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

#--- Plots

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

B.3 Random polynomials and spectral analysis of digit distributions

For a real number $\xi > 1$ with binary digits $d_k(\xi)$, $k = 0, 1, \ldots$ with $d_0(\xi) = 1$, let us define the digit generating function of complex argument z as

$$G(z, \xi) = \sum_{k=0}^{\infty} d_k(\xi) z^k, \quad d_k(\xi) \in \{0, 1\}, \quad \|z\| < 1. \tag{B.17}$$

Let $\xi_* = 2^{-\mu} G\big(\frac{1}{2}, \xi\big)$ where $\mu = \lfloor \log_2 \xi \rfloor$ is an integer power of 2. Then ξ and ξ_* have the same binary digits: ξ_* is the rescaled version of ξ. For convenience, define the polynomial G_n of degree at most n as

$$G_n(z, \xi) = \sum_{k=0}^{n} d_k(\xi) z^k \to G(z, \xi) \text{ as } n \to \infty. \tag{B.18}$$

Now for fixed n and ξ, let $\rho_k(n)$, $k = 0, 1, \ldots$, denote the roots of $G_n(z, \xi)$, sorted in descending order based on the distance $d(\|\rho_k(n)\|, 1)$. Then,

$$G_n(z, \xi) = \pm \prod_k \Big(\rho_k(n) - z\Big). \tag{B.19}$$

The sign in (B.19) is chosen so that $G_n(0, \xi) = +1$. In particular, $G_n(1, \xi)$ is the number of 1 in the first n binary digits of ξ, that is, the digit sum function associated to ξ.

B.3.1 Orbits of the digit generating function

I now define the orbit of G at radius $\rho < 1$ as the set of complex values

$$\Gamma_{\rho,\xi} = \{\, G(z, \xi) \text{ such that } ||z|| = \rho \,\}.$$

See the case $\rho = 0.80$ in Figures B.16 and B.17, respectively for $\xi = \pi$ and $\xi = \frac{24}{37}$. By contrast, the curve is a simple oval when $\xi = \frac{4}{7}$. For $\rho = 0.99$, see Figures B.14 and B.15, respectively for $\xi = \pi$ and $\xi = \frac{24}{37}$. The precise definition of the orbit at ρ is as follows. Let $z_t = \rho \cdot (\cos 2\pi t + i \sin 2\pi t)$. Then $\Gamma_{\rho,\xi}$ is the parametric curve

$$x_t = \Re\big[G(z_t, \xi)\big], \quad y_t = \Im\big[G(z_t, \xi)\big], \quad \text{with } 0 \le t \le 1. \tag{B.20}$$

The color on the orbit plots changes from blue to red as t increases from 0 to 1, with green for $t \approx 0.5$. Here $\Re, \Im$ denote respectively the real and imaginary parts of a complex number. In the code, I used the approximation G_n to G, with n set to `ndigits=1000` and with 200,000 evenly spaced values of t. As ρ gets closer to 1, you need to increase `ndigits` accordingly. For most ξ, there is no convergence when $\rho \to 1$. And unlike the Riemann zeta function rescaled by the factor $1 - 2^{1-z}$ to give rise to the Dirichlet eta function that converges on

the unit circle, there is no such generic scaling factor applicable to $G(z, \xi)$ to make it converge there. Notable exceptions are integers and dyadic rationals ξ for which $G(z, \xi)$ always exists regardless of z's modulus.

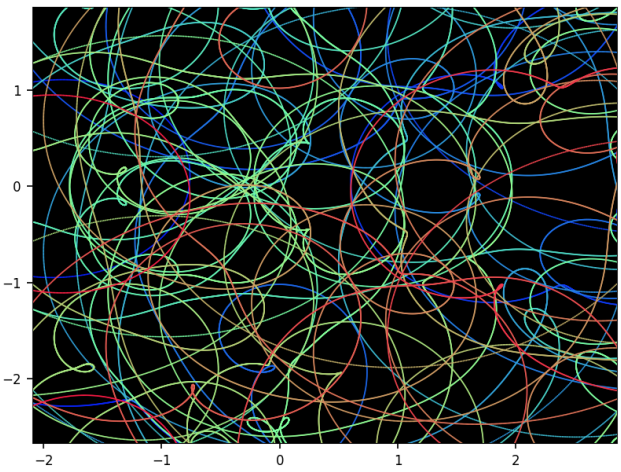

Figure B.14: $\Gamma_{\rho,\xi}$ with $\rho = 0.99$, $\xi = \pi$

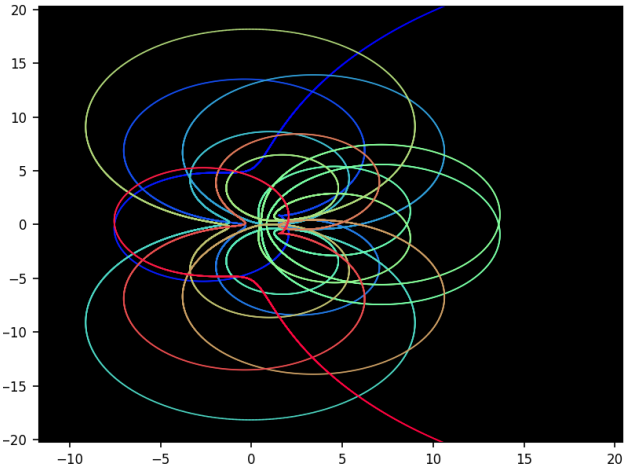

Figure B.15: $\Gamma_{\rho,\xi}$ with $\rho = 0.99$, $\xi = \frac{24}{37}$

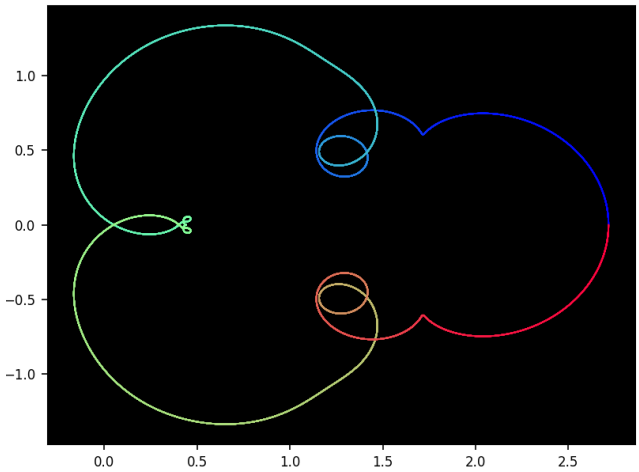

Figure B.16: $\Gamma_{\rho,\xi}$ with $\rho = 0.80$, $\xi = \pi$

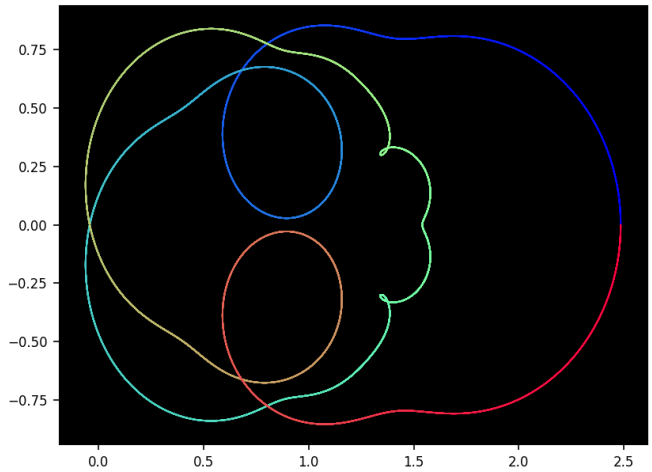

Figure B.17: $\Gamma_{\rho,\xi}$ with $\rho = 0.80$, $\xi = \frac{24}{37}$

Another way to characterize a digit distribution specific to a number ξ, is pictured in Figure B.23 representing $G(z, \xi)$ for 4 different ξ, this time with z real in $[-0.75, 0.75]$. These pictures convey more information about the digits than standard analyses, where normal numbers such as π or $\sqrt{2}$ seem indistinguishable when comparing their raw digit distributions. In particular, when ρ approaches 1, the shape of the orbit is an indicator of how scrambled the digits of ξ are.

See below the code to produce the plots just discussed. It contains special characters such as π and ξ, yet they do not cause problems when running Python. Also on GitHub, here.

```
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np
import gmpy2

ndigits = 1000
ctx = gmpy2.get_context()
ctx.precision = ndigits

mpl.rcParams['axes.linewidth'] = 0.5
plt.rcParams['xtick.labelsize'] = 8
plt.rcParams['ytick.labelsize'] = 8
plt.rcParams['legend.fontsize'] = 'x-small'
plt.figure(facecolor='black')
plt.gcf().set_facecolor("white") # outside the plot
plt.gca().set_facecolor("black") # plot area
```

```
xi1 = gmpy2.const_pi()
xi2 = gmpy2.exp(1)
xi3 = gmpy2.log(2)
xi4 = gmpy2.mpfr(24)/37
arr_xi = [(xi1, 'π'), (xi2, 'exp(1)'), (xi3, 'log(2)'), (xi4, '4/7')]

for m in range(len(arr_xi)):
   value = arr_xi[m]
   xi = value[0]
   xi = gmpy2.mpz(2**(2*ndigits) * xi)
   xi_bin = bin(xi)[2:ndigits+2]
   text = value[1]
   arr_z = []
   arr_G = []
   for x in np.arange(-0.75, 0.75, 0.001):
      sum = 0
      for k in range(ndigits):
         sum += int(xi_bin[k]) * x**k
      arr_z.append(x)
      arr_G.append(sum)
   plt.plot(arr_z, arr_G, linewidth=1.4, label = f"ξ = {text}")
plt.grid(True, color = (0.3, 0.3, 0.3))
plt.legend(loc='upper left', shadow=True)
plt.show()


#--- G(z) orbit on the unit circle

plt.gcf().set_facecolor("white") # outside the plot
plt.gca().set_facecolor("black") # plot area

xi = gmpy2.mpz(2**(2*ndigits) * xi1)
xi_bin = bin(xi)[2:ndigits+2]

## arr_z = []
arr_G_re = []
arr_G_im = []
arr_colors = []
npoints = 200000 # points generated on the circle
radius = 0.99
for t in range(npoints):
   tc = t/npoints
   color = [tc, 4*tc*(1-tc), 0.25+0.75*(1-tc)]
   z_re = radius * np.cos(2*np.pi*tc)
   z_im = radius * np.sin(2*np.pi*tc)
   if t % 1000 == 0:
      print("...",t)
   sum = 0
   for k in range(ndigits):
      sum += int(xi_bin[k]) * complex(z_re, z_im)**k
   arr_G_re.append(sum.real)
   arr_G_im.append(sum.imag)
   arr_colors.append(color)

plt.scatter(arr_G_re, arr_G_im, s=0.8, color = arr_colors, edgecolors = 'none')
plt.show()
```

B.3.2 Connection to Littlewood, Shapiro, and Newman Polynomials

Whether series (B.17) is finite or infinite, for all practical purposes, I work with a finite number of digits. The truncated version defined by (B.18) is a Newman polynomial of degree at most n: one with coefficients in $\{0, 1\}$. They are related to Littlewood polynomials with coefficients in $\{1, -1\}$, and special roots called Salem and Pisot numbers. Special examples include the Shapiro polynomials defined by a simple recursion [Wiki]. The infinite Thue-Morse sequence consisting of 0 and 1, and its generating function [Wiki], also satisfy a simple recurrence. It is used in fractal music. All these polynomials have numerous applications, in particular in signal processing, and are well studied. There are many articles about their complex roots, known to be concentrated around the unit circle. Those with little amplitude on the unit circle are called flat polynomials and involved in the study of the Riemann Hypothesis.

Much of my research focuses on the same topics: the behavior of $G_n(z, \xi)$ when $||z|| = 1$ in section B.3.1 via the orbit $\Gamma_{\rho,\xi}$. And then its roots in section B.3.3, to establish a link to normal numbers. Building a set of roots

that leads to random-like digits $d_k(\xi)$ with a known function $G_\infty(z, \xi)$, would be a spectacular accomplishment, and the reverse approach to proving that some known constant ξ is normal. However, it seems out of reach. Even finding a Taylor series with bounded integer coefficients, representing a well-known non-rational function that can be evaluated in closed form at $z = \frac{1}{2}$, seems impossible.

I also looked at what happens when you replace the binary digits 0 and 1 in (B.18) by two arbitrary values not necessarily integer or rational, and most of the properties I am interested in are preserved. An interesting example with digits in $\{-1, 0, 1\}$ showing up in uneven proportions, is the Möbius function [Wiki]. Finally, my techniques to compare rational and irrational numbers, offer an alternative to the irrationality measure [Wiki].

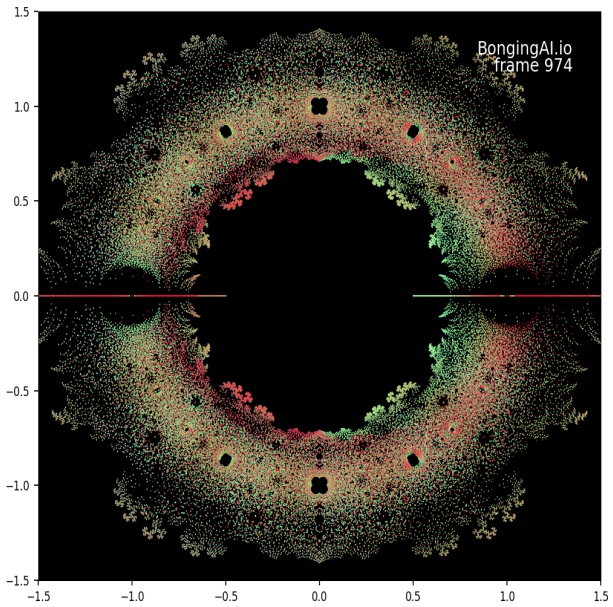

Figure B.18: Roots of all Littlewood polynomials of degree $n \leq 14$

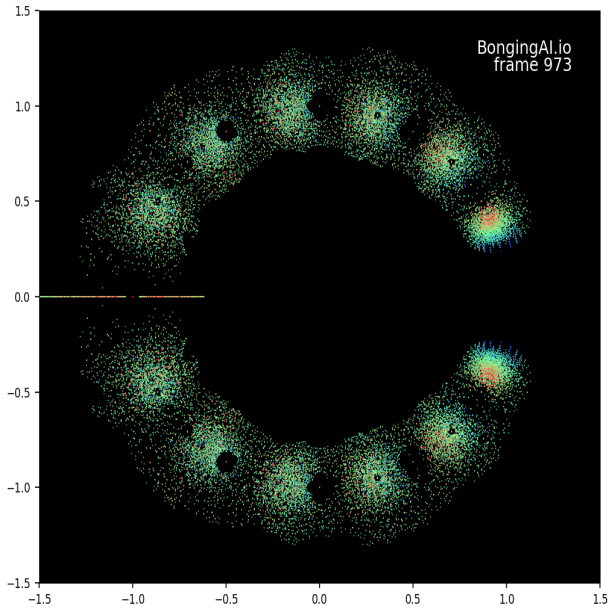

Figure B.19: Roots of sampled Newman polynomials of degree $n = 14$

B.3.3 Spectral signature of even versus uneven bit strings

Roots of random polynomials have been well studied for decades. Our interest is when the coefficients $d_k(\xi)$ in the polynomial $G_n(z, \xi)$ are distributed as i.id. Bernoulli variables on $\{0, 1\}$ with parameter p. Here $0 < p < 1$ represents the proportion of 1, and the polynomial has degree n: the first and last digits are 1, the other ones are random. The coefficients in question are the binary digits of ξ. We are concerned with the case $n \to \infty$.

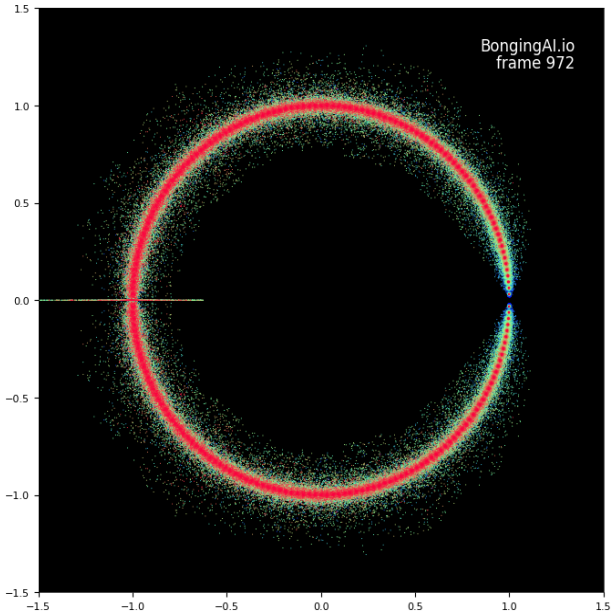

Figure B.20: Roots of $G_n(z, \xi)$ for 2000 ξ's. p ranges from 0 (blue) to 1 (red); $n = 200$.

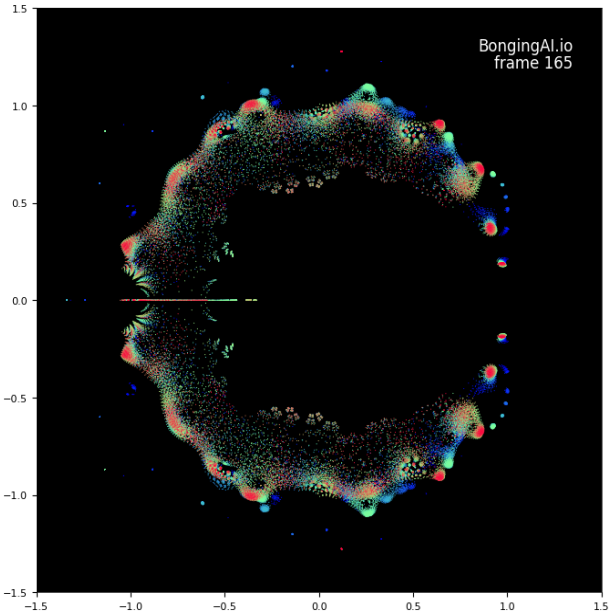

Figure B.21: Roots of $G_n(z, \xi)$ for 2000 ξ's based on params in frame #165, $n = 35$.

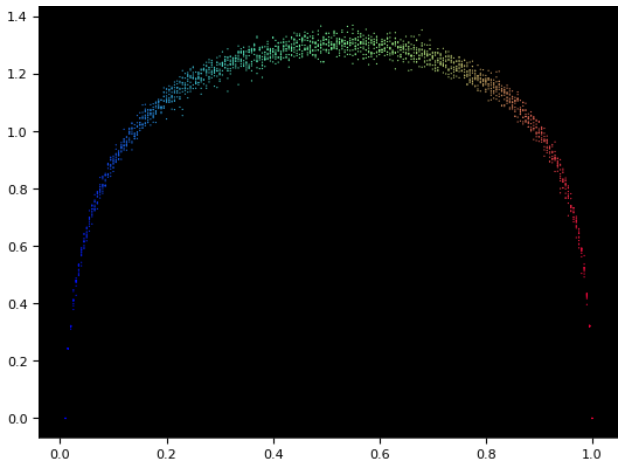

Figure B.22: $\Delta_\xi(p)$ with p on the X-axis, showing a peak when the proportion of 1 is around $p = \frac{1}{2}$.

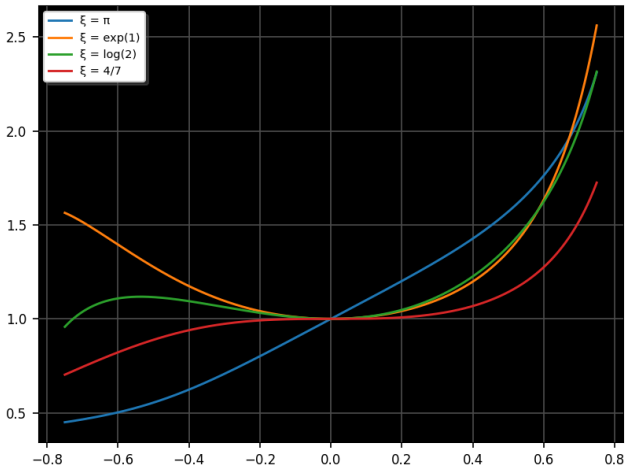

Figure B.23: $G(z, \xi)$ with $-0.75 < z < 0.75$, for $\xi = \pi, e, \log 2$ and $4/7$, and z on X-axis

The idea is as follows: if ξ is not a normal number in base 2, more specifically if $p \neq \frac{1}{2}$ and departure from normality is strong enough to be detectable in the roots, then it should show up in the root distribution of the digit generating function $G(z, \xi)$ and its polynomial approximation $G_n(z, \xi)$. The literature on random polynomials focuses on coefficients – the binary digits in our case – distributed as Gaussian with zero mean or Bernoulli(p) on $\{-1, 1\}$, but rarely on $\{0, 1\}$. Many results are distribution-independent but some assume that the coefficients have zero expectation. When the expected proportion of 1 is $p = \frac{1}{2}$, the following is known:

- At the limit as $n \to \infty$, the complex roots are uniformly distributed on the unit circle, see [28, 42, 44].
- The proportion of real roots (among all n roots) tends to zero as $n \to \infty$, see [50].
- Complex roots repel each other: the chance of a multiple root is zero, see [31].

Based on simulations (no proof), I show that the situation is much different when p is close to 0 or 1. It provides a new mechanism to test whether a number ξ satisfies some of the requirements for normality in base 2. There are many potential indicators in the root distribution to flag non-normality. The one I chose as the most striking is the scaled average absolute distance

$$\Delta_\xi = \lim_{n\to\infty} \frac{\lambda(n)}{n} \sum_{k=0}^{n-1} \Big|1 - ||\rho_k(n)||\Big| \quad \text{with } \lambda(n) = \sqrt{n}\log n, \tag{B.21}$$

computed on the n roots $\rho_0(n), \ldots, \rho_{n-1}(n)$ in (B.19) and looking at the limit $n \to \infty$. Here $||\cdot||$ represents the modulus of a complex number. Also, the scaling factor $\lambda(n)$ counteracts the fact that the roots drift towards the unit circle as n increases. While leading to a correct conclusion in all my tests, it is still subject to refinement and theoretical investigation. Now I visually show the link between the roots of $G(z, \xi)$ and the proportion of 1 in the digits of the number ξ. The associated Python code is in section B.3.4.

- For a random number ξ generated with a proportion p of 1 in its binary digit expansion, let's replace the notation Δ_ξ by $\Delta_\xi(p)$ to put the emphasis on p. Figure B.22 shows $\Delta_\xi(p)$ with p on the X-axis. I tested 2000 values of p evenly spaced between 0 and 1, with one random ξ for each p (that is, one ξ with a proportion p of 1). Each ξ has n digits, thus the degree of the polynomials is n, here with $n = 200$ specified by the parameter `N` in the code. Clearly, $\Delta_\xi(p)$ is maximum when $p = \frac{1}{2}$. Thus Δ_ξ is an indicator of normality (or lack of, to be precise). In other words, the roots behave very differently if too few or too many polynomial coefficients are 1.
- Figure B.20 shows the combined roots of all the K polynomials for the same set of ξ with the same degree n. The number of polynomials is set to `K=2000` in the code. Thus, there is a total of $n \times K$ roots in the plot, each represented by a pixel. The few real roots are on the horizontal segment. The color ranges from red when the proportion p of 1 is close to 1, to blue when p is close to 0. Green corresponds to middle values closer to $p = \frac{1}{2}$. Green roots are more spread out compared to red ones, thus with a higher $\Delta_\xi(p)$ in Figure B.22. Blue roots are more concentrated near $(1, 0)$ while red roots are distributed in a narrow annulus close to the unit circle. Thus, both blue and red roots have a lower $\Delta_\xi(p)$ compared to the green ones, reflected in Figure B.22.

The roots of Newman polynomials of degree n mostly lie in n small discs with centers evenly spread across the unit circle. This well-known phenomenon (roots repelling each other) is visible when $n \leq 20$. You can

see it in Figure B.19. By contrast, the classic representation [Wiki] lacks this feature because it shows all polynomials instead of sampled ones. Figure B.18 is a replica of the iconic Wikipedia picture. The color shows the corresponding ξ. Because Figure B.19 is based on coefficients in $\{0,1\}$ instead of $\{1,-1\}$, it misses the disc at $(1,0)$ and the positive real roots found in Figure B.18. To produce Figure B.18, set `mode='full'` in the code along with `K=2**N` (2 at power `N`), and finally `a=2` and `b=-1` if you want the coefficients to be in $\{1,-1\}$ rather than $\{0,1\}$. The code is listed in section B.3.4.

B.3.4 Mesmerizing video featuring 1000 cases, with Python code

Figures B.18, B.19, B.20, B.21 featuring the roots and Figure B.22 for the spectrum are produced by the code in section B.3.4 (this section) while Figures B.14, B.15, B.16, B.17 and B.23 linked to the orbits are generated by the code in section B.3.1. The code in this section also produces a spectacular video with 975 frames numbered from 0 to 974. Each frame shows a root configuration for a specific set of ξ. Also the binary digits of ξ are the coefficients of the associated polynomial. You can watch the video on YouTube, here.

The special frames 974, 973 and 972 are displayed respectively in Figures B.18, B.19, B.20 and discussed in section B.3.2. All the other ones are based on setting `mode='deterministic'`, including frame 165 shown in Figure B.21. Each one features the combined roots of K polynomials $G_n(z,\xi_1),\ldots,G_n(z,\xi_K)$ with $\xi_k = f(k;\theta)$. In the code, the function f is denoted as `f_bin` and the bivariate parameter $\theta=(\theta_1,\theta_2)$ is denoted as `options`; $f(k,\theta)$ returns the first n binary digits of ξ_k, which are also the coefficients of $G_n(z,\xi_k)$. Finally, ξ_k is built as follow:

- First, θ_1 allows you to select a number $\xi \in \{\pi, e, \log 2, \frac{4}{7}\}$. For instance, if $\theta_1 = 0$ then $\xi = \pi$.
- Then, $\xi_k = \xi + k$, $\xi + k^2$, $\xi + \frac{1}{k}$ or ξk depending on whether $\theta_2 = 0, 1, 2$ or 3.

The digits of ξ and ξ_k are quite similar but shifted to the left of right with some perturbation at the beginning or at the end depending on the operation, except for the multiplication $\xi_k = \xi k$ which introduces significant scrambling. As a result, in a given video frame, the roots of $G_n(z,\xi_k)$ for $k = 1,\ldots,K$ are somewhat correlated, producing a spectacular distribution as in Figure B.21. It consists of a fractal-like border loosely following the unit circle, with most roots peppered either inward or outward depending on θ_2. Unless $\theta_2 = 3$ ($\xi_k = k\xi$) where the scrambling produces a dull image.

./...	k	p	Checksum	Digits of ξ_k
0.050	100	0.543	1.00000	11001001011000101110010000101111111
0.100	200	0.514	1.00000	11001000101100010111001000010111111
0.150	300	0.514	1.00000	10010110010110001011100100001011111
0.200	400	0.486	1.00000	11001000010110001011100100001011111
0.250	500	0.571	1.00000	11111010010110001011100100001011111
0.300	600	0.486	1.00000	10010110001011000101110010000101111
0.350	700	0.543	1.00000	10101111001011000101110010000101111
0.400	800	0.457	1.00000	11001000001011000101110010000101111
0.450	900	0.486	1.00000	11100001001011000101110010000101111
0.500	1000	0.543	1.00000	11111010001011000101110010000101111
0.550	1100	0.457	1.00000	10001001100101100010111001000010111
0.600	1200	0.457	1.00000	10010110000101100010111001000010111
0.650	1300	0.457	1.00000	10100010100101100010111001000010111
0.700	1400	0.514	1.00000	10101111000101100010111001000010111
0.750	1500	0.543	1.00000	10111011100101100010111001000010111
0.800	1600	0.429	1.00000	11001000000101100010111001000010111
0.850	1700	0.486	1.00000	11010100100101100010111001000010111
0.900	1800	0.457	1.00000	11100001000101100010111001000010111
0.950	1900	0.543	1.00000	11101101100101100010111001000010111

Table B.3: Prop. p of 1 in digits of 20 sampled ξ_k, video frame 165

The colors range from blue ($k=0$) to red ($k=K$) with green for middle values. The fact that each frame has areas with strongly distinct colors – as opposed to colors randomly distributed – means that close polynomials, that is $G_n(z,\xi_k)$ and $G_n(z,\xi_l)$ with $|k-l|$ small, have at least some roots that are also close. Again, this is not true if $\theta_2 = 3$; in that case, index proximity does not translate into coefficients proximity due to the scrambling induced by the multiplication operator.

Besides the `f_bin` options θ_1, θ_2 applicable only to `mode='deterministic'`, there are two other parameters a, b available in all modes. The default is $a = 1, b = 0$. Any other values, integer or not, positive or not, change the binary digit $d \in \{0,1\}$ into $d^* = ad + b \in \{a+b, b\}$. Or equivalently, it changes the polynomial

$G_n(z,\xi)$ with coefficients $d_k(\xi), k = 0,\ldots,n$ into the polynomial $G_n^*(z,\xi)$ with coefficients $d_k^*(\xi) = ad_k(\xi) + b$. The following result is straightforward:

$$G_n^*(z,\xi) = a\,G_n(z,\xi) + b(1 + z + \cdots + z^n). \tag{B.22}$$

Roots of $G_n(z,\xi)$ and $G_n^*(z,\xi)$ are connected as follows. Let u, v be two roots of $G_n(z,\xi)$, and u_*, v_* two roots of $G_n^*(z,\xi)$. Then, for any fixed ξ, by virtue of (B.22) we have

$$\frac{G_n^*(u,\xi)}{1+u+\cdots+u^n} = \frac{-a\,G_n(u_*,\xi)}{1+u_*+\cdots+u_*^n} \tag{B.23}$$

$$\frac{G_n^*(u,\xi)}{1+u+\cdots+u^n} = \frac{G_n^*(v,\xi)}{1+v+\cdots+v^n} \tag{B.24}$$

$$\frac{G_n(u_*,\xi)}{1+u_*+\cdots+u_*^n} = \frac{G_n(v_*,\xi)}{1+v_*+\cdots+v_*^n}. \tag{B.25}$$

Note that (B.23) does not depend on b. Then, the formulas (B.24) and (B.25) do not depend on either a or b, leading to universal rather than model-specific properties.

The Python code displays a statistical summary after each generated frame. It consists of the parameters $N, K, a, b, \theta_1, \theta_2$ attached to the frame in question, as well as Table B.3, in this example for frame 165 shown in Figure B.21. The `checksum` value should always be 1 unless the polynomial coefficient attached to z^n is zero. A value not close enough to 1 indicates lack of accuracy when computing the roots. To fix it, increase the precision or check other parameters (number of steps and so on) in the root–finding algorithm `polynomial.polyroots` from the Numpy library. With my parameter set, none of the generated frames faces this problem. To speed up the computations, process the polynomials in parallel. Also, the code is designed to resume without losing previously generated images if it crashes at any time for any reason. All the results and illustrations are fully replicable. The code is also on GitHub, here.

This research is part of a project at BondingAI.io to build an AI agent specializing in scientific and high-performance computing, as well as AI–generated art. Also, one of the goals is to share beautiful mathematical results with the public at large, thanks to the video and the connection to polynomial roots. The latter is familiar to most high school students but the classroom material is usually dry and lacks exciting experiments. I hope that my contribution fills this gap and will boost the interest in mathematics, probability, and engineering, as these fields are typically considered as unwelcoming by many.

Finally, my work raises an interesting question: when you automatically generate thousands of examples (frames here,) can AI select which ones are the most beautiful and enriching, to include them in the final video? The answer is yes if you train AI, but can it do it well with little training and detect on its own a set of diversified images that are universally considered spectacular or enlightening by everyone? Finally, if you are interested in adding an AI-generated mathematical soundtrack (music) to my video, matching the frames, please contact me at vincent@bondingAI.io.

```
import random
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'

def extract_roots(N, xbin, a, b):

    arr_re = []
    arr_im = []
    delta = 0
    prop = xbin.count('1')/N
    arr_coeffs = []
    for k in range(N):
       digit = int(xbin[k])
       arr_coeffs.append(a*digit+b)
    # coefficients in that order: 1 + 3x + 5x^2 --> [1, 3, 5]
    roots = np.polynomial.polynomial.polyroots(arr_coeffs)

    checksum = 1
    for r in roots:
       re = r.real
```

```
        im = r.imag
        norm = (re**2 + im**2)**0.5
        arr_re.append(re)
        arr_im.append(im)
        delta += abs(norm-1)
        checksum *= norm
    if len(roots) == 0:
        delta = 0
    else:
        delta /= len(roots)
    return(arr_re, arr_im, delta, checksum)


def generate_bit_string(N, prop):

    num_ones = int(0.5 + (N-2) * prop)
    num_zeros = (N-2) - num_ones
    bits = ['1'] * num_ones + ['0'] * num_zeros
    random.shuffle(bits)
    xbin = "".join(bits)
    xbin = '1' + xbin + '1'
    return(xbin)


def f_bin(k, N, options):

    constant = options[0]
    formula = options[1]

    if constant == 0:
        cst = gmpy2.const_pi()
    elif constant == 1:
        cst = gmpy2.exp(1)
    elif constant == 2:
        cst = gmpy2.log(2)
    elif constant == 3:
        cst = gmpy2.mpfr(4)/gmpy2.mpfr(7)

    if formula == 0:
        eta = cst + k
    elif formula == 1:
        eta = cst + k*k
    elif formula == 2:
        eta = cst + gmpy2.mpfr(1)/k
    elif formula == 3:
        eta = cst * k

    x = gmpy2.mpz(2**(2*ndigits) * eta)
    xbin = bin(x)[2:N+2]
    return(xbin)


#--- Core function

def set_of_polynomials(K, N, a, b, mode, f_options):

    arr_delta = []
    arr_prop = []
    arr_col = []

    for k in np.arange(1, K):

        p = k/K
        color = [p, 4*p*(1-p), 0.25+0.75*(1-p)]
        if mode == 'random':
            xbin = generate_bit_string(N, p)
        elif mode == 'deterministic':
            xbin = f_bin(k, N, f_options)
        elif mode == 'full':
            string = bin(k)[2:]
            xbin = string + "0" * (N - len(string))

        p1 = xbin.count('1')/N # should be close to prop
        arr_re, arr_im, delta, checksum = extract_roots(N, xbin, a, b)

        # checksum (product of all roots) must be equal to 1
```

```
104         if k % 100 == 0:
105            string = xbin
106            if len(string) > 60:
107               string = string[:60] + "..."
108            print("%5.3f %5d %5.3f %9.7f %s" % (p, k, p1, checksum, string))
109
110         arr_prop.append(p1)
111         arr_delta.append(np.sqrt(N)*np.log(N)*delta)
112         arr_col.append(color)
113         ax.scatter(arr_re, arr_im, s=0.6, color=color, edgecolors='none')
114
115      return(arr_prop, arr_delta, arr_col)
116
117
118   #--- Main
119
120   width_px, height_px = 800, 800
121   dpi = 100
122   width_in = width_px / dpi
123   height_in = height_px / dpi
124   random.seed(42)
125
126   show_axes = False
127   show_spectrum = False
128   save_spectrum = True
129   show_roots = False
130   save_roots = True
131   saved_frames = []
132
133   """
134   param entry: [N, K, a, b, mode, frame_ID, f_options]
135
136       • N: number of digits (polynomial degree < N)
137       • K: number of polynomials used in a set or video frame
138       • a, b: binary digit replaced by a*digit+b
139       • mode: 'deterministic' or 'random'
140       • frame_ID: index number attached to video frame
141       • f_options: parameters (u, v) used in function f_bin:
142          • u: 0 for Pi, 1 for exp(1), 2 for log(2), 3 for 4/7
143          • v: 0 for u+k, 1 for u+k*k, 2 for u+(1/k), 3 for u*k,
144   """
145
146   params = []
147   frame_ID = 0
148   mode = 'deterministic'
149   K = 2000
150   a = 4
151
152   for b in range(-7, 2, 1):
153      for u in range(0, 4):
154         for v in range(0, 3):
155            for N in range(20, 61, 5):
156               f_options = (u, v)
157               param = [N, K, a, b, mode, frame_ID, f_options]
158               params.append(param)
159               frame_ID += 1
160
161   N = 200
162   K = 2000
163   param = [N, K, 1, 0, 'random', frame_ID, (0,0)]
164   params.append(param)
165   frame_ID += 1
166
167   N = 14
168   K = 12000
169   param = [N, K, 1, 0, 'random', frame_ID, (0,0)]
170   params.append(param)
171   frame_ID+=1
172
173   N = 14
174   K = 2**N
175   param = [N, K, 2, -1, 'full', frame_ID, (0,0)]
176   params.append(param)
177   frame_ID+=1
178
179   nframes = len(params)
```

```
print(nframes, "frames")

for param in params:

   N = param[0]
   K = param[1]
   a = param[2]
   b = param[3]
   mode = param[4]
   frame_ID = param[5]
   f_options = param[6]

   # ndigits (precision) must be larger than N
   ndigits = 5*N // 4
   ctx = gmpy2.get_context()
   ctx.precision = ndigits

   if frame_ID in (0, 165, nframes-3, nframes-2, nframes-1):

      print("\n----------------\nFrame:", frame_ID, "/", nframes)
      print("\ndegree: N = %d\nnumber of ξ's: K = %d\na: %7.4f\nb: %7.4f\nmode: %s\nf_options: %s"
          %(N, K, a, b, mode,str(f_options)))
      print("\n./... k  p   checksum digits of ξ_k")
      if not show_axes:
         plt.style.use('dark_background')
      fig, ax = plt.subplots(figsize=(width_in, height_in))
      arr_prop, arr_delta, arr_col = set_of_polynomials(K, N, a, b, mode, f_options)
      ax.text(0.95, 0.95, 'BongingAI.io', transform=ax.transAxes, ha='right', va='top',
          fontsize=12, color='white')
      ax.text(0.95, 0.92, 'frame ' + str(frame_ID), transform=ax.transAxes, ha='right', va='top',
          fontsize=12, color='white')

      plt.xlim(-1.5, 1.5)
      plt.ylim(-1.5, 1.5)
      if not show_axes:
         plt.axis('off')
         plt.margins(0)
         plt.subplots_adjust(top=1, bottom=0, right=1, left=0, hspace=0, wspace=0)
      else:
         ax.set_facecolor('black')
      if save_roots:
         filename = 'nroots' + str(frame_ID) + '.png'
         plt.savefig(filename, bbox_inches='tight', pad_inches=0, dpi=dpi)
         saved_frames.append(filename)
      if show_roots:
         plt.show()
      else:
         plt.close()

      plt.rcParams['axes.facecolor'] = 'black'
      plt.scatter(arr_prop, arr_delta, s=0.8, c = arr_col, edgecolors = 'none')
      if save_spectrum:
         filename = 'nroots_spectrum' + str(frame_ID) + '.png'
         plt.savefig(filename, bbox_inches='tight', pad_inches=0, dpi=dpi)
      if show_spectrum:
         plt.show()
      else:
         plt.close()


#--- Produce video

import moviepy.video.io.ImageSequenceClip
clip = moviepy.video.io.ImageSequenceClip.ImageSequenceClip(saved_frames, fps=4)
clip.write_videofile('nroots_v1.mp4')
```

Bibliography

[1] Franklin T. Adams-Watters and Frank Ruskey. Generating functions for the digital sum and other digit counting sequences. *Journal of Integer Sequences*, 12:1–9, 2009. [Link]. 12, 14, 23, 32, 45

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

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

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

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

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

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

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

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

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

[11] Faiza Firdousi, Syeda Iram Batool, and Muhammad Amin. A novel construction scheme for nonlinear component based on quantum map. *International Journal of Theoretical Physics*, 58:3871–3898, 2019. [Link]. 12, 23, 32, 45

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[28] C. P. Hughes and A. Nikeghbali. The zeros of random polynomials cluster uniformly near the unit circle. *Compositio Mathematica*, 144:734–746, 2008. [Link]. 118

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

[30] 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

[31] Marcus Michelen and Oren Yakir. Limit law for root separation in random polynomials. *Preprint*, pages 1–77, 2025. arXiv:2505.02723 [Link]. 118

[32] 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

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

[34] 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

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

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

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

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

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

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

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

[42] Terence Tao. Limit law for root separation in random polynomials. *Tao's blog*, 2013. [Link]. 118

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

[44] Van Vu Terence Tao. Limit law for root separation in random polynomials. *Preprint*, pages 1–56, 2014. arXiv:1307.4357 [Link]. 118

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

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

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

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

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

[50] Nhan D. V. Nguyen Yen Q. Do. Limit law for root separation in random polynomials. *Electronic Journal of Probability*, 30:1–37, 2025. [Link]. 118

[51] 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