Chapter 5

Test of Normality and Digit Distribution of Algebraic Numbers

The previous chapters feature new discoveries about the binary digit distribution of numbers related to Euler's number e, based on quadratic dynamical systems defined via simple auto-convolutions. In this chapter, the focus is on algebraic numbers, with binary digit sequences generated by a similar mechanism. A new fundamental theorem is introduced with proof, setting a new bar in what qualifies as "deep result" about these digit sequences, moving forward. Finally, in section 5.1, I discuss a test of randomness – more specifically about normality – based on a considerably simplified version of Weyl's criterion. Again, it relies on quadratic dynamical systems, this time with a state space consisting of 2×2 matrices with determinant and spectral radius equal to 1. The problem is strongly connected to asymptotic bounds of Chebyshev polynomials on some subsets of $[0, 1]$.

5.1 Simple normality test with application to PRNGs

The context is testing whether the binary digits of an irrational number, say $x_0 = \pi$, look random enough. This would make its digit sequence a good candidate to generate random bits. The methodology described here is new. It starts with a simplified version of the Weyl criterion for normality, along with introducing $y_0 = \cos 2\pi x_0$ to transform irrational numbers into rational ones via Chebyshev polynomials. Assuming $x_0 \in [0, 1]$, let start with the dyadic map $x_{n+1} = \{2x_n\}$ where the brackets denote the fractional part. The k-th binary digit of x_0 is the integer part of $2x_k$. For non-zero integers τ, the formula $x_{n+1} = \{2x_n\}$ can successively be rewritten as

$$\begin{aligned} x_{k+1} &= 2x_k \bmod 1 \\ 2\pi\tau x_{k+1} &= 2 \cdot (2\pi\tau x_k) \bmod 2\pi \\ \exp(2\pi i\tau x_{k+1}) &= \exp(2 \cdot 2\pi i\tau x_k) \\ \exp(2\pi i\tau x_{k+1}) &= \big(\exp(2\pi i\tau x_k)\big)^2 \\ \exp(2\pi i\tau x_k) &= \big(\exp(2\pi i\tau x_0)\big)^{2^k} \end{aligned}$$

Now, using the notation $z_k = \exp(2\pi i\tau x_k) = z_0^{2^k}$, we have

$$\prod_{k=0}^{n-1}(1+z_k) = \prod_{k=0}^{n-1}\left(1+z_0^{2^k}\right) = \sum_{k=0}^{2^n-1} z_0^k = \frac{1-z_0^{2^n}}{1-z_0}. \tag{5.1}$$

We say that x_0 is q-normal in base 2 if and only if

$$\lim_{n\to\infty}\left[\prod_{k=0}^{n-1}\left(1+z_0^{2^k}\right)\right]^{1/n} = 1 \tag{5.2}$$

for all non-zero integer τ. Taking the logarithm, the convergence in (5.2) is equivalent to the following:

$$\lim_{n\to\infty}\frac{1}{n}\sum_{k=0}^{n-1} z_0^{2^k} = 0, \quad \text{that is ,} \quad \lim_{n\to\infty}\frac{1}{n}\sum_{k=0}^{n-1}\exp\left(2\pi i\tau 2^k x_0\right) = 0 \tag{5.3}$$

for all non-zero integer τ. Interestingly, the right part in formula (5.3) is the Weyl criterion for the normality of x_0 in base 2. Thus the concepts of q-normality and normality are identical. Thanks to (5.1), we can go one step further to considerably simplify the Weyl criterion. The result is stated in the following theorem.

Theorem 5.1.1 *A number $x_0 \in [0,1]$ is normal in base 2 if and only if the following condition is satisfied, for all non-zero integer τ:*

$$\lim_{n\to\infty} \frac{1}{n} \log \Big[1 - \cos(2\pi\tau 2^n x_0)\Big] = 0 \tag{5.4}$$

Proof
This is a consequence of the fact that the Weyl criterion for normality is equivalent to (5.2) which itself relies on (5.1). Taking the logarithm of the complex norm in (5.2) and combining with (5.1), the convergence criterion can be rewritten as

$$\frac{1}{n}\log \Bigg|\Bigg|\prod_{k=0}^{n-1}\Big(1+z_0^{2^k}\Big)\Bigg|\Bigg|^2 = \frac{1}{n}\log\Bigg|\Bigg|\frac{1-z_0^{2^n}}{1-z_0}\Bigg|\Bigg|^2 \to 0 \text{ as } n\to\infty.$$

To conclude, note that

$$\frac{1}{n}\log\Bigg|\Bigg|\frac{1-z_0^{2^n}}{1-z_0}\Bigg|\Bigg|^2 = \frac{1}{n}\log\big|\big|1-z_0^{2^n}\big|\big|^2 - \frac{1}{n}\log\big|\big|1-z_0\big|\big|^2$$

with

$$\frac{1}{n}\log\big|\big|1-z_0\big|\big|^2 \to 0, \quad \big|\big|1-z_0^{2^n}\big|\big|^2 = 2\Big(1-\cos(2\pi i\tau 2^n x_0)\Big), \quad \frac{1}{n}\log 2 \to 0.$$

Extra care is needed when taking the n-th root or the logarithm of complex numbers as these are not uniquely defined. The details are beyond the scope of this presentation. The product in (5.2) represents the geometric mean of a set of complex numbers. ▮

A consequence is that a rational number $x_0 = a/b$ cannot be normal. To prove it, use $\tau = b$ in (5.4). Then, the limit is $-\infty$, violating the criterion.

5.1.1 High performance computing with Chebyshev polynomials

The computation of $\cos(2\pi\tau 2^n x_0)$ in formula (5.4) is not trivial when n is large, say $n = 10^5$. The problem is compounded by the fact that x_0 is irrational, for instance $x_0 = \log 2$. However, we can focus on a class of irrationals x_0 that are a lot easier to deal with. This is the case is $x_0 = (2\pi)^{-1}\arccos y_0$, with $y_0 = p/q$ a rational number and arccos the inverse cosine function also called arc cosine. That is,

$$x_0 = \frac{1}{2\pi}\arccos y_0, \quad y_0 = \cos 2\pi x_0, \quad y_0 = \frac{p}{q} \text{ with } 0 < p < q. \tag{5.5}$$

Here p, q are coprime integers with $q > 2$. Then, x_0 is still an irrational number. We then have

$$\cos(2\pi m x_0) = \cos(m \arccos y_0) = T_m(y_0), \text{ with } m = \tau 2^m. \tag{5.6}$$

Here T_m is the Chebyshev polynomial of degree m, $T_m(y_0)$ is a rational number, and (5.4) can be restated as

$$\lim_{n\to\infty}\frac{1}{n}\log\Big[1 - T_m(y_0)\Big] = 0, \text{ with } m = \tau 2^n. \tag{5.7}$$

Chebyshev polynomials satisfy the recursion $T_{m+1}(y) = 2yT_m(y) - T_{m-1}(y)$ with the initial conditions $T_0(y) = 1$ and $T_1(y) = y$. Now let $V_m(y) = T_{m-1}(y)$, with $V_0(y) = T_{-1}(y) = y$. It is easy to prove that

$$\begin{bmatrix} T_m(y) \\ V_m(y)\end{bmatrix} = A\begin{bmatrix} T_{m-1}(y) \\ V_{m-1}(y)\end{bmatrix} = A^m\begin{bmatrix} T_0(y) \\ V_0(y)\end{bmatrix} = A^m\begin{bmatrix} 1 \\ y\end{bmatrix}, \text{ with } A = \begin{bmatrix} 2y & -1 \\ 1 & 0\end{bmatrix}. \tag{5.8}$$

Another expression, not used here, is also available:

$$T_m(y) = \frac{1}{2}\Big[\Big(y + i\sqrt{1-y^2}\Big)^m + \Big(y - i\sqrt{1-y^2}\Big)^m\Big]. \tag{5.9}$$

Note that if $0 < y_0 < 1$, then $|T_m(y_0)| \leq 1$. To check the normality of x_0 using (5.7), we want to know how close $T_m(y_0)$ can get to 1 when y_0 is a rational number. Too close (for some non-zero integer τ as $n \to \infty$) implies that x_0 is not normal. The converse is true. Again, $m = \tau 2^n$. The topic of asymptotic bounds ($m \to \infty$) for $|T_m(y)|$ when $y \in [-1, 1]$ is studied in the literature [4, 30] and linked to the concept of logarithmic capacity. However, I could not find how this theory helps solve our problem.

Since $m = \tau 2^n$, there is a very efficient way to compute $T_m(y_0)$ based on formula (5.8). First, let $A_0(\tau) = A^\tau$. Then iteratively compute $A_{k+1}(\tau) = A_k^2(\tau)$. We have $A^m = A_n(\tau)$, and $T_m(y_0)$ is the first component of the bivariate vector $A^m \cdot (1, y_0)^T$. See the Python code in section 5.1.2, featuring high performance computing with the gmpy2 library.

5.1.2 Application with test of randomness and Python code

Figure 5.1 shows, for any n up to $n = 5000$, the upper and lower bounds of $\lambda_n(\tau, y_0)$ over all $\tau \in \{1, \dots, 100\}$, with n on the X axis and $y_0 = \frac{3}{5}$, based on

$$\lambda_n(\tau, y_0) = \frac{1}{n} \log \Big[1 - T_m(y_0)\Big] = \frac{1}{n} \log \Big[1 - \cos(2\pi m x_0)\Big], \text{ with } m = \tau 2^n. \tag{5.10}$$

The convergence of both curves to zero empirically shows that $x_0 = (2\pi)^{-1} \arccos \frac{3}{5}$ is normal in base 2. It also means that the binary digits of x_0 are equidistributed. This is a weak form or randomness, yet superior to what is currently implemented in standard random number generators based on rational numbers with a finite period. The upper bound (orange curve) is straightforward as $\lambda_n(\tau, y_0) \leq (\log 2)/n$ regardless of τ or y_0. The challenge is with the lower bound (blue curve) for which no asymptotic minimum (lim inf) is guaranteed.

Failure to satisfy normality happens when the cosine term in (5.10) gets too close to 1 as n gets increasingly large, that is, when the fractional part of mx_0 gets either too close to 1 or too close to 0. Therefore, the Weyl criterion for normality of x_0 in base 2 can further be simplified to

$$\lim_{n\to\infty} \frac{1}{n} \log \Big(\big|\{\tau 2^n x_0\}\big|\Big) = 0 \tag{5.11}$$

for all non-zero integer τ. The brackets $\{\cdot\}$ represent the fractional part function. It may still be impossible to verify if x_0 is a number such as π, $\sqrt{2}$ or $\log 2$. Thus our focus on numbers such as $x_0 = (2\pi)^{-1} \arccos \frac{3}{5}$.

For a stronger concept of normality, called strong normality, see chapter 4 in [13]. It better captures strong randomness. Finally, computing $T_m(y_0)$ is based on the recursion $A_{k+1}(\tau) = A_k^2(\tau)$ with $A_0(\tau) = A^\tau$ where A is the 2×2 matrix defined in formula (5.8). The determinant and spectral radius of A are both equal to 1. Thus, this is also true for any integer power of A. The recursion in question, preserving the determinant and spectral radius, corresponds to a quadratic dynamical system also called quadratic map where the state space consists of 2×2 real matrices with determinant and spectral radius equal to 1. There are strong connections to the material presented in chapter 1, where I also use a quadratic map to compute the digits of a special irrational number, using integers only and a truncation mechanism similar to that in the code below, with the same goal of assessing whether or not the binary digits look random.

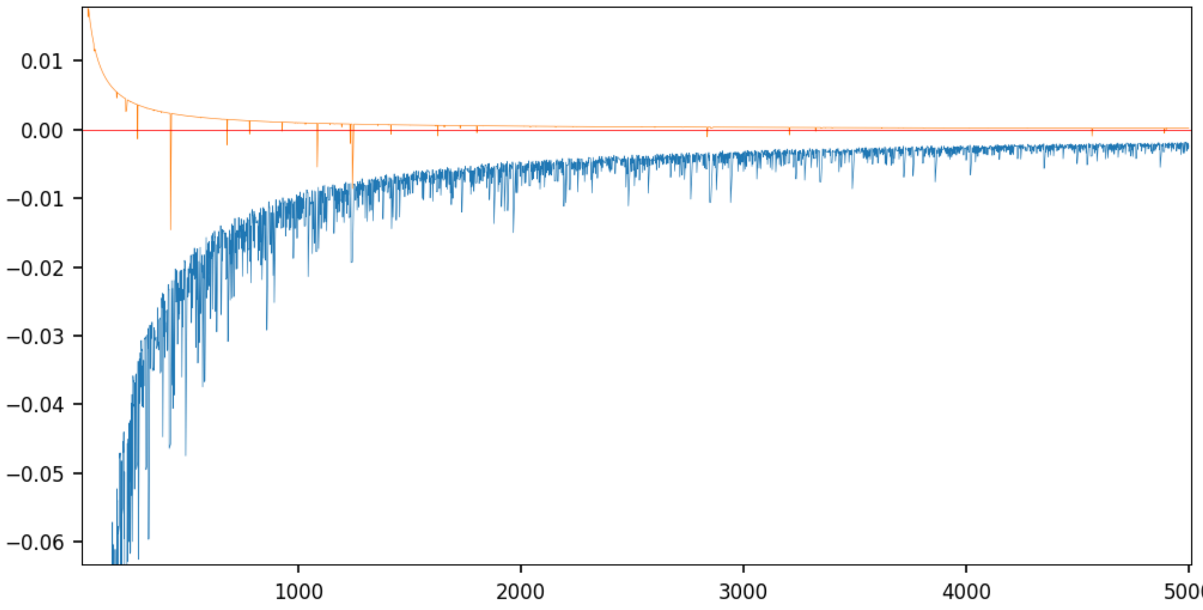

Figure 5.1: Upper and lower bounds for $\lambda_n(\tau)$ on Y axis, with n on X-axis, based on 100 values of τ

About truncation, the Python code below, despite dealing with irrational numbers, manipulates integer numbers only when `mode` is set to `'arccos'`. Yet these numbers become insanely large when n grows to 5000 and we compute A^m with $m = \tau 2^n$ and $\tau = 100$. Thus truncation is unavoidable. Yet, how do we make sure that we still preserve at least 12 digits of accuracy until the very end? In chapter 1, there is a theoretical framework that guarantees the desired precision. But not here. However, the determinant, always equal to 1, plays the role of a checksum. When the precision drops below a certain threshold, it shows in the digits of the determinant. Likewise, when testing with `mode='direct'`, $x_0 = 3/7$ and $\tau = 7$, we must have $T_m(y_0) = 1$ at all times. When this is no longer true, it means that we have precision issues and must increase the precision parameter `ctx.precision` in the code. By default, its value is 10^4 bits at the starting point ($n = 0$).

The program below named `normal_numbers5.py` is available on my Google drive, here. The rational y_0 corresponds to `y0_1` in the code. As a rule of thumb, you start with full precision in the inner loop when $n = 0$, and you lose about one digit at each iteration as n increases. This is consistent with the similar algorithm in chapter 1. So if the precision is set to 600 bits and `n_max` is set to 400, in the last iteration the precision on $T_m(y_0)$ is about $600 - 400 = 200$ bits. Also, $T_m(y_0)$ is denoted as `T_frac` in the code, with $m = \tau 2^n$ as usual.

```python
import numpy as np
import gmpy2

import matplotlib.pyplot as plt
import matplotlib as mpl

mpl.rcParams['axes.linewidth'] = 0.5
plt.rcParams['xtick.labelsize'] = 8
plt.rcParams['ytick.labelsize'] = 8
plt.rcParams['legend.fontsize'] = 'x-small'

ctx = gmpy2.get_context()
ctx.precision = 10000

mode = 'arccos' # options: 'direct', 'arccos'
y0_0 = gmpy2.mpfr(1)
if mode == 'direct':
   # x0 must be in ]-1, 1[
   # if x0 = a/b rational and tau=b, lim=0 always unless precision error
   x0 = gmpy2.mpfr(3)/gmpy2.mpfr(7) ## use trancendental number
   pi = gmpy2.const_pi()
   y0_1 = gmpy2.cos(2*pi*x0)
else:
   # 0 < p < q, both integers
   p = gmpy2.mpz(3)
   q = gmpy2.mpz(5)
   qn = gmpy2.mpz(q)
   y0_1 = gmpy2.mpfr(p/q)

A0 = [[gmpy2.mpfr(2*y0_1), gmpy2.mpfr(-1)],
     [gmpy2.mpfr(1), gmpy2.mpfr(0)]]
y0 = [y0_0, y0_1]

n_max = 5000
tau_max = 100
arr_lim = np.zeros((n_max, tau_max))
for tau in range(1,tau_max):
   A = np.linalg.matrix_power(A0, tau)
   lim = 0
   for n in range(0, n_max):
      T = np.matmul(A, y0)
      T_frac = T[0]
      # determinant must always be 1, used as checksum
      det = A[0,0]*A[1,1] - A[0,1]*A[1,0]
      if n > 0:
         #lim = (1 - T_frac)**(1/n)
         lim = gmpy2.log2(1 - T_frac)/n
      arr_lim[n, tau] = lim
      if n % 100 == 0:
         print("n: %5d tau: %3d T_frac: %12.9f lim %12.9f det %12.9f"
                % (n, tau, T_frac, lim, det))
      A = np.matmul(A, A)

xval = []
arr_min = []
arr_max = []
for n in range(n_max):
   tmin = 999999999.99
   tmax = -999999999.99
   for tau in range(1,tau_max):
      lim = arr_lim[n,tau]
      if lim < tmin:
         tmin = lim
      if lim > tmax:
         tmax = lim
   xval.append(n)
```

```
    arr_min.append(tmin)
    arr_max.append(tmax)
    print("n: %5d tmin: %12.9f tmax: %12.9f" % (n, tmin, tmax))

plt.plot(xval, arr_min, linewidth = 0.3, alpha=1)
plt.plot(xval, arr_max, linewidth = 0.3, alpha=1)
plt.axhline(y=0.0, color='r', linewidth=0.4)
plt.show()
```

5.1.3 Problem and solution

Write a version of the Python code that performs exact computations when $y_0 = p/q$ is a rational number with p, q coprime and $p < q$. In this case, $T_m(y_0)$ is also rational number, with numerator and denominator denoted respectively as p_n and q_n, with $p_n < q_n$. Again, $m = \tau 2^n$. Try $(p, q) = (3, 5), (1, 3)$ and $(1, 4)$. Look at $q_n - p_n$ and how close it can get to zero as n grows, depending on τ. If too close to zero for a specific τ and large n, then $x_0 = (2\pi)^{-1} \arccos y_0$ may not be normal. Find patterns in p_n. Note that $q_n = q^m$.

Below is my version for the code. But it only works with small values of n as the number of digits in p_n, q_n grows extremely fast when n increases, quickly eating all the available memory. Don't look at my solution until after you wrote and tested your code. Hopefully, you can write a version that works with bigger n, or a least be able to find patterns in $q_n - p_n$ even if you cannot compute all the digits. Even better, find patterns confirming that x_0 is normal in base 2, after a formal proof. My code normal_numbers.py is posted online, here.

```
import numpy as np
import gmpy2

ctx = gmpy2.get_context()
ctx.precision = 10000

# y0 = p/q
p = gmpy2.mpz(3)
q = gmpy2.mpz(5)
qn = q

# array with 'dtype=object' to store bigint
A = np.array([[2*p, -q], [q,0]], dtype=object)
y0 = np.array([q, p], dtype=object)
tau = 1
A = np.linalg.matrix_power(A, tau)
numlog = 0
denumlog = 0
delta = 0

for n in range(0, 25):
    T = np.matmul(A, y0)
    num = T[0]//q
    denum = qn**tau
    if n > 0:
        numlog = gmpy2.log(abs(num))
        denumlog = gmpy2.log(abs(denum))
    T_frac = gmpy2.mpfr(num/denum)
    if n > 0:
        delta = gmpy2.log2(1-T_frac)/n
    print("n: %5d T_frac: %12.9f delta: %12.9f numlog: %15f denumlog: %15f"
          % (n, T_frac, delta, numlog, denumlog))
    qn = qn * qn
    A = np.matmul(A, A)
```

5.2 Another interesting discrete quadratic dynamical system

Starting with $p_0 = 1$ and $q_0 = 2$, I build a sequence (p_n, q_n) with the three steps below, in that order at each iteration, based on two positive integer parameters μ, ν and an associated sequence of integers (δ_n) discussed later:

$$p^*_{n+1} = (p_n + q_n)^2 + \delta_n, \tag{5.12}$$

$$q_{n+1} = 2^\mu \cdot q_n^2, \tag{5.13}$$

$$p_{n+1} = \varphi(p^*_{n+1}, q_n, \nu), \tag{5.14}$$