On the Randomness of Bits Generated by Sufficiently Smooth Functions

Damien Stehlé
LORIA / Université Nancy 1.
http://www.loria.fr/~stehle — stehle@maths.usyd.edu.au

Abstract. Elementary functions such as sin or exp may naively be considered as good generators of random bits: the bit-runs output by these functions are believed to be statistically random most of the time. Here we investigate their computational hardness: given a part of the binary expansion of $\exp x$, can one recover x? We describe a heuristic technique to address this type of questions. It relies upon Coppersmith's heuristic technique — itself based on lattice reduction — for finding the small roots of multivariate polynomials modulo an integer. For our needs, we improve the lattice construction step of Coppersmith's method: we describe a way to find a subset of a set of vectors that decreases the Minkowski theorem bound, in a rather general setup including Coppersmith-type lattices.

1 Introduction

Using expansions of real numbers is a natural idea to build pseudo-random number generators (PRNG). In the paper [2] in which Blum, Blum and Shub introduce the celebrated "$x^2 \bmod N$" PRNG, they first investigate the so-called "$1/N$" generator. A secret integer N is chosen, and the output bits are consecutive bits of the binary expansion of $1/N$, starting from a specified rank (the most significant bits are hidden, otherwise one would recover N by simply applying the inverse function). The PRNG is efficient but, unfortunately, it is cryptographically insecure: with a run of $2\log_2 N + O(1)$ bits, one can recover N in time polynomial in the bit-size of N, and thus compute the bits that are to come next. This bit-run is not random since one can predict the remainder from the beginning of it. Instead of rationals, one could use algebraic numbers, i.e., roots of a degree d monic univariate integer polynomial $P(x)$, where d and a bound H on the magnitude of the coefficients are specified. This question was raised by Manuel Blum and answered negatively in [10]: if the first $O(d^2 + d\log H)$ bits of a root of P are known then one can recover P in polynomial time and thus compute the sequence himself.

In the present paper, we address a generalization of this type of questions to smooth mathematical functions like trigonometric functions, exponentials, logarithms, … Let f be such a function over $[0,1]$ and x an n-bit long secret integer. The output of the PRNG is the bit-run of the binary expansion of $f\left(\frac{x}{2^n}\right)$, starting from a specified rank: the first bits (or digits, the base being irrelevant) are kept hidden to make impossible the use of f^{-1} if it exists. These generated bits are believed to be statistically random. They would correspond to so-called normal numbers [1]. They were introduced by Borel [5] who showed they are overwhelming. Besides, statistical randomness is far weaker than unpredictability. Here we will consider the PRNG as weak if from a sequence of length polynomial in n one can recover the integer x in time polynomial in n.

The problem described above is connected to the so-called table maker's dilemma [11], a difficulty encountered while implementing an elementary mathematical function f in a given floating-point (*fp* for short) precision — for example in double precision (with 53-bit long mantissæ). In tight to all cases, the image $f(x)$ of the *fp*-number x cannot be represented exactly. The value $f(x)$ has to be rounded to $\diamond(f(x))$, where $\diamond(a)$ is a closest *fp*-number to a. Unfortunately, the exact value $f(x)$ can be extremely close to the middle of two consecutive representable numbers (or even worse, it could be exactly the middle) and thus many bits of $f(x)$, that is to say a very sharp approximation to $f(x)$, may be needed to compute $\diamond(f(x))$. The maximum of the number of needed bits, taken over the input *fp*-numbers, helps getting an efficient implementation of f: for any input x, one computes a close enough approximation to $f(x)$, and then round this approximation to the closest representable number. This quantity is usually computed by finding the *fp*-numbers x for which $f(x)$ has a long run of zeros, or a long run of ones, starting just after the rounding bit. Instead of "inverting" an arbitrary sequence of bits in the context of the PRNG described above, we "invert" a sequence of zeros or a sequence of ones. The overall approach we describe generalizes and improves the technique developed in [21] to find bad cases for the rounding of functions.

We tackle these issues with Coppermith's lattice-based technique for calculating the small roots of multivariate polynomials modulo an integer [7]. It is heuristic, which is the only reason why our result is heuristic. For our needs, we improve the lattice construction step of this technique. In Coppersmith's technique, a family of polynomials is first derived from the polynomial whose roots are wanted. This family naturally gives a lattice basis and short vectors of this lattice possibly provide the wanted roots. The main difficulty is to choose cleverly the family: the goal is to find polynomials for which the Minkowski bound of the corresponding lattice is as low as possible, making possible the computation of larger roots. We present a general technique to choose a good subset of polynomials within a family of polynomials. Boneh and Durfee already presented such a technique in [3] but it applies only to very specific lattice bases. Our technique is more general, though slightly less powerful in the case of [3], and could be of interest wherever Coppersmith's method is used.

Road-map of the Paper. In Section 2 we describe the problem we tackle and related issues. In Section 3 we give the minimal background on lattices and Coppersmith's method. We describe our algorithm in Section 4, give its complexity analysis in Section 5 and demonstrate experimentally its efficiency in Section 6. In Section 7, we discuss a few generalizations and open problems.

Notation. We define $[\![a, b]\!]$ as the set of integers in $[a, b]$. For any integer n, we let $[a, b)_n$ denote the $\frac{m}{2^n}$'s where $a \leq \frac{m}{2^n} < b$ and $m \in \mathbb{Z}$. For example $[1/2, 1)_{53}$ corresponds to the positive *fp*-numbers in double precision with exponent -1. For any real x, we let $\lfloor x \rfloor$, $\lceil x \rceil$, $\lfloor x \rceil$ and (x cmod 1) denote its floor, ceiling, closest integer and centered fractional part $x - \lfloor x \rceil$. For $\lfloor x \rceil$ and (x cmod 1), if x is half an odd integer, we choose any of the two possibilities. In the following, vectors are written in bold and for a vector $\mathbf{x} \in \mathbb{R}^n$, $\|\mathbf{x}\|$ and $\|\mathbf{x}\|_1$ are its L_2 and L_1 norms, i.e., $\sqrt{\sum_{i=1}^{n} x_i^2}$ and $\sum_{i=1}^{n} |x_i|$. For the complexity results, we use the bit complexity model. The notation $\mathcal{P}(x_0, \ldots, x_k)$ shall be read as "polynomial in $x_0, \ldots, x_k$". Finally, all logarithms are in base 2.

2 Bits Generated by Mathematical Functions

In the present section, after describing the equation we tackle, we explain how it relates to the problems mentioned above: the PRNG and the search of bad cases for the rounding of functions. We describe our results afterwards.

2.1 The Equation to be Solved

Let $f : [0, 1] \to [\alpha, \beta]$ (for some reals α and β) be a function, and N_1, N_2, M be three integers. Let $c \in \mathbb{R}$. We are interested in solving the following equation:

$$\left| \left[N_2 \cdot f\left(\frac{x}{N_1}\right) - c \right] \text{ cmod } 1 \right| \leq \frac{1}{M}, \text{ for } x \in [\![0, N_1]\!]. \tag{1}$$

We are given an approximation c to the exact value $f(x/N_1)$, but the most significant bits are hidden. If the outputs of f behave sufficiently randomly, then as soon as $M = \Omega(N_1)$, the solution x, if there is one, should be unique.

The Pseudo-Random Number Generator. We study the following PRNG, based on a given function f. Fix two security parameters n_1 and n_2. Choose a secret seed $x_0 \in [\![0, 2^{n_1}]\!]$. Compute $f(x_0/2^{n_1})$, throw away the first bits up to the one of weight 2^{-n_2}, and then output as many as needed of the following ones. More precisely, the output corresponds to the bits of $(2^{n_2} \cdot f(x_0/2^{n_1}) \text{ cmod } 1)$, up to some rank. One must choose a large enough n_2 to make out of reach the guess of the hidden bits and the use of f^{-1} if it exists. To obtain an efficient PRNG one may choose an efficiently computable function f. For example $\sin, \log, \exp$ are computable in time quasi-linear in n_1, n_2 and the number of output bits [6]. In this context, n_1 and n_2 are thus polynomially related. Breaking this PRNG can be reduced to (1). Indeed, suppose that one has seen the first $m + 1$ output bits, giving some $y_0 \in [\![2^m, 2^{m+1} - 1]\!]$. The seed x_0 satisfies:

$$\left| \left[2^{n_2} \cdot f\left(x_0/2^{n_1}\right) - y_0/2^{m+1} \right] \text{ cmod } 1 \right| \leq 2^{-m}.$$

It is an instance of (1): take $N_1 = 2^{n_1}, N_2 = 2^{n_2}, M = 2^m$ and $c = y_0/2^{m+1}$.

The Table Maker's Dilemma. Let f be an elementary function, and n be the input-output precision for an implementation of f over $[1/2, 1)_n$. The input *fp*-numbers such that $\diamond(f(x))$ is hard to compute are the x's in $[1/2, 1)_n$ with:

$$|[2^{n+1} \cdot f(x) + 1/2] \text{ cmod } 1| \leq \epsilon \text{ in the case of a rounding to nearest,}$$
$$|2^n \cdot f(x) \text{ cmod } 1| \leq \epsilon \text{ in the case of a directed rounding,}$$

where directed roundings are the roundings towards $\infty, -\infty$ and 0. Obviously these equations are instances of (1). The smaller ϵ, the more difficult the computation of $\diamond(f(x))$, because a tighter approximation of the exact value needs to be computed to decide the last bit of $\diamond(f(x))$. If f behaves randomly enough and we want to find the worst input, then ϵ will be set to $\approx 2^{-n}$: we expect $O(1)$ solutions (the bits should be independent and uniformly distributed), containing the worst input. If this equation cannot be solved efficiently for this ϵ, it may be interesting to show that it has no solution for a much smaller ϵ, in order to ensure that f has no "exact output": such outputs (at the middle between two *fp*-numbers in the case of the rounding to nearest, and exactly a *fp*-number in the case of a directed rounding) are in full generality impossible to handle because even very accurate approximations do not help deciding the rounding direction.

Other Related Problems. Integer factorization can also be reduced to (1). Take an n-bit long integer $N = pq$ with $p \leq q$. Then p is a solution to:

$$\left|2^{\lfloor n/2 \rfloor} \cdot \left(\frac{N}{2^n} \frac{1}{x/2^{\lceil n/2 \rceil}}\right) \text{ cmod } 1\right| = 0 \text{ for } x \in \left[\!\left[0, 2^{\lceil n/2 \rceil}\right]\!\right].$$

Similarly, solving (1) could help obtaining integer points on curves. For example, take two integers a and b and suppose we want to find the pairs of integers (x, y) satisfying $y^2 = x^3 + ax + b$ and $0 \leq x \leq 2^n$ for an even n. Then we can consider (1) with $f(x) = \sqrt{x^3 + (a2^{-2n})x + (b2^{-3n})}$ and $N_2 = 2^{3n/2}$.

Unfortunately, our heuristic method seems to fail for algebraic functions and does not help solving the two problems above.

2.2 Description of the Results

From now on, we fix $f : [0, 1] \to [\alpha, \beta]$ and suppose that f is $\mathcal{C}^\infty$ and that its successive derivatives satisfy: $\forall i \geq 0, \forall x \in [0, 1], |f^{(i)}(x)| \leq i!K$ for some K. We suppose that the derivatives of f are efficiently computable (the first n_2 bits of $f^{(i)}(x)$ where x is n_1-bit long shall be computable in time $\mathcal{P}(i, n_1, n_2)$). We also suppose that the quantities $\max_{x \in [0,1]} |f^{(i)}(x)|$ are efficiently computable. For example, we can choose $f = \exp$ or $f = \sin$.

In the following sections, we describe and analyze an algorithm that, given as inputs N_1, N_2, M and c satisfying $MN_2 \geq N_1$, finds all the solutions to (1) in essentially (see Theorem 3 for the exact statement):

$$\mathcal{P}(\log(N_1 N_2 M)) \cdot 2^{\frac{\log^2(N_1 N_2)}{4 \log(M N_2)}} \text{ bit operations.}$$

Some comments need to be made on this statement. Firstly, the algorithm always finishes and gives the correct output but its running time is only heuristic: in the worst case, it might fall down to an exhaustive search. The heuristic assumption under which the result holds will be made explicit below. Secondly, notice that in the case of the table maker's dilemma, we get a (heuristic) $\mathcal{P}(n)2^{n/2}$ running time by choosing $N_1 = N_2 = M = 2^n$. This improves the best complexity bound previously known, i.e., a (heuristic) $\mathcal{P}(n)2^{4n/7}$ running time [21]. Finally, by choosing $M = 2^{\log^2(N_1 N_2)}$, we obtain a polynomial time algorithm that breaks the PRNG: a run of $(n_1 + n_2)^2$ output bits suffices to recover the seed efficiently. Amazingly, this quadratic bound matches the result of Nesterenko and Waldschmidt [17] for exp when specialized to our context: their result implies that for $f = \exp, c = 0$ and $M \geq 2^{k(n_1+n_2)^2}$ for some constant k, there is no non-trivial solution to (1). Our work can be seen as a constructive variant of [17].

3 Preliminaries

We start this section by stating some algorithmic results on lattices (see [13] for more details) before describing Coppersmith's technique and our method for selecting a good subset of polynomials within a set of polynomials.

3.1 Lattices and the L^3 Algorithm

A lattice L is a set of all linear integer combinations of $d \leq n$ linearly independent vectors $\mathbf{b}_i$ over $\mathbb{R}$, that is $L = \{\sum_{i=1}^{d} x_i \mathbf{b}_i, x_i \in \mathbb{Z}\}$. The $\mathbf{b}_i$'s are called a basis of L. A given lattice has infinitely many bases (as soon as $d \geq 2$). The lattice dimension $\dim L = d$ does not depend on the choice of the basis, neither does the embedding dimension n. The determinant of the lattice L is defined by:

$$\det L = \prod_{i=1}^{d} \|\mathbf{b}_i^*\|, \tag{2}$$

where $[\mathbf{b}_1^*, \ldots, \mathbf{b}_d^*]$ is the Gram-Schmidt orthogonalization of $[\mathbf{b}_1, \ldots, \mathbf{b}_d]$, that is: $\mathbf{b}_1^* = \mathbf{b}_1$, and $\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\|\mathbf{b}_j^*\|^2} \mathbf{b}_j^*$. This definition extends the usual definition of the determinant to non-square matrices (except for the sign). The determinant is a lattice invariant: it is independent of the chosen basis of L.

Most often, bases of interest are made of rather short vectors. Minkowski [16] showed that any lattice L contains a vector $\mathbf{b} \neq \mathbf{0}$ satisfying the so-called Minkowski bound:

$$\|\mathbf{b}\| \leq \sqrt{\dim L} \cdot (\det L)^{\frac{1}{\dim L}}.$$

Unfortunately, Minkowski's proof is not constructive and no efficient way to find such a short vector is known. In 1982, Lenstra, Lenstra and Lovász [12] gave a polynomial time algorithm computing a so-called L^3-reduced basis that, among others, contains a vector that satisfies a weakened version of Minkowski's bound.

Theorem 1 ([18]). *Let $\mathbf{B}_1, \ldots, \mathbf{B}_d \in \mathbb{Z}^n$ be independent vectors with lengths smaller than B, and $d = O(\log B)$. In $O(d^4 n \log^2 B)$ bit operations, one can find a basis $\mathbf{b}_1, \ldots, \mathbf{b}_d$ of the lattice spanned by the $\mathbf{B}_i$'s that satisfies:*

$$\|\mathbf{b}_1\| \leq 2^d (\det L)^{\frac{1}{d}} \quad \textit{and} \quad \|\mathbf{b}_2\| \leq 2^d (\det L)^{\frac{1}{d-1}}.$$

This theorem covers up all we need to know about the L^3 algorithm for our current needs: lattice reduction is to be used as a black box. Of course, for practical issues, one should dismantle the black box and tune it for the application.

3.2 Small Roots of Bivariate Polynomials Modulo an Integer

Coppersmith's method [7] is a general technique to find all small roots of polynomial equations modulo an integer. It heavily relies upon the L^3 algorithm, that dominates the running time. It is provable in the case of univariate polynomials: if P is a degree d monic polynomial in $(\mathbb{Z}/N\mathbb{Z})[x]$, then one can find all its roots smaller than $N^{1/d}$ in time polynomial in d and $\log N$. It is only heuristic for multivariate polynomials. It has proved very powerful in public-key cryptography: the univariate variant [4, 7, 15] as well as the multivariate one [3, 8, 14].

Suppose we search the solutions to the equation:

$$P(x, y) = 0 \mod N, \tag{3}$$

where P is a bivariate polynomial with integer coefficients, the modulus N is an integer, and x and y are integer unknowns. Since in general solving such a polynomial equation is hard, we restrict ourselves to finding the small solutions: $|x| \leq X$ and $|y| \leq Y$, for some bounds X and Y that are as large as possible.

Coppersmith's technique depends on an integer parameter $\alpha \geq 1$ to be chosen to maximize the reachable bounds X and Y (most often, α growing to infinity is asymptotically the optimal choice). The method is made of four main steps.

1. First, a large set $\mathbb{P}$ of polynomial equations modulo N^α is derived from (3). We use powers of P shifted by powers of variables: $N^{\alpha-i}P(x, y)^i x^j y^k$ for $i \in [|0, \alpha|]$ and $j, k \geq 0$. This is the *polynomials selection step.* If (x_0, y_0) is a solution to (3), then it must be a root modulo N^α of all the polynomials in $\mathbb{P}$.
2. In the *polynomials-to-lattice step*, we transform the family of polynomials $\mathbb{P}$ into a lattice $L_{X,Y}[\mathbb{P}]$. We list and sort (arbitrarily) the monomials $x^j y^k$ appearing in the polynomials of $\mathbb{P}$. Suppose there are n such monomials. If a polynomial $Q(x, y)$ has all its monomials belonging to the monomials appearing in the selected family, then we map it to an n-dimensional vector whose coefficient corresponding to the monomial $x^j y^k$ is the coefficient of $Q(x \cdot X, y \cdot Y)$ for this monomial. This map is obviously a bijection. The lattice $L_{X,Y}[\mathbb{P}]$ we consider is spanned by the vectors of $\mathbb{R}^n$ that are obtained from the selected polynomials *via* the map described above. Since any vector of this lattice is an integral linear combination of the vectors corresponding to the selected polynomials, any solution (x_0, y_0) to (3) is a root modulo N^α of all the polynomials corresponding to the vectors of $L_{X,Y}[\mathbb{P}]$.
3. If the polynomials of $\mathbb{P}$ were linearly independent, we got a lattice basis in the previous step, and we now run an L^3-type algorithm on it. If the polynomials are linearly dependent, this is not a problem since L^3 can be modified to manage generating vectors instead of basis vectors (but the analysis of the method becomes more intricate). After this *lattice-reduction step*, which is the computationally dominating step, we have a basis of L made of vectors whose lengths are related to $\det(L_{X,Y}[\mathbb{P}])$, as described in Theorem 1.
4. In the reduced basis of $L_{X,Y}[\mathbb{P}]$, we take all the vectors of L_1 norm $< N^\alpha$. Any solution (x_0, y_0) to (3) modulo N is a root over $\mathbb{Z}$ of all the polynomials corresponding to these vectors. It therefore remains to solve these equations over the integers. We call this the *root-finding step.* There are several ways to perform this step: with any variable elimination method (for example through a resultant computation) or with Hensel's lifting. In all cases, we need at least two lattice vectors with small L_1 norm to solve the system of equations. Furthermore these polynomials can share factors, in which case it may be impossible to recover any useful information. It is not known how to work

around this difficulty. This step is the heuristic one in Coppersmith's method for bivariate polynomials. This is the reason why the running-time bound of our method for solving (1) is only heuristic. At the end, all the possible solutions to (3) need to be checked, since some might be spurious.

The heuristic assumption. In the present paper, we do the following heuristic assumption: if two polynomials correspond to the first two vectors of an L^3-reduced basis computed during any lattice reduction step of Coppersmith's bivariate method, then they do not share any factor.

Theorem 1 ensures we will obtain two sufficiently short vectors, as long as:

$$\sqrt{n}2^{\dim L[\mathbb{P}]} \cdot (\det L[\mathbb{P}])^{\frac{1}{\dim L[\mathbb{P}]-1}} < N^\alpha.$$

When expliciting the above inequality as a relation on X and Y, we obtain what we call the *Coppersmith equation*. The goal of the analysis of a particular use of Coppersmith's method is to find the family $\mathbb{P}$ providing the best Coppersmith equation, that is to say the one allowing the largest reachable X and Y. The target is thus to minimize Minkowski's quantity $(\det L[\mathbb{P}])^{\frac{1}{\dim L[\mathbb{P}]-1}}$.

3.3 Finding a Good Family of Polynomials

As we have seen above, the strength of Coppersmith's method is determined by the polynomials selection. Often, the family $\mathbb{P}$ is chosen so that the corresponding lattice basis is square and triangular. This makes the determinant computation simple (the determinant being in this case the product of the absolute values of the diagonal entries) and this often gives the "good" bound, which means that after many trials, one could not find a better family of polynomials. Nevertheless, in some cases, one can improve Minkowski's bound by choosing polynomials giving a non-square matrix. This is the case for example in [3] and in our present situation. Here we give a mean to bound the determinant of lattices given by bases that are not necessarily square and triangular.

Suppose we have a $d \times n$ matrix B whose rows span a lattice L. Suppose further that the entries of B satisfy: $|B_{i,j}| \leq wr_i \cdot wc_j$, for some wr_i's and wc_j's (with $i \leq d$ and $j \leq n$). This is the case for all Coppersmith-type lattice bases. We say that such a matrix is bounded by the products of the wr_i's and wc_j's.

Theorem 2. *Let B be a $d \times n$ matrix bounded by the product of some quantities wr_i's and wc_j's. Let $L[B]$ be the lattice spanned by the rows of the matrix B, and P the product of the d largest wc_j's. We have:*

$$\det L[B] \leq 2^{\frac{d(d-1)}{2}} \cdot \sqrt{n} \cdot \left(\prod_{i=1}^{d} wr_i\right) \cdot P.$$

The result follows from basic row and column operations and Hadamard's bound $\det L[B] \leq \prod_{i \leq d} \|\mathbf{b}_i\|$, where the $\mathbf{b}_i$'s are the vectors corresponding to the rows of the matrix B. The proof is given in appendix. Theorem 2 can be used to find a subset of a given set of vectors that improves the term "$(\det L)^{\frac{1}{\dim L}}$" in Minkowski's bound. Suppose we have a $d \times n$ matrix B bounded by the product of some wr_i's and wc_j's. We consider the d vectors given by the rows of B. We begin by ordering the wc_j's decreasingly and the wr_i's increasingly. Then the subset of cardinality k that gives the best bound *via* Theorem 2 is the one corresponding to any k smallest wr_k's. We compute the quantities $\left(\prod_{i=1}^{k} wr_i \cdot wc_{n-i}\right)^{1/k}$ for all $k \leq d$, and keep the vectors giving the smallest value.

Notice that our method does not necessarily gives the best subset of a given set of vectors: in particular, it makes no use of the possibly special shape (not even triangular) of the coefficients. E.g., it fails to give the 0.292 bound of [3].

4 The Algorithm and its Correctness

The algorithm we study is described in Figure 1. It takes as input the quantities N_1, N_2, M and c, as well as three parameters T, d and α that will be chosen in order to improve the efficiency of the algorithm. The output are all the solutions to (1) for the given N_1, N_2, M and c. The overall architecture of the algorithm is as follows. The initial search interval $[\![0, N_1]\!]$ is divided into $\frac{N_1}{2T}$ subintervals of length $2T$. Each subinterval is considered independently (possibly on different machines): for each of them, we approximate the function f by a degree d polynomial P; we solve (1) for P instead of f with a smaller M (to take care of the distance between f and P); to perform this last step, we use Coppersmith's method with the bivariate polynomial $P(x) + y$.

```
Input: $N_1, N_2, M \in \mathbb{Z}, c \in \mathbb{R}$ with finitely many bits. Three parameters $t, d, \alpha \in \mathbb{Z}$.
Output: All the $x$'s in $[\![0, N_1]\!]$ that are solution to (1).
 1. $n := \frac{(\alpha+1)(d\alpha+2)}{2}$, $\{e_1, \ldots, e_n\} := \{x^i y^j, i + dj \leq d\alpha\}$, $T := 2^t, T' := T, S := \emptyset$.
 2. $t_0 := 0$. While $t_0 \leq N_1$, do
 3.    If $t_0 + 2T' \geq N_1$, $T' := \lfloor \frac{N_1 - t_0}{2} \rfloor$.
 4.    If $T' = 0$, then
 5.       Add $t_0$ in $S$ if it is solution to (1).
 6.       $t_0 := t_0 + 1, T' := T$.
 7.    Else
 8.       $t_m := t_0 + T', P(x) := c + f\left(\frac{t_m}{N_1}\right) + f'\left(\frac{t_m}{N_1}\right)\frac{x}{N_1} + \ldots + \frac{1}{d!} f^{(d)}\left(\frac{t_m}{N_1}\right)\frac{x^d}{N_1^d}$.
 9.       $\epsilon := \left(\max_{x \in [0,1]} \left| f^{(d+1)}(x) \right|\right) \cdot \frac{N_2}{(d+1)!} \left(\frac{T'}{N_1}\right)^{d+1}$, $M' := \frac{1}{\epsilon + 1/M}$.
10.       $\left\{g_1, \ldots, g_{\frac{\alpha(\alpha+1)}{2}}\right\} := \{x^i (N_2 P(x) + y)^j, i + j \leq \alpha\}$.
11.       Create the $\frac{\alpha(\alpha+1)}{2} \times n$ matrix $B$ such that $B_{k,l}$ is the coefficient of the
          monomial $e_l$ in the polynomial $g_k\left(xT', \frac{1}{M'}y\right)$.
12.       L$^3$-reduce the rows of $B$. Let $\mathbf{b}_1, \mathbf{b}_2$ be the first two output vectors.
13.       $test := 1$. If $\|\mathbf{b}_1\|_1 \geq 1$ or $\|\mathbf{b}_2\|_1 \geq 1$, $test := 0$.
14.       Let $Q_1(x,y), Q_2(x,y)$ be the polynomials corresponding to $\mathbf{b}_1$ and $\mathbf{b}_2$.
15.       $R(x) := \mathrm{Res}_y(Q_1(x,y), Q_2(x,y))$. If $R(x) = 0$, then $test := 0$.
16.       If $test = 0$, then $T' := \lfloor T'/2 \rfloor$.
17.       Else, for any root $x_0$ of $R$ belonging to $[\![-T', T']\!]$, add $t_m + x_0$ in $S$ if
          it is a solution to (1), $t_0 := t_0 + 2T' + 1, T' := T$.
18. Return $S$.
```

Fig. 1. The algorithm solving (1).

At Step 1, we compute the embedding dimension n of the lattices we will reduce, and the list of the monomials that will appear in the polynomials generated during Coppersmith's method. During the execution of the algorithm, the integer t_0 increases: at any moment, all the solutions to (1) below t_0 have already been found and it remains to find those that are between t_0 and N_1. The set S contains all the solutions to (1) that are $\leq t_0$. Finally, the value T' is half the size of the current subinterval in which we are searching solutions to (1): usually, we have $T' = T$, but if Coppersmith's method fails, then we halve T'. The quantity T' makes the algorithm valid even if Coppersmith's method repeatedly fails. As a drawback the running time bound only heuristic. If there are too many consecutive failures of Coppersmith's method, then we might have $T' = 0$: in this case we test if the current value t_0 is a solution to (1) (Step 5). At Step 8, we approximate the function f by its degree d Taylor expansion P at the center of the considered interval. At Step 9, we compute the error ϵ made by approximating f by P. We update M accordingly. At Step 10, we generate the family of polynomials that will be used in Coppersmith's method for the bivariate polynomial $N_2P(x)+y$: we are searching the roots (x_0, y_0) of $N_2P(x)+y$ modulo 1 such that $|x_0| \leq T'$ and $|y_0| \leq 1/M'$. Coppersmith's method can fail for two reasons: either we do not find two vectors of small enough L_1 norm (this is detected at Step 13), or the two bivariate polynomials corresponding to these vectors share a factor (this is detected at Step 15). If Coppersmith's method does not fail, we go to Step 17: all the solutions of (1) that are in the considered subinterval must be roots of the y-resultant of the polynomials corresponding to the two small vectors that we found. If this polynomial has no integer root in the considered subinterval, then it means there were no solution to (1) in this subinterval; if it has roots, we test them to avoid those that are not solution to (1).

In the algorithm of Figure 1, the calculations are described with real numbers (at Steps 8 to 15). It is possible to replace these real numbers by others with finitely many bits, or by integers. At Step 8, we can replace the polynomial P by a polynomial $\tilde{P}$ whose coefficients approximate those of P: it suffices that $\max_{x\in[-T',T']} |N_2 \cdot [P(x) - \tilde{P}(x)]| = O(1/M)$. This can be ensured by taking the $O(\log M + \log N_2 + d \log T) = O(\log M + \log N_2 + d \log N_1)$ most significant bits of each coefficient. At Step 9, we can take $M' := \left\lfloor \frac{1}{2\epsilon+1/M} \right\rfloor$ instead of $M' := \frac{1}{\epsilon+1/M}$. The computations of Step 10 and 11 are then performed exactly. At Step 12, we need integer entries to use Theorem 1. Since all the entries of the matrix B are reals with finitely many bits, it suffices to multiply them by a sufficiently large power of 2: since we took reals with $O(\log M + \log N_2 + d \log N_1)$ bits to construct the polynomial $\tilde{P}$, multiplying by 2^ℓ with $\ell = O(\alpha(\log M + \log N_2 + d \log N_1))$ is sufficient. Once these modifications are performed, the remaining steps of the algorithm compute over the integers. In the following, we will keep the initial description (of Figure 1) to avoid unnecessary technicalities.

The main result of the paper is the following:

Theorem 3. *Let $N_1, N_2, M \in \mathbb{Z}$ and $c \in \mathbb{R}$ with finitely many bits. Let $t, d, \alpha \geq 0$. Given $N_1, N_2, M, c, t, d, \alpha$ as input, the algorithm of Figure 1 outputs all the solutions $x \in [\![0, N_1]\!]$ to (1). If test is never set to 0 at Step 15, the algorithm* *finishes in time* $\mathcal{P}(n_1, n_2, m, d, \alpha)\frac{N_1}{2^t}$*, as long as* $N_1 \leq MN_2, \log d = O(\alpha)$ *and:*

$$t \leq \min\left(n_1 - \frac{m+n_2+O(1)}{d+1}, n_1 - \frac{(n_1+n_2)^2}{4(m+n_2)}(1+\epsilon_1)+\epsilon_2\right),$$

with $n_1 = \log N_1, n_2 = \log N_2, m = \log M$ *and, for* α *growing to* ∞*:*

$$\begin{aligned}\epsilon_1 &= O(1/\alpha) + dO(1/\alpha^2)\\ \epsilon_2 &= \frac{1}{m+n_2}O(\alpha^2) + \frac{n_1}{m+n_2}(O(\alpha)+dO(1/\alpha))\\ &\quad +(n_1+n_2)(O(1/\alpha)+d(1/\alpha^3)) + mO(1/\alpha^2).\end{aligned}$$

Corollary 4 (Table Maker's Dilemma). *Let* $N_1 = 2^n$ *and* $N_2 = 2^{n+e}$ *with* $e \in \{0,1\}$*. Let* $\epsilon > 0$*. Suppose that* $d = 3, M = 2^n$*, and* $t = \frac{n}{2}(1-\epsilon)$*. Suppose also that test is never set to* 0 *at Step 15. Then one can choose* α, d *and* t *such that the algorithm of Figure 1 finds the solutions to (1) in time* $\mathcal{P}(n) \cdot 2^{\frac{n}{2}(1+\epsilon)}$*.*

Corollary 5 (Inverting the PRNG). *Let* $N_1 = 2^{n_1}$ *and* $N_2 = 2^{n_2}$*. Suppose that* $M = 2^{(n_1+n_2)^2}$*. Suppose also that test is never set to 0 at Step 15. Then one can choose* α, d *and* t *such that the algorithm of Figure 1 finds the solutions to (1) in time polynomial in* n_1 *and* n_2*.*

The following table gives the parameters providing the two corollaries above.

	M	α	d	t
Corollary 4	2^n	$O(n)$	3	$n/2$
Corollary 5	$2^{(n_1+n_2)^2}$	$O(n_1+n_2)$	$O((n_1+n_2)^2)$	$n_1+O(1)$

We can give a precise complexity estimate in Corollary 5 by using Theorem 1. The most expensive step of the algorithm is Step 12. The dimension of the lattice is $O(\alpha^2) = O((n_1+n_2)^2)$, its embedding dimension is $O(d\alpha^2) = O((n_1+n_2)^4)$, and the entries, when considered as integers (see the discussion above) are of length $O((n_1+n_2)^4)$. Therefore, the overall cost is $O((n_1+n_2)^{20})$.

Proof of correctness of the algorithm. Since we test any returned solution, it suffices to check that we do not miss any. Let $x_0 \in [\![0, N_1]\!]$ be such a solution. The integer x_0 belongs to at least one of the considered subintervals: let $[\![t_m - T', t_m + T']\!]$ be the smallest of them. For this subinterval, at Step 15 we have *test* = 1. Besides, for any $x \in [\![t_m - T', t_m + T']\!]$ we have, with $x' = x - t_m$:

$$\begin{aligned}|N_2P(x') \text{ cmod } 1| &\leq |N_2P(x') - N_2f(x/N_1) - c| + |(N_2f(x/N_1)+c) \text{ cmod } 1|\\ &\leq \epsilon + 1/M \leq 1/M'.\end{aligned}$$

As a consequence, we have $|N_2P(x_0-t_m) \text{ cmod } 1| \leq 1/M'$. Let $y_0 = -N_2P(x_0 - t_m) \text{ cmod } 1$. The pair (x_0-t_m, y_0) is a root of the bivariate polynomial $N_2P(x)+y$ modulo 1. It is therefore a root of any of the g_k's modulo 1, and of their integer linear combinations. In particular, it is a root of Q_1 et Q_2 modulo 1.

Besides, we have the inequalities $|x_0-t_m| \leq T'$ and $|y_0| \leq 1/M'$. Since the L_1 norms of the vectors $\mathbf{b}_1$ and $\mathbf{b}_2$ are smaller than 1, the pair (x_0-t_m, y_0) is a root of the polynomials Q_1 and Q_2 over $\mathbb{Z}$. It implies that if $R = \text{Res}_y(Q_1(x,y), Q_2(x,y))$ is non-zero, then we find x_0 at Step 17 of the algorithm.

5 Analysis of the Algorithm

This section is devoted to proving the complexity statement of Theorem 3. We suppose that *test* is never set to 0 at any Step 15. It implies that with a correct choice of the input parameters we always have $T' = T$ (except possibly when t_0 is close to N_1). The definition of M' gives $M' = O(M)$. Besides, Taylor's theorem and the condition $t \leq n_1 - \frac{m+n_2+O(1)}{d+1}$ of Theorem 3 ensure that for any $x \in [\![t_m - T, t_m + T]\!]$ the quantity ϵ computed at Step 9 satisfies $\epsilon = O(1/M)$. We thus have $M' = \Theta(M)$. To simplify, we identify M' with M and T' with T.

Our goal is to prove that if the conditions of Theorem 3 are fulfilled, then *test* if never set to 0 at Step 13, which means that the L_1 norms of the first two vectors output by L^3 are smaller than 1. Theorem 1 ensures that if the following condition is satisfied, then $\mathbf{b}_1$ and $\mathbf{b}_2$ will be short enough:

$$\sqrt{\frac{d\alpha(\alpha+1)}{2}} \cdot 2^{O\left(\frac{\log M + \log N_2 + d \log N_1}{\alpha - 1}\right)} \cdot 2^{\frac{\alpha(\alpha+1)}{2}} \cdot (\det L[\mathbb{P}])^{\frac{2}{\alpha(\alpha-1)}} < 1, \qquad (4)$$

where we used the classical relation between the L_1 and L_2 norms, took care of the fact that we have to scale the lattice to the integers to use Theorem 1, and $\mathbb{P}$ is the family of polynomials $\{x^i(N_2P(x) + y)^j, 0 \leq i + j \leq \alpha\}$.

Let B be the $\frac{\alpha(\alpha+1)}{2} \times \frac{d\alpha(\alpha+1)}{2}$ matrix where the entry $B[(i,j);(i',j')]$ is the coefficient of the polynomial $Q_{i,j}(x,y) = (xT)^i(N_2P(xT)+y/M)^j$ corresponding to the monomial $x^{i'}y^{j'}$. We order the rows and columns of B by increasing value of $i + dj$, and by increasing value of j in case of equality. We can write:

$$B = \begin{pmatrix} B_1 & 0 \\ B_2 & B_3 \end{pmatrix},$$

where B_1 is square, lower-triangular and corresponds to the rows such that $0 \leq i + dj \leq \alpha$, and B_3 is rectangular and corresponds to the rows such that $\alpha < i + dj \leq d\alpha$ and $0 \leq i + j \leq \alpha$. We easily obtain $\det L[\mathbb{P}] = |\det B_1| \cdot |\det B_3|$.

Lemma 6. *We have* $|\det B_1| = T^{\frac{1}{6d}(\alpha^3+O(\alpha^2))} M^{-\frac{1}{6d^2}(\alpha^3+O(\alpha^2))}$.

We are to bound the determinant of the matrix B_3 by using Theorem 2.

Lemma 7. *The matrix B_3 is bounded by the product of the quantities:*

$$wr_{i,j} = 2^j(d+1)^jK^jN_1^iN_2^j \quad \text{and} \quad wc_{i',j'} = T^{i'}/(M^{j'}N_1^{i'}N_2^{j'}),$$

with $i + j \in [\![0,\alpha]\!], i + dj \in [\![\alpha+1, d\alpha]\!]$ *and* $i' + j' \in [\![\alpha+1, d\alpha]\!]$.

Theorem 2 gives the inequality $|\det B_3| \leq 2^{O(\alpha^4)} \cdot \sqrt{d} \cdot R \cdot P$, where R is the product of the $wr_{i,j}$'s with $i + j \in [\![0,\alpha]\!]$ and $i + dj \in [\![\alpha+1, d\alpha]\!]$, and P is the product of the $(\dim B_3)$ largest $wc_{i,j}$'s with $i + dj \in [\![\alpha+1, d\alpha]\!]$.

Lemma 8. *We have the following relations, for α growing to infinity:*

$$\dim B_3 = \left(\tfrac{1}{2} - \tfrac{1}{2d}\right)(\alpha^2 + O(\alpha)),$$

$$R = 2^{O(\alpha^3)}(d+1)^{O(\alpha^3)} N_1^{\frac{d-1}{6d}(\alpha^3+O(\alpha^2))} N_2^{\frac{d^2-1}{6d^2}(\alpha^3+O(\alpha^2))}.$$

In order to bound P, we write $P \leq 2^{O(\alpha^3)} \prod_{k=\tau}^{k_{\max}} 2^{k \cdot c_k}$, with $c_k = \sharp\{(i,j), \alpha < i + dj \leq d\alpha$ and $k \leq (t - n_1)i - (m + n_2)j < k + 1\}$. We also define the parameter $\tau = -\sqrt{(n_1 - t)(m + n_2)}\alpha$. It is fixed so that $\sum_{k=\tau}^{k_{\max}} c_k \approx \dim B_3$, which means that we take sufficiently many columns. Finally we define:

$$k_{\max} = \begin{cases} \lfloor \alpha(t-n_1) \rfloor & \text{if } MN_2 \geq (N_1/T)^d, \\ \lfloor -(m+n_2)\alpha/d \rfloor & \text{if } MN_2 \leq (N_1/T)^d. \end{cases}$$

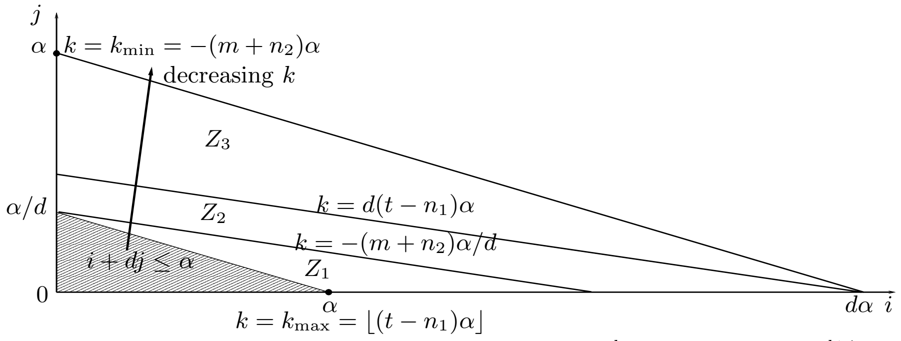

Fig. 2. Relations between i, j and k when $(N_1/T)^d \leq MN_2 \leq (N_1/T)^{d+1}$.

Figure 2 shows the relations between the variables i, j and k when $MN_2 \in [(N_1/T)^d, (N_1/T)^{d+1}]$. The dashed area corresponds to the submatrix B_1 of B and is not considered in the study of B_3. We split the set of valid pairs (i, j) (the largest triangle, without the dashed area) into three zones Z_1, Z_2 and Z_3 depending on the value of k. In the lemma below we consider only Z_1 and Z_2 since $\tau\alpha$ corresponds to an index k belonging to Z_2.

Lemma 9. *If $MN_2 \in [N_1/T, (N_1/T)^{d+1}]$ and α grows to infinity, then:*

$$\sum_{k=\tau}^{k_{\max}} c_k = \left(\frac{1}{2} - \frac{1}{2d}\right)\left(\alpha^2 + O(\alpha)\right) = (\dim B_3)(1 + O(1/\alpha))$$

$$\sum_{k=\tau}^{k_{\max}} kc_k = \frac{(n_1-t)d + (m+n_2) - 2d^2\sqrt{(n_1-t)(m+n_2)}}{6d^2}(\alpha^3 + O(\alpha^2)).$$

We can now batch the partial results of Lemmata 6, 8 and 9.

Theorem 10. *If $N_1/T \leq MN_2 \leq (N_1/T)^{d+1}$ and α grows to infinity, we have:*

$$\det L[\mathbb{P}] \leq 2^{O(\alpha^4)} \cdot (N_1N_2)^{\frac{\alpha^3}{6}+O(\alpha^2)} \cdot 2^{-\frac{1}{3}\sqrt{(n_1-t)(m+n_2)}(\alpha^3+O(\alpha^2))} \cdot (MT)^{O(\alpha^2)}.$$

By using (4) and Theorem 10, it is easy to end the proof of Theorem 3.

6 Experimental Data

A C implementation of the algorithm is available at the URL `http://www.loria.fr/~stehle/bacsel.html`. The code relies on GNU MP [9] for the integer arithmetic, on MPFR [19] for the *fp*-arithmetic and on a *fp*-L^3 available at the URL `http://www.loria.fr/~stehle/fpLLL-1.3.tar.gz`. The code is not meant to be efficient, but to be a proof of feasibility. The timings given in Figure 3 are thus overestimations of what may be possible with a more accurate implementation. Tuning the code is not an obvious task since the algorithm depends on many parameters. A previous implementation for the particular parameters $d = \alpha = 2$ was written by Paul Zimmermann and is available at the URL `http://www.loria.fr/~zimmerma/free/wclr-1.6.1.tar.gz`. This former implementation was used to find the worst cases for the correct rounding of the function 2^x over $[1/2, 1)$ in double extended precision (64-bits mantissæ) for all rounding modes. For example, the worst case for the rounding to nearest is:

$$2^{\frac{15741665614440311501}{2^{64}}} = \underbrace{1.110\ldots110}_{64}1\underbrace{0\ldots0}_{63}11\ldots$$

The corresponding computation lasted a time equivalent to ≈ 7 years on a single Opteron 2.4 GhZ. With the new code and $d = 3$ and $\alpha = 2$, this computation should be speeded up significantly. Nevertheless, for the application to the table maker's dilemma, $n = 64$ seems to be the bound of feasibility. In particular, the quadruple precision (113-bit mantissæ) remains far out of reach.

n	d	α	M	T	time
53	3	2	2^{53}	$2^{20.45}$	7.7 days
64	2	2	2^{64}	$2^{23.95}$	3.2 years (estimated time)
64	3	2	2^{64}	$2^{24.60}$	2.6 years (estimated time)
113	3	2	2^{113}	$2^{44.45}$	$3 \cdot 10^9$ years (estimated time)

Fig. 3. Estimated time to find a worst case of exp over $[1/2, 1)$, on an Opteron 2.4 GhZ.

For the inversion of the PRNG, the complexity, though polynomial, remains too high for extensive computations with a growing value of n. As an example, we found the seed $x_0 = 17832265507654526358 \cdot 2^{-64} \in [1/2, 1)_{64}$ such that:

$$\exp x = b_{-1}b_0.b_1b_2\ldots b_{64}c_1c_2c_3\ldots c_{400}\ldots,$$

with the 400 bits c_i known. The computation was performed in time equivalent to less than one week on an Opteron 2.4 GhZ.

For the transcendental functions we tried, we did not encounter a single failure of Coppersmith's method. On the contrary, the method failed for all algebraic functions f that we tried: in this case the running-time is between what is expected and that of an exhaustive search, and it does not seem to decrease when we increase the parameter d. We have two heuristic explanations for this phenomenon: firstly, for some algebraic functions, there can be too many solutions to write them in polynomial time; secondly, as described in Section 2, it would give a polynomial time algorithm for integer factorization.

7 Generalizations and Open Problems

One can extend the algorithm and its analysis to the case of functions of several variables [20]. For the PRNG, if the input precision is of n bits for all variables, and the first n bits of the output are kept hidden, the number of bits needed to recover the multivariable seed is $O(n^{k+1})$, where k is the number of variables.

An open problem related to our algorithm is to prove Theorem 3 without the heuristic assumption on the resultant computation. Since the method fails in practice for some functions (in particular for algebraic functions), the task would be to give a sufficient condition on the function f for the method to work.

An intermediate question is to determine under which conditions Coppersmith's method can be made provable for multivariate polynomials.

Another interesting problem is to determine if the $O(n^2)$ bound is the best possible: can we invert in polynomial time the PRNG with significantly fewer than n^2 bits? This bound matches the one of [17] and might thus be considered as somehow natural. The gap between the probabilistic injectivity of the PRNG ($m = O(n)$) and its polynomial-time invertibility ($m = O(n^2)$) is puzzling.

Acknowledgments. The author thanks Guillaume Hanrot, Vincent Lefèvre and Paul Zimmermann for their helpful comments. The writing of the present paper was completed while the author was visiting the University of Sydney, whose hospitality is gratefully acknowledged.

References

1. D. H. Bailey and R. E. Crandall. Random generators and normal numbers. *Experimental Mathematics*, 11(4):527–546, 2002.
2. L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*, 15(2):364–383, 1986.
3. D. Boneh and G. Durfee. Cryptanalysis of RSA with private key d less than $N^{0.292}$. *IEEE Transactions on Information Theory*, 46(4):233–260, 2000.
4. D. Boneh, G. Durfee, and N. Howgrave-Graham. Factoring $n = pq^r$ for large r. In *Proc. of Eurocrypt 1999*, volume 1666 of *LNCS*, pages 326–337. Springer-V., 1999.
5. É. Borel. Les probabilités dénombrables et leurs applications arithmétiques. *Rendiconti del Circolo Matematico di Palermo*, 27:247–271, 1909.
6. R. Brent. Fast multiple precision zero-finding methods and the complexity of elementary function evaluation. *Journal of the ACM*, 23:242–251, 1976.
7. D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4):233–260, 1997.
8. M. Ernst, E. Jochens, A. May, and B. de Weger. Partial key exposure attacks on RSA up to full size exponents. In *Proc. of Eurocrypt 2005*, number 3494 in LNCS, pages 371–386. Springer-V., 2005.
9. T. Granlund. The GNU MP Bignum Library. Available at `http://www.swox.com/`.
10. R. Kannan, A. K. Lenstra, and L. Lovász. Polynomial factorization and nonrandomness of bits of algebraic and some transcendental numbers. In *Proc. of STOC 1984*, pages 191–200. ACM, 1984.
11. V. Lefèvre. *Moyens arithmétiques pour un calcul fiable.* PhD thesis, ÉNS Lyon, 2000.
12. A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:513–534, 1982.
13. L. Lovász. *An Algorithmic Theory of Numbers, Graphs and Convexity.* SIAM Publications, 1986. CBMS-NSF Regional Conference Series in Applied Mathematics.
14. A. May. *New RSA Vulnerabilities Using Lattice Reduction Methods.* PhD thesis, University of Paderborn, 2003.
15. A. May. Computing the RSA secret key is deterministic polynomial time equivalent to factoring. In *Proc. of Crypto 2004*, volume 3152 of *LNCS*, pages 213–219. Springer-V., 2004.
16. H. Minkowski. *Geometrie der Zahlen.* Teubner-Verlag, 1896.
17. Yu. V. Nesterenko and M. Waldschmidt. On the approximation of the values of exponential function and logarithm by algebraic numbers. *Matematicheskie Zapiski*, 2:23–42, 1996.
18. P. Nguyen and D. Stehlé. Floating-point LLL revisited. In *Proc. of Eurocrypt 2005*, volume 3494 of *LNCS*, pages 215–233. Springer-V., 2005.
19. The SPACES Project. MPFR, a LGPL-library for multiple-precision floating-point computations with exact rounding. Available at `http://www.mpfr.org/`.
20. D. Stehlé. *Algorithmique de la réduction de réseaux et application à la recherche de pires cas pour l'arrondi de fonctions mathématiques.* PhD thesis, Université Nancy 1, 2005.
21. D. Stehlé, V. Lefèvre, and P. Zimmermann. Searching worst cases of a one-variable function. *IEEE Transactions on Computers*, 54(3):340–346, 2005.

Missing Proofs

Proof of Theorem 2. Recall that the determinant is defined by (2). If one of the wr_i's is zero, the determinant is obviously zero and the result holds. Suppose now that all wr_i's are positive. Let B' be the matrix B after having divided the i-th row by wr_i for all i. Wlog we suppose that $wc_1 \geq wc_2 \geq \ldots \geq wc_n$ (otherwise we perform a permutation of the columns, which does not change the determinant). It suffices to show that: $\det B' \leq 2^{\frac{d(d-1)}{2}} \sqrt{n} \cdot wc_1 \cdot \ldots \cdot wc_d$.

We prove it by induction on d and n. If $n < d$, this is obvious since the rows must be linearly dependent, so that $\det B' = 0$. If $d = 1$, the determinant is exactly the length of the unique vector. Suppose now that $n \geq d \geq 2$. We apply a permutation on the rows of the matrix B' in order to have $|B'_{1,1}| \geq |B'_{i,1}|$ for any i. This last operation does not change the determinant. For all $i \geq 2$, we perform the transformation $B'_i := B'_i - \frac{B'_{i,1}}{B'_{1,1}} B'_1$, which does not change the determinant either (B'_i is the i-th row of the matrix B'). Wlog we suppose that $B'_{1,1} \neq 0$: otherwise all the $B'_{i,1}$'s are zero, and we obtain the result by induction on n by removing the first column. The transformations we have performed on the matrix B' have given a new matrix B' for which $|B'_{1,1}| \leq wc_1$, $B'_{i,1} = 0$ for any $i \geq 2$ and $|B'_{i,j}| \leq 2wc_j$, for any pair (i,j). This last statement follows from the fact that for any $i \leq d$, we have $\left|\frac{B'_{i,1}}{B'_{1,1}}\right| \leq 1$. We apply the result inductively on the $(d-1) \times n$ matrix B'' at the bottom of the matrix B':

$$\det B'' \leq 2^{\frac{(d-1)(d-2)}{2}} \sqrt{n} \cdot (2wc_2) \cdot \ldots \cdot (2wc_d) \leq 2^{\frac{d(d-1)}{2}} \sqrt{n} \cdot wc_2 \cdot \ldots \cdot wc_d.$$

Proof of Corollary 4. Let $m = n_1 = n_2 = n$ and $d = 3$. If α grows to infinity, we have: $\epsilon_1 = O(1/\alpha), \epsilon_2 = 1/n \cdot O(\alpha^2) + O(\alpha) + nO(1/\alpha)$. We fix α sufficiently large to ensure that the terms "$O(1/\alpha)$" of ϵ_1 and ϵ_2 become smaller than ϵ (with absolute values). We get, for n growing to infinity: $|\epsilon_1| \leq \epsilon, |\epsilon_2| \leq O(1) + n\epsilon$. Finally, the equation to be satisfied becomes, for n growing to infinity:

$$t \leq \min\left(\tfrac{n}{2} - O(1), n - \tfrac{n}{2}(1+\epsilon) - n\epsilon + O(1)\right).$$

For n larger than some constant, this inequality is satisfied if $t = n(1-4\epsilon)/2$.

Proof of Corollary 5. Let $d = O((n_1+n_2)^2)$ and $\alpha = O(n_1+n_2)$. Then for $n_1 + n_2$ growing to infinity, we have $\epsilon_1 = O(1)$ and $\epsilon_2 = O(1)$. The equation to be satisfied becomes $t \leq n_1 - O(1)$, for $n_1 + n_2$ growing to infinity.

Proof of Lemma 6. The determinant of B_1 is:

$$\prod_{i+dj \leq \alpha} (xT)^i \cdot (P(xT) + \frac{y}{M})^j [x^i y^j] = \prod_{i+dj \leq \alpha} T^i M^{-j} = (T^{\frac{1}{6d}} M^{\frac{-1}{6d^2}})^{\alpha^3 + O(\alpha^2)}.$$

Proof of Lemma 7. The row (i,j) of B_3 with $i + j \in [\![0, \alpha]\!]$ and $i + dj \in [\![\alpha+1, d\alpha]\!]$ corresponds to the polynomial $(xT)^i \left(P(xT) + \frac{y}{M}\right)^j$. Its column $(i' + j')$ with $i' + j' \in [\![\alpha+1, d\alpha]\!]$ corresponds to the coefficient of this polynomial for the monomial $x^{i'} y^{j'}$. We have the following inequalities:

$$\begin{aligned}
|B_3[(i,j);(i',j')]| &\leq (xT)^i \left(P(xT) + \frac{y}{M}\right)^j [x^{i'} y^{j'}] \\
&\leq T^i \left[\sum_{k=0}^{j} \binom{j}{k} (P(xT))^{j-k} M^{-k} y^k\right] [x^{i'-i} y^{j'}] \\
&\leq 2^j T^i M^{-j'} \left[\sum_{k=0}^{d} a_k T^k x^k\right]^{j-j'} [x^{i'-i}].
\end{aligned}$$

Besides, we have $a_k \leq K\frac{N_2}{N_1^k}$, which gives that:

$$\begin{aligned}
|B_3[(i,j);(i',j')]| &\leq 2^j K^{j-j'} T^i N_2^{j-j'} M^{-j'} \left[\sum_{k=0}^{d} \left(\frac{T}{N_1}\right)^k x^k\right]^{j-j'} [x^{i'-i}]. \\
&\leq 2^j K^{j-j'} T^i N_2^{j-j'} M^{-j'} \left(\frac{T}{N_1}\right)^{i'-i} \left[\sum_{k=0}^{d} x^k\right]^{j-j'} [x^{i'-i}] \\
&\leq \left(2^j (d+1)^j K^j N_1^i N_2^j\right) \cdot \left(\frac{T^{i'}}{M^{j'} N_1^{i'} N_2^{j'}}\right).
\end{aligned}$$

Proof of Lemma 8. For the first relation, one can write:

$$\dim B_3 = \dim B - \dim B_1 = \tfrac{1}{2}(\alpha^2 + O(\alpha)) - \tfrac{1}{2d}(\alpha^2 + O(\alpha)).$$

The proof of the second relation is similar:

$$\begin{aligned}
\prod_{\substack{i+j\,\in\,[\![0,\alpha]\!]\\ i+dj\,\in\,[\![\alpha+1,d\alpha]\!]}} N_1^i N_2^j &= \left(\prod_{i+j\leq\alpha} N_1^i N_2^j\right) \cdot \left(\prod_{i+dj\leq\alpha} N_1^i N_2^j\right)^{-1} \\
&= N_1^{\frac{d-1}{6d}(\alpha^3+O(\alpha^2))} N_2^{\frac{d^2-1}{6d^2}(\alpha^3+O(\alpha^2))}.
\end{aligned}$$

Proof of Lemma 9. We restrict ourselves to the first statement and to the case where $MN_2 \in [(N_1/T)^d, (N_1/T)^{d+1}]$. The other proofs are similar. We have:

$$\begin{aligned}
\sum_{k=\tau}^{k_{\max}} c_k &= \sum_{i=0}^{\alpha-1} \sharp \left[\!\!\left[\left\lfloor \frac{\alpha-i}{d} \right\rfloor, \left\lfloor \frac{(t-n_1)i-\tau}{m+n_2} \right\rfloor\right]\!\!\right] + \sum_{i=\alpha}^{\left\lfloor \frac{\tau}{t-n_1} \right\rfloor} \sharp \left[\!\!\left[0, \left\lfloor \frac{(t-n_1)i-\tau}{m+n_2} \right\rfloor\right]\!\!\right] \\
&= \sum_{i=0}^{\left\lfloor \frac{\tau}{t-n_1} \right\rfloor} \frac{(t-n_1)i-\tau}{m+n_2} - \sum_{i=0}^{\alpha-1} \frac{\alpha-i}{d} + O(\alpha) \\
&= \frac{t-n_1}{2(m+n_2)} \left[\frac{\tau}{t-n_1} + O(1)\right]^2 - \frac{\tau}{m+n_2} \left[\frac{\tau}{t-n_1} + O(1)\right] - \frac{\alpha^2}{2d} + O(\alpha) \\
&= \frac{\alpha^2}{2}(1 + O(1/\alpha)) - \frac{\alpha^2}{2d} + O(\alpha) = \left(\frac{1}{2} - \frac{1}{2d}\right)(\alpha^2 + O(\alpha))
\end{aligned}$$

End of the Proof of Theorem 3. Since $d = 2^{O(\alpha)}$, Theorem 10 gives that:

$$|\det L[\mathbb{P}]| \leq 2^{O(\alpha^4) + (n_1+n_2)\left(\frac{\alpha^3}{6} + O(\alpha^2)\right) - \frac{1}{3}\sqrt{(n_1-t)(m+n_2)}(\alpha^3+O(\alpha^2)) + (m+t)O(\alpha^2)}.$$

As a consequence, to ensure that 4 is satisfied, it suffices that:

$$\begin{aligned}
O(\alpha^2) + (m+n_2+dn_1)O(1/\alpha) + (n_1+n_2)(\alpha+O(1)) + (m+t)O(1)& \\
< 2\sqrt{(n_1-t)(m+n_2)}(\alpha+O(1)),&
\end{aligned}$$

which is implied by the simpler equation:

$$O(\alpha) + (n_1+n_2)\left(1 + O\left(\frac{1}{\alpha}\right) + dO\left(\frac{1}{\alpha^2}\right)\right) + mO\left(\frac{1}{\alpha}\right) < 2\sqrt{(n_1-t)(m+n_2)}.$$

This last equation is itself implied by the condition of Theorem 3.