

# 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

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

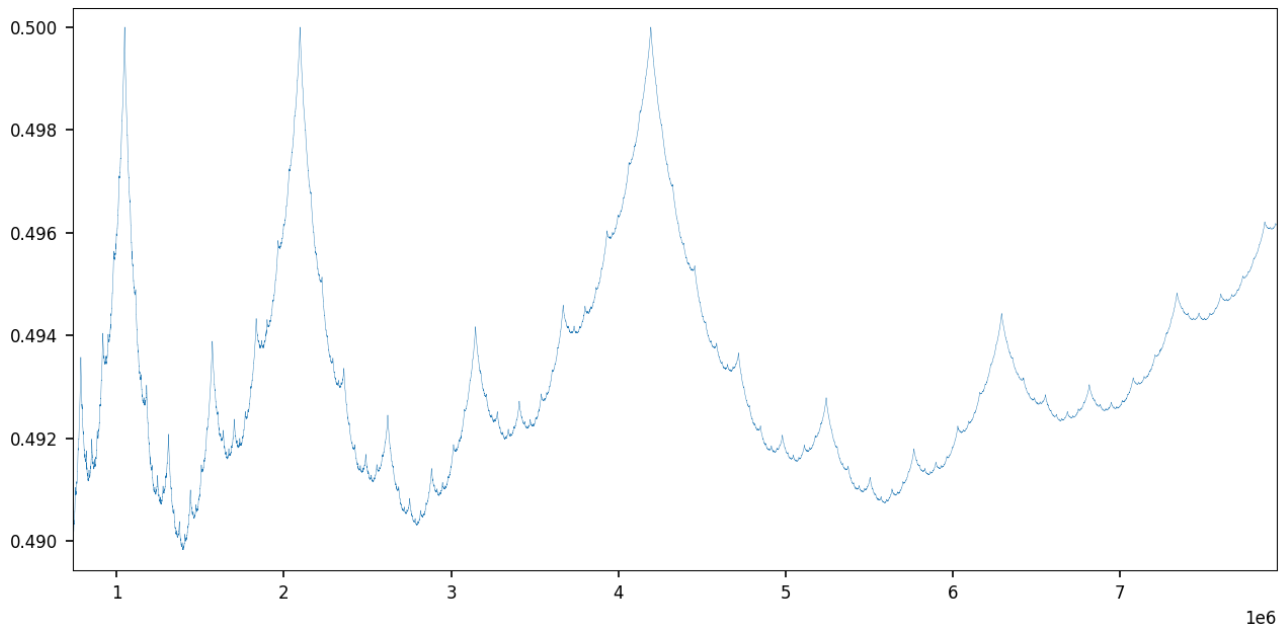


Figure B.1: Hamming weights normalized cumulative function  $S_H(n)$  with  $n \leq 8 \times 10^6$  on the X-axis

Clearly,  $S_H(n) \rightarrow \frac{1}{2}$  as  $n \rightarrow \infty$  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](#).

---

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import matplotlib as mpl
4
5 mpl.rcParams['axes.linewidth'] = 0.5
6 plt.rcParams['xtick.labelsize'] = 8
7 plt.rcParams['ytick.labelsize'] = 8
8 plt.rcParams['legend.fontsize'] = 'x-small'
9
10 def countSetBits(n):
11     # compute H(n)
12     count = 0
13     while(n):
14         count += n & 1
15         n >>= 1
16     return(count)
17
18 sum = 0
19 arr_k = []
20 arr_sum = []
21
22 for k in range(0,8000000):
23     d = countSetBits(k)
24     sum += d
25     arr_k.append(k)
26     if k < 2:
27         arr_sum.append(0)
28     else:
29         arr_sum.append(sum/(k*np.log2(k)))
30     if k % 10 == 0:
31         print("sss", k, d, sum)
32
33 plt.plot(arr_k, arr_sum, linewidth = 0.2)
34 plt.show()

```

---

Now I discuss some interesting properties related to the [generating functions](#) attached to  $H_n$ . It is easy to prove that

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

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

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

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

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

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 [34]. See also [37]. The main result is Theorem 1 in [34], stating that

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

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  $\gamma$  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 [\[21\]](#). 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

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

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  $\rho$  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  $\rho$ . 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  $\rho$  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  $\rho$  as  $n \rightarrow \infty$ , see [Figure B.2](#) where  $\rho_n$  on the Y-axis has a different color depending on  $n \bmod 7$ , with  $n$  on the X-axis. We jump from one branch to another each time  $n$  is incremented, yet eventually  $\rho_n$  converges to  $\rho$ . 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  $\rho$  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 [\[21\]](#). 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  $\rho$ . 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  $\rho$ . Interestingly, some LLMs erroneously believe that the answer is  $\rho = \sqrt{3}$ . For a deep dive into this particular case, see section B.2.3.

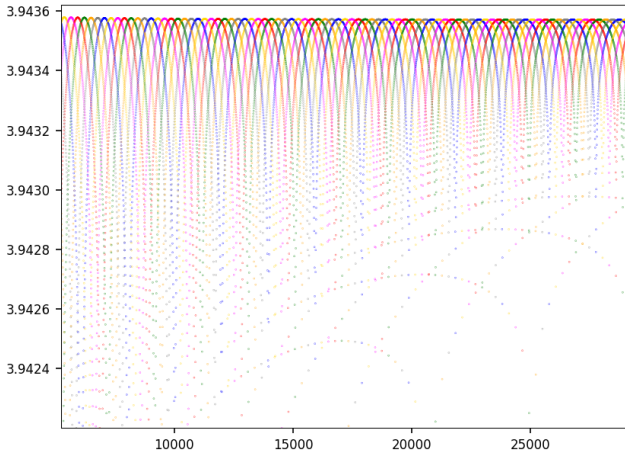


Figure B.2: Constant jumps in  $|x_n|^{1/n}$  gives the illusion of 7 curves but there is only one (section B.2.1).

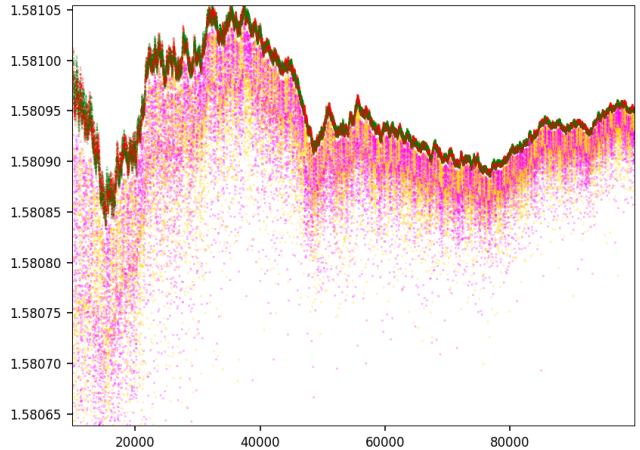


Figure B.3: 4-colors quantum regime similar to that of Figure B.2 but now with chaos (section B.2.2).

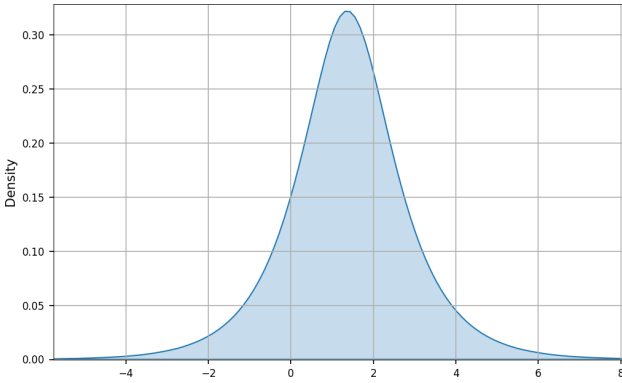


Figure B.4: Empirical PDF for  $\log |x_{n+1}/x_n|$ , case featured in Figure B.2

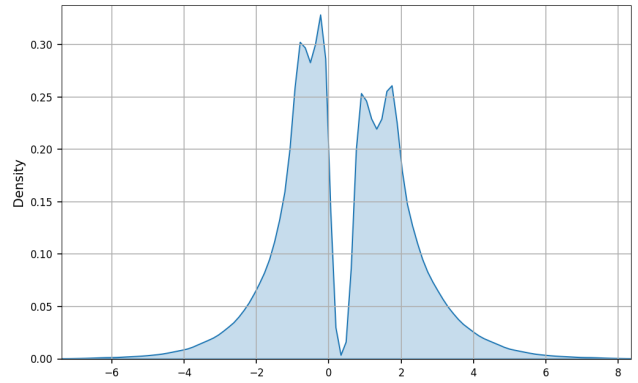


Figure B.5: Empirical PDF for  $\log |x_{n+1}/x_n|$ , case featured in Figure B.3

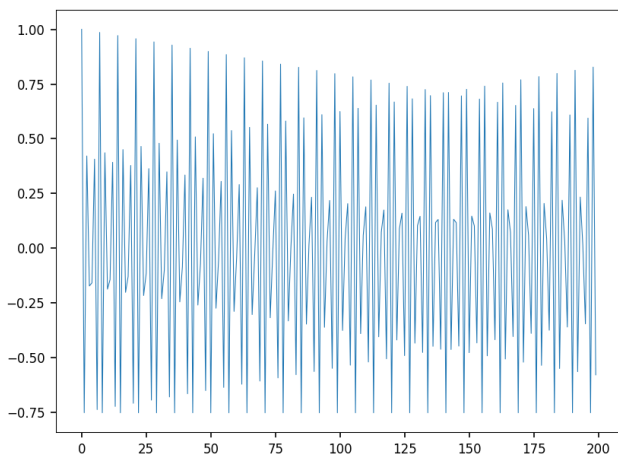


Figure B.6: Autocorrelation function for  $\log |x_{n+1}/x_n|$ , case featured in Figure B.2

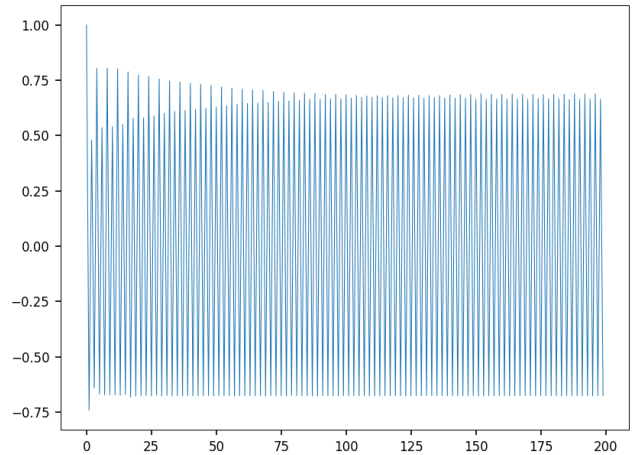


Figure B.7: Autocorrelation function for  $\log |x_{n+1}/x_n|$ , case featured in Figure B.3

The curve  $\rho_n = |x_n|^{1/n}$  has 4 chaotic bands. For a specific  $n$ , the band that  $\rho_n$  belongs to depends on  $n \bmod 4$ . 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 [14].

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.

---

```

1 import numpy as np
2 import gmpy2
3 import matplotlib.pyplot as plt
4 import matplotlib as mpl
5
6 mpl.rcParams['axes.linewidth'] = 0.5
7 plt.rcParams['xtick.labelsize'] = 8
8 plt.rcParams['ytick.labelsize'] = 8
9 plt.rcParams['legend.fontsize'] = 'x-small'
10
11 ndigits = 10000
12 ctx = gmpy2.get_context()
13 ctx.precision = ndigits
14 z = gmpy2.log(2)
15
16 # choose N < ndigits/2 to avoid losing precision
17 N = 5000
18 mode = 'quantum' # options: 'quantum' or 'chaotic'
19
20 x0 = gmpy2.mpfr(0)
21 x1 = gmpy2.mpfr(1)
22 x2 = gmpy2.mpfr(1)
23 arr_k = []
24 arr_x = []
25 arr_col = []
26 arr_log_rho = []
27
28 def set_color(k, mode):
29
30     if mode == 'quantum':
31         if k % 7 == 0:
32             color='red'
33         elif k % 7 == 1:
34             color='blue'
35         elif k % 7 == 2:
36             color='green'
37         elif k % 7 == 3:
38             color='gold'
39         elif k % 7 == 4:
40             color='orange'
41         elif k % 7 == 5:
42             color='magenta'
43         elif k % 7 == 6:
44             color='gray'
45
46     elif mode == 'chaotic':
47         if k % 4 == 0:
48             color = 'red'
49         elif k % 4 == 1:
50             color = 'gold'
51         elif k % 4 == 2:
52             color = 'green'
53         elif k % 4 == 3:
54             color = 'magenta'
55         else:
56             color = 'gray'
57     else:

```

```

58     print("Unsuported mode:", mode)
59     exit()
60     return(color)
61
62
63 #--- Main
64
65 modulus = 4
66 hash_modulo = {}
67
68 for k in range(2,N):
69     color = set_color(k, mode)
70     if mode == 'quantum':
71         x = 2*x2 - 16*x1 + 4*x0
72         delta = 0
73     elif mode == 'chaotic':
74         x = abs(x2 - 3*x1)
75     v = gmpy2.log(abs(x))/k
76     w = gmpy2.exp(v)
77     rho = float(x/x2)
78     if mode == 'chaotic':
79         old_rho = float(x2/x1)
80         # delta should be 0 for k < ndigits/2
81         delta = rho**2 - 1 + 6/old_rho - 9/old_rho**2
82     log_rho = np.log(abs(rho))
83     if k > N/10:
84         print("%6d log_rho: %8.5f %8.5f %f" % (k, log_rho, delta, rho))
85         arr_k.append(k)
86         arr_x.append(w)
87         arr_col.append(color)
88         arr_log_rho.append(log_rho)
89         residue = k % modulus
90         if residue in hash_modulo:
91             arr_local = hash_modulo[residue]
92             arr_local.append(log_rho)
93             hash_modulo[residue] = arr_local
94         else:
95             hash_modulo[residue] = [log_rho]
96         x0 = x1
97         x1 = x2
98         x2 = x
99
100 plt.scatter(arr_k, arr_x, c=arr_col, s=0.02)
101 plt.show()
102 plt.scatter(arr_k, arr_log_rho, c=arr_col, s=0.02)
103 plt.show()
104 plt.plot(arr_k, arr_log_rho, linewidth=0.4)
105 plt.show()
106 print("\nRho[residue] with modulus = %2d" % (modulus))
107 avg = 1
108 for residue in range(modulus):
109     arr_vals = hash_modulo[residue]
110     arr_vals = np.array(arr_vals)
111     arr_exp = np.exp(arr_vals)
112     iqr_rho = np.quantile(arr_exp,0.75) - np.quantile(arr_exp, 0.25)
113     mu = np.average(arr_vals)
114     mean_rho = np.exp(mu)
115     median = np.std(arr_vals)
116     avg *= mean_rho
117     print("residue %2d | mu = %8.5f | rho = %8.5f | irq = %8.5f | median = %8.5f"
118           % (residue, mu, mean_rho, iqr_rho, median))
119 avg = avg**(1/modulus)
120 print("Rho: %8.5f" % (avg))
121
122
123 #--- Plot EPDFs
124
125 np_log_rho = np.array(arr_log_rho)
126 import seaborn as sns
127 plt.figure(figsize=(8, 5))
128 # sns.ecdfplot(np_rho)
129 sns.kdeplot(data=np_log_rho, fill=True)
130 plt.grid(True)
131 plt.show()
132
133 meanlog = np.mean(np_log_rho)

```

```

134 medianlog = np.median(np_log_rho)
135 mean = np.exp(meanlog)
136 median = np.exp(medianlog)
137 print("\nRho: %8.5f [median = %8.5f | meanlog = %8.5f]" % (mean, median, meanlog))
138 print()
139
140
141 #--- Compute autocorrel for log(x/x2)
142
143 nobs = len(np_log_rho)
144 arr_lag = []
145 arr_autocorrel = []
146
147 for lag in range(200):
148     mean1 = np.mean(np_log_rho[0: nobs-lag])
149     mean2 = np.mean(np_log_rho[lag: nobs])
150     std1 = np.std(np_log_rho[0: nobs-lag])
151     std2 = np.std(np_log_rho[lag: nobs])
152     dotprod = np.dot(np_log_rho[0: nobs-lag], np_log_rho[lag: nobs])
153     dotprod /= (nobs-lag)
154     autocorrel = (dotprod - mean1*mean2)/(std1*std2)
155     arr_lag.append(lag)
156     arr_autocorrel.append(autocorrel)
157     print("Autocorrel lag %3d: %8.5f" % (lag, autocorrel))
158
159 plt.plot(arr_lag, arr_autocorrel, linewidth = 0.5)
160 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:

$$r_{n+1} = \left| 1 - \frac{3}{r_n} \right| \quad (\text{B.8})$$

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

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

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

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  $\mu$  in (B.10), with  $|x_n|^{1/n} \rightarrow \rho = e^\mu$ . In the code, the number N of iterations must 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  $\rho$ .

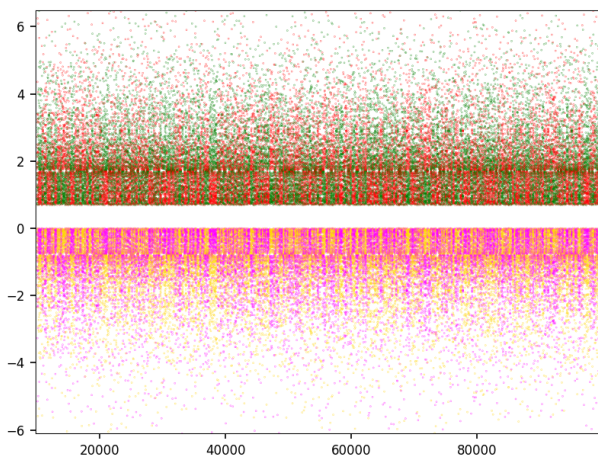


Figure B.8: Same as Figure B.3 but now with  $s_n$  instead of  $|x_n|^{1/n}$  on the Y-axis ( $n$  on the X-axis)

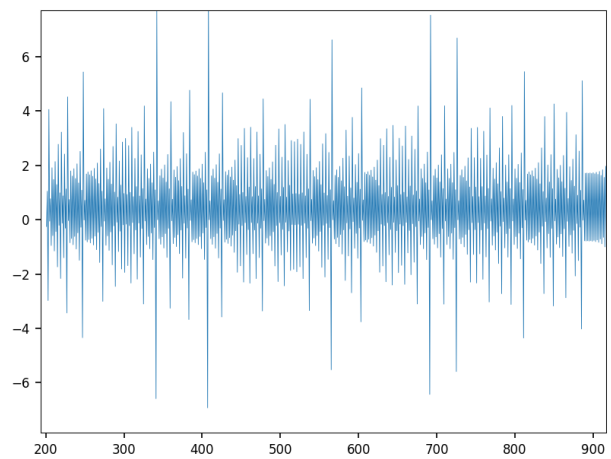


Figure B.9: Same as Figure B.8 with standard plot instead of scatterplot, thus missing the empty band

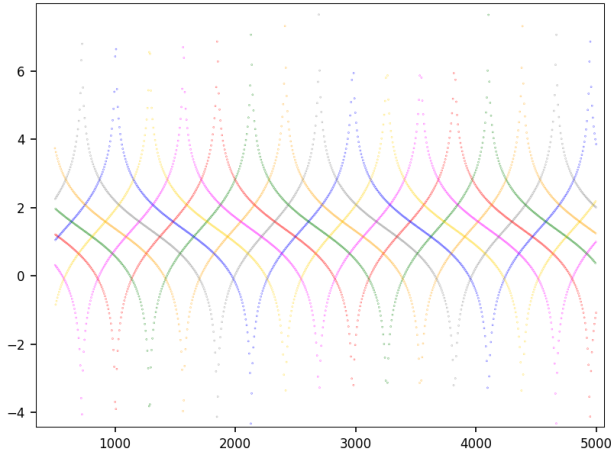


Figure B.10: Same as Figure B.2 but now with  $s_n$  instead of  $|x_n|^{1/n}$  on the Y-axis ( $n$  on the X-axis)

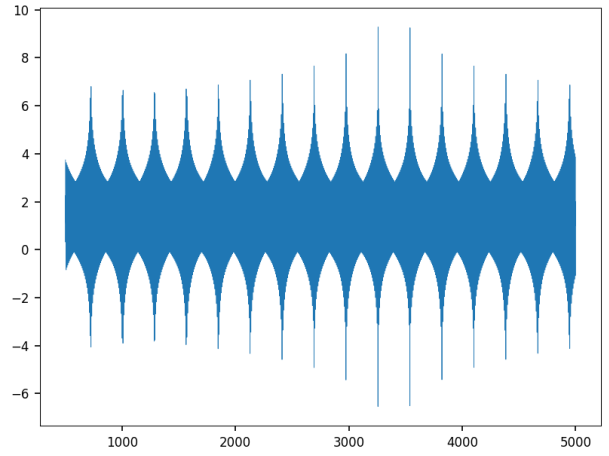


Figure B.11: Same as Figure B.10 with standard plot instead of scatterplot. Provided solely as a piece of art

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  $S_0, \dots, S_{m-1}$ , where  $m$  may be finite or infinite. Let  $p_j$  be the proportion of indices in set  $S_j$ , with  $p_0 + \dots + p_{m-1} = 1$ . 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

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

I used (B.11) to compute  $\rho$  for random Fibonacci sequences discussed in section 7.4 in [21]. 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  $\mu_j$  ( $j = 0, 1, \dots, m-1$ ) 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  $\mu$ . 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](#)

$$F_Z(z) = F_Z \left( \log \left| 1 - 3 \exp(-z) \right| \right). \quad (\text{B.12})$$

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 [14]. 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  $E[Z]$  attached to  $F_Z$ . I asked LLMs to find  $F_z$  or  $\mu = E[Z]$  and mostly got erroneous theoretical results yet good approximation for  $\mu$ . Finally, (B.11) can be rewritten as

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

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  $x_{n+1} > \log 2$ .

# 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] 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](#)
- [11] P. M. Gauthier. Approximating the Riemann zeta-function by polynomials with restricted zeros. *Canadian Mathematical Bulletin*, 62(3):475–478, 2018. [\[Link\]](#). [95](#)
- [12] Vincent Granville. *Stochastic Processes and Simulations: A Machine Learning Perspective*. MLT, 2022. [\[Link\]](#). [77](#), [94](#)
- [13] Vincent Granville. *Synthetic Data and Generative AI*. MLT, 2022. [\[Link\]](#). [77](#)
- [14] 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](#)
- [15] Vincent Granville. *Building Disruptive AI & LLM Technology from Scratch*. MLT, 2024. [\[Link\]](#). [15](#), [23](#), [32](#), [45](#), [100](#)
- [16] Vincent Granville. *State of the Art GenAI & LLMs, Creative Projects & Solutions*. MLT, 2024. [\[Link\]](#). [20](#), [84](#)
- [17] Vincent Granville. *Statistical Optimization for AI and Machine Learning*. MLT, 2024. [\[Link\]](#). [78](#)
- [18] Vincent Granville. *Synthetic Data and Generative AI*. Elsevier, 2024. [\[Link\]](#). [82](#)
- [19] Vincent Granville. *Blueprint: Next-Gen Enterprise RAG & LLM 2.0 – Nvidia PDFs Use Case*. 2025. MLT [\[Link\]](#). [100](#)
- [20] Vincent Granville. Simple, efficient, secure, accurate enterprise AI xLLM 2.0 architecture & operating system. 2025. BondingAI internal report bdai-scores.pdf, July 2025. [100](#)
- [21] Vincent Granville. *No-Blackbox, Secure, Efficient AI and xLLM Solutions*. MLT, 2026. [\[Link\]](#). [95](#), [100](#), [105](#), [110](#)
- [22] Vincent Granville and Richard L Smith. Disaggregation of rainfall time series via Gibbs sampling. *NISS Technical Report*, pages 1–21, 1996. [\[Link\]](#). [94](#)
- [23] Emil Grosswald. Oscillation theorems of arithmetical functions. *Transactions of the American Mathematical Society*, 126:1–28, 1967. [\[Link\]](#). [84](#)

- [24] Adam J. Harper. Moments of random multiplicative functions, II: High moments. *Algebra and Number Theory*, 13(10):2277–2321, 2019. [\[Link\]](#). 85
- [25] 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
- [26] Adam J. Harper. Almost sure large fluctuations of random multiplicative functions. *Preprint*, pages 1–38, 2021. arXiv [\[Link\]](#). 85
- [27] Christopher Lutsko, Athanasios Sourmelidis, and Niclas Technau. Pair correlation of the fractional parts of  $\alpha n^\theta$ . *Journal of the European Mathematical Society*, 27:4069–4082, 2025. arXiv:2106.09800 [\[Link\]](#). 71
- [28] 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
- [29] 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
- [30] Mohammadamin Moradi et al. Data-driven model discovery with Kolmogorov-Arnold networks. *Preprint*, pages 1–6, 2024. arXiv:2409.15167 [\[Link\]](#). 24, 32, 45
- [31] 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
- [32] Alan Oppenheim and Ronald W. Schaffer. *Discrete-Time Signal Processing*. Prentice Hall, 1999. 97
- [33] Michel Planat and Patrick Solé. Efficient prime counting and the Chebyshev primes. *Preprint*, pages 1–15, 2011. arXiv:1109.6489 [\[Link\]](#). 84
- [34] Frank Ruskey. Generating functions for the digital sum and other digit counting sequences. *Journal of Integer Sequences*, 12:1–9, 2009. [\[Link\]](#). 104
- [35] Klaus Schiefermayr and Maxim Zinchenko. Norm estimates for Chebyshev polynomials, i. *Journal of Approximation Theory*, 265, 2021. [\[Link\]](#). 56
- [36] Jan-Christoph Schlage-Putcha and Jasson Vindas. The prime number theorem for Beurlings generalized numbers – new cases. pages 1–26, 2011. [\[Link\]](#). 77
- [37] Maxwell Schneider and Robert Schneider. Digit sums and generating functions. *Preprint*, pages 1–10, 2018. arXiv:1807.06710 [\[Link\]](#). 104
- [38] Terence Tao. Biases between consecutive primes. *Tao’s blog*, 2016. [\[Link\]](#). 84
- [39] Yury V. Tiumentsev and Mikhail V. Egorchev. *Neural Network Modeling and Identification of Dynamical Systems*. Elsevier, 2019. 23, 32, 45
- [40] Chukwudubem Umeano and Oleksandr Kyriienko. Ground state-based quantum feature maps. *Preprint*, pages 1–8, 2024. arXiv:2024.07174 [\[Link\]](#). 12, 23, 32, 45
- [41] Joseph Vandehey. On the binary digits of  $\sqrt{2}$ . *Preprint*, pages 1–6, 2017. arXiv:1711.01722 [\[Link\]](#). 11, 23, 32, 45, 61
- [42] Troy Vasiga and Jeffrey Shallit. On the iteration of certain quadratic maps over  $\text{GF}(p)$ . *Discrete Mathematics*, 277:219–240, 2004. [\[Link\]](#). 23, 32, 44
- [43] Henry S. Warren. *Hacker’s Delight*. Addison-Wesley Professional, second edition, 2012. 12, 23, 32, 45
- [44] 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