Piercing the Deepest Mathematical Mystery

Vincent Granville, Ph.D.
vincentg@MLTechniques.com
www.MLTechniques.com
Version 1.0, January 2025

1 Introduction

The style of my presentation is unusual, keeping the mystery secret until later in this article, focusing first on new theory created to help solve this problem. The material shared here is accessible to engineers, computer scientists, and AI practitioners. The problem in question has been attacked for a long time, with no progress. It is deemed more difficult than proving the Riemann Hypothesis, yet its formulation can be understood by kids in elementary school. This article sets a significant milestone towards a final solution, with the light visible at the end of the tunnel. This document serves as blueprint, featuring the architecture of a highly constructive yet difficult proof, based on new theoretical developments and concepts.

The goal is to transition from experimental to theoretical number theory. That said, I use illustrations with numbers that have more than 2^{10000} digits, that is, larger than 2^m where $m = 2^{10000}$. No amount of computing power will ever be able to handle such numbers. Obviously, I use some tricks here, and large numbers make the patterns and their complexity easier to detect with the naked eye. But the same strong pattern structure is present in much smaller numbers, as well as in numbers with infinitely many digits. And most importantly these patterns can be explained with theoretical arguments based on a new theory: the iterated auto-convolutions of infinite strings of characters and their congruence classes, akin to p-adic numbers. Finally, there is a strong connection to the deepest aspects of discrete dynamical systems approaching their chaotic regime. Also, there are practical applications to cryptography, and computations are doubled-checked using external libraries.

2 A new type of string operators

By string, I mean a sequence of characters as found in any programming language. I focus on binary strings, with an alphabet consisting of two characters, labeled as '0' and '1' for simplicity. Strings with at least two characters must start with '1'. It is tempting to introduce the theory that follows using very abstract concepts, also generalizing to alphabets with more than two characters. However, since the goal is to prove a result in number theory, I will focus only on the minimum needed to make the desired progress, using familiar terminology. Note that strings can have an infinite number of characters, on the right side.

2.1 String class

The first concept is that of string class. Two finite strings are said to be equivalent, that is, belonging to a same class, if they are identical after stripping any trailing '0' that follows the last '1' on the right side. For instance, the strings '10110100' and '101101' are equivalent. The version that ends with '1' on the right is called the canonical form. String classes are similar to modulo classes in arithmetic. Whenever I use the word string in the remaining of my discussion, it is to be understood as the associated string class in its entirety. You may use the notation $\{S\}$ to distinguish the string S from its class. We will stick to S in both cases, for simplicity.

2.2 Truncation, number representation, and convergence

The concept is trivial to engineers and programmers, but less obvious to mathematicians. In our context, truncation is always performed on the right side, keeping the left part of the string unchanged. From a mathematical viewpoint, a truncated string is similar to a residue in modulo classes, except that arithmetic residues consist of digits on the right, while string residues (the result of truncation) consists of characters on the left. In some sense, string classes have analogies to p-adic numbers; both can be infinite.

I now introduce the concept of number representation. I use it to define string convolution, but it could be avoided at the expense of increased abstraction. A string S represents the number x if the characters in S match the binary digits of x in the same order. The notations $S(x)$ or $S \equiv x$ mean that S represents x. The mapping is one-to-one at the class level: for instance, '1011' or '10110' represents both $13, 26, 6.5, 3.25$ and so on. The correspondence is up to a factor 2^m where m is a positive or negative integer.

Finally, a sequence of strings (S_n) converges to S if the number ρ_n of consecutive identical characters at the beginning (on the left) between S_n and S_{n+1} increases and becomes infinite as $n \to \infty$, matching those of S. If x_n represents S_n with $x_n \to x$, the following notations (shortcuts) may be used: $S_n \to S$ or $S_n \to x$. The notation is abusive since we are dealing both with string and number classes rather than specific instances. For instance, convergence to $\sqrt{2}$ means convergence to any number of the form $2^m\sqrt{2}$ where m is a positive or negative integer, including $m = \infty$.

2.3 String convolution and square root

If the string classes S and T represent respectively the numbers x and y, then the convolution is defined as $S * T = x \cdot y$. Again, this is up to a factor 2^m where m is a positive or negative integer. When $S = T$, I use the word auto-convolution. Moving forward, we deal with auto-convolution exclusively. The core mechanism in this product is the carry-over computations that propagate chaos from right to left, but usually with dissipation. This will soon become evident.

The inverse of the auto-convolution is the square root operator. More specifically, S is the square root of T if $S * S = T$. Both '1' are $S(\sqrt{2})$ = '1011010100...' are square roots of '1'. The square roots of '11' are $S(\sqrt{3})$ and $S(\sqrt{6})$. Note that $2\sqrt{3}$ also represents the square root of '11', but $S(2\sqrt{3}) = S(\sqrt{3})$; the two numbers belong to the same class and have identical strings.

In mathematical terms, we have a topological homeomorphism between string classes and number classes. The latter are similar to p-adic numbers. Both form a multiplicative abelian group, with the convolution product for strings (including infinite strings), equivalent to the algebraic product for numbers. The concepts of limit and convergence are well defined in both spaces, with strings inheriting the definition from the homeomorphism.

2.4 Well-balanced strings

A formal definition is not needed at this stage. In layman's terms, well-balanced means the absence of strong patterns in the string in question. Many definitions are possible for finite strings. For an infinite string $S(x)$, the classic definition is that x (the class of numbers that it represents) must be normal in base 2. It follows that the set of infinite strings that are not well-balanced have Lebesgue measure zero: they are incredibly rare, yet still as numerous as real numbers based on Cantor's definition of countable.

In practice, we encounter the following situation: $S_1 \subset S_2 \subset \ldots$ is an infinite sequence of embedded strings of increasing lengths. Each finite string is well-balanced (in some sense), making the limit S_∞ well-balanced, that is, free of obvious biases and patterns.

One would think that the auto-convolution of a well-balanced string is well-balanced, as this operator tends to increase or maintain mixing in the resulting sequence. While this is true in most cases, there are infinitely many exceptions. The most notorious counterexample is $S(\sqrt{2}) * S(\sqrt{2}) =$ '1'. All mathematicians believe that $S(\sqrt{2})$ is well-balanced. However this fact is still unproved to this day, not even in this paper, even when using the weakest possible definition of well-balanced.

3 Infinite sequences of iterated auto-convoluted strings

In this section, I discuss the new machinery leading to the mathematical breakthrough. The link to the number theory conjecture addressed here is established in section 4. Readers with a good mathematical intuition may be able to assess the significance of the material shared here, and what it really is all about. If not, the secret is revealed in section 4.

For each integer $n > 0$, we create an infinite sequence of iterated auto-convoluted strings, where the initial string S_n^0 consists of $n+1$ characters, all equal to '0' except the first and last one equal to '1'. In other words,

$$S_n^{k+1} = S_n^k * S_n^k, \quad k = 0, 1, \ldots \tag{1}$$

If for instance $n = 1000$, it very quickly leads to titanic strings of incredible size, unmanageable even if you had access to the entire computing power in the universe for trillions of years. To avoid this problem, I truncate the strings, keeping only the first $2n$ characters on the left at each iteration. One can prove that this truncation preserves the first $n-3$ characters in the following sense:

Theorem 3.1 *Let x_n (defined up to a factor 2^m where m can be any positive or negative integer) be the number representing the string S_n^n using truncation to first $2n$ characters at each iteration in recursion (1). Then, the first $n-3$ digits of x_{n+k} are identical regardless of $k = 0, 1, 2$ and so on. This is also true when $k \to \infty$.*

This theorem is easy to prove. I choose m such that $2 \leq x_n < 4$, regardless of n. Thus m depends on n. Also this condition uniquely defines x_n. The reason for this choice will become obvious in section 4, but I prefer to

keep the secret for now. An obvious consequence of the truncation is the fact that for a fixed n, the sequence (S_n^k) indexed by k eventually becomes periodic, though the period is gigantic. For an empirical verification of Theorem 3.1, the Python code also performs a precision check using an external library for computations, based on the exact value of x_∞.

Theorem 3.1 can be restated as follows: the string S_n^n converges as $n \to \infty$, gaining about one additional character that will remain unchanged when moving from n to $n+1$, despite the truncation. In a nutshell, the truncation consists of replacing the characters beyond the first $2n$, by an infinite string of '0'. But you could use any arbitrary string instead of zeros, without impacting the first $n-3$ characters in subsequent iterations. The following is a direct consequence of the definition of auto-convolution and its inverse (square root operator).

Theorem 3.2 *Let k be a fixed integer and x_∞ be the limit number representing S_n^n as $n \to \infty$, assuming that the sequence (S_n^n) converges. Then (S_n^{n+k}) converges to $(x_\infty)^{2^k}$ as $n \to \infty$. Here, k can be positive or negative. This is true with or without truncation, and regardless of the initial configurations S_n^0 for $n = 1, 2$ and so on. Extra care is needed with $k < 0$ as fractional powers of a string are not uniquely defined.*

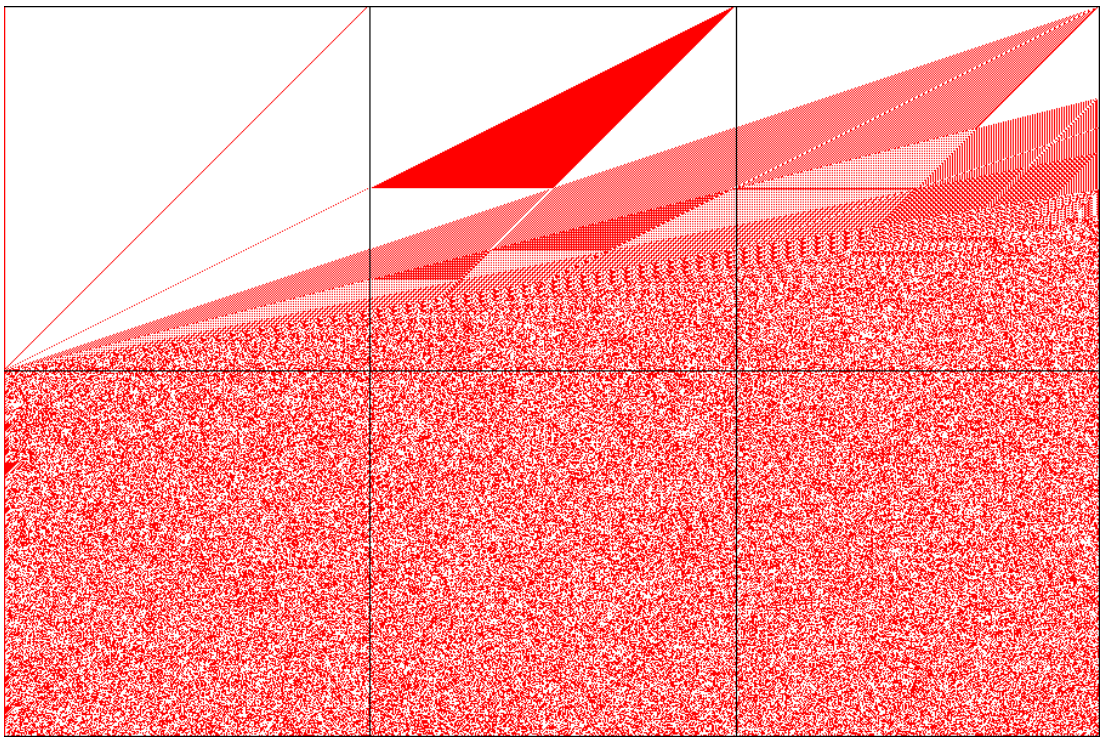

Figure 1: First $3n$ characters of S_n^k, for $n = 300$ and $1 \leq k \leq 2n$ (one line per k)

3.1 Visualizing iterated auto-convoluted string sequences

Figure 1 shows S_n^k truncated to $3n$ characters, with $n = 300$. Each line in the picture corresponds to a specific value of k, starting with $k = 1$ at the top, and ending with $k = 2n$ at the bottom. Thus the size of the image is 900×600 pixels. A red pixel represents '1', and a white one represents '0'. The initial string S_n^0, also called the seed, consists of $n+1$ characters all zeros except the first and last ones. This explains why there is so much white on the top half, especially on the left side.

As k increases, more and more '1' start to pop up from the right side, and propagate to the left. Patterns become more and more complex until $k = n$. Beyond $k = n$, we are in full chaotic mode. We do not care about the bottom half of the picture (the chaotic zone). Likewise, we do not care about the rightmost n pixels. They could be anything, yet they have no impact on the final conclusion. Indeed, we only need the first $2n$ pixels on the left, and we only care about the top left block ($n \times n$ pixels). However the color of these pixels is impacted by those in the middle block.

The framework is identical to the dynamical systems described in my book about chaos [5]. If we keep $2n$ digits at all times, the precision decreases by about one digit per iteration when incrementing k, thus we can expect to still have n correct digits after n iterations. Beyond that point, the precision continues to deteriorate, and when $k = 2n$, we have lost all precision. The process still continues as k increases, but it is like starting with a new seed every n iterations or so.

Finally, Figure 2 shows a more granular view, zooming in on a portion of the top left block in Figure 2 to further reveal the patterns. Here I kept only the $2n$ first leftmost digits when truncating (rather than $3n$ in Figure 2), yet the pixels are identical.

As a side note, it would be interesting to check if the sequence (S_n^k) indexed by k (with n fixed) is dense, ergodic, and find its invariant measure in the string distribution space. For each k, you first need to choose a unique number to represent S_n^k, (say) the one in $[2, 4[$, and then check if the corresponding sequence of numbers is dense in $[2, 4[$. Here truncation is not allowed, otherwise the answer to the first question is obviously negative.

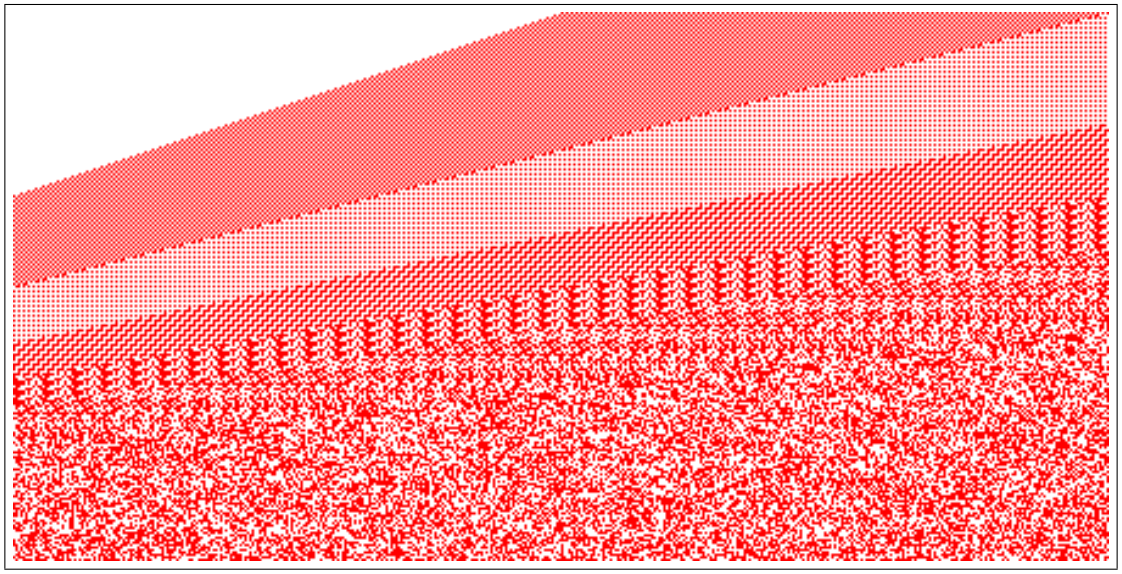

Figure 2: Zoom in on bottom right corner of top left block in Figure 1

3.2 Fundamental result about the number of zeros and ones

I briefly discussed chaotic dynamical systems in section 3.1, mentioning that for the string sequence (S_n^k) indexed by k with fixed n, chaos starts at $k = n$. One of the novelties is that we are dealing with strings rather than real numbers or vectors. The choice of the seed S_n^0, in this case the most rudimentary string with $n+1$ characters, is entirely responsible for the observed behavior. While patterns are obvious in Figure 1 and especially Figure 2, in this section they are amplified to a whole knew level, with deeper connections to dynamical systems. This is possible thanks to working with one of the simplest iterated auto-convolution systems for strings, with truncation that luckily preserves the interesting properties throughout the first n iterations ($k \leq n$, but not thereafter). It leads to spectacular discoveries. Before starting, let me introduce the following constants, linked to the abscissas of the dashed vertical lines in Figure 3:

$$\rho_1 = \frac{1}{2}, \quad \rho_2 = \frac{2}{3}, \quad \rho_3 = \frac{3}{4}, \quad \rho_4 = \frac{4}{5}, \quad \rho_5 = \frac{5}{6}, \quad \rho_6 = \frac{6}{7}, \quad \ldots \tag{2}$$

In Figure 3, if you look at the red dots, the proportion of '1' in the first n characters of the string S_n^k (with n fixed), using the usual truncation mechanism, follows this path:

- It starts at a very low close to 0% (after all, the seed only has two '1'),
- then abruptly starts growing steadily at about $k = \rho_2 n$.
- Around $k = \rho_3 n$, the blue and red curves separate, but the slope stays the same on the red curve until $k = \rho_4 n$. At this point, about 20% of the characters are '1' on the red trajectory.
- Suddenly at $k = \rho_4 n$ until $k = \rho_5 n$, the slope is steeper, bringing the proportion of '1' to about 30% at $k = \rho_5 n$.
- From there, two branches open up, with the proportion of '1' still increasing.
- At $k = \rho_6 n$, more bifurcations appear. Eventually the growth tapers off, more and more branches open up faster and faster,
- and when $k = \rho_\infty n = n$, we enter the fully chaotic zone with the proportion of '1' oscillating around 50% moving forward.

At the same time (not shown in the picture), the proportion of '0' moves in the opposite direction (flipped side), starting at almost 100% to eventually drop and likewise, oscillate around 50% when reaching the chaotic zone.

While not visible on the picture, S_n^k has exactly two '1' between $k = 0$ (the seed) and $k = \rho_1 n$, and exactly three '1' between $k = \rho_1 n$ and $k = \rho_2 n$ making the proportion of '1' virtually zero when n is large and $k < \rho_2 n$. The red dots correspond to even values of k, and the blue dots to odd values. All of this is classic in dynamical systems that start gentle and evolve to chaos. I summarize it in the following theorem.

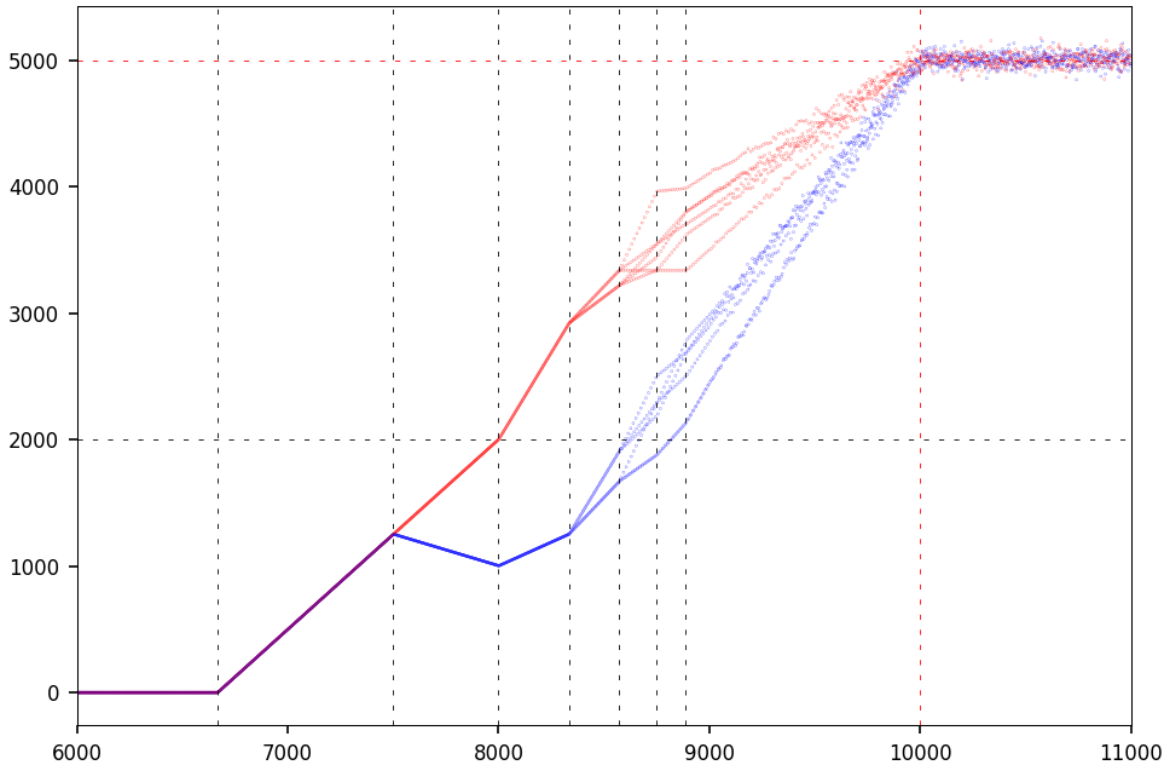

Figure 3: Number of '1' in first n chars of S_n^k, with $n = 10^4$ and k on the X-axis

Theorem 3.3 *The string sequence (S_n^k) indexed by k, with n fixed, constitutes a dynamical system in non-chaotic phase when $k < n$. In this phase, the number of ones (or zeros) follows a regime with changing points governed by schedule (2). In particular:*

- *Prior to branching ($k < \rho_5 n$ if k is even) the proportion of ones steadily increases as k increases, to reach above 29.1% (approximately), and never drops below that threshold thereafter when $k > \rho_5 n$.*
- *The 29.1% threshold in question is absolute. It holds for any n large enough, including $n = \infty$.*

Determining a precise value for the 29.1% threshold may prove difficult. A proof targeting 20% or 25% is likely to be easier to reach. Table 1 displays values of ν_3, ν_4 and ν_5 with n ranging from 1000 to 8000. These values represent the proportion of '1' in the first n digits of S_n^k when k is the closest even integer respectively to $\frac{3}{4}n$, $\frac{4}{5}n$ and $\frac{5}{6}n$. That is, on the red orbit in Figure 3, when it crosses respectively the second, third, and fourth vertical dashed lines from the left. The takeaway is that these values are absolute as stated in theorem 3.3, not really depending on n. In particular, the 29.1% mentioned earlier corresponds to $\nu_5(n)$.

n	1000	2000	4000	8000
$\nu_3(n)$	0.127	0.126	0.125	0.125
$\nu_4(n)$	0.202	0.201	0.200	0.200
$\nu_5(n)$	0.294	0.291	0.292	0.291

Table 1: Sample values of ν_3, ν_4, ν_5

Bumps or dips (if there are any) on the smooth sections of the blue and red paths in Figure 3 are of very small amplitude and duration (invisible), essentially having no impact. Also, the only numbers that matter are those attached to the red path. Ignoring the blue path has no impact on the final conclusion, about the guaranteed minimum proportion of '1' when $k = n$: if n is odd, then at the next sequence $n + 1$ is even and on a red path, with S_n^n and S_{n+1}^{n+1} having the same number of '1' in the first n digits.

3.2.1 Mechanics of the bifurcation process

So far I mentioned two paths: the red, and the blue one. They are actually multi-paths, since more and more bifurcations show up as we move to the right in Figure 3. Let's focus on the red multi-path (similar arguments apply to the blue multi-path). At $k = \rho_5 n$, we have a new split, with 2 options: one path steeper than the other one. The steeper one corresponds to $k \equiv 0 \bmod 4$, and the other one to $k \equiv 2 \bmod 4$.

Let us follow the steeper path. The next split, taking place at $k = \rho_6 n$, now offers 3 options: steep climb, moderate climb, or nearly flat (until the next split at $k = \rho_7 n$). Which path to follow depends this time on the value of k modulo 12, with $k \equiv 4 \bmod 12$ for the steeper climb. At the end of that climb, the proportion of '1' jumped from 29.1% to above 33.4%. Moving further to the right, subsequent splits offer more bifurfactions, and the full combinatorial nature of the problem becomes evident (and also discussed in section 4.1).

To address the most challenging aspects of the problem in order to get the strongest possible conclusion, one also has to look at paths going down, and how many segments are needed to recover from a decline. Eventually, the width of the chaotic band beyond $k = n$ shrinks and becomes a flat line when $n \to \infty$ (if the plot size is kept constant), because that width is proportional to $\sqrt{n}$ and $\sqrt{n}/n \to 0$. But proving this fact still seems completely out of reach at this point. If proved, it would show that the proportion of '1' at $k = n$ tends to 50%, a conjecture that all mathematicians believe in. Looking at the link between infinite iterated self-convolutions and the limiting Gaussian distribution, could shed more light, with the nearly-Gaussians converging to a singular distribution centered at 50% as $n = \infty$.

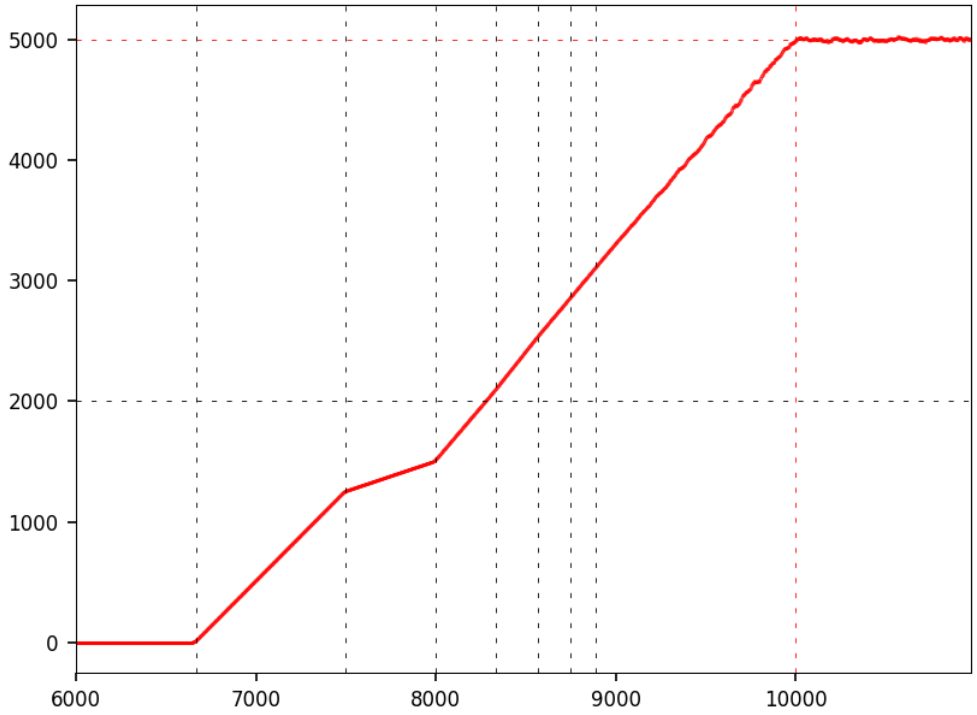

Figure 4: Averaging the paths within each band in Figure 3

3.2.2 Interesting variable-length moving averages

The goal here is to get an idea of the general trend in Figure 3. I focus on counting the '1's, but it also applies in a mirrored way to the character '0'.

Each path within a band delimited by two consecutive vertical dashed lines in Figure 3, is linear (perfectly linear when $n = \infty$). Also, the number of paths in each band, starting on the left and moving to the right, are 1, 1, 2, 2, 4, 12, and so on. Except for the first two values, these are factorials multiplied by 2. Then, based on (2), the widths of each band, from left to right, are proportional to

$$\frac{1}{1 \cdot 2}, \quad \frac{1}{2 \cdot 3}, \quad \frac{1}{3 \cdot 4}, \quad \frac{1}{4 \cdot 5}, \quad \cdots$$

The proportionality factor is n. If you average all the paths within each band, and concatenate the resulting segments across all the bands, you get a variable-length moving average that shows how the number of '1's is trending up as k increases. The result is pictured in Figure 4. Slight departure from a perfect straight line on the right is due to me using some approximations to produce this plot.

3.2.3 The inverse system

You might ask the following question. What if everything goes well until S_n^{n-1} but suddenly at S_n^n you lose everything: the well balanced string obtained in the previous iteration suddenly becomes imbalanced in just one step. This is not a theoretical question, it happens in practice. For instance if $S_n^{n-1} \equiv \sqrt{2}$ then $S_n^n \equiv 1$, the most imbalanced of all strings. The question then is how can you end up in a situation like that and how to make sure it is not the case with the seeds that we use?

The situation could be much more subtle than the extreme case just described. To reconstruct the seed of each sequence (S_n^k) with n fixed, assume that $S_n^n \equiv x_n$ and $x_n \to x$. Then $S_1^0 \equiv x^{1/2}$, $S_2^0 \equiv x^{1/4}$, $S_3^0 \equiv x^{1/8}$ and so on results in $S_n^n \to x$. Keep in mind that the square root of a non-zero string always have two values, so there are many solutions. Also, if S_n^n is imbalanced as in our extreme case, S_n^{n-1} may not be, and it converges to $\sqrt{x}$. Note that if $S_n^{n-1} \equiv \sqrt{2}$ and thus $S_n^n \equiv 1$, the seeds are well balanced. In our case, the situation is opposite, with extremely imbalanced seeds consisting only of zeros with a '1' at each end.

Now, it is interesting to make an analogy with the logistic map, another quadratic system, governed by the recursion $s_k = 4s_{k-1}(1 - s_{k-1})$ with a seed s_0 in $[0, 1[$. Our system consists of infinitely many recursions (one for each n), with the n-th recursion being $S_n^k = S_n^{k-1} * S_n^{k-1}$ with each iterate S_n^k, when represented as a real number, constrained to lie (say) in $[2, 4[$. The logistic map also has a binary numeration system attached to it, see section 3.1.1 in [5]. The seed $s_0 = 1/3$, while extremely imbalanced, leads to the well balanced number $(2\pi)^{-1} \arcsin(\sqrt{1/3})$, known to be irrational. The logistic map is homeomorphic to the dyadic map that produces the standard binary digits, and thus, homeomorphic to each of our dynamical systems (one system for each n).

3.2.4 Invariant measure

An invariant measure of an ergodic dynamical system, also called equilibrium or attractor distribution, is a probability distribution defined as follows. All dynamical systems are governed by a mapping h that specifies the recursion. In our case, $S_n^{k+1} = h(S_n^k) = S_n^k * S_n^k$. When the string S_n^k is represented by a number lying in $[1, 2[$, the function h is defined as

$$h(x) = \begin{cases} x^2 & \text{if } 1 \le x < \sqrt{2}, \\ \frac{1}{2}x^2 & \text{if } \sqrt{2} \le x < 2. \end{cases}$$

Note that h is the same for all n since all our dynamical systems are identical except for the seed S_n^0. The invariant measure is the distribution F that satisfies $F(x) = F(h(x))$. Finding F involves solving a functional equation, but in our case, $F(x) = \log_2 x$ with $x \in [1, 2[$. This is known as the reciprocal distribution. For the logistic map, the invariant measure is a beta distribution, and for the dyadic map, a uniform distribution.

The standard approach to solve our problem is to use the target number as the seed, show that it is in the attraction domain of the main invariant measure F, and thus conclude that the number in question (the seed) is normal. No one has ever succeeded to prove anything with this technique. To the contrary, our approach uses trivial seeds apparently unrelated to the target number, exhibiting no sign of being in the attraction domain until after k iterations with $k > n$. We don't care if it is or not in the attraction domain, we only care about the behavior when $k \le n$, for each dynamical system (S_n^k) with n fixed. By looking at all n, we make the connection to the target number and have compelling arguments about its digit distribution.

4 Solving one of the greatest mathematical mysteries

For any positive integer n, the seed string S_n^0 is equivalent to the number $2^n + 1$. Thus, the string S_n^k obtained via k successive auto-convolutions is equivalent to $2^n + 1$ at power 2^k. Since we work with classes (similar to modulo classes), the magnitude does not matter: we can choose any equivalent number by multiplying by a factor 2^m (with m a positive or negative integer). In particular, we can choose m so that the resulting number always lies in $[2, 4[$, and is thus uniquely defined, without changing the characters in the string S_n^k. You achieve this goal with $m = -\lfloor 2^k \log_2(2^n + 1) \rfloor + 1$, which is equal to $-n \cdot 2^n$ when $k = n$. Here $\lfloor \cdot \rfloor$ stands for the integer part function. From there, we obtain

$$S_n^n \equiv \left(1 + \frac{1}{2^n}\right)^{2^n} \to e = 2.718281828\ldots \quad \text{as } n \to \infty, \tag{3}$$

where e is Euler's number. Simple computations using logarithms and Taylor series expansions show that the approximation yields about n correct binary digits, in accordance with earlier claims made in this paper. It is also straightforward to verify that $S_n^{n-1}, S_n^{n+1}, S_n^{n+2}, S_n^{n+3}$ tend respectively to $\sqrt{e}, e^2, e^4$ and e^8 as $n \to \infty$, in accordance with theorem 3.2, using the definition of convergence in section 2.2.

But what would happen with a different seed S_n^0? With $S_n^0 \equiv 2^n - 1$ and $S_n^0 \equiv 2^n + 3$, the limit constants are respectively e^{-1} and e^3, instead of e. However, there is no guarantee that the observed patterns are identical: it needs to be verified. Then, using $S_n^0 \equiv 2^n + 2 = 2 \cdot (2^{n-1} + 1)$ does not bring anything new as we are staying within the same original string class. Now, I can state the main result – the very first *deep* result about the digit distribution associated to popular math constants.

Theorem 4.1 *For n large enough, the proportion p_n of ones in the first n binary digits of $e = 2.7182\ldots$ satisfies $p_n > \mu$, and the proportion q_n of zeros satisfies $q_n < 1 - \mu$, where $\mu > 0$ does not depend on n even as $n \to \infty$.*

Theorem 4.1 is stated in a very weak form given all the advances shared in this paper. I present it as a starting point rather than the best that we have obtained so far. Yet this weak version is a phenomenal result in itself. In particular, the statement that μ stays strictly positive even when n is infinite is the strongest result ever obtained for any major mathematical constant such as $\pi, e, \log 2$ or $\sqrt{2}$, by a long shot. For instance, it has been established that the proportion of ones in the first n binary digits of $\sqrt{2}$ is larger than $\sqrt{2n}$, see [8]. However, since $\sqrt{2n}/n \to 0$ as $n \to \infty$, it leads to $\mu = 0$. Also the proof is rather simple and cannot be improved to get a stronger result.

About theorem 4.1, "n large enough" may be interpreted as $n > 50$. Also, a low hanging fruit is $\mu = 10\%$, while improving to $\mu = 20\%$ (more challenging), and even $\mu = 30\%$ (significantly more challenging), appears to be within reach based on results in section 3.2. In the end, theorem 4.1 is a weaker version of theorem 3.3, applied to Euler's number instead of the original formulation that deals with auto-convoluted string sequences. Formula (3) is the connection between both.

For additional references, see my book on chaos and dynamical systems, discussing a stronger concept of normality in various numeration systems [5]. Andrew Granville and Davig Bailey [2] are also good references on this topic. The Wolfram entry for the digit count function (see here) features an exact closed-form formula for the number of digits equal to 1 in the binary expansion of any integer, with more references. It is also known as the Hamming weight, with a very fast algorithm described here and a full chapter in [9]. It is particularly relevant to our problem. Regarding recent publications on normal numbers, see Verónica Becher [3] and [1]. For a discussion about the carry digit function (a 2-cocycle) that propagates 1's from right to left in the successive iterations, see [4]. Another application of the digit count function – also called sum-of-digits function, for binary sequences – is featured in [7] in the context of genotype maps, with processes not unlike the dynamical systems discussed in this article, and blancmange curves almost identical to Figure 3.3 in my book on numeration systems [5].

Also look for references on infinite iterated self-convolutions of a function, with re-scaling, and convergence to a Gaussian distribution. In our case, the function in question is equivalent to a mapping from $[2, 4[$ onto itself. The implicit non-continuous re-scaling is done at each iteration to keep the iterates in that interval. Related material include discrete dynamical systems, the logistic map in particular, with bifurcations at $\frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}$ and so on. In short, the central argument to the proof of theorem 4.1 and its stronger version, say with $\mu = 29.1\%$, is to analyze an infinite number of infinite iterated sequences of truncated auto-convoluted binary string classes similar to p-adic numbers, look at the converging diagonal entries in the associated topological space, and use re-scaling. The guaranteed minimum of at least 29.1% of ones in the binary digits of Euler's number, comes as a special constant in a specific dynamical system in its non-chaotic phase just before chaos arises, starting with the simplest seed string of length $n+1$ in each sequence (S_n^k) indexed by k, with fixed n, and then let $n \to \infty$.

4.1 Another interesting sequence

In order to obtain interesting results with my framework, you need to have string convergence. Few mathematical constants fit the bill. It works for e (possibly the simplest case) because e can very easily be approximated by a sequence A_n/B_n where A_n is an integer, and B_n of power of 2. In this case, assuming x is an integer,

$$A_n = (2^n + x)^{2^n},\ B_n = 2^{n \cdot 2^n} \Rightarrow \frac{A_n}{B_n} \to \exp(x) \text{ as } n \to \infty.$$

One might think that since A_n can be expanded using the binomial theorem, a simpler case consists of working with just one coefficient – the central one – in the binomial expansion. While there is a way to do it, the solution is probably far more complicated and only works with $x = 1$. The starting point could be this:

$$A_n = n \cdot \binom{2n}{n}^2,\ B_n = 2^{4n} \Rightarrow \frac{A_n}{B_n} \to \frac{1}{\pi} \text{ as } n \to \infty.$$

The savvy reader is invited to play with it! If the denominator B_n is a power of b rather than 2, you may get interesting results in base b rather than in the binary numeration system, especially if b is prime or a power of a prime. In short, my scheme allows you to entirely ignore the denominator and focus just on the digits of A_n.

4.2 Application to cryptography

If you look at Figure 1, the bottom half (where the chaos truly starts) seems to produce decent random bits. However, if you use this system to generate pseudo-random numbers, you face the same issues as with congruential random generators. First, the truncation defined in section 2.2 is a type of modulo operator, that is, generating congruence classes of some sort. Then, while the period may be extremely large, these systems have security vulnerabilities. In addition, the way the bits are computed in my system is not very efficient, as the goal is to solve theoretical problems rather than designing fast apps. Yet, the Python code in section 4.3 uses a library to speed up some computations by several orders of magnitude, and I use it to double check my results with external algorithms. You might want to have a look at it.

That said, I designed and tested a few fast, secure and strong random number generators based on irrational numbers. For details, see chapter 13 in [6], entitled "Strong Random Generators for Reproducible AI", as well as chapter 4 in [5], entitled "Random Numbers Based on Quadratic Irrationals".

4.3 Python code

The Python code `number_theory.py` is also on GitHub, here. To analyze the growing complexity of the patterns as n increases, I highly recommend to use AI. These patterns are governed by universal dynamical system thresholds, such as 29.1% mentioned earlier. These thresholds are estimated in section [3] in the code. In section [2], I use the Mpmath library to compute the digits of e and to double check that my algorithm yields at least $n-3$ correct digits at each n.

```
from PIL import Image, ImageDraw

n = 1000
L = 2*n
H = int(1.1*n)

height,width = (H+1, L+1)
img1 = Image.new( mode = "RGBA", size = (L+3, H+2), color = (0, 0, 0) )
pix1 = img1.load()
draw1 = ImageDraw.Draw(img1,"RGBA")

prod = 2**n + 1
offset_H = 0
arr_count1 = []
arr_count0 = []
arr_count1.append(2)
arr_count0.append(n-2)


#--- [1] --- Main

for k in range(1, H+1):

   prod = prod*prod # this is (2^n + 1) at power 2^k (truncated)
   pstri = bin(prod)
   stri = pstri[0: L+2]
   prod = int(stri, 2)
   lsr = len(stri)

   if k == n+1:
      offset_H = 1
      for l in range(L+1):
         pix1[l, k-1] = (0, 0, 0)

   stri = stri[2: len(stri)]
   if k == n:
      e_approx = stri
   rstri = stri[n:2*n] # rightmost n digits in first 2n digits
   rcnt0 = rstri.count('0')
   rcnt1 = rstri.count('1')
   estri = stri[0:n] # leftmost n digits
   ecnt0 = estri.count('0')
```

```
    ecnt1 = estri.count('1')
    bnum = 0

    for l in range(n):
        bnum += int(estri[l]) / 2**(l-1)
    offset_L = 0
    for l in range(min(L, len(stri))):
        if l == n+1:
            pix1[l, k-1] = (0, 0, 0)
            offset_L = 1
        elif l == 2*n+2:
            pix1[l+1, k-1] = (0, 0, 0)
            offset_L = 2
        if stri[l] == '1':
            pix1[l+offset_L, k-1+offset_H] = (255, 0, 0)
        elif stri[l] == '0':
            pix1[l+offset_L, k-1+offset_H] = (255, 255, 255)

    arr_count1.append(ecnt1)
    arr_count0.append(ecnt0)

    print("%3d %3d %3d %3d %3d %f" %(k, ecnt0, ecnt1, rcnt0, rcnt1, bnum))

img1.save("img_1.png")
img2 = img1.crop((n-n/2, n-n/4, n, n)) # left, top, right, bottom
img2.save("img_2.png")


#--- [2] --- Fast computation of binary digits of e

from mpmath import mp
import numpy as np

# Set precision for n binary digits
mp.dps = int(n*np.log2(10))
e_value = mp.e # Get e in decimal

# Convert to binary
e_binary = bin(int(e_value * (2 ** n)))[2:]

k = 0
print()
while e_approx[k] == e_binary[k]:
    k += 1
print("%d correct digits (n = %d)" %(k, n))


#--- [3] --- Compute nu values in Table 1

arr_rho = []
print()
for j in range(1,6):
    threshold = int(n*j/(j+1))
    if threshold % 2 == 1:
        threshold += 1
    p1_even = arr_count1[threshold]/n
    print("Rho %1d: %7.5f" %(j, p1_even))


#--- [4] --- Create the plots

import matplotlib.pyplot as plt
import matplotlib as mpl

mpl.rcParams['axes.linewidth'] = 0.5
plt.rcParams['xtick.labelsize'] = 8
```

```
plt.rcParams['ytick.labelsize'] = 8

xvalues = np.arange(1, H+2, 1)
arr_color = []
for k in range(H+1):
   if k%2 == 0:
      arr_color.append((1,0, 0))
   else:
      arr_color.append((0, 0, 1))

xmin = int(0.6*n)
xmax = int(1.1*n)

plt.scatter(xvalues[xmin:xmax], arr_count1[xmin:xmax], s = 0.01, c =
    arr_color[xmin:xmax])
plt.axvline(x=2*n/3, color='black', linestyle='--', linewidth = 0.4, dashes=(5, 10))
plt.axvline(x=3*n/4, color='black', linestyle='--', linewidth = 0.4, dashes=(5, 10))
plt.axvline(x=4*n/5, color='black', linestyle='--', linewidth = 0.4, dashes=(5, 10))
plt.axvline(x=5*n/6, color='black', linestyle='--', linewidth = 0.4, dashes=(5, 10))
plt.axvline(x=6*n/7, color='black', linestyle='--', linewidth = 0.4, dashes=(5, 10))
plt.axvline(x=7*n/8, color='black', linestyle='--', linewidth = 0.4, dashes=(5, 10))
plt.axvline(x=8*n/9, color='black', linestyle='--', linewidth = 0.4, dashes=(5, 10))
plt.axhline(y=n/5, color='black', linestyle='--', linewidth = 0.4, dashes=(5, 10))
plt.axvline(x=n, color='red', linestyle='-', linewidth = 0.4, dashes=(5, 10))
plt.axhline(y=n/2, color='red', linestyle='--', linewidth = 0.4, dashes=(5, 10))
plt.xlim([xmin, xmax])
plt.show()
```

References

[1] Christoph Aistleitner et al. Normal numbers: Arithmetic, computational and probabilistic aspects. 2016. Workshop [Link]. 8

[2] David Bailey, Jonathan Borwein, and Neil Calkin. *Experimental Mathematics in Action*. A K Peters, 2007. 8

[3] Verónica Becher, A. Marchionna, and G. Tenenbaum. Simply normal numbers with digit dependencies. *Mathematika*, 69:988–991, 2023. arXiv:2304.06850 [Link]. 8

[4] James Dolan. Carrying is a 2-cocycle. *Preprint*, pages 1–9, 2023. [Link]. 8

[5] Vincent Granville. *Gentle Introduction To Chaotic Dynamical Systems*. MLTechniques.com, 2023. [Link]. 3, 7, 8, 9

[6] Vincent Granville. *Building Disruptive AI & LLM Technology from Scratch*. MLTechniques.com, 2024. [Link]. 9

[7] Vaibhav Mohanty et al. Maximum mutational robustness in genotype–phenotype maps follows a self-similar blancmange-like curve. *The Royal Society Publishing*, pages 1–16, 2023. [Link]. 8

[8] Joseph Vandehey. On the binary digits of $\sqrt{2}$. *Preprint*, pages 1–6, 2017. arXiv:1711.01722 [Link]. 8

[9] Henry S. Warren. *Hacker's Delight*. Addison-Wesley Professional, second edition, 2012. 8