# Extension of the SLZ algorithm to bivariate functions

Lauriane Turelier

# EXTENSION OF THE SLZ ALGORITHM TO BIVARIATE FUNCTIONS

LAURIANE TURELIER

Abstract. I propose an extension of the SLZ algorithm to find worst cases for the correct rounding of mathematical functions of two variables. First, I recall the principle of the algorithm for univariate functions. Then, I detail the extension to the bivariate functions and I prove that the algorithm is correct. Finally, I test the algorithm with the power function in single and double precision, I calculate its computation time and I expose some improvements to make the algorithm faster.

## 1. Introduction

The SLZ algorithm [1] is an algorithm to search worst cases of mathematical functions. A worst case for a function $f$ is a number $x$ such that $f(x)$ has $m$ identical bits after the round bit.

We suppose that $f$ is defined on $[1/2, 1[^2$ and $\forall x, y \in [1/2, 1[^2, f(x) \in [1/2, 1[$. We can always come back to this case. Indeed, if we consider an interval of the form $[2^n, 2^{n+1}[$ and $f$ defined on it and $f(x) \in [2^k, 2^{k+1}[$, by setting $g(x) = \dfrac{f(\frac{x}{2^{n+1}})}{2^{k+1}}$, $g$ is well defined on $[1/2, 1[$ and $g(x) \in [1/2, 1[$.

To find worst cases, the SLZ algorithm solves this modular equation:

$$\left| Nf\left(\frac{t}{N}\right) \mod 1 \right| < \frac{1}{M}$$

with $N$, $T$ and $M$ known and for $t$ in $[\![-T, T]\!]$.

To solve this equation, the function $f$ is approximated by its Taylor polynomial $P$. We build the lattice from a certain family of polynomials based on the polynomial $P$. After reducing this lattice, we have some polynomials whose integer roots are candidates to be a worst case for $f$. This algorithm is implemented in BacSel [3]. The SLZ algorithm has been detailed only for univariate functions, some hints to extend it to the $2D$ case are given in Damien Stehlé's PhD thesis [2] but the implementation was never done. In this article we detail an extension for bivariate functions, implement and experiment it on the power function in single and double precision.

## 2. Extension to the $2D$ case

We have the same modular equation to solve, but with $f$ a bivariate function. There are some details that change compared to the univariate case.

1. **The algorithm.**
**Input:** a function $f$, positive integers $N, T, M, d, \alpha$
**Output:** all $x, y \in [\![-T, T]\!]^2$ such that $|Nf(\frac{x}{N}, \frac{y}{N}) \bmod 1| < 1/M$

1. Let $P(x, y)$ be the expansion of $Nf\left(\dfrac{x}{N}, \dfrac{y}{N}\right)$ up to order $d$ included and $n = \dfrac{(2\alpha^2 d^2 + \alpha d^2 + 9\alpha d + 12)(\alpha + 1)}{12}$
2. Compute $\varepsilon$ such that $\left|P(x, y) - Nf\left(\dfrac{x}{N}, \dfrac{y}{N}\right)\right| < \varepsilon$ for $|x|, |y| \leqslant T$

3. Let $M' = \left\lfloor \frac{1/2}{1/M+\varepsilon} \right\rfloor$, $C = \frac{(d+1)(d+2)}{2} M'$ and $\tilde{P}(\tau,\nu) = \lfloor CP(T\tau, T\nu) \rceil$ (where the coefficients of $\tilde{P}(\tau,\nu)$ are the coefficients of $CP(T\tau,T\nu)$ rounded to the nearest integer)
4. Let $\{e_1, \cdots, e_n\} \longleftarrow \{\tau^{i_1}\nu^{i_2}z^j\}$ for $0 \leqslant i_1+i_2+dj \leqslant d\alpha$ (in ascending order of the sum $i_1+i_2+dj$ taking the $j$ increasing in case of a tie and finally taking $i_1$ increasing in the last case of a tie)
5. Let $\{g_1, \cdots, g_n\} \longleftarrow \left\{ T^{i_1+i_2}\tau^{i_1}\nu^{i_2} \left( \tilde{P}(\tau,\nu) + \frac{(d+1)(d+2)}{2} z \right)^j C^{\alpha-j} \right\}$ (the order is the same as in the previous step)
6. Form the $n \times n$ integral matrix $L$ where $L_{k,\ell}$ is the coefficient of the monomial $e_k$ in $g_\ell$
7. Reduce the lattice $L$ using LLL and let $V$ be the reduced lattice
8. Let $v_i$ be the vectors from $V$ which have a 1-norm smaller that $C^\alpha$ and $Q_i(T\tau, T\nu, z)$ the corresponding polynomials
9. If the number of $Q_i$ is less than 3 then return **FAIL**
10. Else, for each triplet of $Q_i, Q_j, Q_k$, compute $p_1(x,y) = \mathrm{Res}_z(Q_i(x,y,z), Q_j(x,y,z))$ and $p_2(x,y) = \mathrm{Res}_z(Q_i(x,y,z), Q_k(x,y,z))$. If $p_1 = 0$ or $p_2 = 0$ then continue with the next triplet. If it was the last triplet then return **FAIL**
11. Else, compute $p(x) = \mathrm{Res}_y(p_1(x,y), p_2(x,y))$. If $p = 0$ then continue to the previous step
12. Else, for each $x_0$ integer root of $p$ do for each $y_0$ integer root of $p_1(x_0,y)$ and $p_2(x_0,y)$ do if $|Nf(\frac{x_0}{N}, \frac{y_0}{N}) \bmod 1| < 1/M$ and $|x_0|, |y_0| \leqslant T$ then output $(x_0, y_0)$ . Return **SUCCESS**

## 2. The proof of the algorithm.

**Theorem 2.1.** *In case this algorithm returns **SUCCESS** (there is a triplet whose resultants are not 0), it outputs exactly all integers $x, y \in [\![-T, T]\!]^2$ such that $\left| Nf\left(\frac{x}{N}, \frac{y}{N}\right) \bmod 1 \right| < \frac{1}{M}$.*

*Proof.* Because of the test at the final step of the algorithm, we only have to check if any worst case is missed. Suppose there are $x_0, y_0 \in [\![-T, T]\!]^2$ with $\left| Nf\left(\frac{x_0}{N}, \frac{y_0}{N}\right) \bmod 1 \right| < \frac{1}{M}$.

On the one hand, we have

$$\begin{aligned} |P(x_0,y_0) \bmod 1| &= \left| P(x_0,y_0) - Nf\left(\frac{x_0}{N}, \frac{y_0}{N}\right) + Nf\left(\frac{x_0}{N}, \frac{y_0}{N}\right) \bmod 1 \right| \\ &\leqslant \left| P(x_0,y_0) - Nf\left(\frac{x_0}{N}, \frac{y_0}{N}\right) \right| + \left| Nf\left(\frac{x_0}{N}, \frac{y_0}{N}\right) \bmod 1 \right| \\ &< \varepsilon + \frac{1}{M} \\ &\leqslant \frac{1}{2M'}. \end{aligned}$$

On the other hand, we have

$$|CP(T\tau, T\nu) - \tilde{P}(\tau,\nu)| \leqslant \frac{(d+1)(d+2)}{4}.$$

Indeed, between the coefficients of the same monomial in $CP(T\tau,T\nu)$ and $\tilde{P}(\tau,\nu)$, there is a difference of at most $1/2$. Moreover, there are at most $\frac{(d+1)(d+2)}{2}$ coefficients in each polynomial. Finally, as $|\tau|, |\nu| \leqslant 1$, we find the previous inequality.

So we have:

$$\begin{aligned}\left|\tilde{P}\left(\frac{x_0}{N},\frac{y_0}{N}\right) \bmod C\right| &= \left|\tilde{P}\left(\frac{x_0}{N},\frac{y_0}{N}\right) - CP(x_0,y_0) + CP(x_0,y_0) \bmod C\right| \\ &\leqslant \left|\tilde{P}\left(\frac{x_0}{N},\frac{y_0}{N}\right) - CP(x_0,y_0)\right| + |CP(x_0,y_0) \bmod C| \\ &< \frac{(d+1)(d+2)}{4} + \frac{C}{2M'} \\ &\leqslant \frac{(d+1)(d+2)}{2}.\end{aligned}$$

Therefore, it exists $z_0 \in ]-1,1[$ such that $\tilde{P}\left(\frac{x_0}{N},\frac{y_0}{N}\right) + z_0\frac{(d+1)(d+2)}{2} = 0 \bmod C$. So $(x_0,y_0,z_0)$ is a root of $\tilde{P}\left(\frac{x}{N},\frac{y}{N}\right) + \frac{(d+1)(d+2)}{2}z \bmod C$. Thus, it exists $k \in \mathbb{N}$ such that $\tilde{P}\left(\frac{x_0}{N},\frac{y_0}{N}\right) + \frac{(d+1)(d+2)}{2}z_0 = kC$. We inject $x_0, y_0, z_0$ in $g_{i_1,i_2,j}$ and we obtain:

$$\begin{aligned}g_{i_1,i_2,j}(x_0,y_0,z_0) &= x_0^{i_1}y_0^{i_2}\left(\tilde{P}\left(\frac{x_0}{N},\frac{y_0}{N}\right) + z_0\frac{(d+1)(d+2)}{2}\right)^j C^{\alpha-j} \\ &= x_0^{i_1}y_0^{i_2}k^jC^\alpha.\end{aligned}$$

As $x_0$ and $y_0$ are integers, $x_0^{i_1}y_0^{i_2}k^j \in \mathbb{Z}$. So $(x_0,y_0,z_0)$ is a root of $g_{i_1,i_2,j}$ modulo $C^\alpha$. As $Q_1, Q_2$ and $Q_3$ are integer linear combinations of the $g_{i_1,i_2,j}$, $(x_0,y_0,z_0)$ is a real root of $Q_1, Q_2$ and $Q_3$. So $(x_0,y_0)$ is an integer root of $p_1$ and $p_2$. So $x_0$ is an integer root of $p$. So $x_0$ is found at the last step of the algorithm and $y_0$ too in the continuation of this same step. □

## 3. Experimental results

I have implemented the algorithm from section 2 in SAGE with the version 9.5. I have essentially worked with the power function. To not lose time during the execution of the algorithm, the computation of the error bound $\varepsilon$ in Step 2 is hardcoded for small values of $d$.

1. **The function $x, y \longrightarrow x^y$ in double precision.**
The double precision is a computer number format in the IEEE-754 standard. Each floating-point number in double precision is encoded on 64 bits distributed as follows:

- 1 bit of sign
- 11 bits of exponent
- 53 bits of mantissa with one implicit bit

An example of worst case of this function: $x = 4783716528592059/9007199254740992, y = 17/32$. We can find it with these parameters:

- $N = 2^{53}$
- $M = 2^{49}$
- $d = \alpha = 2$
- $T = 2^{13}$

and the algorithm returns $(-2885, -8192)$.

1. *Original setting.*
All tests are made with these parameters:

- $N = 2^{53}$
- $M = 2^{54}$ or $M = 2^{108}$
- $(d,\alpha) = (1,1), (2,2), (3,2)$ or $(3,3)$
- on a machine with 4 processors IntelCore i5-4570 CPU 3.20 GHz, with SAGE 9.5

(1) **Find the optimal $T$ and $(d, \alpha)$ pair and computation time on all $[1/2, 1]^2$**

For this study, I worked with $M = 2^{54}$ and then $M = 2^{108}$. For four pairs $(d, \alpha)$, I searched the largest $T$ for which the algorithm does not return **FAIL**, of form $\lfloor 2^{n+k} \rceil$ with $n \in \mathbb{N}$ and $k \in \{0, 0.1, \cdots, 0.9\}$. I found these results:

| $(d, \alpha)$ | $(1, 1)$ | $(2, 2)$ | $(3, 2)$ | $(3, 3)$ |
|---|---|---|---|---|
| optimal $T$ | $\lfloor 2^{11.2} \rceil = 2353$ | $\lfloor 2^{13.4} \rceil = 10809$ | $\lfloor 2^{13.4} \rceil = 10809$ | $2^{14} = 16384$ |

For this value of $T$, I calculated the computation time of the algorithm around one hundred points chosen randomly in $[1/2, 1[^2$ and averaged it. Then I calculated the total time on the entire domain $[1/2, 1[^2$ by proportionality. I call "square" a domain of the form $[-\frac{T}{N}, \frac{T}{N}]^2$ and "pad" the entire domain $[1/2, 1]^2$. I had these results:

| $(d, \alpha)$ | $(1, 1)$ | $(2, 2)$ |
|---|---|---|
| time on a square | $0.01s$ | $0.08s$ |
| time on the pad | $2.91 \times 10^{14}$ years | $1.10 \times 10^{14}$ **years** |

| $(d, \alpha)$ | $(3, 2)$ | $(3, 3)$ |
|---|---|---|
| time on a square | $0.23s$ | $1.93s$ |
| time on the pad | $3.17 \times 10^{14}$ years | $1.16 \times 10^{15}$ years |

I did the same study with $M = 2^{108}$ and I obtained these results:

| $(d, \alpha)$ | $(1, 1)$ | $(2, 2)$ | $(3, 2)$ | $(3, 3)$ |
|---|---|---|---|---|
| optimal $T$ | $\lfloor 2^{11.2} \rceil = 2353$ | $\lfloor 2^{14.2} \rceil = 18820$ | $\lfloor 2^{14.1} \rceil = 17560$ | $\lfloor 2^{15.4} \rceil = 43238$ |

Computation time:

| $(d, \alpha)$ | $(1, 1)$ | $(2, 2)$ |
|---|---|---|
| time on a square | $0.01s$ | $0.08s$ |
| time on the pad | $2.91 \times 10^{14}$ years | $3.63 \times 10^{13}$ **years** |

| $(d, \alpha)$ | $(3, 2)$ | $(3, 3)$ |
|---|---|---|
| time on a square | $0.25s$ | $3.53s$ |
| time on the pad | $1.30 \times 10^{14}$ years | $3.04 \times 10^{14}$ years |

In the rest of this section (double precision), the parameter $T$ will be fixed to the optimal $T$ calculated previously.

(2) **Splitting of the calculation time**

There are three steps that take time in the algorithm:

- the lattice reduction with the LLL algorithm (Step 7)
- the computation of the error $\varepsilon$ (Step 2)
- the treatment of triplets to find roots (Step 10-12)

So, I calculated the time spent on these steps, for $M = 2^{54}$ for different pairs $(d, \alpha)$. I calculated the computation time of the algorithm around one hundred points chosen randomly in $[1/2, 1[^2$ and averaged it. I found these results:

| $(d, \alpha)$ | $(1, 1)$ | $(2, 2)$ | $(3, 2)$ | $(3, 3)$ |
|---|---|---|---|---|
| dimension of the lattice | 4 | 22 | 39 | 94 |
| time of SLZ on a square | $0.01s$ | $0.08s$ | $0.23s$ | $1.93s$ |
| time of LLL on a square | $0.0022s$ | $0.062s$ | $0.19s$ | $1.37s$ |
| percentage time of LLL | 22% | 78% | 83% | 71% |
| time of computation $\varepsilon$ on a square | $0.0003s$ | $0.0002s$ | $0.0008s$ | $0.0009s$ |
| percentage of computation $\varepsilon$ | 3% | 0.25% | 0.34% | 0.047% |
| time of treatment of triplets on a square | $0.0015s$ | $0.0049s$ | $0.0051s$ | $0.46s$ |
| percentage of treatment of triplets | 15% | 6% | 2% | 24% |

We remark that more the pair $(d, \alpha)$ is large, more time is spent in LLL or the treatment of triplets. We study in the following some methods to reduce this time.

2. *Improvements.*

(1) **Reduction of the dimension of the lattice**

To reduce the calculation time, we can take a smaller lattice than the one we use for $(d, \alpha)$ fixed. Indeed, it is not necessary to take as many polynomials $g_i$ as members of the basis $\{e_1, \cdots, e_n\}$, we can take only the $g_i$s such that $0 \leqslant i_1 + i_2 + j \leqslant \alpha$, as proposed by D. Stehlé in chapter III-1-6 of [2].We have a rectangular lattice instead of a square lattice. Firstly, I calculated the computation time on the entire domain for $M = 2^{54}$ and $M = 2^{108}$. I obtained these results:

For $M = 2^{54}$:

| $(d, \alpha)$ | $(1, 1)$ | $(2, 2)$ |
|---|---|---|
| time on a square | $0.012s$ | $0.051s$ |
| time on the pad | $3.48 \times 10^{14}$ years | $7.02 \times 10^{13}$ **years** |

| $(d, \alpha)$ | $(3, 2)$ | $(3, 3)$ |
|---|---|---|
| time on a square | $0.087s$ | $0.85s$ |
| time on the pad | $1.20 \times 10^{14}$ years | $5.09 \times 10^{14}$ years |

And for $M = 2^{108}$:

| $(d, \alpha)$ | $(1, 1)$ | $(2, 2)$ |
|---|---|---|
| time on a square | $0.010s$ | $0.051s$ |
| time on the pad | $2.91 \times 10^{14}$ years | $2.32 \times 10^{13}$ **years** |

| $(d, \alpha)$ | $(3, 2)$ | $(3, 3)$ |
|---|---|---|
| time on a square | $0.093s$ | $1.36s$ |
| time on the pad | $4.85 \times 10^{13}$ years | $1.17 \times 10^{14}$ years |

We remark that except for the pair $(1, 1)$, this method has a positive impact on the computation time. Let's now look at the distribution of time in the different steps. Here, $M = 2^{54}$, and I obtained these results:

| $(d, \alpha)$ | $(1, 1)$ | $(2, 2)$ | $(3, 2)$ | $(3, 3)$ |
|---|---|---|---|---|
| dimension of the lattice | 4 | 10 | 10 | 20 |
| time of SLZ on a square | $0.012s$ | $0.051s$ | $0.087s$ | $0.85s$ |
| time of LLL on a square | $0.0024s$ | $0.030s$ | $0.053s$ | $0.26s$ |
| percentage time of LLL | 20% | 59% | 61% | 31% |
| time of computation $\varepsilon$ on a square | $0.0003s$ | $0.0002s$ | $0.0008s$ | $0.0008s$ |
| percentage of time of computation $\varepsilon$ | 2.5% | 0.39% | 0.91% | 0.094% |
| time of treatment of triplets on a square | $0.0024s$ | $0.006s$ | $0.006s$ | $0.56s$ |
| percentage of treatment of triplets | 20% | 12% | 7% | 66% |

Without surprise, the time spent in LLL is less important than before. Yet, the time spent on the treatment of triplets is slightly more important than before. But the time saving in LLL is greater than the time wasted in treatment of triplets.

(2) **Computation of the resultant on a finite field**

To try to reduce the time of treatment of triplets, we can use a finite field. So, for each $T$ used, I took the smallest prime number $p$ larger than $2T + 1$, and I calculated resultants and roots in $\mathbb{Z}/p\mathbb{Z}$. Indeed, if we find a root of the resultant that is not in $[\![-T, T]\!]$, we can bring it back into $[\![-T, T]\!]$ by subtracting $p$. Firstly, I calculated the computation time on the pad for $M = 2^{54}$ and $M = 2^{108}$. I obtained these results:

For $M = 2^{54}$:

| $(d, \alpha)$ | $(1, 1)$ | $(2, 2)$ |
|---|---|---|
| time on a square | $0.013s$ | $0.09s$ |
| time on the pad | $3.78 \times 10^{14}$ years | $1.24 \times 10^{14}$ **years** |

| $(d, \alpha)$ | $(3, 2)$ | $(3, 3)$ |
|---|---|---|
| time on a square | $0.24s$ | $1.46s$ |
| time on the pad | $3.3 \times 10^{14}$ years | $8.75 \times 10^{14}$ years |

For $M = 2^{108}$:

| $(d, \alpha)$ | $(1,1)$ | $(2,2)$ |
|---|---|---|
| time on a square | $0.012s$ | $0.086s$ |
| time on the pad | $3.49 \times 10^{14}$ years | $3.90 \times 10^{13}$ years |

| $(d, \alpha)$ | $(3,2)$ | $(3,3)$ |
|---|---|---|
| time on a square | $0.25s$ | $2.7s$ |
| time on the pad | $1.30 \times 10^{14}$ years | $2.32 \times 10^{14}$ years |

Unlike the method with a smaller lattice, the computation time is not reduced compared to the initial method, except for the pair $(3,3)$. Let's now look at the distribution of time in the different steps. Here, $M = 2^{54}$, and I obtained these results:

| $(d, \alpha)$ | $(1,1)$ | $(2,2)$ | $(3,2)$ | $(3,3)$ |
|---|---|---|---|---|
| time of SLZ on a square | $0.013s$ | $0.09s$ | $0.24s$ | $1.46s$ |
| time of LLL on a square | $0.0021s$ | $0.061s$ | $0.19s$ | $1.35s$ |
| percentage time of LLL | 16% | 68% | 79% | 92% |
| time of computation $\varepsilon$ on a square | $0.0003s$ | $0.0002s$ | $0.0008s$ | $0.0008s$ |
| percentage of time of computation $\varepsilon$ | 2% | 0.22% | 0.33% | 0.055% |
| time of treatment of triplets on a square | $0.0044s$ | $0.015s$ | $0.014s$ | $0.023s$ |
| percentage of treatment of triplets | 34% | 17% | 6% | 2% |

We remark that the time of the treatment of triplets is less important than the initial method for the pair $(3,3)$ but not for the other pairs.

(3) **Combination of these two methods**

Now, we combine the two methods: a reduced lattice and calculation in a finite field. I did the same tests as before: computation time on the pad and the repartition of time in the different steps.

For $M = 2^{54}$:

| $(d, \alpha)$ | $(1,1)$ | $(2,2)$ |
|---|---|---|
| time on a square | $0.012s$ | $0.051s$ |
| time on the pad | $3.49 \times 10^{14}$ years | $7.02 \times 10^{13}$ **years** |

| $(d, \alpha)$ | $(3,2)$ | $(3,3)$ |
|---|---|---|
| time on a square | $0.083s$ | $0.29s$ |
| time on the pad | $1.14 \times 10^{14}$ years | $1.73 \times 10^{14}$ years |

For $M = 2^{108}$:

| $(d, \alpha)$ | $(1,1)$ | $(2,2)$ |
|---|---|---|
| time on a square | $0.013s$ | $0.049s$ |
| time on the pad | $3.78 \times 10^{14}$ years | $2.22 \times 10^{13}$ **years** |

| $(d, \alpha)$ | $(3,2)$ | $(3,3)$ |
|---|---|---|
| time on the square | $0.081s$ | $0.3s$ |
| time on the pad | $4.22 \times 10^{13}$ years | $2.58 \times 10^{13}$ years |

We remark a large reduction of the computation time. Indeed, with these two methods, we reduce the time in LLL and the time in the treatment of the triplets. We can see this improvement with the repartition of time in the different steps.

| $(d, \alpha)$ | $(1,1)$ | $(2,2)$ | $(3,2)$ | $(3,3)$ |
|---|---|---|---|---|
| time of SLZ on a square | $0.012s$ | $0.051s$ | $0.083s$ | $0.29s$ |
| time of LLL on a square | $0.0021s$ | $0.027s$ | $0.048s$ | $0.24s$ |
| percentage time of LLL | 18% | 53% | 58% | 83% |
| time of computation $\varepsilon$ on a square | $0.0003s$ | $0.0002s$ | $0.0007s$ | $0.0007s$ |
| percentage of time of computation $\varepsilon$ | 3% | 0.39% | 0.84% | 0.23% |
| time of treatment of triplets on a square | $0.0047s$ | $0.015s$ | $0.012s$ | $0.020s$ |
| percentage of treatment of triplets | 39% | 22% | 14% | 7% |

As expected, the time spent in LLL has greatly decreased and only for the pair $(3,3)$ the treatment of triplets has decreased too, and this is due to the reduction of the lattices used and the use of finite fields.

2. **The function $x, y \longrightarrow x^y$ in single precision.**
The single precision is a computer number format in the IEEE-754 standard. Each floating-point number in single precision is encoded on 32 bits distributed as follows:

- 1 bit of sign
- 8 bits of exponent
- 24 bits of mantissa with one implicit bit

A study was made for the pow function in single precision. Some worst cases were found for $M = 2^{57}$ and $N = 2^{24}$ [4]. An example of worst case of this function:
$x = 296454119304300942313540655513 60, y = 15706239/8589934592$. We can find it with these parameters:

- $f(x, y) = (x2^{105})^{y2^{-9}}/2$
- $N = 2^{24}$
- $M = 2^{57}$
- $d = \alpha = 2$
- $T = 2^5$

and the algorithm returns $(31, 31)$.

I made the same study as before for the double precision: I found the optimal $T$ for different pairs $(d, \alpha)$, I calculated the computation time on the entire domain $[1/2, 1[^2$ and the repartition of time in the different steps. I did the tests with $M = 2^{57}$.

For the optimal $T$, I found these results:

| $(d, \alpha)$ | $(1,1)$ | $(2,2)$ | $(3,2)$ | $(3,3)$ |
|---|---|---|---|---|
| optimal $T$ | $2^4 = 16$ | $\lfloor 2^{5.9} \rceil = 60$ | $\lfloor 2^{5.9} \rceil = 60$ | $\lfloor 2^{6.5} \rceil = 91$ |

For the computation time on the entire domain, I had these results:

| $(d, \alpha)$ | $(1,1)$ | $(2,2)$ | $(3,2)$ | $(3,3)$ |
|---|---|---|---|---|
| time on a square | $0.0073s$ | $0.041s$ | $0.068s$ | $0.28s$ |
| time on the pad | 15.91 years | 6.41 **years** | 10.64 years | 19.07 years |

For the repartition of time in the different steps, I obtained these results:

| $(d, \alpha)$ | $(1,1)$ | $(2,2)$ | $(3,2)$ | $(3,3)$ |
|---|---|---|---|---|
| time of SLZ on a square | $0.0073s$ | $0.041s$ | $0.068s$ | $0.28s$ |
| time of LLL on a square | $0.0022s$ | $0.029s$ | $0.049s$ | $0.24s$ |
| percentage time of LLL | 30% | 71% | 72% | 86% |
| time of computation $\varepsilon$ on a square | $0.0003s$ | $0.0002s$ | $0.0007s$ | $0.0007s$ |
| percentage of time of computation $\varepsilon$ | 4% | 0.5% | 1% | 0.25% |
| time of treatment of triplets on a square | $0.0021s$ | $0.0065s$ | $0.0063s$ | $0.015s$ |
| percentage of treatment of triplets | 29% | 16% | 9% | 5% |

## 4. Conclusion

I extend the SLZ algorithm to bivariate functions and propose some improvements. It appears from the study that the optimal parameters are $d = \alpha = 2$, and the associated $T$ ($T = \lfloor 2^{13.4} \rceil$ for $M = 2^{54}$ and $T = \lfloor 2^{14.2} \rceil$ for $M = 2^{108}$ on double precision and $T = \lfloor 2^{5.9} \rceil$ for $M = 2^{57}$ on single precision) whatever the method used. But it is only a little step, because I only worked with the power function but there are other bivariate functions to study like $x, y \longrightarrow \arctan\left(\frac{x}{y}\right)$ or $x, y \longrightarrow \sqrt{x^2 + y^2}$. It could be interesting to make a generalization of the algorithm for functions in $n$ variables.

**References.**


[1 ] STEHLÉ D., LEFÈVRE V. and ZIMMERMANN P., *Searching Worst Cases of a One-Variable Function Using Lattice Reduction*, IEEE Transactions on computers, 54, 3 (2005), p.340-346

[2 ] STEHLÉ D., *Algorithmique de la réduction de réseaux et application à la recherche de pires cas pour l'arrondi de fonctions mathématiques.* PhD thesis, Université Henri Poincaré Nancy 1, 2005

[3 ] https://gitlab.inria.fr/zimmerma/bacsel.git

[4 ] SIBIDANOV A., ZIMMERMANN P. and GLONDU S., *The CORE-MATH project*, ARITH 2022-29TH IEEE Symposium on Computer Arithmetic, 2022, https://hal.inria.fr/hal-03721525