Appendix B

Quantum, chaotic and fractal types of algorithmic convergence

The problems in this appendix complement the material discussed in the book. Section B.1 deals with the digit sum, in particular with its generating and cumulative functions. It leads to an interesting fractal convergence behavior featured in Figure B.1. Section B.2 discusses linear recurrences with non-fractal, chaotic convergence, illustrated in Figures B.2 and B.3.

B.1 Digit count generating function and fractal convergence

Let H_k be the number of 1 in the binary expansion of the positive integer k, also called the hamming weight of k. These weights are listed as the sequence A000120 in the online encyclopedia of integer sequences (OEIS), see here. The Hamming weight normalized cumulative function is pictured in Figure B.1 and defined as

$$S_H(n) = \frac{1}{n \log_2 n} \sum_{k=0}^{n} H_k, \quad n = 2, 3, 4, \ldots \tag{B.1}$$

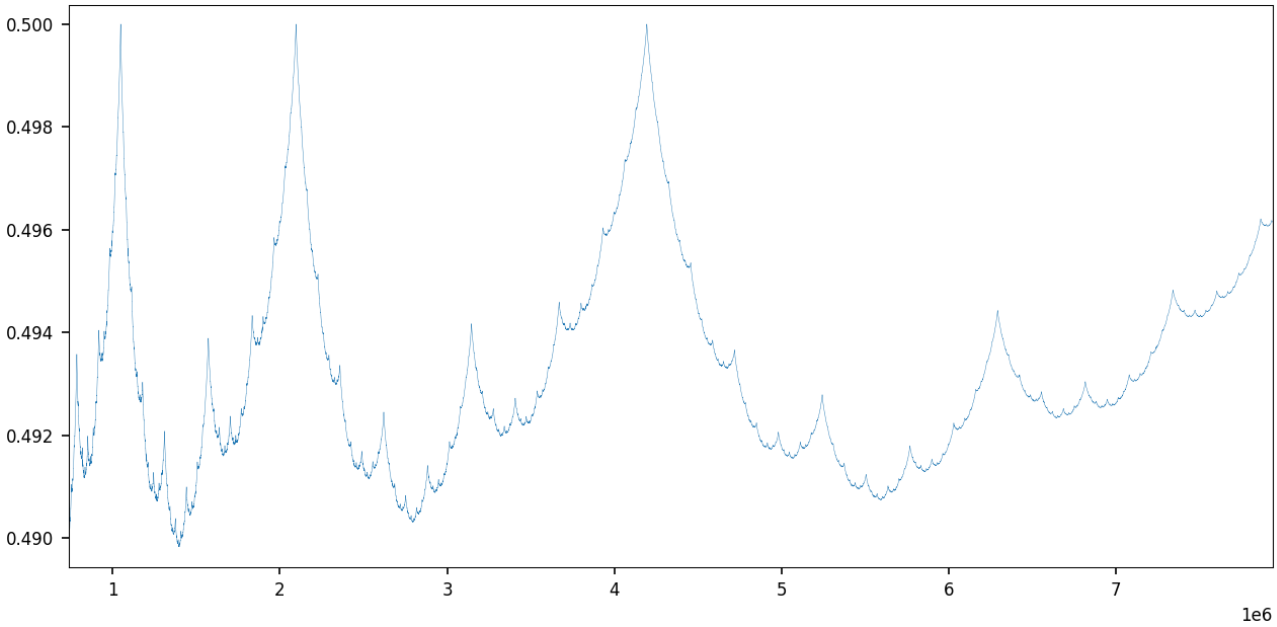

Figure B.1: Hamming weights normalized cumulative function $S_H(n)$ with $n \leq 8 \times 10^6$ on the X-axis

Clearly, $S_H(n) \to \frac{1}{2}$ as $n \to \infty$ with the peaks occurring when n is a power of 2. The curve is fractal-like: the graph between two successive peaks is identical to that between the two previous peaks, but stretched horizontally by a factor 2, and with minima increasing over time, thus causing some vertical compression as n increases. See below the code for the computations and plot generation. The code is also on GitHub, here.

```
import numpy as np
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
plt.rcParams['legend.fontsize'] = 'x-small'

def countSetBits(n):
    # compute H(n)
    count = 0
    while(n):
        count += n & 1
        n >>= 1
    return(count)

sum = 0
arr_k = []
arr_sum = []

for k in range(0,8000000):
    d = countSetBits(k)
    sum += d
    arr_k.append(k)
    if k < 2:
        arr_sum.append(0)
    else:
        arr_sum.append(sum/(k*np.log2(k)))
    if k % 10 == 0:
        print("sss", k, d, sum)

plt.plot(arr_k, arr_sum, linewidth = 0.2)
plt.show()
```

Now I discuss some interesting properties related to the generating functions attached to H_n. It is easy to prove that

$$\Lambda_n(\lambda, x) := \prod_{k=0}^{n-1} \left(1 + e^{\lambda} x^{2^k}\right) = \sum_{k=0}^{2^n-1} e^{\lambda H_k} x^k \tag{B.2}$$

Thus $\Lambda_n(0, x) = (1 - x^{2^n})/(1 - x)$. By virtue of (B.2), we also have

$$\Lambda_n(\lambda, 1) = \left(1 + e^{\lambda}\right)^n = \sum_{k=0}^{n} \binom{n}{k} e^{k\lambda} = \sum_{k=0}^{2^n-1} e^{\lambda H_k} \tag{B.3}$$

Now, taking the m-th derivative with respect to λ and then setting $\lambda = 0$, we obtain

$$\sum_{k=0}^{2^n-1} \left(H_k\right)^m = \sum_{k=0}^{n} \binom{n}{k} k^m = 2^n n! \sum_{k=0}^{m} \frac{S(m,k)}{2^k (n-k)!}, \quad m = 0, 1, 2 \ldots \tag{B.4}$$

where $S(\cdot\,, \cdot)$ denotes the Stirling numbers of the second kind. The left identity in (B.4) follows from (B.3) and is true even if m is not an integer. The expression on the right in (B.4) corresponds to the m-th moment of a Bernoulli distribution of parameters $(n, \frac{1}{2})$, multiplied by 2^n.

The standard form of the Hamming weights generating function, identical to that of binary digit sums, is studied in [36]. See also [39]. The main result is Theorem 1 in [36], stating that

$$\sum_{k=0}^{\infty} H_k x^k = \frac{1}{1-x} \sum_{m=0}^{\infty} \frac{x^{2^m} - 2x^{2^{m+1}} + x^{3 \cdot 2^m}}{(1 - x^{2^m})(1 - x^{2^{m+1}})}. \tag{B.5}$$

The "digit sum" entry in Wolfram also features beautiful results, see here. Among others:

$$\sum_{k=1}^{\infty} \frac{H_k}{k(k+1)} = \log 2, \quad \sum_{k=1}^{\infty} \frac{(2k+1) H_k}{k^2 (k+1)^2} = \frac{\pi^2}{9}$$

$$\sum_{k=1}^{\infty} \frac{H_k + H'_k}{2k(2k+1)} = \gamma, \quad \sum_{k=1}^{\infty} \frac{H_k - H'_k}{2k(2k+1)} = \log \frac{4}{\pi}$$

where γ is the Euler-Mascheroni constant and H'_k is the number of 0 in the binary expansion of k.

The Hamming weights H_n can be computed with the recursion $H_{2n} = H_n$, $H_{2n+1} = H_n + 1$ with $H_0 = 0$. A related sequence with many interesting properties is Stern's diatomic series (A_n) with $A_0 = 0$ and $A_1 = 1$, defined by the recursion $A_{2n} = A_n, A_{2n+1} = A_n + A_{n+1}$. Finally, the integer sequence defined by $C_n = \frac{1}{2}C_{n-1}$ if C_{n-1} even, $C_n = 3C_{n-1} + 1$ otherwise, is presumed to converge to 1 regardless of the initial condition $C_0 > 0$. This is known as the Collatz conjecture.

B.2 Other examples of chaotic convergence

For centuries, mathematicians worked on problems where the convergence is either smooth or does not happen. Now the concept of chaotic convergence is mainstream, popularized by the stochastic gradient descent in deep neural networks, central to LLMs. In this book, most cases also involve various types of chaotic convergence, for instance in Figures 4.16, 5.1, 6.7, 6.8, and 6.11. In some examples such as Figure B.1, the chaos exhibits a fractal structure. In other examples such as Figures 2.3, 2.7, 6.4, and 6.9, it looks like we have multiple curves converging to a same limit, when in reality there is only one, with jumps from one level to another every millisecond: this pattern is strikingly similar to quantum states. This section features more illustrations falling in these various categories.

For examples of chaotic convergence in the context of explainable deep neural networks, see Figure 4.6 featuring stochastic gradient descent, Figure 4.14 featuring swarm optimization, and Figure 4.2 featuring adaptive loss function all in my book on no-Blackbox LLM architectures [22]. Yet another type is asymptotic periodicity where x_n converges to multiple values depending on the residue class of n modulo some integer. See section 5.2.1.

B.2.1 Smooth convergence but with multiple branches

The first example deals with the recursion $x_{n+3} = 2x_{n+2} - 16x_{n+1} + 4x_n$ with initial conditions $x_0 = 0$, $x_1 = 1$, and $x_2 = 1$. We are interested in the limit

$$\rho := \lim_{n\to\infty} |x_n|^{1/n} \tag{B.6}$$

if it exists. It does in this example. To compute it, proceed as follows:

- Find the roots of the characteristic polynomial, in this case the cubic equation $x^3 = 2x^2 - 16x + 4$.
- Identify the root with the largest norm. Here we have two complex conjugate roots with the same norm, and one real root. The complex roots have the largest norm.
- Then ρ is the square root of the norm in question. More specifically,

$$\rho = \sqrt{\frac{12\omega}{2\omega - \omega^2 + 44}}, \quad \text{with } \omega = \sqrt[3]{82 + 6\sqrt{2553}} \tag{B.7}$$

First, I used AI to find ρ. The LLMs that I tested correctly identify $r \approx 0.2572$ as the real root of the cubic equation, and $\rho \approx 3.9436$ as the solution. Perplexity even mentioned that $\rho = 2\,||r||^{-1/2}$, which is correct. But they all failed when providing the exact values for the roots, thus the exact value for ρ was also incorrect. To be fair, I used the free version of these tools. Then, I used the `solve` function from the SymPy Python library. It is based on symbolic mathematics to find exact solutions. That's how I obtained (B.7).

The most interesting part is how $\rho_n = |x_n|^{1/n}$ converges to ρ as $n \to \infty$, see Figure B.2 where ρ_n on the Y-axis has a different color depending on n mod 7, with n on the X-axis. We jump from one branch to another each time n is incremented, yet eventually ρ_n converges to n. The associated Python code is listed in section B.2.2.

B.2.2 Chaotic convergence with multiple branches

I now discuss the case $x_{n+2} = |x_{n+1} - 3x_n|$ with initial conditions $x_0 = 1$ and $x_1 = 1$. Again, we are interested in the same limit ρ defined by (B.6). At first glance, this case seems easier as the characteristic polynomial is now of degree 2 instead of 3. However, the absolute value in the recursion makes it different with a dramatic shift in behavior as seen in Figure B.3, by contrast to Figure B.2. Both plots show $|x_n|^{1/n}$ on the Y-axis with n on the X-axis.

It is reminiscent of the problem about random Fibonacci sequences discussed in section 7.4 in [22]. In that example, the equivalent of the ratio $r_n = x_{n+1}/x_n$ asymptotically oscillates between 5 different values. Taking the geometric mean of these values yields the correct, exact value for ρ. However the situation is a lot more complicated here. Even though empirical evidence points to the limit $\rho \approx 1.58010$, it is not obvious to

prove convergence, let alone putting an exact value on ρ. Interestingly, some LLMs erroneously believe that the answer is $\rho = \sqrt{3}$. For a deep dive into this partular case, see setion B.2.3.

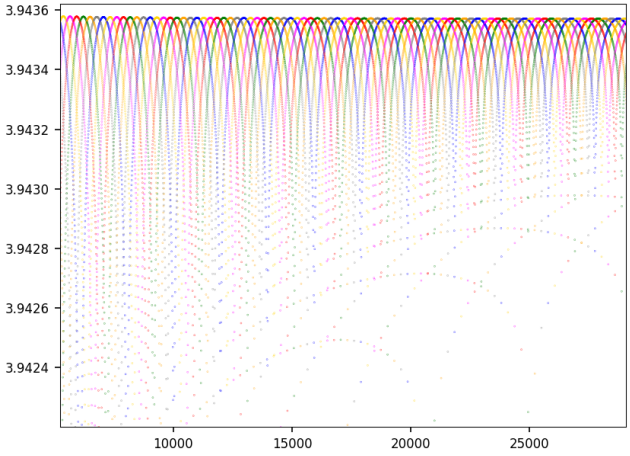

Figure B.2: Constant jumps in $|x_n|^{1/n}$ gives the illusion of 7 curves but there is only one (section B.2.1).

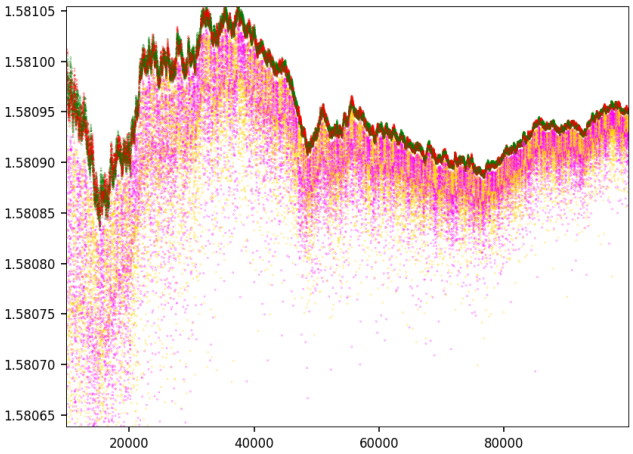

Figure B.3: 4-colors quantum regime similar to that of Figure B.2 but now with chaos (section B.2.2).

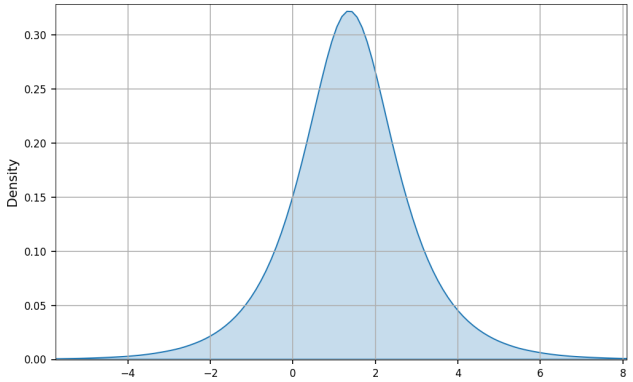

Figure B.4: Empirical PDF for $\log|x_{n+1}/x_n|$, case featured in Figure B.2

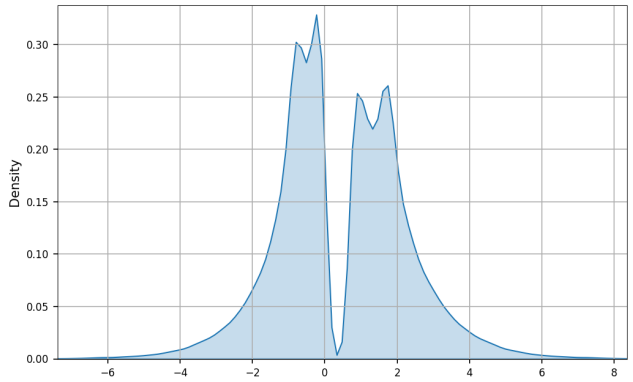

Figure B.5: Empirical PDF for $\log|x_{n+1}/x_n|$, case featured in Figure B.3

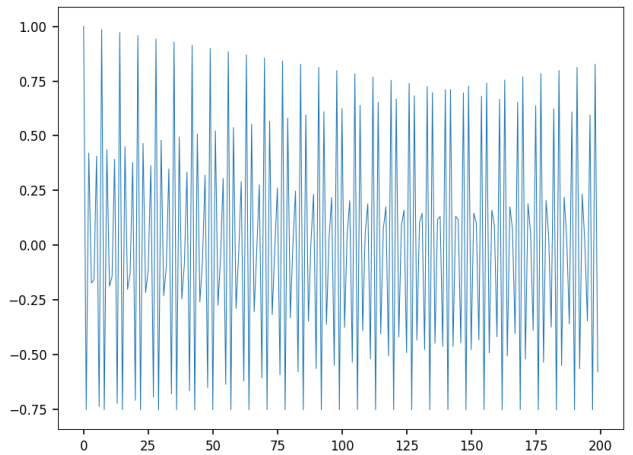

Figure B.6: Autocorrel function for $\log|x_{n+1}/x_n|$, case featured in Figure B.2

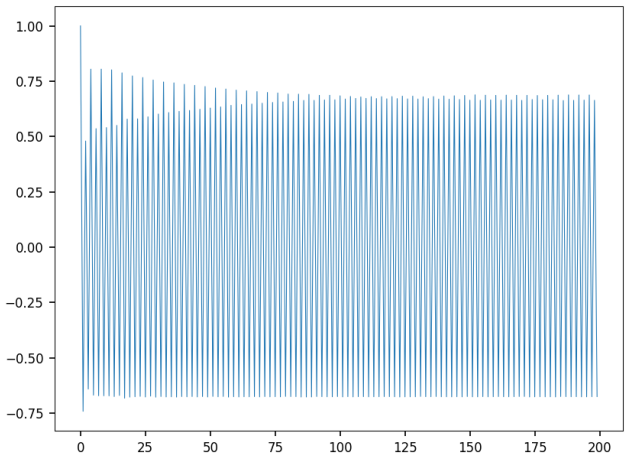

Figure B.7: Autocorrel function for $\log|x_{n+1}/x_n|$, case featured in Figure B.3

The curve $\rho_n = |x_n|^{1/n}$ has 4 chaotic bands. For a specific n, the band that ρ_n belongs to depends on n mod 4. The green and red bands are interlaced but distinct. The yellow and pink bands are somewhat more separated. This is further confirmed when looking at the empirical autocorrelation function pictured in Figure B.7, showing the correlation between the sequences $\log|x_{n+1}/x_n|$ and $\log|x_{n+k+1}/x_{n+k}|$ for $0 \leq k < 200$. A cycle of length 4

is clearly visible. By contrast, the case studied in section B.2.1 has a cycle of length 7, as pictured in Figure B.6, which also explains the 7 branches in Figure B.2.

Finally, I looked at the empirical probability density function (EPDF in Python), computed on the first 10^5 values of $\log|x_{n+1}/x_n|$, skipping the first 10,000 ones. Not surprisingly, the EPDF is smooth and classic for the case featured in Figure B.2, but bumpy with 4 main peaks and a deep valley for the example discussed here. You can see the contrast by comparing Figures B.4 and B.5 respectively. To conclude, the dark upper boundary in Figure B.3 looks like a Brownian motion to the layman, but it is in fact very different. Ours converges (the variance tends to zero as n increases) while the variance of a Brownian motion keeps growing. However, this is a small issue: rescaling would easily fix it. The main difference is that our curve is significantly more chaotic, and thus suitable as a model when increased chaos is needed. Metrics measuring the amount of chaos are discussed in section 2.3 in my book on chaotic dynamical systems [15].

Below is the Python code used to do the analyses and producing the plots in sections B.2.1 and B.2.2. The code is also on GitHub, here. The `mode` parameter allows you to choose between the recursion in section B.2.1 and that in section B.2.2.

```
import numpy as np
import gmpy2
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
plt.rcParams['legend.fontsize'] = 'x-small'

ndigits = 10000
ctx = gmpy2.get_context()
ctx.precision = ndigits
z = gmpy2.log(2)

# choose N < ndigits/2 to avoid losing precision
N = 5000
mode = 'quantum' # options: 'quantum' or 'chaotic'

x0 = gmpy2.mpfr(0)
x1 = gmpy2.mpfr(1)
x2 = gmpy2.mpfr(1)
arr_k = []
arr_x = []
arr_col = []
arr_log_rho = []

def set_color(k, mode):

   if mode == 'quantum':
      if k % 7 == 0:
         color='red'
      elif k % 7 == 1:
         color='blue'
      elif k % 7 == 2:
         color='green'
      elif k % 7 == 3:
         color='gold'
      elif k % 7 == 4:
         color='orange'
      elif k % 7 == 5:
         color='magenta'
      elif k % 7 == 6:
         color='gray'

   elif mode == 'chaotic':
      if k % 4 == 0:
         color = 'red'
      elif k % 4 == 1:
         color = 'gold'
      elif k % 4 == 2:
         color = 'green'
      elif k % 4 == 3:
         color = 'magenta'
      else:
         color = 'gray'
   else:
```

```
        print("Unsuported mode:", mode)
        exit()
    return(color)


#--- Main

modulus = 4
hash_modulo = {}

for k in range(2,N):
    color = set_color(k, mode)
    if mode == 'quantum':
        x = 2*x2 - 16*x1 + 4*x0
        delta = 0
    elif mode == 'chaotic':
        x = abs(x2 - 3*x1)
    v = gmpy2.log(abs(x))/k
    w = gmpy2.exp(v)
    rho   = float(x/x2)
    if mode == 'chaotic':
        old_rho = float(x2/x1)
        # delta should be 0 for k < ndigits/2
        delta = rho**2 - 1 + 6/old_rho - 9/old_rho**2
    log_rho = np.log(abs(rho))
    if k > N/10:
        print("%6d log_rho: %8.5f %8.5f %f" % (k, log_rho, delta, rho))
        arr_k.append(k)
        arr_x.append(w)
        arr_col.append(color)
        arr_log_rho.append(log_rho)
        residue = k % modulus
        if residue in hash_modulo:
            arr_local = hash_modulo[residue]
            arr_local.append(log_rho)
            hash_modulo[residue] = arr_local
        else:
            hash_modulo[residue] = [log_rho]
    x0 = x1
    x1 = x2
    x2 = x

plt.scatter(arr_k, arr_x, c=arr_col, s=0.02)
plt.show()
plt.scatter(arr_k, arr_log_rho, c=arr_col, s=0.02)
plt.show()
plt.plot(arr_k, arr_log_rho, linewidth=0.4)
plt.show()
print("\nRho[residue] with modulus = %2d" %(modulus))
avg = 1
for residue in range(modulus):
    arr_vals = hash_modulo[residue]
    arr_vals = np.array(arr_vals)
    arr_exp = np.exp(arr_vals)
    iqr_rho = np.quantile(arr_exp,0.75) - np.quantile(arr_exp, 0.25)
    mu = np.average(arr_vals)
    mean_rho = np.exp(mu)
    median = np.std(arr_vals)
    avg *= mean_rho
    print("residue %2d | mu = %8.5f | rho = %8.5f | irq = %8.5f | median = %8.5f"
              % (residue, mu, mean_rho, iqr_rho, median))
avg = avg**(1/modulus)
print("Rho: %8.5f" %(avg))


#--- Plot EPDFs

np_log_rho = np.array(arr_log_rho)
import seaborn as sns
plt.figure(figsize=(8, 5))
# sns.ecdfplot(np_rho)
sns.kdeplot(data=np_log_rho, fill=True)
plt.grid(True)
plt.show()

meanlog = np.mean(np_log_rho)
```

```
medianlog = np.median(np_log_rho)
mean = np.exp(meanlog)
median = np.exp(medianlog)
print("\nRho: %8.5f [median = %8.5f | meanlog = %8.5f]" % (mean, median, meanlog))
print()


#--- Compute autocorrels for log(x/x2)

nobs = len(np_log_rho)
arr_lag = []
arr_autocorrel = []

for lag in range(200):
   mean1 = np.mean(np_log_rho[0: nobs-lag])
   mean2 = np.mean(np_log_rho[lag:nobs])
   std1 = np.std(np_log_rho[0: nobs-lag])
   std2 = np.std(np_log_rho[lag:nobs])
   dotprod = np.dot(np_log_rho[0: nobs-lag], np_log_rho[lag:nobs])
   dotprod /= (nobs-lag)
   autocorrel = (dotprod - mean1*mean2)/(std1*std2)
   arr_lag.append(lag)
   arr_autocorrel.append(autocorrel)
   print("Autcorrel lag %3d: %8.5f" %(lag, autocorrel))

plt.plot(arr_lag, arr_autocorrel, linewidth = 0.5)
plt.show()
```

B.2.3 Deep dive into the chaotic case

Let's $r_n = x_{n+1}/x_n$ and $s_n = \log r_n$, with $r_n \geq 0$ for all n. Then the case $x_{n+2} = |x_{n+1} - 3x_n|$ can be rewritten in two different ways, as follows:

$$r_{n+1} = \left|1 - \frac{3}{r_n}\right| \tag{B.8}$$

$$s_{n+1} = \log\left|1 - 3\exp\left(-s_n\right)\right| \tag{B.9}$$

Also, with $x_0 = x_1 = 1$, we have:

$$x_n = \prod_{k=0}^{n-1} r_k, \qquad \log x_n = \sum_{k=0}^{n-1} s_k, \qquad \rho = \exp\left(\mu\right) \text{ with } \mu = \lim_{n\to\infty} \frac{1}{n}\sum_{k=0}^{n-1} s_k \tag{B.10}$$

Both (B.8) and (B.9) are discrete chaotic dynamical systems. The latter is much better behaved, ergodic, with a non-singular invariant measure and a finite ergodic mean equal to μ in (B.10), with $|x_n|^{1/n} \to \rho = e^{\mu}$. In the code, the number `N` of iterations most be much smaller than the precision (set by `ndigits`) otherwise you quickly get values of x_k that are completely wrong. However, thanks to the ergodicity, you may use a small `ndigits` and a large `N` to massively accelerate the computations without impacting the precision on ρ.

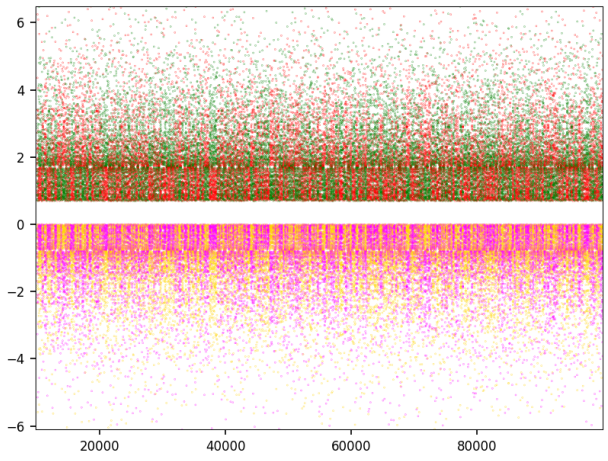

Figure B.8: Same as Figure B.3 but now with s_n instead of $|x_n|^{1/n}$ on he Y-axis (n on the X-axis)

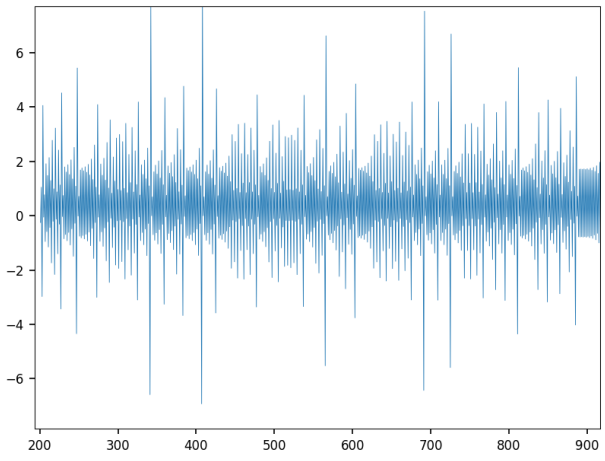

Figure B.9: Same as Figure B.8 with standard plot instead of scatterplot, thus missing the empty band

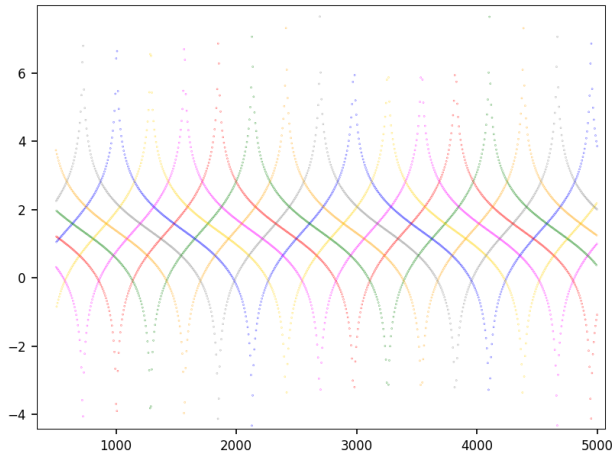

Figure B.10: Same as Figure B.2 but now with s_n instead of $|x_n|^{1/n}$ on he Y-axis (n on the X-axis)

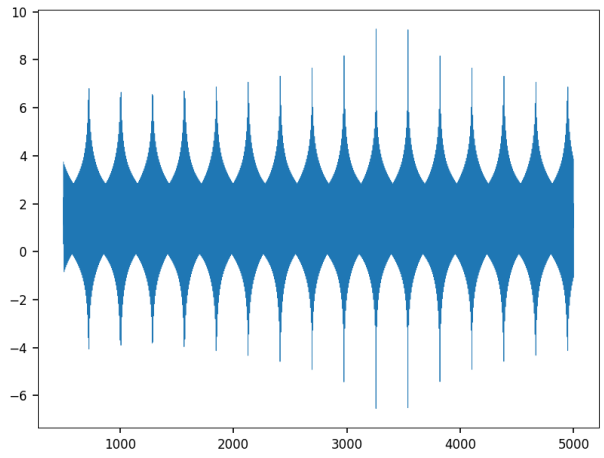

Figure B.11: Same as Figure B.10 with standard plot instead of scatterplot. Provided solely as a piece of art

I now state a fundamental result, which is a direct application of the pigeonhole principle. Let the indices $0, 1, 2$ and so on (that is, all the positive integers) be partitioned into m non-overlapping sets $S_0, \ldots, S_{m-1}$, where m may be finite or infinite. Let p_j be the proportion of indices in set S_j, with $p_0 + \cdots + p_{m-1} = 1$. Also, let $S_j(n) = \{i \in S_j \text{ such that } i \leq n\}$, and $|S_j(n)|$ be the number of elements in $S_j(n)$. Then

$$\rho = \exp\left[\sum_{j=0}^{m-1} p_j \mu_j\right], \quad \text{with } \mu_j = \lim_{n\to\infty} \mu_j(n) \text{ and } \mu_j(n) := \frac{1}{|S_j(n)|} \sum_{i\in S_j(n)} s_i. \tag{B.11}$$

I used (B.11) to compute ρ for random Fibonacci sequences discussed in section 7.4 in [22]. In this section, S_j consists of the positive integers congruent to j modulo m. The goal is to find a modulus m for which the ergodic means μ_j ($j = 0, 1, \ldots, m-1$) are distinct. This is true if $m \in \{2, 4\}$, but not for $m \in \{3, 5\}$. It shows a pattern, pictured with $m = 4$ colors in Figures B.8 and B.3. In the Python code in section B.2.2, $n, j, m, \mu_j(n)$ are denoted respectively as `N`, `residue`, `modulus` and `mu`. This is further summarized in table B.1.

$m=2$		$m=3$		$m=4$		$m=5$	
j	$\mu_j(n)$	j	$\mu_j(n)$	j	$\mu_j(n)$	j	$\mu_j(n)$
0	1.92912	0	0.46528	0	1.89867	0	0.45958
1	-1.01310	1	0.45667	1	-1.04106	1	0.45089
		2	0.45204	2	1.96136	2	0.46003
				3	-0.98514	3	0.45794
						4	0.46153

Table B.1: $\mu_j(n)$ for $2 \leq m \leq 5$, with $n = 10^5$

In each column in Table B.1, the average $\mu_j(n)$ is about 0.458, which is a good approximation to the theoretical μ. But variations between columns are large, pointing to 4 distinct values when $m = 4$. This is reflected in Figure B.5 with two major peaks, each one being a double summit.

I did not compute the invariant measure attached to (B.9), defined as the CDF or cumulative distribution function $F_Z(z)$ solution to the functional equation

$$F_Z(z) = F_Z\Big(\log\Big|1 - 3\exp\big(-z\big)\Big|\Big). \tag{B.12}$$

In many examples like the logistic map, F_z has a closed-form expression. But this is not the case here, though you can solve (B.12) numerically with an iterative algorithm, to get an approximation: the empirical CDF. An example is discussed in section 2.2.1 in my book on chaotic dynamical systems [15]. If you were able to find a closed-form expression, you can verify it via the Perron-Frobenius operator, also called transfer operator [Wiki]. Also, the theoretical ergodic mean is the expectation $\mathrm{E}[Z]$ attached to F_Z. I asked LLMs to find F_z or $\mu = \mathrm{E}[Z]$ and mostly got erroneous theoretical results yet good approximation for μ. Finally, (B.11) can be rewritten as

$$\rho = e^{\mu}, \quad \mu = \mathrm{E}[Z] = \sum_{j=0}^{m-1} p_j \mu_j. \tag{B.13}$$

Finally, Figures B.10 and B.11 are related to the smooth example discussed in section B.2.1 and should be compared respectively with Figures B.8 and B.9 related to the chaotic case discussed in this section. It also illustrates the fact that a standard plot hides some important patterns, thus the preference for a scatterplot.

Technical note: The horizontal empty band in Figure B.8 is between the values 0 and $\log 2$ on the Y-axis. There are other horizontal bands with low density but non-empty, some visible to the naked eye or by zooming in. These bands are the same regardless of the initial conditions $x_0, x_1 > 0$. The empty band is responsible for the gap between the two main peaks in Figure B.5. The explanation is simple: if $x_n > \log 2$ then $x_{n+1} < 0$, and if $x_n < 0$ then $x_{n+1} > \log 2$. One way to get rid of the empty band is by splitting the dynamical system into two subsystems $t_n = s_{2n}, t'_n = s_{2n+1}$ with the same recursion $t_{n+1} = g(g(t_n)$, $t'_{n+1} = g(g(t'_n)$, with $g(t) = \log|1 - 3\exp(-t)|$.

B.2.4 Interesting connection between 3^n and the digits of $\sqrt{2}$

The findings in this section apply to the positive integer powers q^n of an integer $q > 1$ not a power of 2. It is not limited to just $q = 3$, although I spent most of my research on the latter. I start with an established theorem.

Theorem B.2.1 *Let $q > 1$ be an integer, not a power of 2. For $n = 0, 1$ and so on, let ν_n be the dyadic rational*

$$\nu_n = \frac{q^n}{2^{\lfloor n \log_2 q \rfloor}}. \tag{B.14}$$

Let $\nu_{(n)}$ be the median computed on the first $2n+1$ values $\nu_0, \ldots, \nu_{2n}$. Then $\nu_{(n)} \to \sqrt{2}$ as $n \to \infty$.

Proof

The symbols $\lfloor \cdot \rfloor$ and $\{\cdot\}$ denote respectively the integer part and fractional part functions. Also, $q^n = 2^{n \log_2 q}$ where $\log_2$ is the logarithm in base 2. Thus, ν_n is a dyadic rational in $[1, 2]$, equal to

$$\nu_n = 2^{\{n \log_2 q\}}. \tag{B.15}$$

Now, if α is irrational, then the sequence $\{\alpha n\}$ ($n = 0, 1$ and so on) is equidistributed modulo 1 and dense in $[0, 1]$ by virtue of Weyl's equidistribution theorem. Thus the sequence v_n is dense in $[1, 2]$ because $\alpha = \log_2 q$ is irrational. That is, there are integer subsequences $0 \leq \sigma_1 < \sigma_2 < \sigma_3$ and so on such that ν_{σ_n} gets arbitrarily close to any pre-specified constant in $[1, 2]$ including $\sqrt{2}$, as $n \to \infty$..

Finally, due to the equidistribution, the median values $\nu_{(n)}$ tend to the median of 2^U, where U is a random variable with uniform distribution on [0, 1]. Its median is $\sqrt{2}$. ∎

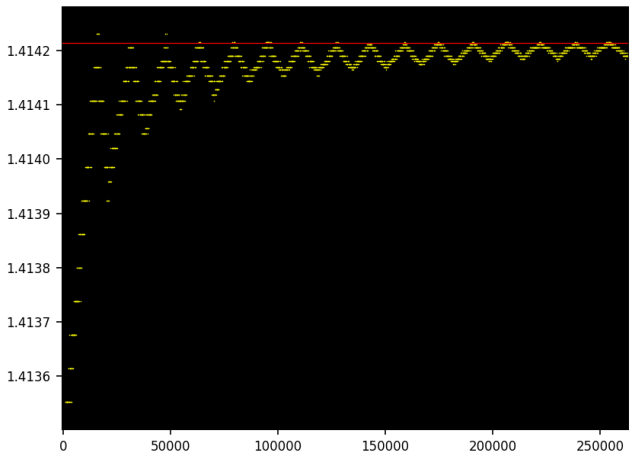

Figure B.12: $\nu(n) = \text{Median}(\nu_0, \ldots, \nu_{2n})$ on the Y-axis, with abcissa $2n + 1$ on the X-axis

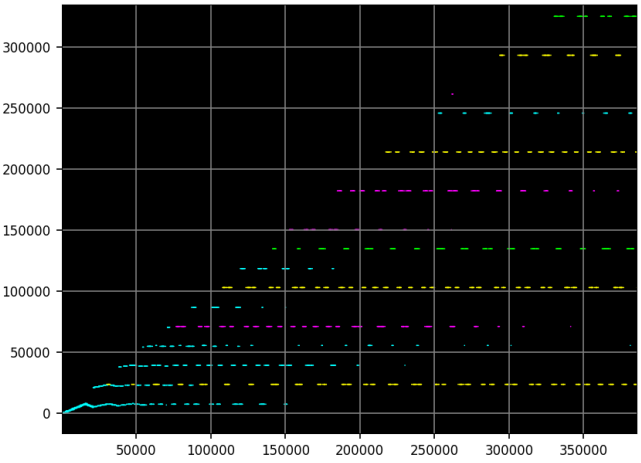

Figure B.13: Index σ_n satisfying $\nu_{(n)} = \nu_{\sigma_n}$ on the Y-axis, with abcissa $2n + 1$ on the X-axis

Theorem B.2.1 states that you can build a subsequence of ν_n that convergences to $\sqrt{2}$. It provides an explicit solution based on the medians $\nu_{(n)}$. The binary digits of $\nu_{(n)}$ match those of $\sqrt{2}$ at the limit. And for each n, $\nu_{(n)}$ is one of the values (the median) in $\{\nu_0, \nu_1, \ldots, \nu_{2n}\}$ since this set has an odd number of elements. Let σ_n be the index of the median value $\nu_{(n)}$, uniquely characterized by $\nu_{(n)} = \nu_{\sigma_n}$, with $0 \leq \sigma_n \leq 2n$. Now I discuss some insights, based on $q = 3$.

- The sequence σ_n defined by $\nu_{(n)} = \nu_{\sigma_n}$ goes up and down, jumping from one horizontal level to another as pictured in Figure B.13. The general trend is up, and eventually $\sigma_\infty = \infty$.

- The colors in Figure B.13 show how close ν_{σ_n} is to the limit $\sqrt{2}$. Green means that only the first $\kappa = 18$ binary digits of $\nu(n)$ match those of $\sqrt{2}$. Yellow, magenta and cyan correspond respectively to $\kappa = 17$, $\kappa = 16$ and $\kappa \leq 15$.
- Figure B.12 shows how slowly $\nu_{(n)}$ convergences to the limit $\sqrt{2}$, staying below the limit (the red line) most of the time. However, it periodically gets close and hits the red line, when n is a multiple of 8,000 (approximately). If you replace $q = 3$ by $q = 5$, the pattern is quite different.
- If you replace the median $\nu_{(n)}$ by an average, then convergence is towards $1/\log 2$ instead of $\sqrt{2}$. Indeed,

$$\frac{1}{n}\sum_{k=1}^{n}\nu_k = \frac{1}{n}\sum_{k=1}^{n}\frac{q^k}{2^{\lfloor k\log_2 q\rfloor}} = \frac{1}{n}\sum_{k=1}^{n}2^{\{k\log_2 q\}} \to \frac{1}{\log 2} \text{ as } n\to\infty. \tag{B.16}$$

  This result is particularly interesting when n is a power of 2. Then all the denominators $2^{\lfloor k\log_2 q\rfloor}$ and n in (B.16) are a power of 2.

For each n, the binary digits of ν_n start with 1, followed by a decimal point. The digits on the right starting after the decimal point are called leading digits. One would think that for any digit string of length κ, its frequency is $2^{-\kappa}$. However, this is not the case. For instance, the leading digit '0' has frequency $\log_2(3/2) \approx 58\%$, different from 50%. Similarly, the string '0000' appears about twice as frequently as '1111' in the leading digits of ν_n. Leading digits frequently follow the Benford's law [Wiki]. But this is not the case here.

Finally, for a fixed $\kappa > 0$ and $q = 3$, let $\varphi(\kappa)$ be the minimum n such that the first κ leading binary digits of ν_n match those of $\sqrt{2}$. That is, the number of binary digits of $q^{\varphi(\kappa)}$ is $\lfloor\varphi(\kappa)\log_2 q\rfloor + 1$, of which only the first κ match those of $\sqrt{2}$. Table B.2 shows how $\varphi(\kappa)$ grows exponentially fast as a function of κ.

κ	$\varphi(\kappa)$	κ	$\varphi(\kappa)$	κ	$\varphi(\kappa)$	κ	$\varphi(\kappa)$
6	6	11	512	17	23,734	21	4,247,415
7	47	13	6,803	18	134,936	24	5,200,100
9	206	15	8,133	20	2,342,045	27	5,390,637

Table B.2: Exponential growth of $\varphi(\kappa)$ as a function of κ

A common mistake when proving that (say) $\sqrt{2}$ is a normal number is as follows. First, one proves that the binary digits of 3^n are equidistributed when $n \to \infty$. Let us pretend that this fact has been proved. Also, for each n, there is an integer σ_n such that $\nu_{(n)}$ and 3^{σ_n} have the same digits. And $\nu_{(n)}$ converges to $\sqrt{2}$. Thus $\sqrt{2}$ is simply normal in base 2. However this argument is fallacious for the following reasons:

- The digits of 3^{σ_n} and $\nu_{(n)}$ are identical. Thus, if the former has about 50% of 1, this is also true for the latter. But what if most of the 1's are on the right half in the digit expansion? At best, the first $O(\log n)$ digits on the left match those of $\sqrt{2}$ while 3^{σ_n} has $O(n)$ digits.
- It is possible to find many infinite subsequences of ν_n that converge to any constant ξ in $[1, 2]$. For instance to $\xi = \frac{3}{2}$. Despite the 50% of 1 guaranteed in 3^n as $n \to \infty$, the limit ξ has a very different proportion.
- Actually, no one knows if the digits of 3^n are equidistributed. It is a major open problem, see [10, 35].

The Python code below performs the computations and generates the plots used in section B.2.4. While I work with large numbers such as 3^n with $n = 10^7$, I only need the first $O(\log n)$ leading digits. Thus the `ndigits` parameter is set to a small value (50), far smaller than n. The latter is represented by `N` in the code. In modern AI, this truncation process is known as quantization. Here, it tremendously accelerates the computations. The code is also on GitHub, here.

```
import numpy as np
import gmpy2
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
plt.rcParams['legend.fontsize'] = 'x-small'

ndigits = 50 # maximum possible match
ctx = gmpy2.get_context()
ctx.precision = ndigits
N = 200000
```

```
string_tests = ('0', '1', '00', '01', '10', '11', '0000', '1111')

power3 = gmpy2.mpz(1)
isqrt2 = gmpy2.isqrt(2 * 2**(2*ndigits))
rescaled2 = isqrt2/ 2**int(gmpy2.log2(isqrt2))
sqrt2 = bin(isqrt2)[2:]

def update_hash(hash, key, count):
   if key in hash:
      hash[key] += count
   else:
      hash[key] = count
   return(hash)

def match_strings(bin3, sqrt2):
   # match = number of consecutive identical chars on the left
   match = 0
   Flag = True
   while match < min(len(bin3)-1, len(sqrt2)-1) and Flag:
      if bin3[match] == sqrt2[match]:
         match += 1
      else:
         Flag = False
   return(match)

arr_match = []
arr_nu = []
hash_strings = {}
hash_match = {}
q = 3
lg23 = np.log2(q)
lg2 = np.log(2)
lg3 = np.log(q)
e2 = 2**ndigits


for k in range(N):

   rescaled3 = gmpy2.exp(k*lg3 - int(k*lg23)*lg2)
   arr_nu.append(float(rescaled3))
   power3 = gmpy2.mpz(e2 * rescaled3)
   bin3 = bin(power3)[2:]
   for string in string_tests:
      if bin3[1:len(string)+1] == string:
         update_hash(hash_strings, string, 1)
   match = match_strings(bin3, sqrt2)
   if match not in hash_match:
      hash_match[match] = k
   arr_match.append(match)

#--- Show frequency of some leading digit strings

print()
for string in hash_strings:
   count = hash_strings[string]
   print("String frequency %5s: %8.6f" % (string, count/N))

#--- Show phi(kappa)

print()
for kappa in hash_match:
   print("phi(%3d) = %6d" %(kappa, hash_match[kappa]))

#--- Plot convergence of median to sqrt(2)

arr_median_k = []
arr_median_idx = []
arr_median_match = []
arr_median_value = []
arr_median_color = []

for k in range(len(arr_nu)//20, len(arr_nu), 100):
   # compute nu((k-1)/2)
   arr = np.array(arr_nu[0: k])
   median_idx = np.argpartition(arr, len(arr) // 2)[len(arr) // 2]
   match = arr_match[median_idx]
```

```
    if match == 18:
       arr_median_color.append('lime')
    elif match == 17:
       arr_median_color.append('yellow')
    elif match == 16:
       arr_median_color.append('magenta')
    else:
       arr_median_color.append('cyan')
    arr_median_k.append(k)
    arr_median_idx.append(median_idx)
    arr_median_match.append(match)
    arr_median_value.append(arr[median_idx])
    if k % 10000 == 1:
       print("Convergence: %6d %6d %8.5f %8.5f"
             %(k, median_idx, arr[median_idx], arr_match[median_idx]))

#--- Plots

plt.rcParams['axes.facecolor'] = 'black'
plt.scatter(arr_median_k, arr_median_value,s=0.6, c='yellow', linewidths=0)
plt.axhline(y=np.sqrt(2), color='r', linewidth = 0.6)
plt.show()
plt.scatter(arr_median_k, arr_median_idx, s=0.6, linewidths=0, c = arr_median_color)
plt.grid(color='gray')
plt.show()
```

Bibliography

[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]. 12, 14, 23, 32, 45

[2] Christoph Aistleitner et al. Normal numbers: Arithmetic, computational and probabilistic aspects. 2016. Workshop [Link]. 12, 23, 32, 45

[3] Adel Alamadhi, Michel Planat, and Patrick Solé. Chebyshev's bias and generalized Riemann hypothesis. *Preprint*, pages 1–9, 2011. arXiv:1112.2398 [Link]. 84

[4] Gökalp Alpan and Maxim Zinchenko. Lower bounds for weighted Chebyshev and orthogonal polynomials. *Preprint*, 2024. arXiv:2408.11496v [Link]. 56

[5] David Bailey, Jonathan Borwein, and Neil Calkin. *Experimental Mathematics in Action*. A K Peters, 2007. 11, 23, 32, 45, 61

[6] Verónica Becher, A. Marchionna, and G. Tenenbaum. Simply normal numbers with digit dependencies. *Mathematika*, 69:988–991, 2023. arXiv:2304.06850 [Link]. 12, 23, 32, 45

[7] Frederik Broucke. On zero-density estimates for Beurling zeta functions. *Preprint*, pages 1–24, 2024. arXiv:2409:1051v1 [Link]. 77

[8] James Dolan. Carrying is a 2-cocycle. *Preprint*, pages 1–9, 2023. [Link]. 12, 23, 32, 45

[9] David Doty, Jack H. Lutz, and Satyadev Nandakumar. Finite-state dimension and real arithmetic. *Information and Computation*, 205:1640–1651, 2007. arXiv:cs/0602032 [Link]. 70

[10] Taylor Dupuy and David E. Weirich. Bits of in binary, Wieferich primes and a conjecture of Erdős. *Journal of Number Theory*, 158:268–280, 2016. [Link]. 112

[11] 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]. 12, 23, 32, 45

[12] P. M. Gauthier. Approximating the Riemann zeta-function by polynomials with restricted zeros. *Canadian Mathematical Bulletin*, 62(3):475–478, 2018. [Link]. 95

[13] Vincent Granville. *Stochastic Processes and Simulations: A Machine Learning Perspective*. MLT, 2022. [Link]. 77, 94

[14] Vincent Granville. *Synthetic Data and Generative AI*. MLT, 2022. [Link]. 77

[15] Vincent Granville. *Gentle Introduction To Chaotic Dynamical Systems*. MLT, 2023. [Link]. 7, 10, 11, 12, 14, 15, 18, 23, 32, 44, 45, 57, 61, 69, 70, 77, 78, 83, 84, 107, 110

[16] Vincent Granville. *Building Disruptive AI & LLM Technology from Scratch*. MLT, 2024. [Link]. 15, 23, 32, 45, 100

[17] Vincent Granville. *State of the Art GenAI & LLMs, Creative Projects & Solutions*. MLT, 2024. [Link]. 20, 84

[18] Vincent Granville. *Statistical Optimization for AI and Machine Learning*. MLT, 2024. [Link]. 78

[19] Vincent Granville. *Synthetic Data and Generative AI*. Elsevier, 2024. [Link]. 82

[20] Vincent Granville. *Blueprint: Next-Gen Enterprise RAG & LLM 2.0 – Nvidia PDFs Use Case*. 2025. MLT [Link]. 100

[21] Vincent Granville. Simple, efficient, secure, accurate enterprise AI xLLM 2.0 architecture & operating system. 2025. BondingAI internal report `bdai-scores.pdf`, July 2025. 100

[22] Vincent Granville. *No-Blackbox, Secure, Efficient AI and xLLM Solutions*. MLT, 2026. [Link]. 95, 100, 105, 110

[23] Vincent Granville and Richard L Smith. Disaggregation of rainfall time series via Gibbs sampling. *NISS Technical Report*, pages 1–21, 1996. [Link]. 94

[24] Emil Grosswald. Oscillation theorems of arithmetical functions. *Transactions of the American Mathematical Society*, 126:1–28, 1967. [Link]. 84

[25] Adam J. Harper. Moments of random multiplicative functions, II: High moments. *Algebra and Number Theory*, 13(10):2277–2321, 2019. [Link]. 85

[26] Adam J. Harper. Moments of random multiplicative functions, I: Low moments, better than squareroot cancellation, and critical multiplicative chaos. *Forum of Mathematics, Pi*, 8:1–95, 2020. [Link]. 85

[27] Adam J. Harper. Almost sure large fluctuations of random multiplicative functions. *Preprint*, pages 1–38, 2021. arXiv [Link]. 85

[28] Christopher Lutsko, Athanasios Sourmelidis, and Niclas Technau. Pair correlation of the fractional parts of αn^{θ}. *Journal of the European Mathematical Society*, 27:4069–4082, 2025. arXiv:2106.09800 [Link]. 71

[29] M. Madritsch and J. Thuswaldner. The level of distribution of the sum-of-digits function of linear recurrence number systems. *Journal de Théorie des Nombres de Bordeaux*, 34:449–482, 2022. MLT [Link]. 32, 45

[30] 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]. 12, 23, 32, 45

[31] Mohammadamin Moradi et al. Data-driven model discovery with Kolmogorov-Arnold networks. *Preprint*, pages 1–6, 2024. arXiv:2409.15167 [Link]. 24, 32, 45

[32] 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]. 23, 32, 45

[33] Alan Oppenheim and Ronald W. Schafer. *Discrete–Time Signal Processing*. Prentice Hall, 1999. 97

[34] Michel Planat and Patrick Solé. Efficient prime counting and the Chebyshev primes. *Preprint*, pages 1–15, 2011. arXiv:1109.6489 [Link]. 84

[35] Eric S. Rowland. Regularity versus complexity in the binary representation of 3^n. *Complex Systems*, 18:367–377, 2009. [Link]. 112

[36] Frank Ruskey. Generating functions for the digital sum and other digit counting sequences. *Journal of Integer Sequences*, 12:1–9, 2009. [Link]. 104

[37] Klaus Schiefermayr and Maxim Zinchenko. Norm estimates for Chebyshev polynomials, i. *Journal of Approximation Theory*, 265, 2021. [Link]. 56

[38] Jan-Christoph Schlage-Putcha and Jasson Vindas. The prime number theorem for Beurlings generalized numbers – new cases. pages 1–26, 2011. [Link]. 77

[39] Maxwell Schneider and Robert Schneider. Digit sums and generating functions. *Preprint*, pages 1–10, 2018. arXiv:1807.06710 [Link]. 104

[40] Terence Tao. Biases between consecutive primes. *Tao's blog*, 2016. [Link]. 84

[41] Yury V. Tiumentsev and Mikhail V. Egorchev. *Neural Network Modeling and Identification of Dynamical Systems*. Elsevier, 2019. 23, 32, 45

[42] Chukwudubem Umeano and Oleksandr Kyriienko. Ground state-based quantum feature maps. *Preprint*, pages 1–8, 2024. arXiv:2024.07174 [Link]. 12, 23, 32, 45

[43] Joseph Vandehey. On the binary digits of $\sqrt{2}$. *Preprint*, pages 1–6, 2017. arXiv:1711.01722 [Link]. 11, 23, 32, 45, 61

[44] Troy Vasiga and Jeffrey Shallit. On the iteration of certain quadratic maps over GF(p). *Discrete Mathematics*, 277:219–240, 2004. [Link]. 23, 32, 44

[45] Henry S. Warren. *Hacker's Delight*. Addison-Wesley Professional, second edition, 2012. 12, 23, 32, 45

[46] 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]. 23, 32, 45