LLM Challenge with Petabytes of Data to Prove Famous Number Theory Conjecture

Vincent Granville, *Co-Founder, BondingAI.io*

***Abstract*—In my recent article "Piercing the Deepest Mathematical Mystery", I paved the way to proving a famous multi-century old conjecture: are the digits of major mathematical constant such as $\pi, e, \log 2, \sqrt{2}$ evenly distributed? No one before ever managed to prove even the most basic trivialities, such as whether the proportion of '0' or '1' exists in the binary expansions of any of these constants, or if it oscillates indefinitely between 0% and 100%. Here I provide an overview of the new framework built to uncover deep results about the digit distribution of Euler's number e, discuss the latest developments, share an upgraded version of the code, and feature new potential research areas across multiple fields, arising from my discovery.**

I. Introduction

My first article on the topic, published in February 2025, is posted here as paper 51. The methodology relies on material discussed in my book "Gentle Introduction to Chaotic Dynamical Systems" [7]. The original article is now part of that book. I used bit strings to represent binary digit sequences, establishing the link to large language models (LLMs). Here, I will stick to integers instead, to simplify the presentation. The main idea is to work with iterated self-convolutions. For fixed integers n and x, iteratively define the sequence $\{S(n,k,x)\}$ with the recursion

$$S(n,k+1,x) = S^2(n,k,x), \quad k = 0,1,2\ldots \tag{1}$$

with $S(n,0,x) = 2^n + x$. The initial value $S(n,0,x)$ is called the seed. We are interested in $x = \pm 1$ or $x = 3$. It follows that the first $\tau(n)$ binary digits of $S(n,n,x)$ match those of $\exp(x)$, with $\tau(n) = n + O(1)$. This remains true if at each iteration k, we truncate $S(n,k,x)$ and only keep the first $2n$ binary digits on the left, so that $S(n,k,x)$ is an integer with $2n$ digits at all times.

A more traditional approach consists in multiplying the seed $S(n,0,x)$ by an integer power of 2, positive or negative, so that it lies in the interval $[1,2[$. Then, replace Formula (1) by the recursion

$$S(n,k+1,x) = \lambda \cdot S^2(n,k,x) \tag{2}$$

with $\lambda = 1$ if $S(n,k,x) < \sqrt{2}$, otherwise $\lambda = \frac{1}{2}$.

Then, the sequence $\{S(n,k,x)\}$ indexed by k, with fixed n and x, is an ergodic quadratic dynamical system. It is homeomorphic both to the logistic map and the dyadic map, with $S(n,k,x) \in [1,2[$ at all times. In short, it is a mapping from $[1,2[$ onto itself. Its invariant measure is the reciprocal distribution, a probability distribution defined as $F(z) = \log_2 z$ for $1 \le z < 2$. The theory, including the connection to normal numbers, is discussed in chapter 6 in [7].

Whether using (1) or (2), the end results are the same, as we are only interested in the first n binary digits of $S(n,k,x)$ on the left side, for $k = 0,1,2$ and so on. I now introduce the digit sum function, also known as digit count for binary digits, as $\zeta_S(n,k,x)$. For a fixed n and x, it counts the number of '1' in the first n binary digits of $S(n,k,x)$. The normalized version

$$\zeta_S^*(n,k,x) = \frac{1}{n}\zeta_S(n,k,x) \tag{3}$$

takes a value between 0 and 1. It represents the proportion of '1' in the first n digits of $S(n,k,x)$. Therefore, $\zeta_S^*(n,n,x)$ represents the proportion of '1' in the first $\tau(n)$ digits in the binary expansion of $\exp(x)$. It is not difficult to prove that $\tau(n) = n + O(1)$. Finally, to obtain theoretical bounds on the proportion of '1' in $\exp(x)$, here with $x = \pm 1$ or $x = 3$, you need to study the spectacular behavior of $\zeta_S^*(n,n,x)$ for fixed n and x, and then let $n \to \infty$. The remaining of the article focuses on this problem, and its various applications to quantum dynamics, cryptography, AI and LLM evaluation in particular, dynamical systems, high performance computing, and so on.

II. Spectacular behavior of the digit sum function

Before digging into this problem, it is worth noticing that my approach is radically different from all previous attempts to prove that the binary digits of numbers such as e or π are evenly distributed. They all failed even at proving much weaker results. The main innovations in my framework are:

- Focusing on very few digits on the left side, rather than many digits on the right side and ignoring the leftmost part of the digit expansion. Yet, as $n \to \infty$, 'very few' eventually becomes infinite.
- To uncover patterns, experimenting with numbers as large as $2^n + 1$ at power 2^n, with $n = 10^5$, far beyond the capability of any computer system. This is possible thanks to the truncation mechanism.
- Working with systems exhibiting strong and fascinating patterns that can be precisely quantified, rather than navigating in the dark. It helps visualize the steps necessary to build a solid proof.
- Building on expertise with discrete dynamical systems, thus having a clear idea of what to expect, both in terms of leverage and challenges.
- Working with the most basic seeds leading to a unique behavior. We do not care if the seed is attached to the main invariant measure or not, as we stop at iteration

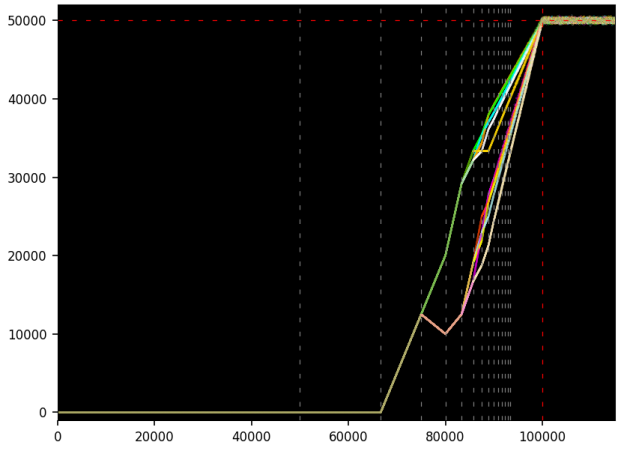

Fig. 1. Digit sum $\zeta_S(n,k,x)$ with $x=1$, $n=10^5$, and k on the X-axis

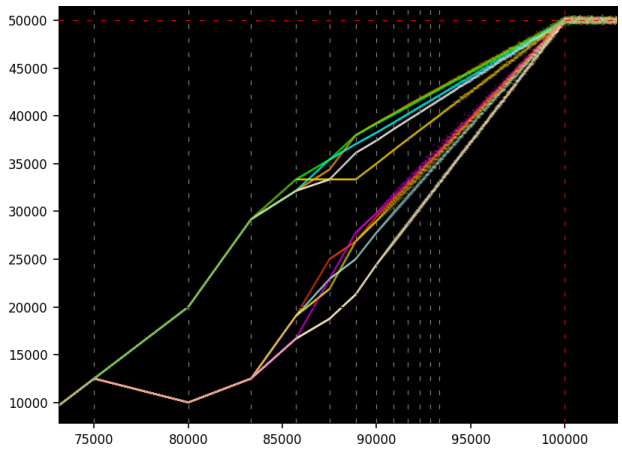

Fig. 2. Zoom in on the right part of Figure 1

$k=n$ in $S(n,k,x)$, that is, at the very end of the non-chaotic regime of the underlying dynamical system, just before full chaos sets in.

A. Potential scenarios

Moving forward, I use $n=10^5$ in all my illustrations. Unless otherwise specified, the integers n and x are fixed. As with all similar dynamical systems (dyadic map, in particular), the main function of interest, in this case the digit sum $\zeta_S(n,k,x)$ as a function of k, can have a wide variety of shapes depending on the seed. Here, the seed corresponds to $S(n,k,x)$ with $k=0$. That is, $S(n,0,x)=2^n+x$.

Figure 1 features an extremely rare behavior, and one of the easiest to predict. Nevertheless, it is the only one of interest to prove deep results about the binary digits of e. The seed corresponds to a string with $n-1$ consecutive '0's, with a '1' at both ends. I will use this case as a basis to describe alternate, much more common scenarios.

At $k=3n/4$, the bifurcation phase starts and lasts until $k=n$. And when $k\geq n$, we are in the full chaotic phase, with the proportion of '1' oscillating around 50%. Interestingly, at $k=n$, the width of the horizontal band visible in the top right corner is conjectured to be $O(\sqrt{n\log\log n})$ at most, a consequence of the law of the iterated logarithm, and $O(\sqrt{n})$ on average, a consequence of the central-limit theorem.

The limit of the digit sum function when $n\to\infty$, when plotted on a fixed-size window, is a continuous time process with infinitesimal increments. Yet, unlike a Brownian motion (the limit of a random walk), it is nowhere continuous, jumping from one state to another as k increases. The finite number of potential states also increases as k increases. Each state has its own color in Figure 1, and corresponds to a particular residue of k in a specific congruential class. Despite being nowhere continuous, the graph appears to be much less erratic than a Brownian motion. In particular, the segments are connected with no apparent discontinuity, a property specific to the seed 2^n+x with $x=\pm1$ or $x=3$, but not true in other cases. Due to the potential states, the multi-branch function is called a quantum function.

I can now share a number of possible scenarios for the behavior of the digit sum function, depending on the seed, for a large variety of seeds. The list below does not cover all the cases, but only those that are either the most common, or related to our discussion. The seed must have at least n bits, starting with '1', and ending with '1' unless its length is infinite.

- The most common case by far is when the seed is a random string with n bits. Then there is no warm-up phase. Instead, we enter the chaotic phase at the very beginning, when $k=0$, instead of $k=n$ in Figure 1.
- If the seed consists mostly of '0', with the proportion of '1' above (say) 10%, then the chaotic phase is reached in very few iterations. This is the most interesting case for cryptography.
- Seeds with $n=10^5$ bits with only 3 to 5 bits set to '1's at random locations, are useful for research. The chaotic phase can be reached pretty fast, but sometimes with gaps in the warm-up phase. See Figure 7.
- If the seed is a real number $2^{q/p}$ with p,q coprime integers, p an odd prime, then there is no chaotic phase. The digit sum function is periodic. The length of the period is the multiplicative order of 2 modulo p.

The seeds $S(n,0,x)$ that we are working with to establish deep results about the digits of e, are among the very few that require so many iterations ($k=n$) before reaching the chaotic phase.

B. Dynamics of the unbroken bifurcation process

I conjecture that the seeds $S(n,0,x)=2^n+x$ with $x=\pm1$ or $x=3$, results in no gap during the non-chaotic phase, unlike Figure 7. If gaps were present, proving the desired result would potentially be more difficult. I use the word "unbroken bifurcation process" to describe the absence of gap.

If you look at Figure 1, new forks and changes in the slope occur at specific locations k on the X-axis. The vertical dashed lines show these locations. They are identical regardless of n,

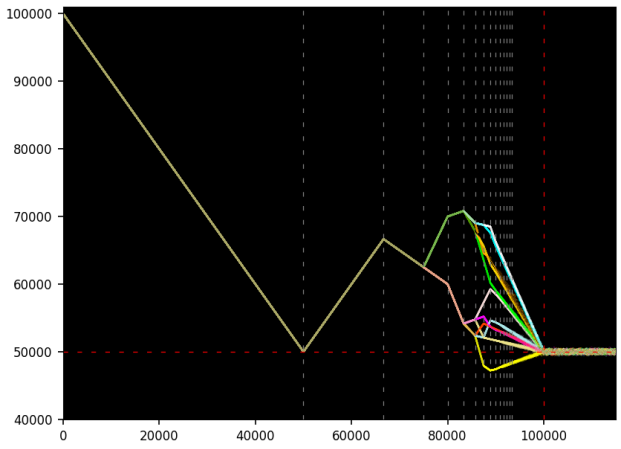

Fig. 3. Digit sum $\zeta_S(n,k,x)$ with $x=-1$, $n=10^5$, and k on the X-axis

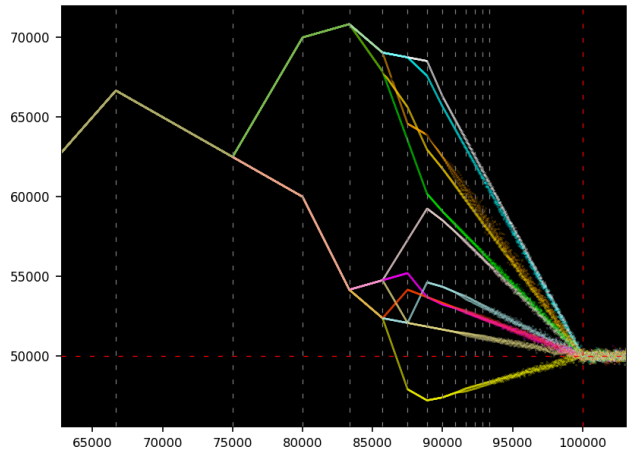

Fig. 4. Zoom in on the right part of Figure 3

and whether $x=\pm1$ or $x=3$. These locations correspond to $k/n \approx \rho_1,\ \rho_2,\ \rho_3$ and so on, with

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

These approximated values are extremely accurate. On the right on the X-axis, beyond $k/n=\rho_7$, the number of change points start exploding with more and more segments, shorter and shorter, eventually evolving in a very narrow range just before reaching the chaotic phase with about 50% of '1'. Within each vertical band delimited by consecutive vertical dashed lines, the number of segments, from left to right, is respectively equal to 1, 1, 2, 2, 4, 12 and so on. Except for the first two values, these are factorials multiplied by 2. Since $S(n,k,x)=2^n+x$ at power 2^k, as k increases this combinatorial explosion is not surprising. Then, based on (4), the widths of each band, from left to right, are proportional to

$$\frac{1}{1\cdot2},\quad \frac{1}{2\cdot3},\quad \frac{1}{3\cdot4},\quad \frac{1}{4\cdot5},\quad \ldots$$

The proportionality factor is n. Paths are determined by congruential residues. For instance, the green path in Figure 2 corresponds to values of k such that $k\equiv 4 \bmod 12$. Finally, the shape of the graph, for a specific x, stays the same regardless of n. But different values of x produce different shapes.

III. Case studies

Figures 1 and 2 feature the case $x=1$. Here, I show the dynamics of the digit sum function for the cases $x=-1$ and $x=3$, associated respectively with the binary digits of e^{-1} and e^3. Then Figure 7 shows the results when using a seed consisting of a string of length $n=10^5$, all '0' except three '1' at random locations, plus a '1' at both ends. Finally, I discuss what happens when averaging values within each vertical band delimited by vertical dashed lines in Figures 1 and 2.

The case $x=-1$ is featured in Figures 3 and 4. The seed 2^n-1 consists of n bits all '1', without any '0'. The graph is more elaborate than the cases $x=1$ and $x=3$, starting at $k=0$ with all '1', with the proportion going down to

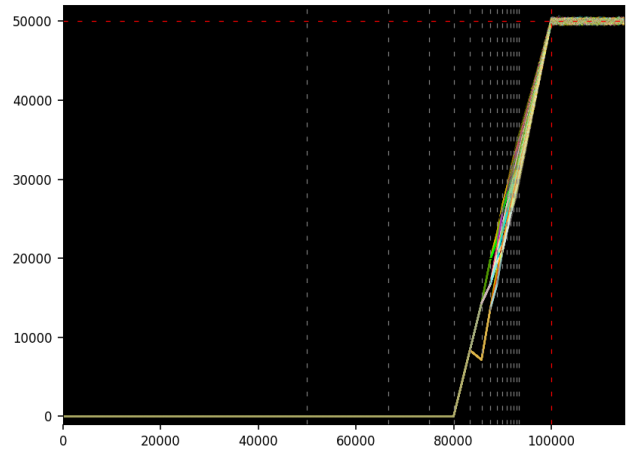

Fig. 5. Digit sum $\zeta_S(n,k,x)$ with $x=3$, $n=10^5$, and k on the X-axis

50% as we reach $k=n/2$, but then bouncing back up before eventually stabilizing around 50% when $k>n$. Despite the different behavior compared to $x=1$ or $x=3$, the vertical dashed lines indicating new bifurcations and slope changes, still have the same abscissas.

At first glance, it seems more difficult to prove a formal theorem about the digit distribution for this case. However, due to the various segments being more spread out, possibly with fewer hidden features, it certainly makes the patterns easier to quantify, maybe facilitating a proof for the digits of e^{-1}, compared to e or e^3.

Figures 5 and 6 correspond to the case $x=3$. The seed has $n+1$ bits, all '0' except the first one on the left and the two rightmost equal to '1'. Despite the seed being apparently more complex than the case $x=1$, having three '1' rather than two, the bifurcation phase (before chaos) evolves in a much narrower range, mostly with increases in the proportion of '1' over time until about 50% is reached, with very few, moderate setbacks. It could make a proof easier than for the case $x=1$. However many features may be hidden, making pattern analysis more complicated. Again, the positions of the vertical dashed lines are unchanged.

Figure 7 features a seed not related to the digits of any

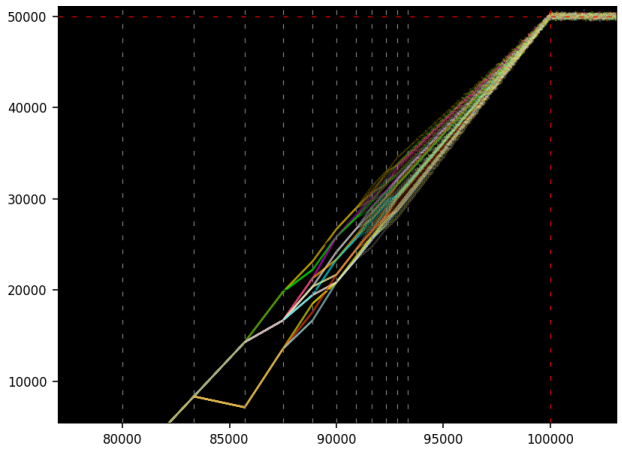

Fig. 6. Zoom in on the right part of Figure 5

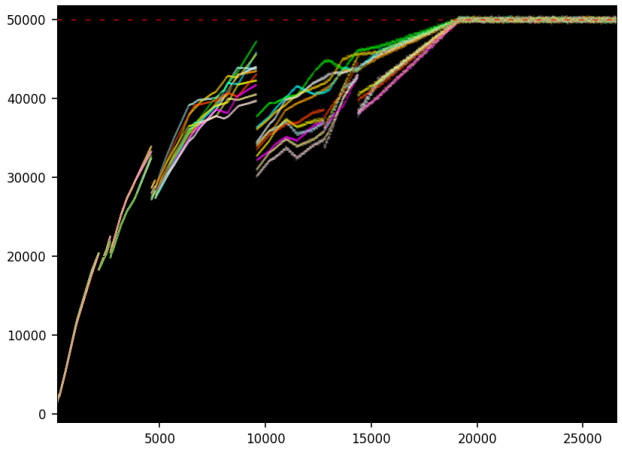

Fig. 7. Digit sum: seed with $n = 10^5$ bits with five '1' at random locations

known number. It is not part of a scheme where increasing n results in convergence of $S(n, n, x)$ to some constant. Unlike the previous examples, the locations of the '1's in the seed of length $n = 10^5$ are spread randomly, yet with a '1' at each end. The total number of '1' is five. Thus, depending on the locations of the random '1's, the integer x can be a very large number, possibly much larger than $2^{n/2}$ yet much smaller than 2^n. Nevertheless, this case is very interesting because it shows that gaps sometimes occur. While mild in most cases, in this example they are rather substantial. Note that the X-axis is truncated to magnify the gaps, as they appear early in the process, with the rightmost visible one at $k < n/6$.

Finally, if you average the segments withing each vertical band delimited by consecutive vertical dashed lines in Figure 1, you obtain Figure 8. Thus, we are approaching a straight line starting at $k = 4n/5$, reaching the chaotic phase at $k = n$ for the number of '1's in the first n binary digits of $2^n + 1$ at power 2^k. Thus, with about 50% of '1' when $n = k$, for the first n digits of e.

To produce Figure 8, I actually used a much simpler technique to get an approximation: the curve is a moving average based on a window with 12 consecutive values of k. In addition, all the pictures and results featured here were

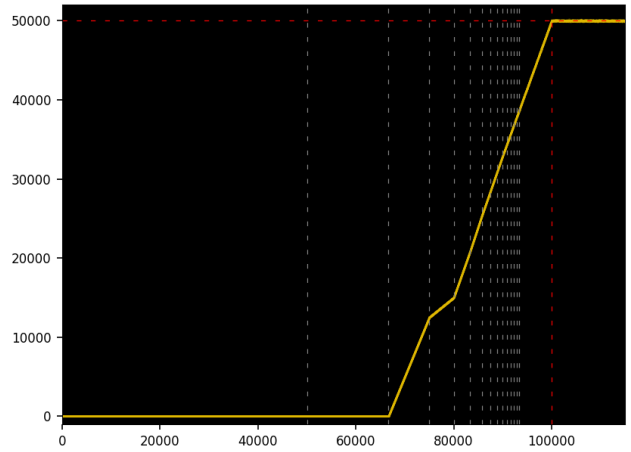

Fig. 8. Averaging the segments within each vertical band in Figure 1

produced using the Python code in section VI. My algorithm is based on formula (1) combined with the truncation mechanism. Also, I double-checked the correctness in the digits of e with an external library. The faster version of the code is on GitHub, here. It runs about twice as fast as the version in section VI, by truncating $S(n, k, x)$ to $2n - k$ bits rather than $2n$, and removing the trailing '0's on the right.

IV. An extreme case

Here $x = \log \mu$ with $\mu = 2$. We need a precision of n bits on $\log \mu$. It is equivalent to using the seed μ^{r_n} with $r_n = 2^{-n}$. Thus, $S(n, n, x) = \mu$ and $S(n, n-1, x) = \sqrt{\mu}$, up to the first n digits. This example shows yet another type of behavior in the digit sum function.

The proportion of '1' in the first n digits at $k = n$ is 0% or 100% depending on whether we approach the limit from under or from above, since $2 = 0.11111\ldots = 1.00000\ldots$ in binary form. For a fixed n, formula (1) – combined with truncating $S(n, k, x)$ to $2n$ bits at each iteration k – guarantees about n correct digits in $S(n, k, x)$ up to $k = n$. Beyond $k = n$, the precision continues to drop. This explains why the rightmost part of the digit sum function in Figure 9, when $k > n$, is not an horizontal line as it should be: it is due to increasing loss in precision as k increases.

Note that in this example, there is no bifurcation process. Also, the coloring scheme is meaningless, and not linked to congruential classes. I used the same code as in section VI to produce Figure 9 and 10. The only difference is about the computation of the seed, done with the following code:

```
ctx.precision = 3*n
prod = 2

for k in range(n):
   prod = gmpy2.sqrt(prod)

prod = int(2**(2*n) * prod)
```

. The seed is denoted as `prod` in the code, and turned into an integer with a precision of about $2n$ bits thanks to setting the precision to $3n$ bits before the loop.

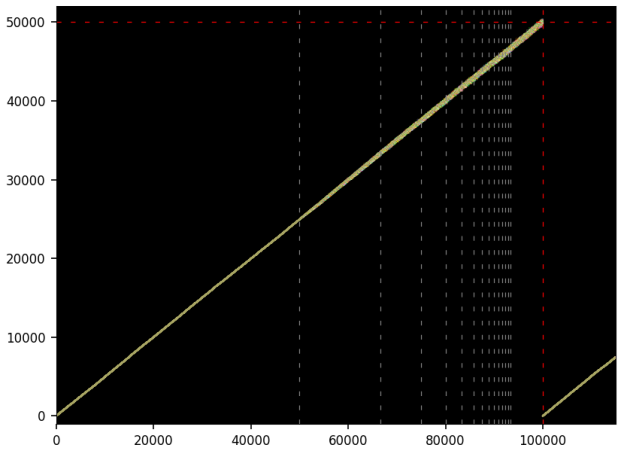

Fig. 9. Digit sum $\zeta_S(n,k,x)$ with $x = \log 2$, $n = 10^5$, k on the X-axis

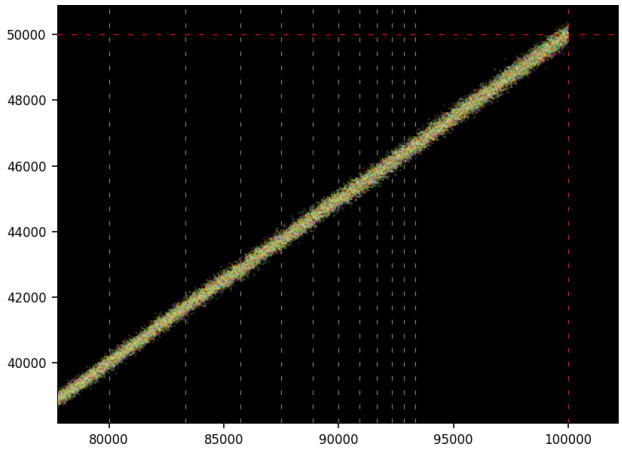

Fig. 10. Zoom in on the right part of Figure 9

V. Applications and AI Challenge with Petabytes Dataset

In this section, I first provide a quick overview of potential applications, with recent references. Then I discuss an interesting challenge: using large language models (LLMs) to leverage my research and uncover deeper insights, test and compare their mathematical and pattern detection capabilities, based on the digit dataset and via reasoning. After all, the iterates $S(n,k,x)$ are highly correlated strings similar to sentences as in English prose or DNA sequences, here with an alphabet consisting of two letters: '0' and '1'. Let's start with the references:

- The framework presented here relies on discrete quadratic dynamical systems. This family also includes the logistic map and the example discussed in [15]. For additional references, see my book on chaos and dynamical systems [7].
- Showing that the binary digits are evenly distributed is the first step towards proving that e is a normal number. Andrew Granville and Davig Bailey [3] are good references on this topic. For recent publications on normal numbers, see Verónica Becher [4] and [2]. One of best results know for any major math constant is the fact that the proportion of ones in the first n binary digits of $\sqrt{2}$ is larger than $\sqrt{2n}$, see [14].
- The digit sum or digit count functions (both are identical for binary digits) is also known as the Hamming weight, with a fast algorithm described here and a full chapter in [16]. The Wolfram entry for the digit sum (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. For a discussion on the carry digit function (a 2-cocycle) that propagates 1's from right to left in the successive iterations $S(n,k,x)$, see [1], [5].
- An interesting application of the digit sum is featured in [9] 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 [7].
- There is a connection to quantum maps and quantum cryptography [6], [13]. For PRNGs (pseudo-random generators) based on irrational numbers, see chapter 13 in [8] or chapter 4 in [7]. Finally, if you use an arbitrary seed instead of $S(n,0,x) = 2^n + 1$, you obtain strings that look random, after very few iterations.
- Deep neural networks have been used to identify the underlying model of dynamical systems, based on available data produced by simulations or from real life observations, see [11], [12], [17]. In our case, the model would be a simple formula that generates the values of the digit sum function, to study its asymptotic properties.

A. AI challenge

The last item in the bullet list brings an interesting challenge. The idea is to use AI and large language models to get the full picture about the patterns. The goal is to formally prove the deepest possible results that you can get from my framework, about the binary digits of the number e, or related numbers such as e^{-1} or e^3. In particular, we want to fully understand and accurately quantify the bifurcation phase shown in all the figures. Questions to answer include

- The number of segments (referred to as quantum states) in each vertical band delimited by successive vertical dashed lines, for instance in Figures 2, 4, and 6. We want to find a general formula for the number of segments based on the location on the X-axis.
- Which values of k correspond to each segment. We know that the answer depends on the residues of k modulo specific integers linked to factorials. What are these residues and the modulo classes in question? It seems like and exact, simple, and general answer can be obtained. The number of segments within a vertical bands is directly linked to the modulo classes in question.
- The slopes of these segments, and the mean slope when averaged within a vertical band, with a general formula applicable to any location on the X-axis. Show that when the slope is moving in the wrong direction (away from the 50% target in the proportion of '1'), it can only do so for so long.

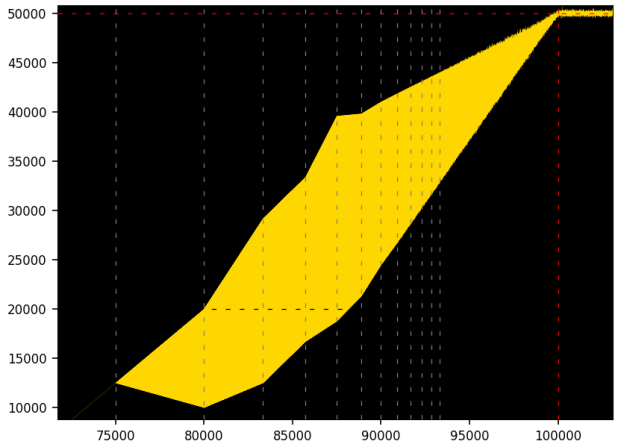

Fig. 11. Upper/lower bounds for $\zeta_S(n,k,x)$ with $x = 1$, $n = 10^5$, and k on the X-axis

- Confirm the absence of gaps when $x = \pm 1$ or $x = 3$, by contrast to Figure 7. What gap-free cases share in common? Also confirm and extend Formula (4) to cover the entire range from $k = 0$ to $k = n$. Finally, what is the exact shape of the envelope pictured in Figure 11, showing the minimum and maximum potential values for the digit sum function at any location k, with $0 \leq k \leq n$.

Rather than smart guesses or predictions, we want actual proofs whenever possible. The purpose of using AI is not to get approximations to model parameters, but exact values when possible, and even a generic formula that generates all the values, such as Formula (4). AI may also be able to identify other applications underlined by the same dynamical models, providing valuable information. This approach is discussed in a recent paper on data-driven model discovery [10].

B. The dataset

Large language models (LLMs) are trained to predict the next tokens given previous tokens, based on a large database of text. In our case, for a fixed n and x, each $S(n,k,x)$ is a string consisting of $2n$ bits. In LLM parlance, it is a block of tokens, each token consisting of (say) 512 bits. We want to predict $S(n,k+1,x), S(n,k+2,x)$ and so on, based on $S(n,k,x), S(n,k-1,x)$ and so on. The training set consists of all $S(n,k,x)$ with $0 \leq k \leq n$. We only care about the first n bits on the left. Indeed are only interested in the digit sum function computed on these first n bits, at any k. The problem is trivial until k gets close to n. The closer to n, the harder. Thus we can restrict training and predictions to $k/n > 0.85$. The predictions must all be exact in this case, and the methodology must use cross-validation.

Detecting the rules that make correct predictions is even more important. Or identifying other well-studied dynamical systems that have a very similar behavior, with known properties. In short, anything that can lead to formally proving a deep result about the digit sum, is highly valuable. Better, an actual proof if some LLMs can come up with one. Perhaps something like this: the proportion of '1' in the binary digits of e is above 10%. This in itself would be a phenomenal result, beating everything obtained previously, by a very long shot. Proving that it is exactly 50% is the best that we could expect.

For that purpose, one can create a dataset consisting of $S(n,k,x)$ strings with $n = 10^7$, for 100 different values of x, some causing gaps and some not in the digit sum function. This requires generating $2 \times 10^7 \times 10^7 \times 10^2$ bits, that is, 2.5 petabytes. LLMs that can detect the patterns with a much smaller dataset, and those that still perform well very close to $k = n$, should be rewarded. The code in section VI, with $n = 10^5$, is a starting point to create the dataset. It runs fast (about a minute) on a small laptop. Each value of x can be run in parallel. I uploaded the values of the digit sum function for $x = 1$, $n = 10^5$, and $0 \leq k \leq n$, on GitHub, here.

VI. Python code

The Python code `number_theory_fast_v2.py` is also on GitHub, here. The variable `p` plays the role of x in the code. If set to 0, it generates a random seed. The variable `H` determines the upper limit for k, that is, the number of iterations. In Part 2 in the code, I use the Mpmath library to compute the digits of e, to double check that my algorithm yields the correct digits as advertised. Finally, the envelope in Figure 11 is produced as a plot in Part 3, rather than the default scatterplot. The code to produce Figure 8 is in Part 4.

```
n = 100000
H = int(1.15*n)

def assign_color(k):

   if k%12 == 0:
      color = 'lime'
   elif k%12 == 2:
      color = 'white'
   elif k%12 == 4:
      color = 'black'
   elif k%12 == 6:
      color = 'cyan'
   elif k%12 == 8:
      color = 'gold'
   elif k%12 == 10:
      color = 'orange'
   elif k%12 == 1:
      color = 'magenta'
   elif k%12 == 3:
      color = 'paleturquoise'
   elif k%12 == 5:
      color = 'khaki'
   elif k%12 == 7:
      color = 'yellow'
   elif k%12 == 9:
      color = 'mistyrose'
   elif k%12 == 11:
      color = 'orangered'
   return(color)


#--- Part 1. Main

import gmpy2
import numpy as np

kmin = 0.00 * n # compute digit sum if k > kmin
kmax = 1.15 * n # compute digit sum if k < kmax
kmax = min(H, kmax)
```

```
# precision set to L bits to keep at least about n
    correct bits till k=kmax
ctx = gmpy2.get_context()
L = n + int(kmax+1)
ctx.precision = L

# p = 1: for e; ~ n correct bits after n iter
#  seed with n+1 bits, all 0 except rightmost and
    leftmost
# p = 3: for e^3, ~ n correct bits after n iter
#  seed with n+1 bits, all 0 except 2 rightmost and
    leftmost
# p = -1: for 1/e, ~ n correct bits after n iter
#  seed with n bits, all 1
# p must be integer != 0; use p=0 for random seed

p = 1 # try -1, 0 (random seed), 1, 3

def create_random_seed(n, cnt1):

   # create random seed of length n-1 with cnt1 '1'
       at random locations
   # and add a 1' at both ends; cnt1 must be <= n-1

   cnt1 = min(cnt1, n-1)
   numpy_seed = 453 # numpy seed to initiate numpy
       PRNG, not the model seed
   np.random.seed(numpy_seed)
   random_locations = np.random.choice(np.arange(1,
       n),size=cnt1,replace=False)
   prod = 2**n + 1 # seed with n+1 '0' except a '1'
       at both ends
   for position in random_locations:
      # add the cnt1 random '1's between both ends
      prod += 2**int(position)
   return(prod)


# create seed with n+1 bits if p>=0, or n bits if
    p<0
if p != 0:
   prod = gmpy2.mpz(2**n + p)
else:
   # random seed with number of '1' in seed to
       cnt1+2
   # test: set n = 30000; cnt1 = 0, 3, 4, 5, 100
       and see what happens!
   cnt1 = 3
   prod = create_random_seed(n, cnt1)

# local variables
arr_count1 = []
arr_colors = []
xvalues = []
ecnt1 = -1

OUT = open("digit_sum.txt", "w")

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

   prod = prod*prod
   pstri = bin(int(prod))
   stri = pstri[0: L+2]
   prod = int(stri, 2)
   prod = gmpy2.mpz(prod)

   if k > kmin and k < kmax:
      stri = stri[2:]
      if k == n:
         e_approx = stri
      estri = stri[0:n] # leftmost n digits
      ecnt1 = estri.count('1')
      arr_count1.append(ecnt1)
      color = assign_color(k)
      arr_colors.append(color)
      xvalues.append(k)
      OUT.write(str(k) + "\t" + str(ecnt1) + "\n")
   if k%1000 == 0:
      print("%3d %3d" %(k, ecnt1))

OUT.close()


#--- Part 2. Double-check the digits of e

from mpmath import mp

# Set precision to L binary digits
mp.dps = int(L*np.log2(10))
e_value = (mp.e)**abs(p) # Get e^|p| in decimal

if p > 0:
   # Convert e_value to binary string
   e_binary = bin(int(e_value * (2 ** n)))[2:]

elif p < 0:
   e_iapprox = int(e_approx, 2) # convert string
       e_approx to integer
   e_ivalue = int(2**(2*n) * e_value)
   one = e_iapprox * e_ivalue
   e_approx = bin(one)[2:]
   e_binary = "1" * (2*n) # string of 2n bits, all
       '1'

if p != 0:
   k = 0
   while e_approx[k] == e_binary[k]:
      k += 1
   # e_binary should be equal to e_approx up to
       about n bits
   print("\n%d correct digits (n = %d)" %(k, n))


#--- Part 3. Create the main plot

import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np

mpl.rcParams['axes.linewidth'] = 0.5
plt.rcParams['xtick.labelsize'] = 8
plt.rcParams['ytick.labelsize'] = 8
plt.rcParams['axes.facecolor'] = 'black'

plt.scatter(xvalues, arr_count1,s=0.01,
    c=arr_colors)
# plt.plot(xvalues, arr_count1,linewidth=0.04,
    c='gold')

plt.axhline(y=n/2,color='red',linestyle='--',
    linewidth=0.6,dashes=(5,10))
plt.axhline(y=n/5, color='black', linestyle='--',
    linewidth = 0.6, dashes=(5, 10))
plt.axvline(x=n, color='red', linestyle='-',
    linewidth = 0.6, dashes=(5, 10))

for k in range(1,15):
   plt.axvline(x=k*n/(k+1),c='gray',
       linestyle='--', linewidth=0.6,dashes=(5, 10))

if p > 0:
   # start with about 0% of 1 going up to about 50%
   ymax = 0.52 * n
   plt.ylim([-0.01 * n, ymax])
elif p < 0:
   # start with 100% of 1 going down to about 50%
   ymax = 1.01 * n
   plt.ylim([0.40 * n, ymax])
elif p == 0:
   ymax = 1.00 * n
   plt.ylim([-0.01 * n, ymax])
plt.xlim([kmin, kmax])
```

```
plt.show()

#--- Part 4. Create the average plot

arr_avg = []
arr_xval = []
arr_count1 = np.array(arr_count1)

for k in range(0, int(kmax-12)):
   y_avg = np.average(arr_count1[k:k+12])
   arr_avg.append(y_avg)
   arr_xval.append(k)

plt.scatter(arr_xval,arr_avg,s=0.0002, c='gold')
plt.axhline(y=n/2,color='red',linestyle='--',
    linewidth=0.6,dashes=(5,10))
plt.axhline(y=n/5, color='black', linestyle='--',
    linewidth = 0.6, dashes=(5, 10))
plt.axvline(x=n, color='red', linestyle='-',
    linewidth = 0.6, dashes=(5, 10))
plt.xlim(0.00*n, kmax-12)
plt.ylim(-0.01*n, ymax)

for k in range(1,15):
   plt.axvline(x=k*n/(k+1),c='gray',linestyle='--',
       linewidth=0.6,dashes=(5, 10))

plt.show()
```

REFERENCES

[1] Franklin T. Adams-Watters and Frank Ruskey. Generating functions for the digital sum and other digit counting sequences. *Journal of Integer Sequences*, 12:1–9, 2009. [Link]. 5

[2] Christoph Aistleitner et al. Normal numbers: Arithmetic, computational and probabilistic aspects. 2016. Workshop [Link]. 5

[3] David Bailey, Jonathan Borwein, and Neil Calkin. *Experimental Mathematics in Action*. A K Peters, 2007. 5

[4] Verónica Becher, A. Marchionna, and G. Tenenbaum. Simply normal numbers with digit dependencies. *Mathematika*, 69:988–991, 2023. arXiv:2304.06850 [Link]. 5

[5] James Dolan. Carrying is a 2-cocycle. *Preprint*, pages 1–9, 2023. [Link]. 5

[6] Faiza Firdousi, Syeda Iram Batool, and Muhammad Amin. A novel construction scheme for nonlinear component based on quantum map. *International Journal of Theoretical Physics*, 58:3871–3898, 2019. [Link]. 5

[7] Vincent Granville. *Gentle Introduction To Chaotic Dynamical Systems*. MLTechniques.com, 2023. [Link]. 1, 5

[8] Vincent Granville. *Building Disruptive AI & LLM Technology from Scratch*. MLTechniques.com, 2024. [Link]. 5

[9] 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]. 5

[10] Mohammadamin Moradi et al. Data-driven model discovery with Kolmogorov-Arnold networks. *Preprint*, pages 1–6, 2024. arXiv:2409.15167 [Link]. 6

[11] K.S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1:4–27, 1990. [Link]. 5

[12] Yury V. Tiumentsev and Mikhail V. Egorchev. *Neural Network Modeling and Identification of Dynamical Systems*. Elsevier, 2019. 5

[13] Chukwudubem Umeano and Oleksandr Kyriienko. Ground state-based quantum feature maps. *Preprint*, pages 1–8, 2024. arXiv:2024.07174 [Link]. 5

[14] Joseph Vandehey. On the binary digits of $\sqrt{2}$. *Preprint*, pages 1–6, 2017. arXiv:1711.01722 [Link]. 5

[15] Troy Vasiga and Jeffrey Shallit. On the iteration of certain quadratic maps over GF(p). *Discrete Mathematics*, 277:219–240, 2004. [Link]. 5

[16] Henry S. Warren. *Hacker's Delight*. Addison-Wesley Professional, second edition, 2012. 5

[17] Rose Yu and Rui Wang. Learning dynamical systems from data: An introduction to physics-guided deep learning. *Proceedings of the National Academy of Sciences of the United States of America*, 121, 2024. [Link]. 5

Vincent Granville Vincent Granville is a pioneering GenAI scientist, co-founder at BondingAI.io, the LLM 2.0 platform for hallucination-free, secure, in-house, lightning-fast Enterprise AI at scale with zero weight and no GPU. He is also author (Elsevier, Wiley), publisher, and successful entrepreneur with multi-million-dollar exit. Vincent's past corporate experience includes Visa, Wells Fargo, eBay, NBC, Microsoft, and CNET. He completed a post-doc in computational statistics at University of Cambridge.