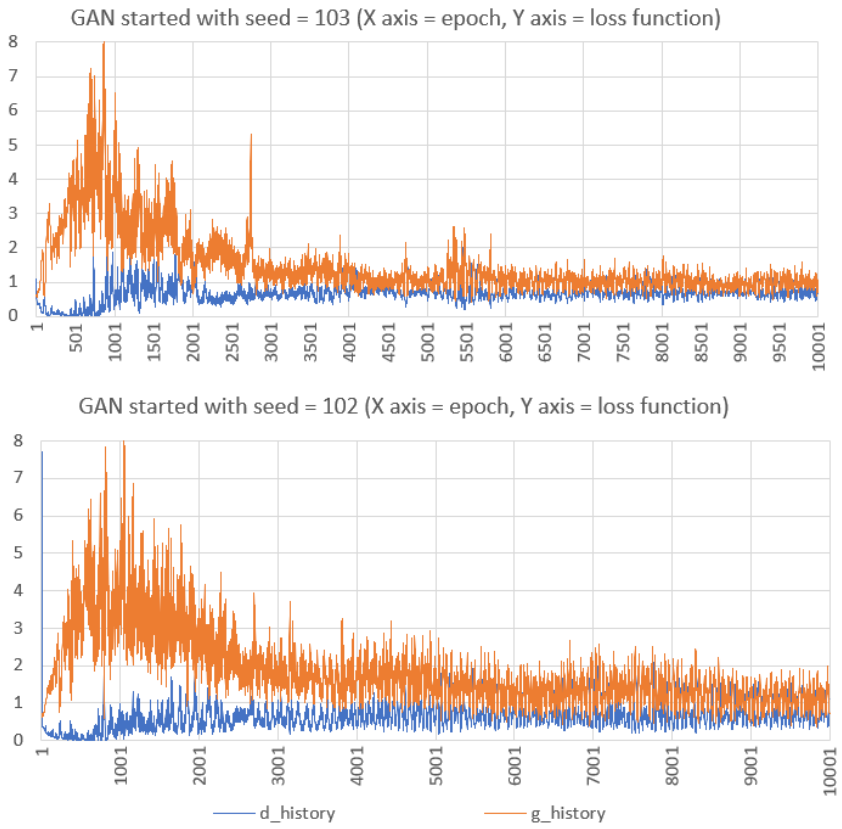
Practical AI & Machine Learning Projects and Datasets



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 serious, 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, it is full of original and open questions, never posted before, encouraging candidates to think outside the box, with applications on real data. The amount of Python code accompanying the solutions is tremendous, 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. The code is also on my GitHub repository.

The topics cover generative AI, synthetic data, machine learning optimization, scientific computing with Python, data visualizations and animations, time series and spatial processes, NLP and large language models, as well as graph applications and more. These projects based on real life data, with solution and enterprise-grade Python code, are a great addition to your portfolio (GitHub) when completed. Hiring managers and instructors can use them as as a complement to their battery of tests and projects, to differentiate themselves from competitors relying on overused, run-of-the-mill exercises.

To see how the program works and earn your certification(s), 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, with just one click. 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 and most well-known experts in machine learning, 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.

```
seed = 105
np.random.seed(seed)
kmax = 1000000
p = 5
q = 3

# local variables
X, pX, qX = 0, 0, 0
d1, d2, e1, e2 = 0, 0, 0, 0
prod, count = 0, 0

# loop over digits in reverse order
for k in range(kmax):

   b = np.random.randint(0, 2) # digit of X
   X = b + X/2

   c1 = p*b
   old_d1 = d1
   old_e1 = e1
   d1 = (c1 + old_e1//2) %2 # digit of pX
   e1 = (old_e1//2) + c1 - d1
   pX = d1 + pX/2

   c2 = q*b
   old_d2 = d2
   old_e2 = e2
   d2 = (c2 + old_e2//2) %2 #digit of qX
   e2 = (old_e2//2) + c2 - d2
   qX = d2 + qX/2

   prod += d1*d2
   count += 1
   correl = 4*prod/count - 1

   if k% 10000 == 0:
      print("k = %7d, correl = %7.4f" % (k, correl))

print("\np = %3d, q = %3d" %(p, q))
print("X = %12.9f, pX = %12.9f, qX = %12.9f" % (X, pX, qX))
print("X = %12.9f, p*X = %12.9f, q*X = %12.9f" % (X, p*X, q*X))
print("Correl = %7.4f, 1/(p*q) = %7.4f" % (correl, 1/(p*q)))
```

4.3 Longest runs of zero in binary digits of $\sqrt{2}$

Studying the longest head runs in coin tossing has a very long history, starting in gaming and probability theory. Today, it has applications in cryptography and insurance [1]. For random sequences or Bernoulli trials, the associated statistical properties and distributions have been studied in details [5], even when the proportions of zero and one are different. Yet, I could barely find any discussion on deterministic sequences, such as the digits or irrational numbers [17]. The case study investigated here fills this gap, focusing on one of the deepest and most challenging problems in number theory: almost all the questions about the distribution of these digits, even the most basic ones such as the proportions of zero and one, are still unsolved conjectures to this day.

In this context, a run is a sequence of successive, identical digits. In random sequences of bits, runs have a specific probability distribution. In particular, the maximum length of a run in a random sequence of n binary digits has expectation $\log_2 n$. For details and additional properties, see [18]. This fact can be used to test if a sequence violates the laws of randomness. Pseudo-random number generators (PRNG) that do not pass this test are not secure.

The focus here is on sequences of binary digits of quadratic irrational numbers of the form $x_0 = \sqrt{p/q}$, where p, q are positive coprime integers. The goal is to show, using empirical evidence, that indeed such sequences pass the test. More precisely, the project consists of computing runs of zero in billions of successive binary digits of x_0 for specific p and q. Let L_n be the length of such a run starting at position n in the digit expansion of x_0. Do we have $L_n / \log_2 n \leq \lambda$ where λ is a positive constant? Based on the computations, is it reasonable to

conjecture that $\lambda = 1$ if n is large enough? Very little is known about these digits. However, before discussing the project in more details, I share a much weaker yet spectacular result, easy to prove. By contrast to the digits investigated here, there is an abundance of far more accurate theoretical results, proved long ago, for random bit sequences. See for instance [4, 15, 16].

4.3.1 Surprising result about the longest runs

For numbers such as $\sqrt{p/q}$ where p, q are coprime positive integers and p/q is not the square of a rational number, a run of zero starting at position n can not be longer than $n + C$ where C is a constant depending on p and q. Due to symmetry, the same is true for runs of one.

Proof

A run of length m starting at position n, consisting of zeros, means that the digit d_n at position n is one, followed by m digits all zero, and then by at least one non-zero digit. For this to happen, we must have

$$\frac{1}{2} < 2^n x_0 - \Big\lfloor 2^n x_0 \Big\rfloor = \frac{1}{2} + \alpha,$$

where

$$x_0 = \sqrt{\frac{p}{q}} \quad \text{and} \quad \frac{1}{2^{m+2}} \leq \alpha < \sum_{k=0}^{\infty} \frac{1}{2^{m+2+k}} = \frac{1}{2^{m+1}}.$$

Here the brackets represent the floor function. Let $K_n = 1 + 2 \cdot \lfloor 2^n x_0 \rfloor$. Then, $K_n < 2^{n+1} x_0 = K_n + 2\alpha$. After squaring both sides, we finally obtain

$$qK_n^2 < 4^{n+1} \cdot p = q(K_n + 2\alpha)^2.$$

Now, let $\delta_n = 4^{n+1}p - qK_n^2$. Note that δ_n is a strictly positive integer, smallest when p, q are coprime. After some simple rearrangements, we obtain

$$\begin{aligned} 2\alpha &= \frac{1}{\sqrt{q}} \left[2^{n+1}\sqrt{p} - \sqrt{4^{n+1}p - \delta_n} \right] \\ &= \frac{1}{\sqrt{q}} \left[\frac{4^{n+1}p - (4^{n+1}p - \delta_n)}{2^{n+1}\sqrt{p} + \sqrt{4^{n+1}p - \delta_n}} \right] \\ &= \frac{1}{\sqrt{q}} \cdot \frac{\delta_n}{2^{n+1}\sqrt{p} + \sqrt{4^{n+1}p - \delta_n}} \\ &\sim \frac{1}{\sqrt{q}} \cdot \frac{\delta_n}{2 \cdot 2^{n+1}\sqrt{p}} \text{ as } n \to \infty. \end{aligned}$$

In the last line, I used the fact that δ_n is at most of the order 2^n, thus negligible compared to $4^{n+1}p$.

Since a run of length m means $2^{-(m+2)} \leq \alpha < 2^{-(m+1)}$, combining with the above (excellent) asymptotic result, we have, for large n:

$$\frac{1}{2^{m+2}} \leq \frac{\delta_n}{4 \cdot 2^{n+1}\sqrt{pq}} < \frac{1}{2^{m+1}}.$$

Taking the logarithm in base 2, we obtain

$$m + 1 \leq -\log_2 \delta_n + \frac{1}{2}\log_2(pq) + n + 3 < m + 2,$$

and thus, assuming L_n denotes the length of the run at position n and n is large enough:

$$L_n = \Big\lfloor -\log_2 \delta_n + \frac{1}{2}\log_2(pq) + n + 2 \Big\rfloor \leq n + 2 + \frac{1}{2}\log_2(pq). \tag{4.1}$$

This concludes the proof. It provides an upper bound for the maximum possible run length at position n: in short, $L_n \leq n + C$, where C is an explicit constant depending on p and q. ∎

Note that if $p = 1$ and q is large, there will be a long run of zero at the very beginning, and thus C will be larger than usual. I implemented the equality in Formula (4.1) in my Python code in section 4.3.2. It yielded the exact run length in all instances, for all n where a new run starts. In the code, I use the fact that δ_n can be written as $u - qv^2$, where $u = 4^{n+1}p$ and v is a positive odd integer, the largest one that keeps $\delta_n = u - qv^2$ positive. It leads to an efficient implementation where L_n is computed iteratively as n increases, rather than from scratch for each new n.

4.3.2 Project and solution

You need to be able to correctly compute a large number of binary digits of numbers such as $\sqrt{2}$. In short, you must work with exact arithmetic (or infinite precision). This is one of the big takeaways of this project. As previously stated, the goal is to assess whether the maximum run length in binary digits of $x_0 = \sqrt{p/q}$, grows at the same speed as you would expect if the digits were random. We focus on runs of zero only. A positive answer would be one more indication (when combined with many other tests) that these digits indeed behave like random bits, and can be used to generate random sequences. Fast, secure random number generators based on quadratic irrationals are described in details in [9].

n	L_n	$\log_2 n$	$L_n/\log_2 n$
1	1	0.0000	
8	5	3.0000	
453	8	8.8234	0.9067
1302	9	10.3465	0.8699
5334	10	12.3810	0.8077
8881	12	13.1165	0.9149
24,001	18	14.5508	1.2370
574,130	19	19.1310	0.9932
3,333,659	20	21.6687	0.9230
4,079,881	22	21.9601	1.0018
8,356,568	23	22.9945	1.0002
76,570,752	25	26.1903	0.9546
202,460,869	26	27.5931	0.9423
457,034,355	28	28.7677	0.9733

Table 4.1: Record runs of zero in binary digits of $\sqrt{2}/2$

The project consists of the following steps:

Step 1: Computing binary digits of quadratic irrationals. Let $x_0 = \sqrt{p/q}$. Here I assume that p, q are positive coprime integers, and p/q is not the square of a rational number. There are different ways to compute the binary digits of x_0, see [9]. I suggest to use the Gmpy2 Python library: it is written in C, thus very fast, and it offers a lot of functions to work with arbitrary large numbers. In particular, `gmpy2.isqrt(z).base(b)` returns the integer square root [Wiki] of the integer `z`, in base `b`. Thus, to get the first N binary digits of x_0, use $z = \lfloor 2^{2N} \cdot p/q \rfloor$ and $b = 2$. The digits are stored as a string.

Step 2: Computing run lengths. Let L_n be the length of the run of zero starting at position n in the binary expansion of x_0. A more precise definition is offered in section 4.3.1. If there is no such run starting at n, set $L_n = 0$. Once the digits have been computed, it is very easy to obtain the run lengths: see Table 4.2, corresponding to $p = 1, q = 2$. However, I suggest using Formula (4.1) to compute L_n. Not only it shows that the formula is correct, but it also offers an alternative method not based on the binary digits, thus also confirming that the binary digits are correctly computed. The end result should be an extension of Table 4.2, ignoring the columns labeled as s_n for now. Try with the first $N = 10^6$ digits, with $p = 1$ and $q = 2$.

Step 3: Maximum run lengths. Use a fast technique to compute the successive records for L_n, for the first $N = 10^9$ binary digits of $\sqrt{2}/2$. Is it reasonable to conjecture that asymptotically, $L_n/\log_2 n$ is smaller than or equal to 1? In other words, could we have $\limsup L_n/\log_2 n = 1$? See [Wiki] for the definition of lim sup. Then instead of using the binary digits of some number, do the same test for random bits. Is the behavior similar? Do we get the same upper bound for the record run lengths? The records for L_n, and when their occur (location n) in the binary digit expansion, are displayed in Table 4.1. You should get the same values.

A fast solution to compute both the digits and the run lengths is to use the Gmpy2 library, as illustrated in the first code snippet below (also available on GitHub, here). The output is Table 4.1. The conclusion is that indeed, $\lambda_n = L_n/\log_2 n$ seems to be bounded by 1, at least on average as n increases. It does not mean that $\limsup \lambda_n = 1$. For instance, if S_n is the number of zeros in the first n digits, assuming the digits are random, then the ratio $\rho_n = |S_n - n/2|/\sqrt{n}$ seems bounded. However $\limsup \rho_n = \infty$. To get a strictly positive, finite constant for lim sup, you need to further divide ρ_n by $\sqrt{\log\log n}$. This is a consequence of the law of the iterated logarithm [Wiki]. More details are available in my book on dynamical systems [9]. For λ_n applied to random

binary digits, see here. This completes Step 1 and Step 3.

```
import gmpy2

p = 1
q = 2
N = 1000000000 # precision, in number of binary digits

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

last_digit = -1
L = 0
max_run = 0

for n in range(0, N):
   d = int(bsqrt[n]) # binary digit
   if d == 0:
      L += 1
   if d == 1 and last_digit == 0:
         run = L
         if run > max_run:
            max_run = run
            print(n-L, run, max_run)
         L = 0
   last_digit = d
```

At the bottom of this section, I share my Python code for Step 2, with the implementation of formula (4.1) to compute L_n. The results are in Table 4.2, and compatible with those obtained in Step 1 and displayed in Table 4.1. The method based on formula (4.1) is a lot slower. So why try it, you may ask? It is slower because Gmpy2 is implemented more efficiently, and closer to machine arithmetic. And the goal is different: formula (4.1) allows you to double check the earlier computations, using a method that does not require producing the binary digits to determine L_n.

n	d_n	L_n	s_n	n	d_n	L_n	s_n	n	d_n	L_n	s_n	n	d_n	L_n	s_n	n	d_n	L_n	s_n
1	1	1	1	21	0		0	41	1		1	61	0		1	81	1	3	1
2	0		0	22	0		1	42	1	1	1	62	1	3	2	82	0		0
3	1		2	23	1		2	43	0		0	63	0		0	83	0		1
4	1	1	1	24	1	2	1	44	1		2	64	0		1	84	0		1
5	0		0	25	0		0	45	1		1	65	0		1	85	1	2	2
6	1	1	2	26	0		1	46	1		1	66	1	1	2	86	0		0
7	0		0	27	1		2	47	1	2	1	67	0		0	87	0		1
8	1	5	2	28	1	2	1	48	0		0	68	1		2	88	1		2
9	0		0	29	0		0	49	0		1	69	1	2	1	89	1	1	1
10	0		1	30	0		1	50	1		2	70	0		0	90	0		0
11	0		1	31	1		2	51	1	2	1	71	0		1	91	1		2
12	0		1	32	1		1	52	0		0	72	1	1	2	92	1	2	1
13	0		1	33	1		1	53	0		1	73	0		0	93	0		0
14	1	2	2	34	1		1	54	1	2	2	74	1		2	94	0		1
15	0		0	35	1		1	55	0		0	75	1		1	95	1		2
16	0		1	36	1		1	56	0		1	76	1		1	96	1	1	1
17	1		2	37	1	2	1	57	1	4	2	77	1		1	97	0		0
18	1		1	38	0		0	58	0		0	78	1	1	1	98	1		2
19	1		1	39	0		1	59	0		1	79	0		0	99	1		1
20	1	2	1	40	1		2	60	0		1	80	1		2	100	1	1	1

Table 4.2: Binary digit d_n, run length L_n for zeros, and steps s_n at position n, for $\sqrt{2}/2$

Most importantly, the slow method is valuable because it is the first step to make progress towards a better (smaller) upper bound than that featured in section 4.3.1. To get a much stronger bound for the run lengths L_n,

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))
```

Bibliography

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

[2] Ramiro Camino, Christian Hammerschmidt, and Radu State. Generating multi-categorical samples with generative adversarial networks. *Preprint*, pages 1–7, 2018. arXiv:1807.01202 [Link]. 72

[3] Fida Dankar et al. A multi-dimensional evaluation of synthetic data generators. *IEEE Access*, pages 11147–11158, 2022. [Link]. 71

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

[5] 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

[6] Vincent Granville. Feature clustering: A simple solution to many machine learning problems. *Preprint*, pages 1–6, 2023. MLTechniques.com [Link]. 60

[7] Vincent Granville. Generative AI: Synthetic data vendor comparison and benchmarking best practices. *Preprint*, pages 1–13, 2023. MLTechniques.com [Link]. 53

[8] Vincent Granville. Generative AI technology break-through: Spectacular performance of new synthesizer. *Preprint*, pages 1–16, 2023. MLTechniques.com [Link]. 12, 15, 78

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

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

[11] Vincent Granville. Massively speed-up your learning algorithm, with stochastic thinning. *Preprint*, pages 1–13, 2023. MLTechniques.com [Link]. 13, 60

[12] Vincent Granville. Smart grid search for faster hyperparameter tuning. *Preprint*, pages 1–8, 2023. MLTechniques.com [Link]. 13, 58, 60

[13] Vincent Granville. *Synthetic Data and Generative AI*. Elsevier, 2024. [Link]. 27, 32, 33, 34, 36, 37, 43, 44, 46, 52, 53, 57, 58, 60, 61, 63, 71, 72

[14] Elisabeth Griesbauer. *Vine Copula Based Synthetic Data Generation for Classification*. 2022. Master Thesis, Technical University of Munich [Link]. 60

[15] 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

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

[17] 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

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

[19] 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]. 71

[20] Ruonan Yu, Songhua Liu, and Xinchao Wang. Dataset distillation: A comprehensive review. *Preprint*, pages 1–23, 2022. Submitted to IEEE PAMI [Link]. 58