck: heuristic test over $\approx 2^n$ hypotenuses

Comparisons with other table-based methods

- Correct rounding in double precision : 2-step Ziv strategy
  - quick phase accurate to 2^{-66}
  - slow phase accurate to 2^{-150}
- Table index size $p = 10$ bits
- Estimate memory accesses (MA), FLOPs, and table size

Method	Table size (B)	Quick phase	Slow phase
Tang	38640	4 MA + 64 FLOP	6 MA + 241 FLOP
Gal	57960	**3 MA + 53 FLOP**	9 MA + 268 FLOP
Proposed	**32200**	**3 MA + 53 FLOP**	**5 MA + 148 FLOP**

- Table-size 17% lower than Tang's, 45% lower than Gal's
- Quick phase 25% + 17% less expensive than Tang's, same as Gal's
- Slow phase 17% + 39% less expensive than Tang's, 45% + 45% than Gal's

Conclusion and Perspectives

Contribution

- a new algorithm for table-based range reductions, for trigonometric functions
  - using error-free values (integers)
  - concentrating the error on the other steps
  - estimated gains of 45% in table-size, FLOPs and memory accesses for correctly-rounded double precision implementations
- a prototype able to pre-compute tables up to 10 indexing bits
- an extension to hyperbolic functions

Perspectives

- evaluation of accuracy and performance in available libraries
- integration in a full code-generation chain (MetaLibm)