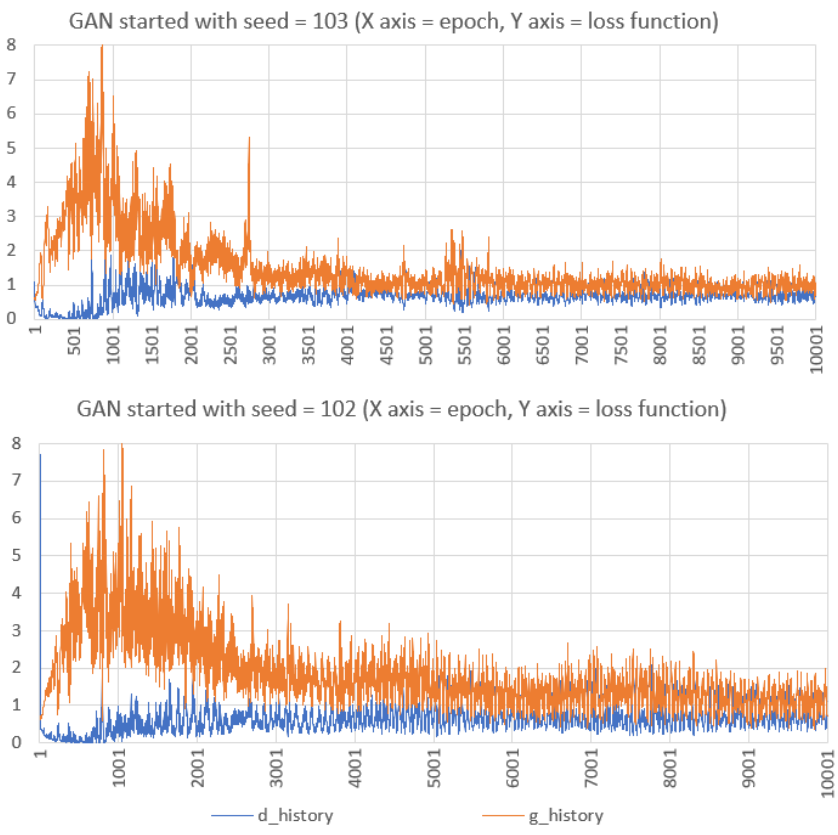
Practical AI & Machine Learning Projects and Datasets



Vincent Granville, Ph.D. | www.MLTechniques.com | Version 2.0, July 2023

Preface

This book is intended to participants in the AI and machine learning certification program organized by my AI/ML research lab MLtechniques.com. It is also an invaluable resource to instructors and professors teaching related material, and to their students. If you want to add enterprise-grade projects to your curriculum, with deep technical dive on modern topics, you are welcome to re-use my projects in your classroom. I provide my own solution to each of them.

This book is also useful to prepare for hiring interviews. And for hiring managers, there is plenty of original questions, encouraging candidates to think outside the box, with applications on real data. The amount of Python code accompanying the solutions is considerable, using a vast array of libraries as well as home-made implementations showing the inner workings and improving existing black-box algorithms. By itself, this book constitutes a solid introduction to Python and scientific programming. The code is also on my GitHub repository.

The topics cover generative AI, synthetic data, machine learning optimization, scientific computing with Python, experimental math, synthetic data and functions, data visualizations and animations, time series and spatial processes, NLP and large language models, as well as graph applications and more. It also includes significant advances on some of the most challenging mathematical conjectures, obtained thanks to modern computations. In particular, intriguing new results regarding the Generalized Riemann Hypothesis, and a conjecture regarding record run lengths in the binary digits of $\sqrt{2}$. For the latter, the author offers a $1m award to prove or disprove the main statement. Most projects are based on real life data, offered with solutions and Python code. Your own solutions would be a great addition to your GitHub portfolio, bringing your career to the next level. Hiring managers, professors, and instructors can use the projects, each one broken down in a number of steps, to differentiate themselves from competitors. Most offer off-the-beaten path material. They may be used as novel exercises, job interview or exam questions, and even research topics for master or PhD theses.

To see how the certification program works, check out our FAQ posted here, or click on the "certification" tab on our website MLtechniques.com. Certifications can easily be displayed on your LinkedIn profile page in the credentials section. Unlike many other programs, there is no exam or meaningless quizzes. Emphasis is on projects with real-life data, enterprise-grade code, efficient methods, and modern applications to build a strong portfolio and grow your career in little time. The guidance to succeed is provided by the founder of the company, one of the top experts in the field, Dr. Vincent Granville. Jargon and unnecessary math are avoided, and simplicity is favored whenever possible. Nevertheless, the material is described as advanced by everyone who looked at it.

The related teaching and technical material (textbooks) can be purchased at MLtechniques.com/shop/. MLtechniques.com, the company offering the certifications, is a private, self-funded AI/ML research lab developing state-of-the-art open source technologies related to synthetic data, generative AI, cybersecurity, geospatial modeling, stochastic processes, chaos modeling, and AI-related statistical optimization.

About the author

Vincent Granville is a pioneering data scientist and machine learning expert, co-founder of Data Science Central (acquired by TechTarget), founder of MLTechniques.com, former VC-funded executive, author and patent owner.

Vincent's past corporate experience includes Visa, Wells Fargo, eBay, NBC, Microsoft, and CNET. Vincent is also a former post-doc at Cambridge University, and the National Institute of Statistical Sciences (NISS). He published in *Journal of Number Theory*, *Journal of the Royal Statistical Society* (Series B), and *IEEE Transactions on Pattern Analysis and Machine Intelligence*. He is also the author of multiple books, available here. He lives in Washington state, and enjoys doing research on stochastic processes, dynamical systems, experimental math and probabilistic number theory.

Contents

one has to investigate δ_n, denoted as `delta` in the code below. The code is also on GitHub, here. Note that the variable `steps` can only take on three values: 0, 1, and 2. It is represented as s_n in Table 4.2. Improving the asymptotic upper bound $L_n/n \leq 1$ in (4.1) as $n \to \infty$, is incredibly hard. I spent a considerable amount of time to non avail, even though anyone who spends a small amount of time on this problem will be convinced that asymptotically, $L_n/\log_2 n \leq 1$ as $n \to \infty$, a much stronger result. Proving the stronger bound, even though verified in Table 4.1 for n up to 10^9, is beyond the capabilities of the mathematical tools currently available. It may as well not be true or undecidable, nobody knows.

```
import math
import gmpy2

# requirement: 0 < p < q
p = 1
q = 2
x0 = math.sqrt(p/q)
N = 1000 # precision, in number of binary digits for x0

# compute and store in bsqrt (a string) the N first binary digits of x0 = sqrt(p/q)
base = 2
bsqrt = gmpy2.isqrt( (2**(2*N) * p) // q ).digits(base)

for n in range(1, N):

   if n == 1:
      u = p * 4**n
      v = int(x0 * 4**n)
      if v % 2 == 0:
         v = v - 1
   else:
      u = 4*u
      v = 2*v + 1
   steps = 0
   while q*v*v < u:
      v = v + 2
      steps += 1 # steps is always 0, 1, or 2
   v = v - 2
   delta = u - q*v*v
   d = bsqrt[n-1] # binary digit of x0 = sqrt(p/q), in position n

   ## delta2 = delta >> (n - 1)
   ## res = 5/2 + n - math.log(delta,2) - math.log(n, 2)

   run = int(n + 1 + math.log(p*q, 2)/2 - math.log(delta, 2) )
   if d == "0" or run == 0:
      run = ""

   print("%6d %1s %2s %1d" % (n, d, str(run), steps))
```

4.4 Quantum derivatives, GenAI, and the Riemann Hypothesis

If you are wondering how close we are to proving the Generalized Riemann Hypothesis (GRH), you should read on. The purpose of this project is to uncover intriguing patterns in prime numbers, and gain new insights on the GRH. I stripped off all the unnecessary math, focusing on the depth and implications of the material discussed here. You will also learn to use the remarkable MPmath library for scientific computing. This is a cool project for people who love math, with the opportunity to work on state-of-the-art research even if you don't have a PhD in number theory.

Many of my discoveries were made possible thanks to pattern detection algorithms (in short, AI) before leading to actual proofs, or disproofs. This data-driven, bottom-up approach is known as experimental math. It contrasts with the top-down, classic theoretical academic framework. The potential of AI and its computing power should not be underestimated to make progress on the most difficult mathematical problems. It offers a big competitive advantage over professional mathematicians focusing on theory exclusively.

My approach is unusual as it is based on the Euler product. The benefit is that you immediately know when the target function, say the Riemann zeta function $\zeta(s)$, has a root or not, wherever the product converges. Also, these products represent analytic function [Wiki] wherever they converge.

I use the standard notation in the complex plane: $s = \sigma + it$, where σ, t are respectively the real and imaginary parts. I focus on the real part only (thus $t = 0$) because of the following result: if for some $s = \sigma_0$, the product converges, then it converges for all $s = \sigma + it$ with $\sigma > \sigma_0$. Now let's define the Euler product. The finite version with n factors is a function of s, namely

$$f(s,n) = \prod_{p \in P_n} \left(1 - \frac{\chi(p)}{p^s}\right)^{-1} = \prod_{k=1}^{n} \left(1 - \frac{\chi(p_k)}{p_k^s}\right)^{-1}.$$

Here $P_n = \{2, 3, 5, 7, 11, \dots\}$ is the set of the first n prime numbers, and p_k denotes the k-th prime with $p_1 = 2$. The function $\chi(p)$ can take on three vales only: $0, -1, +1$. This is not the most generic form, but the one that I will be working with in this section. More general versions are investigated in chapter 17, in [14]. Of course, we are interested in the case $n \to \infty$, where convergence becomes the critical issue. Three particular cases are:

- Rienmann zeta, denoted as $\zeta(s,n)$ or $\zeta(s)$ when $n = \infty$. In this case $\chi(p) = 1$ for all primes p. The resulting product converges only if $\sigma > 1$. Again, σ is the real part of s.
- Dirichlet L-function $L_4(s,n)$ [Wiki] with Dirichlet modular character $\chi = \chi_4$ [Wiki]. Denoted as $L_4(s)$ when $n = \infty$. Here $\chi_4(2) = 0$, $\chi_4(p) = 1$ if $p - 1$ is a multiple of 4, and $\chi_4(p) = -1$ otherwise. The product is absolutely convergent if $\sigma > 1$, but convergence status is unknown if $\frac{1}{2} < \sigma \leq 1$.
- Unnamed function $Q_2(s,n)$, denoted as $Q_2(s)$ when $n = \infty$. Here $\chi(2) = 0$. Otherwise, $\chi(p_k) = 1$ if k is even, and $\chi(p_k) = -1$ if k is odd. Again, p_k is the k-th prime with $p_1 = 2$. The product is absolutely convergent if $\sigma > 1$, and conditionally convergent [Wiki] if $\frac{1}{2} < \sigma \leq 1$.

All these products can be expanded into Dirichlet series [Wiki], and the corresponding χ expanded into multiplicative functions [Wiki] over all positive integers. Also, by construction, Euler products have no zero in their conditional and absolute convergence domains. Most mathematicians believe that the Euler product for $L_4(s)$ conditionally converges when $\frac{1}{2} < \sigma \leq 1$. Proving it would be a massive accomplishment. This would be make L_4 the first example of a function satisfying all the requirements of the Generalized Riemann Hypothesis. The Unnamed function Q_2 actually achieves this goal, with the exception that its associated χ is not periodic. Thus, Q_2 lacks some of the requirements.

The rest of the discussion is about building the framework to help solve this centuries-old problem. It can probably be generalized to L-functions other than L_4, with one notable exception: the Riemann function itself, which was the one that jump-started all this vast and beautiful mathematical theory.

4.4.1 Cornerstone result to bypass the roadblocks

The goal here is to prove that the Euler product $L_4(s,n)$ converges to some constant $c(s)$ as $n \to \infty$, for some $s = \sigma_0 + it$, with $t = 0$ and some $\sigma_0 < 1$. In turns, it implies that it converges at $s = \sigma + it$, for all t and for all $\sigma > \sigma_0$. It also implies that $c(s) = L_4(s)$, the true value obtained by analytic continuation [Wiki]. Finally, it implies that $L_4(s)$ has no zero if $\sigma > \sigma_0$. This would provide a partial solution to the Generalized Riemann Hypothesis, for L_4 rather than the Riemann zeta function $\zeta(s)$, and not with $\sigma_0 = \frac{1}{2}$ (the conjectured lower bound), but a least for some $\sigma_0 < 1$. This is enough to make countless mathematical theorems true, rather than "true conditionally on the fact that the Riemann Hypothesis is true". It also leads to much more precise results regarding the distribution of primes numbers: results that to this day, are only conjectures. The implications are numerous, well beyond number theory.

The chain of implications that I just mentioned, follows mainly from expanding the Euler product into a Dirichlet-L series. In this case, the expansion is as follows, with $s = \sigma + it$ as usual:

$$\prod_{k=1}^{\infty} \left(1 - \frac{\chi_4(p_k)}{p_k^s}\right)^{-1} = \sum_{k=0}^{\infty} \frac{(-1)^{k+1}}{(2k+1)^s}. \tag{4.2}$$

The series obviously converges when $\sigma > 0$. The product converges for sure when $\sigma > 1$. It is believed that it converges as well when $\sigma > \frac{1}{2}$. The goal here is to establish that it converges when $\sigma > \sigma_0$, for some $\frac{1}{2} < \sigma_0 < 1$. When both converge, they converge to the same value, namely $L_4(s)$ as the series is the analytic continuation of the product, for all $\sigma > 0$. And of course, the product can not be zero when it converges. Thus $L_4(s) \neq 0$ if $\sigma > \sigma_0$.

The big question is how to find a suitable σ_0, and show that it must be strictly smaller than 1. I now focus on this point, leading to some unknown σ_0, very likely in the range $0.85 < \sigma_0 < 0.95$, for a number of reasons. The first step is to approximate the Euler product $L_4(s,n)$ with spectacular accuracy around $\sigma = 0.90$, using statistical techniques and a simple formula. This approximation amounts to denoising the irregularities caused by the prime number distribution, including Chebyshev's bias [Wiki]. After this step, the remaining is standard real analysis, trying to establish a new generic asymptotic result for a specific class of functions, and assuring that it encompasses our framework. The new theorem 4.4.1 in question, albeit independent from number theory, has yet to be precisely stated, let alone proved. The current version is as follows:

Theorem 4.4.1 *Let $A_n = \{a_1, \dots, a_n\}$ and $B_n = \{b_1, \dots, b_n\}$ be two finite sequences of real numbers, with $a_n \to 0$ as $n \to \infty$. Also assume that $b_{n+1} - b_n \to 0$. Now, define ρ_n as the ratio of the standard deviations, respectively computed on A_n (numerator) and B_n (denominator). If ρ_n converges to a non-zero value as $n \to \infty$, then b_n also converges.*

The issue to finalize the theorem is to make sure that it is applicable in our context, and add any additional requirements needed (if any). Is it enough to require $\inf \rho_n > 0$ and $\sup \rho_n < \infty$, rather than the convergence of ρ_n to non-zero? A stronger version, assuming $\sqrt{n} \cdot a_n$ is bounded and $\liminf \rho_n = \rho > 0$, leads to

$$\rho b_n - a_n \sim c + \frac{\alpha}{\sqrt{n}} + \frac{\beta}{\sqrt{n \log n}} + \cdots \tag{4.3}$$

where c, α, β are constants. As a result, $b_n \to c/\rho$. For the term $\beta/\sqrt{n \log n}$ to be valid, additional conditions on the asymptotic behavior of a_n and b_n may be required. Note that a_n and $\alpha/\sqrt{n}$ have the same order of magnitude. As we shall see, a_n captures most of the chaotic part of $L_4(s, n)$, while the term $\beta/\sqrt{n \log n}$ significant improves the approximation.

The following fact is at the very core of the GRH proof that I have in mind. Let us assume that b_n depends *continuously* on some parameter σ. If $\rho_n \to 0$ when $\sigma = \sigma_1$, and $\rho_n \to \infty$ when $\sigma = \sigma_2$, then there most be some σ_0 with $\sigma_1 \leq \sigma_0 \leq \sigma_2$ such that ρ_n converges to non-zero, or at least $\limsup \rho_n < \infty$ and $\liminf \rho_n > 0$ when $\sigma = \sigma_0$. This in turn allows us to use the proposed theoretical framework (results such as theorem 4.4.1) to prove the convergence of $L_4(s, n)$ at $\sigma = \sigma_0$. The challenge in our case is to show that there is such a σ_0, satisfying $\sigma_0 < 1$. However, the difficulty is not caused by crossing the line $\sigma = 1$, and thus unrelated to the prime number distribution. Indeed, most of the interesting action – including crossing our red line – takes place around $\sigma = 0.90$. Thus the problem now appears to be generic, rather than specific to GRH.

Now I establish the connection to the convergence of the Euler product $L_4(s, n)$. First, I introduce two new functions:

$$\delta_n(s) = L_4(s, n) - L_4(s), \quad \Lambda_n = \frac{1}{\varphi(n)} \sum_{k=1}^{n} \chi_4(p_k), \tag{4.4}$$

with $\varphi(n) = n$ for $n = 2, 3, 4$ and so on. An important requirement is that $\Lambda(n) \to 0$. I also tested $\varphi(n) = n \log n$. Then, in formula (4.3), I use the following:

$$a_n = \Lambda_n, \quad b_n = \delta_n(s). \tag{4.5}$$

Here, $L_4(s)$ is obtained via analytic continuation, not as the limit of the Euler product $L_4(s, n)$. The reason is because we don't know if the Euler product converges if $\sigma < 1$, although all evidence suggests that this is the case. Convergence of $\delta_n(s)$ translates to $c = 0$ in formula (4.3). Finally, in the figures, the X-axis represents n.

4.4.2 Quantum derivative of functions nowhere differentiable

Discrete functions such as $L_4(s, n)$, when s is fixed and n is the variable, can be scaled to represent a continuous function. The same principle is used to transform a random walk into a Brownian motion, as discussed in section 1.3 in my book on chaos and dynamical systems [10], and pictured in Figure 4.5. This is true whether the function is deterministic or random. In this context, n represents the time.

In general, the resulting function is nowhere differentiable. You can use the integrated function rather than the original one to study the properties, as integration turns chaos into a smooth curve. But what if we could use the derivative instead? The derivative is even more chaotic than the original function, indeed it does not even exist! Yet there is a way to define the derivative and make it particularly useful to discover new insights about the function of interest. This new object called quantum derivative is not a function, but rather a set of points with a specific shape, boundary, and configuration. In some cases, it may consist of multiple curves, or a dense set with homogeneous or non-homogeneous point density. Two chaotic functions that look identical to the naked eye may have different quantum derivatives.

The goal here is not to formally define the concept of quantum derivative, but to show its potential. For instance, in section 3.3.2.1 of the same book [10], I compute the moments of a cumulative distribution function (CDF) that is nowhere differentiable. For that purpose, I use the density function (PDF), which of course is nowhere defined. Yet, I get the correct values. While transparent to the reader, I implicitly integrated weighted quantum derivatives of the CDF. In short, the quantum derivative of a discrete function $f(n)$ is based on $f(n) - f(n-1)$. If the time-continuous (scaled) version of f is continuous, then the quantum derivative corresponds to the standard derivative. Otherwise, it takes on multiple values, called quantum states [Wiki] in quantum physics.

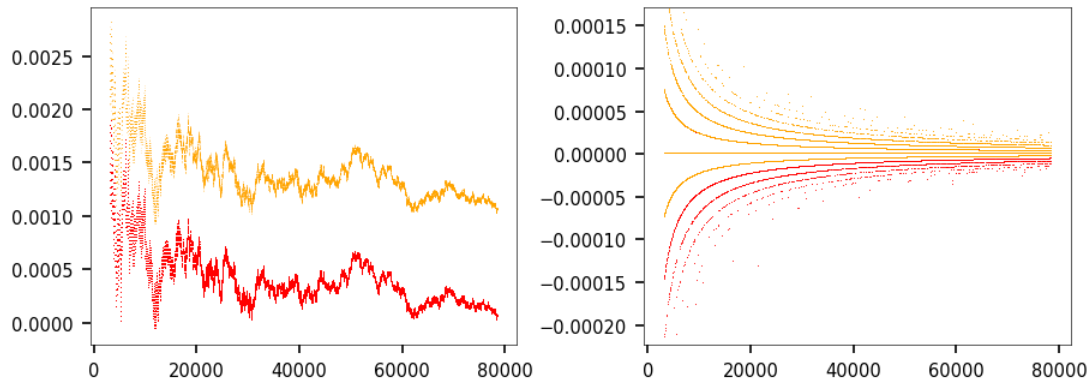


Figure 4.3: Two shifted legs of $\delta_n(s)$ [left], and their quantum derivatives [right]

Now in our context, in Figure 4.3, I show two legs of $\delta_n(s)$: one where $\chi_4(p_n) = +1$, and the other one where $\chi_4(p_n) = -1$. Both give rise to time-continuous functions that are nowhere differentiable, like the Brownian motions in Figure 4.5. Unlike Brownian motions, the variance tends to zero over time. The two functions are almost indistinguishable to the naked eye, so I separated them on the left plot in Figure 4.3. The corresponding quantum derivatives consist of a set of curves (right plot, same figure). They contain a lot of useful information about $L_4(s)$. In particular:

- The left plot in Figure 4.3 shows an asymmetrical distribution of the quantum derivatives around the X-axis. This is caused by the Chebyshev bias, also called prime race: among the first n primes numbers, the difference between the proportion of primes p_k with $\chi_4(p_k) = +1$, and those with $\chi_4(p_k) = -1$, is of the order $1/\sqrt{n}$, in favor of the latter. See [1, 22, 26]. This is known as Littlewood's oscillation theorem [16].
- The various branches in the quantum derivative (same plot) correspond to runs of different lengths in the sequence $\{\chi_4(p_n)\}$: shown as positive or negative depending on the sign of $\chi_4(p_n)$. Each branch has it own point density, asymptotically equal to $2^{-\lambda}$ (a geometric distribution) for the branch featuring runs of length λ, for $\lambda = 1, 2$ and so on. A similar number theory problem with the distribution of run lengths is discussed in section 4.3, for the binary digits of $\sqrt{2}$.

4.4.3 Project and solution

The project consists of checking many of the statements made in section 4.4.1, via computations. Proving the empirical results is beyond the scope of this work. The computations cover not only the case $L_4(s, n)$, but also other similar functions, including synthetic ones and Rademacher random multiplicative functions [17, 18, 19]. It also includes an empirical verification of theorem 4.4.1, and assessing whether its converse might be true. Finally, you will try different functions φ in formula (4.4) to check the impact on approximation (4.3). In the process, you will get familiar with a few Python libraries:

- MPmath to compute $L_4(s)$ for complex arguments when $\sigma < 1$.
- Primepy to obtain a large list of prime numbers.
- Scipy for curve fitting, when verifying the approximation (4.3).

The curve fitting step is considerably more elaborate than the standard implementation in typical machine learning projects. First, it consists of finding the best c, α, β in (4.3) for a fixed n, and identifying the best model: in this case, the choice of $\sqrt{n}$ and $\sqrt{n \log n}$ for the curve fitting function (best fit). Then, using different n, assess whether or not c, α, β, ρ depend on n or not, and whether $c = 0$ (this would suggest that the Euler product converges).

Finally, you will run the same curve fitting model for random functions, replacing the sequence $\{\chi_4(p_n)\}$ by random sequences of independent $+1$ and -1 evenly distributed, to mimic the behavior of $L_4(s, n)$ and Λ_n. One would expect a better fit when the functions are perfectly random, yet the opposite is true. A possible explanation is the fact that the Chebyshev bias in $L_4(n, s)$ is very well taken care of by the choice of Λ_n, while for random functions, there is no such bias, and thus no correction.

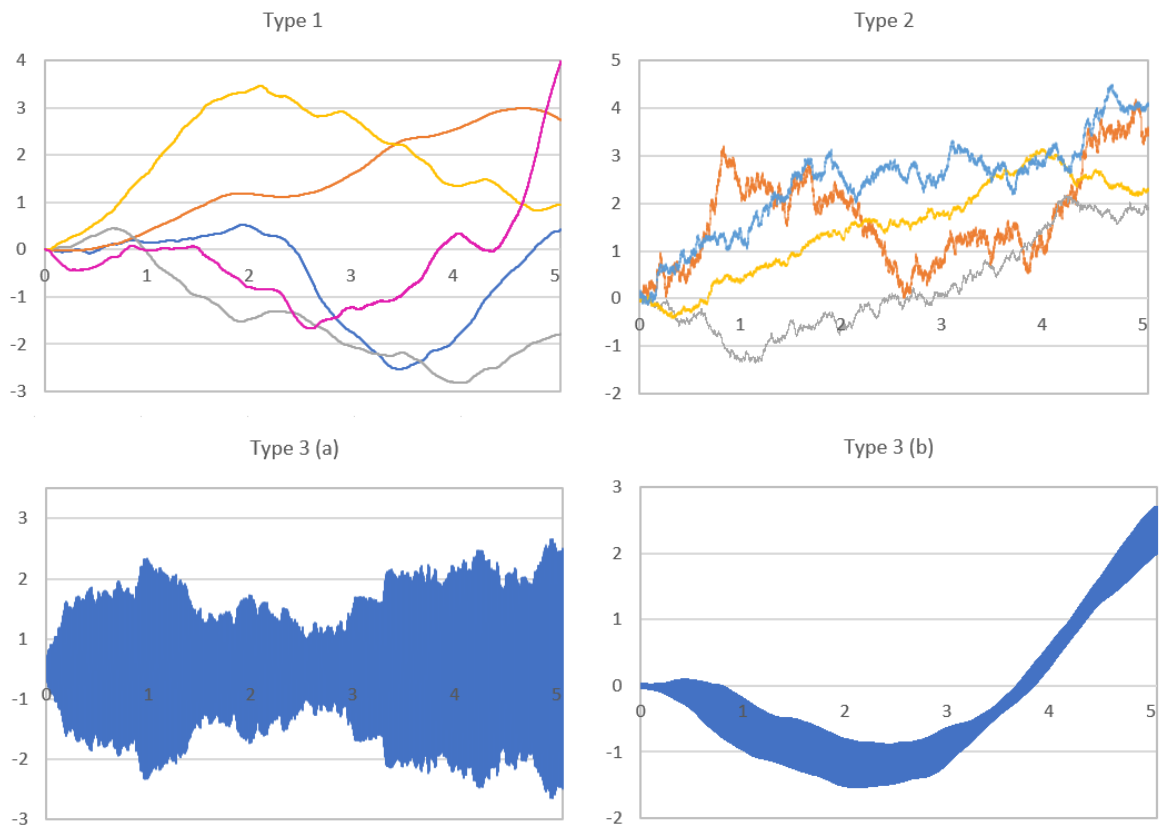


Figure 4.4: Integrated Brownian (top left), Brownian (top right) and quantum derivatives (bottom)

The project consists of the following steps:

Step 1: MPmath library, complex and prime numbers. Compute $L_4(s)$ using the MPmath library. See my code in section 4.4.4. The code is designed to handle complex numbers, even though in the end I only use the real part. Learn how to manipulate complex numbers by playing with the code. Also, see how to create a large list of prime numbers: look at how I use the PrimePy library. Finally, look at the Python `curve_fitting` and `r2_score` functions, to understand how it works, as you will have to use them. The former is from the Scipy library, and the latter (to measure goodness-of-fit) is from the Sklearn library.

Step 2: Curve fitting, part A. Implement Λ_n and the Euler Product $L_4(s, n)$ with n (the number of factors) up to 10^5. My code runs in a few seconds for $n \leq 10^5$. Beyond $n = 10^7$, a distributed architecture may help. In the code, you specify m rather than n, and $p_n \approx m/\log m$ is the largest prime smaller than m. For $s = \sigma + it$, choose $t = 0$, thus avoiding complex numbers, and various values of σ ranging from 0.55 to 1.75. The main step is the curve fitting procedure, similar to a linear regression but for non linear functions. The goal is to approximate $\delta_n(s)$, focusing for now on $\sigma = 0.90$.

The plots of $\delta_n(s)$ and Λ_n at the top in Figure 4.5, with $n =$ 80,000, look very similar. It seems like there must be a strictly positive $\rho_n(s)$ such that $\rho_n(s)\delta_k(s) \approx \Lambda_k$ for $k = 1, 2, \ldots, n$. Indeed, the scaling factor

$$\rho_n(s) = \frac{\text{Stdev}[\Lambda_1, \ldots, \Lambda_n]}{\text{Stdev}[\delta_1(s), \ldots, \delta_n(s)]}$$

works remarkably well. The next step is to refine the linear approximation based on $\rho = \rho_n(s)$, using (4.3) combined with (4.5). This is where the curve fitting takes place; the parameters to estimate are c, α and β, with c close to zero. You can check the spectacular result of the fit, here with $\sigma = 0.90$ and $n \approx 1.25 \times 10^6$, on the bottom left plot in Figure 4.5. Dropping the $\sqrt{n \log n}$ term in (4.3) results in a noticeable drop in performance (test it). For $\rho_n(s)$, also try different options.

Step 3: Curve fitting, part B. Perform the same curve fitting as in Step 2, but this time for different values of n. Keep $s = \sigma + it$ with $t = 0$ and $\sigma = 0.90$. The results should be identical to those in Table 4.3, where $\gamma_n = \sqrt{n} \cdot \Lambda_n$. The coefficients c, α, β and R^2 (the R-squared or quality of the fit) depend on n and s, explaining the notation in the table. Does $\rho_n(s)$ tend to a constant depending only on s, as $n \to \infty$? Or does it stay bounded? What about the other coefficients?

Now do the same with $\sigma = 0.70$ and $\sigma = 1.10$, again with various values of n. Based on your computations, do you think that $\rho_n(s)$ decreases to zero, stays flat, or increases to infinity, depending on whether $s = 0.70$, $s = 0.90$ or $s = 1.10$? If true, what are the potential implications?

Step 4: Comparison with synthetic functions. First, try $\varphi(n) = n \log n$ rather $\varphi(n) = n$, in (4.4). Show that the resulting curve fitting is not as good. Then, replace $\chi_4(p_k)$, both in $L_4(s, n)$ and Λ_n, by independent Rademacher distributions [Wiki], taking the values $+1$ and -1 with the same probability $\frac{1}{2}$. Show that again, the curve fitting is not as good, especially if $n \leq 10^5$. Then, you may even replace p_k (the k-th prime) by $k \log k$. The goal of these substitutions is to compare the results when χ_4 is replaced by synthetic functions that mimic the behavior of the Dirichlet character modulo 4. Also, you want to assess how much leeway you have in the choice of these functions, for the conclusions to stay valid.

The use of synthetic functions is part of a general approach known as generative AI. If all the results remain valid for such synthetic functions, then the theory developed so far is not dependent on special properties of prime numbers: we isolated that problem, opening the path to an easier proof that the Euler product $L_4(s, n)$ converges to $L_4(s)$ at some location $s = \sigma_0 + it$ with $\sigma_0 < 1$ inside the critical strip.

Step 5: Application outside number theory. Using various pairs of sequences $\{a_n\}$, $\{b_n\}$, empirically verify when the statistical theorem 4.4.1 might be correct, and when it might not.

The Python code in section 4.4.4 allows you to perform all the tasks except Step 5. In particular, for Step 4, set `mode='rn'` in the code. As for the curve fitting plot – the bottom left plot in Figure 4.5 – I multiplied both the target function $\delta_n(s)$ and the fitted curve by $\sqrt{n}$, here with $n = 1.25 \times 10^6$. Both tend to zero, but after multiplication by $\sqrt{n}$, they may or may not tend to a constant strictly above zero. Either way, it seems to indicate that the Euler product converges when $\sigma = 0.90$. What's more, the convergence looks strong, non-chaotic, and the second-order term involving $\sqrt{n \log n}$ in the approximation error, seems to be correct.

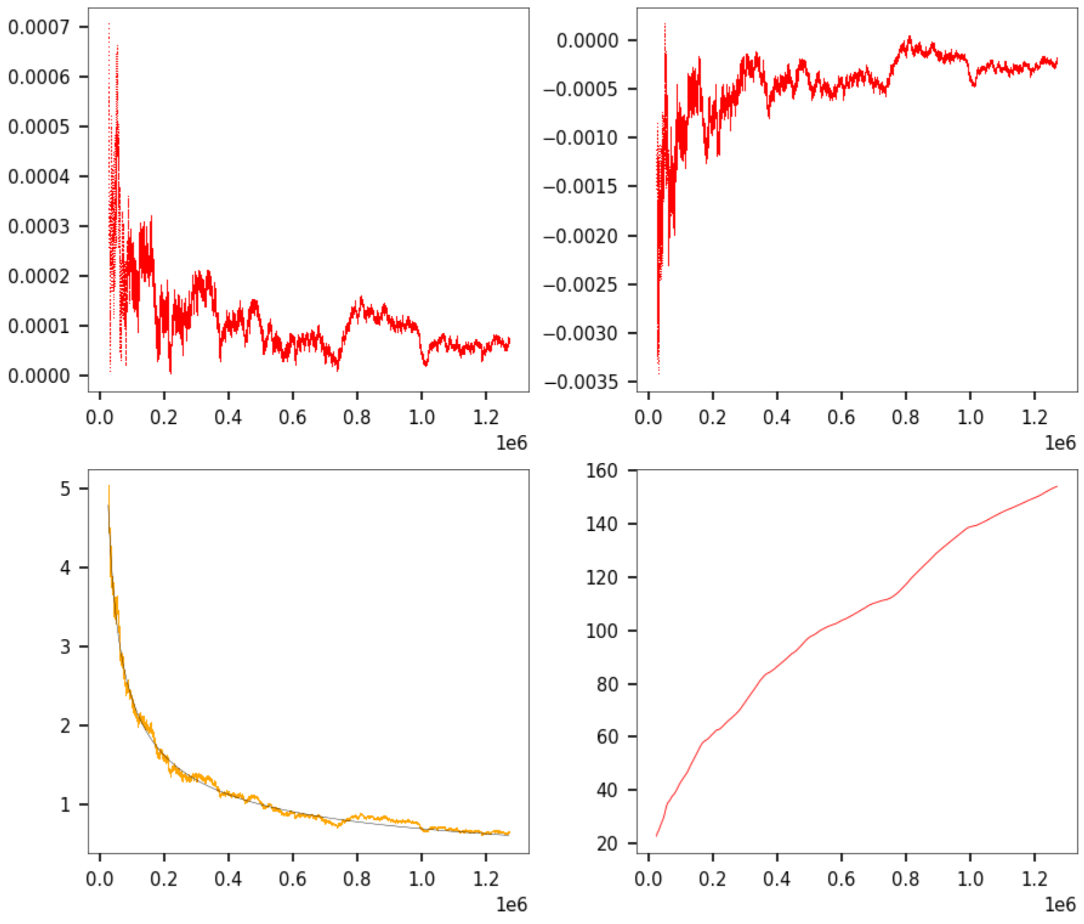


Figure 4.5: Top: $\delta_n(s)$ [left], Λ_n [right]; bottom: fitting $\delta_n(s)$ [left], integrated $\delta_n(s)$ [right]

Regarding Step 3, Table 4.3 is the answer when $\sigma = 0.90$. It seems to indicate that $\rho_n(s)$ convergences (or is at least bounded and strictly above zero) when $\sigma = 0.90$ (remember that $s = \sigma + it$, with $t = 0$). With $\sigma = 0.70$, it seems that $\rho_n(s)$ decreases probably to zero, while with $\sigma = 1.10$, $\rho_n(s)$ is increasing without upper bound.

The highest stability is around $\sigma = 0.90$. There, theorem 4.4.1 may apply, which would prove the convergence of the Euler product strictly inside to critical strip. As stated earlier, this would be a huge milestone if it can be proved, partially solving GRH not for $\zeta(s)$, but for the second most famous function of this nature, namely $L_4(s)$. By partial solution, I mean proving it for (say) $\sigma_0 = 0.90 < 1$, but not yet for $\sigma_0 = \frac{1}{2}$.

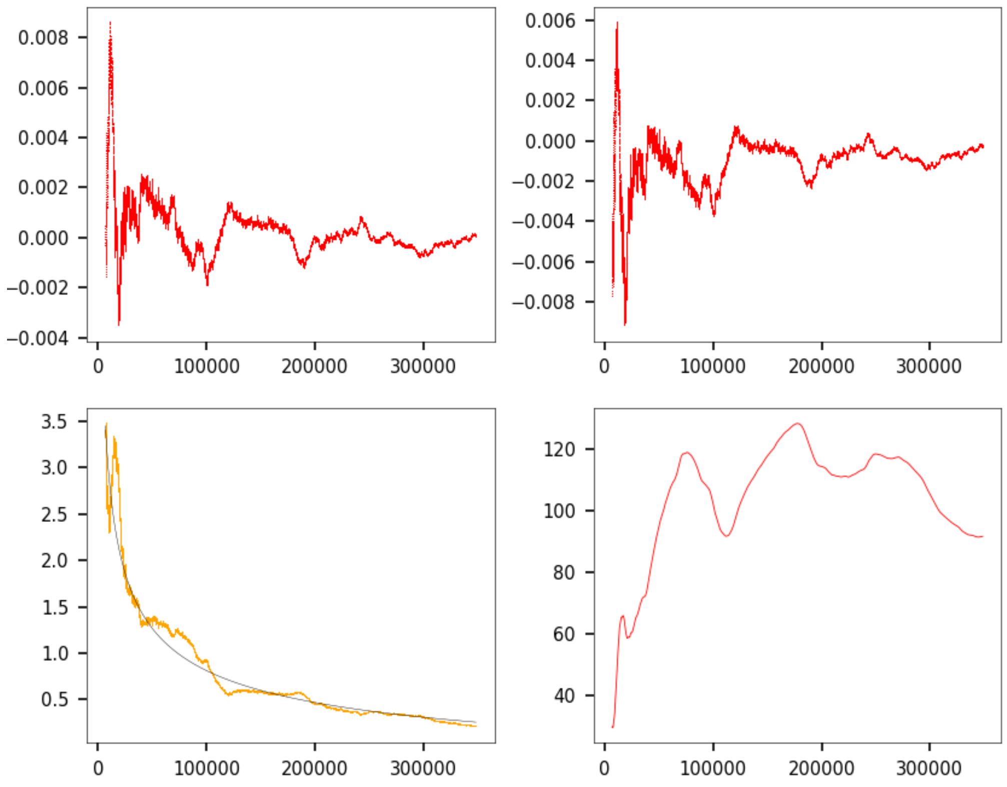


Figure 4.6: Same as Figure 4.5, replacing $\delta_n(s), \Lambda_n$ by synthetic functions

n	γ_n	$c_n(s)$	$\alpha_n(s)$	$\beta_n(s)$	$\rho_n(s)$	R^2
127,040	−0.27	0.00	0.41	0.94	5.122	0.940
254,101	−0.40	0.00	0.39	0.98	5.082	0.976
381,161	−0.35	0.00	0.37	1.03	5.151	0.986
508,222	−0.44	0.00	0.36	1.05	5.149	0.989
635,283	−0.32	0.00	0.36	1.04	5.085	0.989
762,343	−0.22	0.00	0.36	1.03	5.047	0.989
889,404	−0.10	0.00	0.35	1.06	5.123	0.990
1,016,464	−0.44	0.00	0.35	1.07	5.139	0.990
1,143,525	−0.30	0.00	0.34	1.08	5.121	0.991
1,270,586	−0.24	0.00	0.34	1.08	5.111	0.991

Table 4.3: One curve fitting per row, for $\delta_n(s)$ with $s = 0.90$

Unexpectedly,Figure 4.6 shows that the fit is not as good when using a random sequence of $+1$ and -1, evenly distributed, to replace and mimic χ_4. The even distribution is required by the Dirichlet theorem, a generalization of the prime number theorem to arithmetic progressions [Wiki].

Finally, see the short code below as the answer to Step 5. The code is also on GitHub, here. The parameters p, q play a role similar to σ, and `r` represents ρ_n in theorem 4.4.1. The coefficient ρ_n may decrease to zero, increase to infinity, or converge depending on p and q. Nevertheless, in most cases when p, q are not too small, b_n converges. Applied to $L_4(s, n)$, it means that convergence may occur at s even if $\rho_n(s)$ does not converge.

The existence of some σ_1 for which $\rho_n(s)$ decreases to zero, and some σ_2 for which $\rho_n(s)$ increases to infinity, implies that there must be a σ_0 in the interval $[\sigma_1, \sigma_2]$, for which $\rho_n(s)$ converges or is bounded. This in turn implies that the Euler product $L_4(s, n)$ converges at $s = \sigma + it$ if $\sigma > \sigma_0$. The difficult step is to show that the largest σ_1 resulting in $\rho_n(s)$ decreasing to zero, is < 1. Then, $\sigma_0 < 1$, concluding the proof.

```python
import numpy as np

N = 10000000
p = 1.00
q = 0.90
stdev = 0.50
seed = 564
np.random.seed(seed)
start = 20

u = 0
v = 0
a = np.zeros(N)
b = np.zeros(N)

for n in range(2, N):

   u += -0.5 + np.random.randint(0, 2)
   v += np.random.normal(0, stdev)/n**q
   a[n] = u / n**p
   b[n] = v

   if n % 50000 == 0:
      sa = np.std(a[start:n])
      sb = np.std(b[start:n])
      r = sa / sb
      c = r * b[n] - a[n]
      print("n = %7d r =%8.5f an =%8.5f bn =%8.5f c =%8.5f sa =%8.5f sb=%8.5f"
              %(n, r, a[n], b[n], c, sa, sb))
```

4.4.4 Python code

The implementation of the quantum derivatives is in section [4] in the following code. The coefficient $\rho_n(s)$ is denoted as `r`, while the parameter γ in table 4.3 is denoted as `mu`. In addition to addressing Step 1 to Step 4, the computation of the Dirichlet-L series and the Q_2 function are in section [2.1]. For Step 5, see the Python code at the end of section 4.4.3. Finally, to use synthetic functions rather than χ_4, set `mode='rn'`. The code is also on GitHub, here.

```python
# DirichletL4_EulerProduct.py
# On WolframAlpha: DirichletL[4,2,s], s = sigma + it
#    returns Dirichlet L-function with character modulo k and index j.
#
# References:
#    https://www.maths.nottingham.ac.uk/plp/pmzcw/download/fnt_chap4.pdf
#    https://mpmath.org/doc/current/functions/zeta.html
#    f(s) = dirichlet(s, [0, 1, 0, -1]) in MPmath

import matplotlib.pyplot as plt
import matplotlib as mpl
import mpmath
import numpy as np
from primePy import primes
from scipy.optimize import curve_fit
from sklearn.metrics import r2_score
import warnings
warnings.filterwarnings("ignore")

#--- [1] create tables of prime numbers

m = 1000000 # primes up to m included in Euler product
aprimes = []
```

```
for k in range(m):
   if k % 100000 == 0:
      print("Creating prime table up to p <=", k)
   if primes.check(k) and k > 2:
      aprimes.append(k)


#--- [2] Euler product

#--- [2.1] Main function

def L4_Euler_prod(mode = 'L4', sigma = 1.00, t = 0.00):

   L4 = mpmath.dirichlet(complex(sigma,t), [0, 1, 0, -1])
   print("\nMPmath lib.: L4(%8.5f + %8.5f i) = %8.5f + %8.5f i"
        % (sigma, t,L4.real,L4.imag))

   prod = 1.0
   sum_chi4 = 0
   sum_delta = 0
   run_chi4 = 0
   old_chi4 = 0
   DLseries = 0
   flag = 1

   aprod = []
   adelta = []
   asum_delta = []
   achi4 = []
   arun_chi4 = []
   asum_chi4 = []

   x1 = []
   x2 = []
   error1 = []
   error2 = []
   seed = 116 # try 103, 105, 116 & start = 2000 (for mode = 'rn')
   np.random.seed(seed)
   eps = 0.000000001

   for k in range(len(aprimes)):

      if mode == 'L4':
         condition = (aprimes[k] % 4 == 1)
      elif mode == 'Q2':
         condition = (k % 2 == 0)
      elif mode == 'rn':
         condition = (np.random.uniform(0,1) < 0.5)

      if condition:
         chi4 = 1
      else:
         chi4 = -1

      sum_chi4 += chi4
      achi4.append(chi4)
      omega = 1.00 # try 1.00, sigma or 1.10
      # if omega > 1, asum_chi4[n] --> 0 as n --> infty
      # asum_chi4.append(sum_chi4/aprimes[k]**omega)
      asum_chi4.append(sum_chi4/(k+1)**omega)
      # asum_chi4.append(sum_chi4/(k+1)*(np.log(k+2)))

      if chi4 == old_chi4:
         run_chi4 += chi4
      else:
         run_chi4 = chi4
      old_chi4 = chi4
      arun_chi4.append(run_chi4)

      factor = 1 - chi4 * mpmath.power(aprimes[k], -complex(sigma,t))
      prod *= factor
      aprod.append(1/prod)

      term = mpmath.power(2*k+1, -complex(sigma,t))
      DLseries += flag*term
      flag = -flag

   limit = -eps + 1/prod # full finite product (approx. of the limit)
   if mode == 'L4':
      limit = L4 # use exact value instead (infinite product if it converges)

   for k in range(len(aprimes)):

      delta = (aprod[k] - limit).real # use real part
      adelta.append(delta)
      sum_delta += delta
      asum_delta.append(sum_delta)
      chi4 = achi4[k]

      if chi4 == 1:
         x1.append(k)
         error1.append(delta)
      elif chi4== -1:
         x2.append(k)
         error2.append(delta)

   print("Dirichlet L: DL(%8.5f + %8.5f i) = %8.5f + %8.5f i"
      % (sigma, t, DLseries.real, DLseries.imag))
   print("Euler Prod.: %s(%8.5f + %8.5f i) = %8.5f + %8.5f i\n"
      % (mode, sigma, t, limit.real, limit.imag))

   adelta = np.array(adelta)
   aprod = np.array(aprod)
   asum_chi4 = np.array(asum_chi4)
   asum_delta = np.array(asum_delta)
   error1 = np.array(error1)
   error2 = np.array(error2)

   return(limit.real, x1, x2, error1, error2, aprod, adelta, asum_delta,
          arun_chi4, asum_chi4)

#--- [2.2] Main part

mode = 'L4' # options: 'L4', 'Q2', 'rn' (random chi4)
(prod, x1, x2, error1, error2, aprod, adelta, asum_delta, arun_chi4,
    asum_chi4) = L4_Euler_prod(mode, sigma = 0.90, t = 0.00)


#--- [3] Plots (delta is Euler product, minus its limit)

mpl.rcParams['axes.linewidth'] = 0.3
plt.rcParams['xtick.labelsize'] = 7
plt.rcParams['ytick.labelsize'] = 7

#- [3.1] Plot delta and cumulated chi4

x = np.arange(0, len(aprod), 1)

# offset < len(aprimes), used to enhance visualizations
offset = int(0.02 * len(aprimes))
```

```
# y1 = aprod / prod
# plt.plot(x[offset:], y1[offset:], linewidth = 0.1)
# plt.show()

y2 = adelta
plt.subplot(2,2,1)
plt.plot(x[offset:], y2[offset:], marker=',', markersize=0.1,
   linestyle='None', c='red')

y3 = asum_chi4
plt.subplot(2,2,2)
plt.plot(x[offset:], y3[offset:], marker=',', markersize=0.1,
   linestyle='None', c='red')

#- [3.2] Denoising L4, curve fitting

def objective(x, a, b, c):

   # try c = 0 (actual limit)
   value = c + a/np.sqrt(x) + b/np.sqrt(x*np.log(x))
   return value

def model_fit(x, y2, y3, start, offset, n_max):

   for k in range(n_max):

      n = int(len(y2) * (k + 1) / n_max) - start
      stdn_y2 = np.std(y2[start:n])
      stdn_y3 = np.std(y3[start:n])
      r = stdn_y3 / stdn_y2

      # note: y3 / r ~ mu / sqrt(x) [chaotic part]
      mu = y3[n] * np.sqrt(n) # tend to a constant ?
      y4 = y2 * r - y3
      y4_fit = []
      err = -1

      if min(y4[start:]) > 0:
         popt, pcov = curve_fit(objective, x[start:n], y4[start:n],
                    p0=[1, 1, 0], maxfev=5000)
         [a, b, c] = popt
         y4_fit = objective(x, a, b, c)
         err = r2_score(y4[offset:], y4_fit[offset:])
         print("n = %7d mu =%6.2f c =%6.2f a =%5.2f b =%5.2f r =%6.3f err =%6.3f"
                  %(n, mu, c, a, b, r, err))

   return(y4, y4_fit, err, n)

n_max = 10 # testing n_max values of n, equally spaced
start = 20 # use Euler products with at least 'start' factors
if mode == 'rn':
   start = 1000
if start > 0.5 * offset:
   print("Warning: 'start' reduced to 0.5 * offset")
   start = int(0.5 * offset)
(y4, y4_fit, err, n) = model_fit(x, y2, y3, start, offset, n_max)
ns = np.sqrt(n)

if err != -1:
   plt.subplot(2,2,3)
   plt.plot(x[offset:], ns*y4[offset:], marker=',', markersize=0.1,
      linestyle='None', c='orange')
   plt.plot(x[offset:], ns*y4_fit[offset:], linewidth = 0.2, c='black')
else:
   print("Can't fit: some y4 <= 0 (try different seed or increase 'start')")
```

```
#--- [3.3] Plot integral of delta

y5 = asum_delta
plt.subplot(2,2,4)
plt.plot(x[offset:], y5[offset:], linewidth = 0.4, c='red')
plt.show()


#--- [4] Quantum derivative

#- [4.1] Function to differentiated: delta, here broken down into 2 legs

plt.subplot(1,2,1)
shift = 0.001
plt.plot(x1[offset:], error1[offset:], marker=',', markersize=0.1,
  linestyle='None', alpha = 1.0, c='red')
plt.plot(x2[offset:], shift + error2[offset:], marker=',', markersize=0.1,
  linestyle='None', alpha = 0.2, c='orange')

#- [4.2] Quantum derivative

def d_error(arr_error):

   diff_error = [] # discrete derivative of the error
   positives = 0
   negatives = 0
   for k in range(len(arr_error)-1):
      diff_error.append(arr_error[k+1] - arr_error[k])
      if arr_error[k+1] - arr_error[k] > 0:
         positives +=1
      else:
         negatives += 1
   return(diff_error, positives, negatives)

(diff_error1, positives1, negatives1) = d_error(error1)
(diff_error2, positives2, negatives2) = d_error(error2)
ymin = 0.5 * float(min(min(diff_error1[offset:]), min(diff_error1[offset:])))
ymax = 0.5 * float(max(max(diff_error1[offset:]), max(diff_error2[offset:])))

plt.subplot(1,2,2)
plt.ylim(ymin, ymax)
plt.plot(x1[offset:len(x1)-1], diff_error1[offset:len(x1)-1], marker=',', markersize=0.1,
       linestyle='None', alpha=0.8, c = 'red')
plt.plot(x2[offset:len(x2)-1], diff_error2[offset:len(x2)-1], marker=',', markersize=0.1,
       linestyle='None', alpha=0.8, c = 'orange')
plt.show()

print("\nError 1: positives1: %8d negatives1: %8d" % (positives1, negatives1))
print("Error 2: positives2: %8d negatives2: %8d" % (positives2, negatives2))
```

Bibliography

[1] Adel Alamadhi, Michel Planat, and Patrick Solé. Chebyshev's bias and generalized Riemann hypothesis. *Preprint*, pages 1–9, 2011. arXiv:1112.2398 [Link]. 54

[2] K. Binswanger and P. Embrechts. Longest runs in coin tossing. *Insurance: Mathematics and Economics*, 15:139–149, 1994. [Link]. 47

[3] Ramiro Camino, Christian Hammerschmidt, and Radu State. Generating multi-categorical samples with generative adversarial networks. *Preprint*, pages 1–7, 2018. arXiv:1807.01202 [Link]. 83

[4] Fida Dankar et al. A multi-dimensional evaluation of synthetic data generators. *IEEE Access*, pages 11147–11158, 2022. [Link]. 82

[5] Antónia Földes. The limit distribution of the length of the longest head-run. *Periodica Mathematica Hungarica*, 10:301–310, 1979. [Link]. 48

[6] Louis Gordon, Mark F. Schilling, and Michael S. Waterman. An extreme value theory for long head runs. *Probability Theory and Related Fields*, 72:279–287, 1986. [Link]. 47

[7] Vincent Granville. Feature clustering: A simple solution to many machine learning problems. *Preprint*, pages 1–6, 2023. MLTechniques.com [Link]. 71

[8] Vincent Granville. Generative AI: Synthetic data vendor comparison and benchmarking best practices. *Preprint*, pages 1–13, 2023. MLTechniques.com [Link]. 64

[9] Vincent Granville. Generative AI technology break-through: Spectacular performance of new synthesizer. *Preprint*, pages 1–16, 2023. MLTechniques.com [Link]. 12, 15, 89

[10] Vincent Granville. *Gentle Introduction To Chaotic Dynamical Systems*. MLTechniques.com, 2023. [Link]. 49, 53

[11] Vincent Granville. How to fix a failing generative adversarial network. *Preprint*, pages 1–10, 2023. MLTechniques.com [Link]. 14

[12] Vincent Granville. Massively speed-up your learning algorithm, with stochastic thinning. *Preprint*, pages 1–13, 2023. MLTechniques.com [Link]. 13, 71

[13] Vincent Granville. Smart grid search for faster hyperparameter tuning. *Preprint*, pages 1–8, 2023. MLTechniques.com [Link]. 13, 69, 71

[14] Vincent Granville. *Synthetic Data and Generative AI*. Elsevier, 2024. [Link]. 27, 32, 33, 34, 36, 37, 43, 44, 46, 52, 63, 64, 68, 69, 71, 72, 74, 82, 83

[15] Elisabeth Griesbauer. *Vine Copula Based Synthetic Data Generation for Classification*. 2022. Master Thesis, Technical University of Munich [Link]. 71

[16] Emil Grosswald. Oscillation theorems of arithmetical functions. *Transactions of the American Mathematical Society*, 126:1–28, 1967. [Link]. 54

[17] Adam J. Harper. Moments of random multiplicative functions, II: High moments. *Algebra and Number Theory*, 13(10):2277–2321, 2019. [Link]. 54

[18] 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]. 54

[19] Adam J. Harper. Almost sure large fluctuations of random multiplicative functions. *Preprint*, pages 1–38, 2021. arXiv [Link]. 54

[20] Zsolt Karacsony and Jozsefne Libor. Longest runs in coin tossing. teaching recursive formulae, asymptotic theorems and computer simulations. *Teaching Mathematics and Computer Science*, 9:261–274, 2011. [Link]. 48

[21] Tamas Mori. The a.s. limit distribution of the longest head run. *Canadian Journal of Mathematics*, 45:1245–1262, 1993. [Link]. 48

[22] Michel Planat and Patrick Solé. Efficient prime counting and the Chebyshev primes. *Preprint*, pages 1–15, 2011. arXiv:1109.6489 [Link]. 54

[23] M.S. Schmookler and K.J. Nowka. Bounds on runs of zeros and ones for algebraic functions. *Proceedings 15th IEEE Symposium on Computer Arithmetic*, pages 7–12, 2001. ARITH-15 [Link]. 47

[24] Mark Shilling. The longest run of heads. *The College Mathematics Journal*, 21:196–207, 2018. [Link]. 47

[25] Chang Su, Linglin Wei, and Xianzhong Xie. Churn prediction in telecommunications industry based on conditional Wasserstein GAN. *IEEE International Conference on High Performance Computing, Data, and Analytics*, pages 186–191, 2022. IEEE HiPC 2022 [Link]. 82

[26] Terence Tao. Biases between consecutive primes. *Tao's blog*, 2016. [Link]. 54

[27] Ruonan Yu, Songhua Liu, and Xinchao Wang. Dataset distillation: A comprehensive review. *Preprint*, pages 1–23, 2022. Submitted to IEEE PAMI [Link]. 69

Index