Weird Random Walks: Synthetizing, Testing and Leveraging Quasi-randomness

Vincent Granville, Ph.D.
vincentg@MLTechniques.com
www.MLTechniques.com
Version 1.0, August 2022

Abstract

This is a follow-up to my article "Detecting Subtle Departures from Randomness", where I introduced the prime test to identify very weak violations of various laws of large numbers. Pseudo-random sequences failing this test usually pass most test batteries, yet are unsuitable for a number of applications, such as security, strong cryptography, or intensive simulations. The purpose here is to build such sequences with very low, slow-building, long-range dependencies, but that otherwise appear as random as pure noise. They are useful not only for testing and benchmarking tests of randomness, but also in their own right to model almost random systems, such as stock market prices. I introduce new categories of random walks (or quasi-Brownian motions subject to constraints), and discuss the peculiarities of each category. For completeness, I included related stochastic processes discussed in some of my previous articles, for instance integrated and 2D clustered Brownian motions. All the processes investigated here are drift-free and symmetric, yet not perfectly random. They all start at zero.

Contents

1 Symmetric unbiased constrained random walks

The standard symmetric 1D random walk [Wiki] fundamental to this article is a sequence $\{S_n\}$ with $n \geq 0$, starting at $S_0 = 0$, and recursively defined by $S_n = X_n + S_{n-1}$, for $n > 0$. Here X_1, X_2 and so on are independent random variables with $P[X_n = 1] = P[X_n = -1] = \frac{1}{2}$. Thus $\{S_n\}$ is a time-discrete stochastic process, and indeed the most basic one. In sections 1.2 and 1.3, I drop the assumption of independence, leading to modified random walks such as those described in [6, 12]. More general references include [1, 11].

With proper rescaling, a random walk becomes a time-continuous stochastic process S_t called Brownian motion [Wiki], with $t \in \mathbb{R}^+$. See the time series in gray in Figure 1, displaying a particular instance: it shows the first 50,000 values of S_n in a short window, giving the appearance of a Brownian motion. By contrast, each of the orange, red and gray time series represents one instance of a specific type of non-Brownian motion. Sections 1.2 and 1.3 focuses on these three types of processes, which are quasi, but not fully random.

1.1 Three fundamental properties of pure random walks

The standard random walk $\{S_n\}$ (illustrated in gray in Figure 1) is the base or reference process, used to build more sophisticated models. It has too many properties to list in this short article. However, the following are

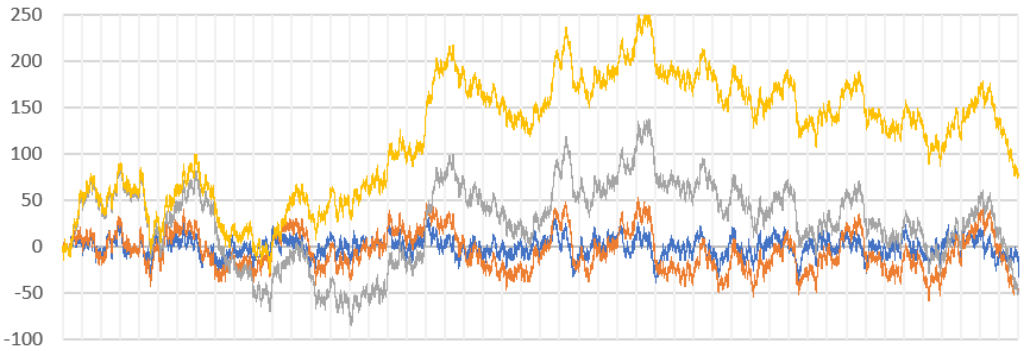

Figure 1: Typical path S_n with $0 \leq n \leq 50{,}000$ for four types of random walks

the most relevant to our discussion.

- Law of the iterated logarithm [Wiki]. In our context, it is stated as follows:

$$\limsup \frac{|S_n|}{\sqrt{2nv \log\log n}} = 1 \quad \text{as } n \to \infty. \tag{1}$$

  Here, as per the Hartman–Wintner theorem [Wiki], $v = \text{Var}[X_1] = 1$. See [11] pages 118–123 for a version adapted to Brownian motions.

- Expected number of zero crossings in $S_1, \ldots, S_n$, denoted as N_n. Here a zero-crossing is an index $0 < k \leq n$ such that $S_k = 0$. For $n > 0$, we have (see here):

$$\text{E}[N_{2n}] = -1 + \frac{2n+1}{4^n}\binom{2n}{n} \sim \frac{2}{\sqrt{\pi}} \cdot \sqrt{n} \quad \text{as } n \to \infty.$$

- Distribution of first hitting time to zero [Wiki], or first zero crossing after $S_0 = 0$, also called time of first return. The random variable in question is denoted as T. It is defined as follows: $T = n$ (with $n > 0$) if and only if $S_n = 0$ and $S_k \neq 0$ if $0 < k < n$. We have $P[T = n] = 0$ if n is odd, and $\text{E}[T] = \text{Var}[T] = \infty$. Yet, our random walks cross the X-axis infinitely many times. We also have the following probability generating function [Wiki] (see here):

$$\sum_{n=1}^{\infty} (2x)^{2n} P[T = 2n] = 1 - \sqrt{1 - 4x^2} \quad \text{if } x \leq \frac{1}{4}.$$

  From there, one can obtain

$$P[T = 2n] = \frac{1}{(2n-1)4^n}\binom{2n}{n} \sim \frac{1}{\sqrt{4\pi}} \cdot n^{-3/2} \quad \text{as } n \to \infty,$$
$$\text{E}[T^{-1}] = \int_0^{1/2} \frac{1 - \sqrt{1 - 4x^2}}{x} dx = 1 - \log 2.$$

Note that $\text{E}[T^{-1}]$ is finite, while $\text{E}[T]$ is infinite. The fact that $\text{E}[T] = \infty$ explains why the sequence S_n can stay above or below the X-axis for incredibly long time periods, as shown in Figure 1 for the gray curve.

The above three statistics $|S_n|/\sqrt{2n \log\log n}$, N_{2n} and T^{-1}can be used to design tests of randomness for pseudo-random number generators. Indeed, the prime test in [7] relies on a number theoretic version of the law of the iterated logarithm (LIL). The purpose is to detect very weak departures from randomness, even in sequences that are random enough to pass the classic LIL test, yet not fully random. In this article, the goal is to simulate quasi random sequences, rather than creating new tests of randomness.

I now describe special types of modified random walks that lack true independence in the sequence $\{X_n\}$. In particular, I discuss why they are special and of great interest, with a focus on applications.

1.2 Random walks with more entropy than pure random signal

One way to introduce dependencies in the sequences is to increase the frequency of oscillations (and thus the entropy) in the gray curve in Figure 1. The gray curve represents a realization of a pure random walk. To achieve this goal, you may want the sequence to violate the law of the iterated logarithm: you want to build a

sequence that would satisfy a modified law of the iterated logarithm with $\sqrt{2n \log \log n}$ in Formula (1) replaced by (say) $n^{2/5}$.

To accomplish this, you need to add constraints when simulating the sequence in question. Yet you want to preserve quasi randomness: the absence of drifts and auto-correlations in the sequence $\{X_n\}$, even though there is some modest lack of independence. So modest indeed that most statistical tests would fail to catch it, even though it can made highly visible to the naked eye: see the red, and especially the blue curve in Figure 1.

1.2.1 Applications

Such sequences can be used to generate synthetic data (see [9]) or to model barely constrained stochastic processes, such as stock price fluctuations in an almost perfect market. See also [4]. Another application is to introduce an undetectable backdoor in some encryption systems without third parties (government or hackers) being able to notice it, depending on the strength of the dependencies. This type of backdoor can help the encryption company decrypt a message when requested by a legitimate user who lost his key, even though the encryption company has no standard mechanism to store or retrieve keys (precisely to avoid government interference).

This assumes that there is a mapping between the type of weak dependencies introduced in a specific sequence, and the type of algorithm (or the key) used to decrypt the sequence in question. The mapping can be made too loose for full decryption even by the parent company, but helpful to retrieve partial data, such as where the sequence originates from: in this case, the type of dependencies is a proxy for a signature. All that is needed is to add some extra bits so that the sequence has the desired statistical behavior.

Ironically, you need a very good, industrial-grade pseudo-random number generator (PRNG) to generate almost perfectly random sequences. PRNG's that are not good enough – such as the Mersenne twister – may introduce irregularities that can interfere with the ones you want to introduce. This is discussed in detail in my article on PRNG's [7].

1.2.2 Algorithm to generate quasi-random sequences

One way to generate such sequences is as follows:

$S = 0$
For $n = 1, 2, \ldots$
  Generate random deviate U on $[0, 1]$
  $M = g(n)$
  If $(S < -M$ and $U < \frac{1}{2} - \epsilon)$ or $(S > M$ and $U < \frac{1}{2} + \epsilon)$ or $(|S| \leq M$ and $U < \frac{1}{2})$
  Then
    $X_n = -1$
  Else
    $X_n = 1$
  $S = S + X_n$
  $S_n = S$

Here $0 < \epsilon < \frac{1}{2}$ and $\alpha > 0$. The function $g(n)$ is positive and growing more slowly than $\sqrt{n}$. Typically, $g(n) = \alpha n^\beta$ with $0 \leq \beta \leq \frac{1}{2}$, or $g(n) = \alpha (\log n)^\beta$ with $\beta \geq 0$. The Python code in section 3.2 performs this simulation: choose the option `deviations='Small'`. You can customize the function $g(n)$, denoted as `G` in the code. The option `mode='Power'` corresponds to $g(n) = \alpha n^\beta$, while `mode='Log'` corresponds to $g(n) = \alpha (\log n)^\beta$.

Results are displayed in Figure 1. The color scheme is as follows:

- Gray curve: $\epsilon = 0$, corresponding to a pure random walk.
- Blue curve: $g(n) = \log n$, $\epsilon = 0.05$.
- Red curve: $g(n) = n^\beta$ with $\beta = 0.35$, $\epsilon = 0.05$.

The yellow curve represents a very different type of process, discussed in section 1.3.

1.2.3 Variance of the modified random walk

The symmetric nature of the modified random walk $\{S_n\}$ defined in section 1.2.2 results in several identities. Let $p_n(m) = P(S_n = m)$, with $-n \leq m \leq n$. Also, let $S_0 = 0$ and $p_0(0) = 1$. Then $p_n(m)$ can be recursively computed using some modified version of the Pascal triangle recursion:

$$p_{n+1}(m) = \Big[\frac{1}{2} + \epsilon \cdot A_n(m-1)\Big] p_n(m-1) + \Big[\frac{1}{2} - \epsilon \cdot A_n(m+1)\Big] p_n(m+1), \tag{2}$$

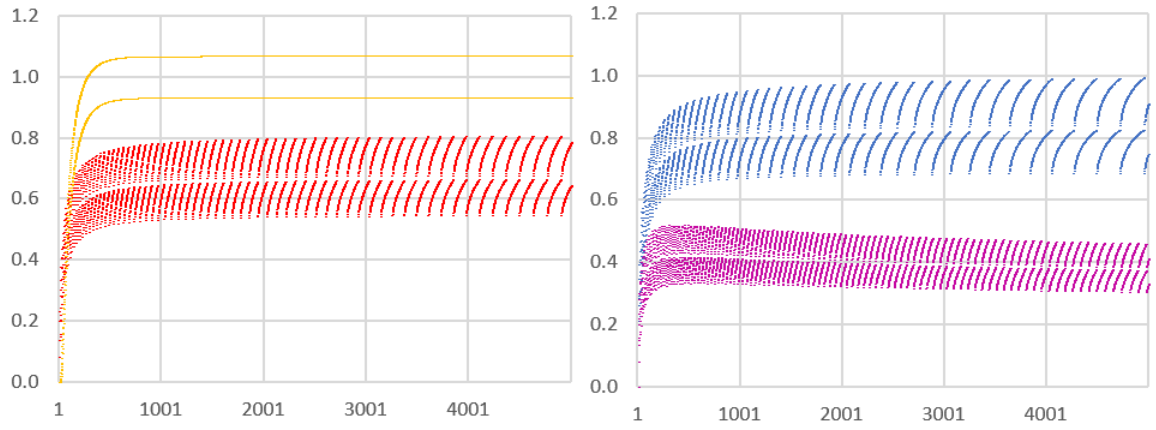

Figure 2: $\delta_n = 1 - \text{Var}[S_{n+1}] + \text{Var}[S_n]$ for four types of random walks, with $0 \leq n \leq 5000$

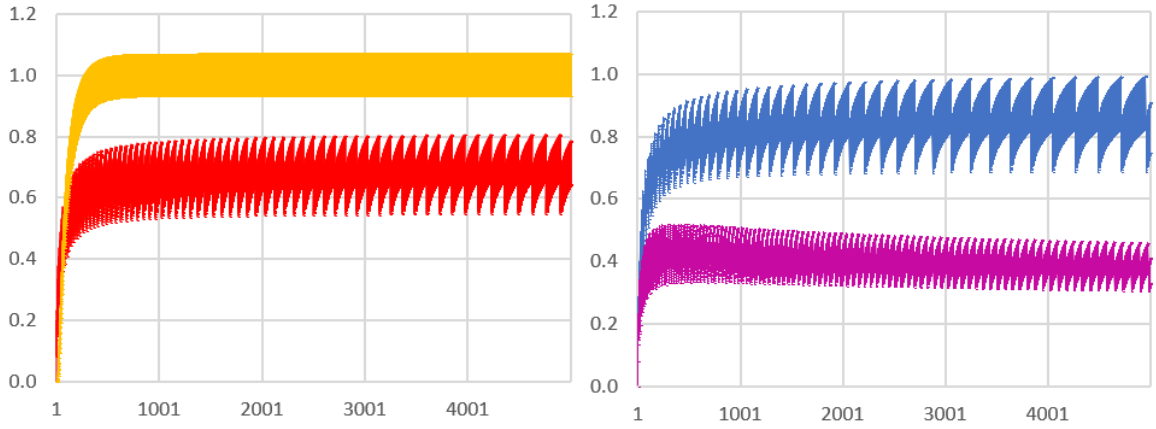

Figure 3: Same as Figure 2, using a more aesthetic but less meaningful chart type

where $A_n(m) = \chi[m < -g(n)] - \chi[m > g(n)]$. Here χ is the indicator function: $\chi(\omega) = 1$ if ω is true, otherwise $\chi(\omega) = 0$. Some of the identities in question include:

$$\sum_{m=-n}^{n} m \cdot p_n(m) = \text{E}[S_n] = 0, \quad \sum_{m=-n}^{n} A_n(m)p_n(m) = 0, \quad \sum_{m=-n}^{n} m^2 A_n(m)p_n(m) = 0,$$

$$\sum_{m=-n}^{n} m^2 \Big[A_{n-1}(m-1)p_{n-1}(m-1) + A_{n-1}(m+1)p_{n-1}(m+1)\Big] = 0.$$

From these identities, it is easy to establish a recursion for the variance:

$$\text{Var}[S_{n+1}] = \text{Var}[S_n] + 1 - \delta_n, \quad \text{with } \delta_n = 8\epsilon \cdot \sum_{m>g(n)} m \cdot p_n(m). \tag{3}$$

The sum for δ_n is finite since $p_n(m) = 0$ if $m > n$. Of course, $\text{Var}[S_0] = 0$. Also, if $\epsilon = 0$, the sequence is perfectly random: $\delta_n = 0$, $\text{Var}[S_n] = n$ and $S_n/\sqrt{n}$ converges to a normal distribution. In turn, the law of the iterated logarithm is satisfied. Conversely, this is violated if $\epsilon > 0$. Formula (3) combined with Hoeffding's inequality [Wiki], may provide some bounds for $\text{Var}[S_n]$.

Figure 3 shows δ_n for four types of modified random walks, using the following color scheme:

- Yellow: $g(n) = 10, \epsilon = 0.05$
- Red: $g(n) = n^\beta, \beta = 0.50, \epsilon = 0.05$
- Blue: $g(n) = n^\beta, \beta = 0.45, \epsilon = 0.05$
- Purple: $g(n) = n^\beta, \beta = 0.55, \epsilon = 0.05$

The curve that coincides with the X-axis ($\delta_n = 0$) corresponds to $\epsilon = 0$, that is, to pure randomness regardless of $g(n)$. It is not colored in Figure 3. Finally, the Python code in section 3.1 computes $\text{Var}[S_n]$ exactly (not via simulations) using two different methods, proving that Formula (3) is correct.

1.3 Random walks with less entropy than pure random signal

In section 1.2, I focused on creating sequences with higher oscillation rates than dictated by randomness, resulting in lower amplitudes. Doing the opposite – decreasing the oscillation rate – is more difficult. For instance, using $g(n) = n^\beta$ with $\beta > \frac{1}{2}$ won't work. You can't do better than $\sqrt{n}$ because of the law of the iterated logarithm: boosting β beyond the threshold $\frac{1}{2}$ is useless.

A workaround is to use the following algorithm:

$S = 0$
For $n = 1, 2, \ldots$
  Generate random deviate U on $[0, 1]$
  $M = g(n)$
  If $(-M < S < 0$ and $U < \frac{1}{2} + \epsilon)$ or $(0 < S < M$ and $U < \frac{1}{2} - \epsilon)$ or $(S = 0$ and $U < \frac{1}{2})$
  Then
    $X_n = -1$
  Else
    $X_n = 1$
  $S = S + X_n$
  $S_n = S$

The Python code in section 3.2, with the option `deviations='Large'`, performs this simulation. The yellow time series in Figure 1 is a realization of such a modified random walk, in this case with $g(n) = \alpha n^\beta$, with $\alpha = 0.30, \beta = 0.54$ and $\epsilon = 0.01$. It is unclear if the yellow curve will ever cross again the horizontal axis after 50,000 iterations, but it is expected to do so. To the contrary, the other three curves (gray, red, blue) are guaranteed to cross the horizontal axis infinitely many times, even though the random variable T measuring the spacing between two crossings (referred to as the hitting time in section 1.1) has infinite expectation.

For pure random walks (the gray curve in Figure 1), the average number of times that $S_k = 0$ when $0 < k \leq 2n$ is asymptotically equal to $\sqrt{4n/\pi}$, as discussed in section 1.1. One would expect this value to be about 178 when $2n = 50{,}000$. For the gray curve, the observed value is 243. Keep in mind that huge variations are expected between different realizations of the same random walk, due to the fact that $\mathrm{E}[T] = \infty$. Indeed, averaged over three realizations, the value 243 was down to 185. Also, a faulty pseudo-random number generator could easily lead to results that are off, in this case.

One would expect much larger values for the "non-random" red and blue curves. The observed values are respectively 747 and 1783, based on a single realization in each case. Likewise, the yellow curve is expected to have a much smaller value: in Figure 1, that value is 105.

2 Related stochastic processes

There are countless types of random walks or quasi-Brownian motions that are – on purpose and by design – not perfectly random. One could write an encyclopedia on this topic. A good reference is the book by Mörters and Peres [11], published in 2010. My goal in this section is to present two examples (one in two dimensions) that are very recent, interesting, and related to the material discussed in section 1. I built these stochastic processes in the last two years, to address modeling issues with fintech applications in mind.

2.1 From Brownian motions to clustered Lévy flights

Here I discuss a 2B Brownian motion generated using some specific probability distributions. Depending on the parameters, these distributions may or may not have an infinite expectation or variance. Things start to get interesting when the expectation becomes infinite (and the Brownian motion is no longer Brownian), resulting in a system exhibiting a strong clustering structure.

In some sense, it is similar to the examples studied earlier, where moving away from the law of the iterated logarithm resulted in unusual patterns: either very strong or very weak oscillations. Note that all the simulations performed here consist of discrete random walks rather than time-continuous Brownian motions. They approach Brownian motions very well, but since modern computers (at least to this date) are "digital" as opposed to "analog", everything is broken down into bits, and is thus discrete, albeit with a huge granularity.

In one dimension, we start with $S_0 = 0$ and $S_n = S_{n-1} + R_n\theta_n$, for $n = 1, 2$ and so on. If the R_n's are independently and identically distributed (iid) with an exponential distribution of expectation $1/\lambda$ and $\theta_n = 1$, then the resulting process is a stationary Poisson point process [Wiki] of intensity function λ on $\mathbb{R}^+$; the R_n's are the successive interarrival times interarrival times. If the θ_n's are iid with $P(\theta_n = 1) = P(\theta_n = -1) = \frac{1}{2}$,

and independent from the R_n's, then we get a totally different type of process, which, after proper re-scaling, represents a time-continuous Brownian motion in one dimension. For general references, see [2, 3].

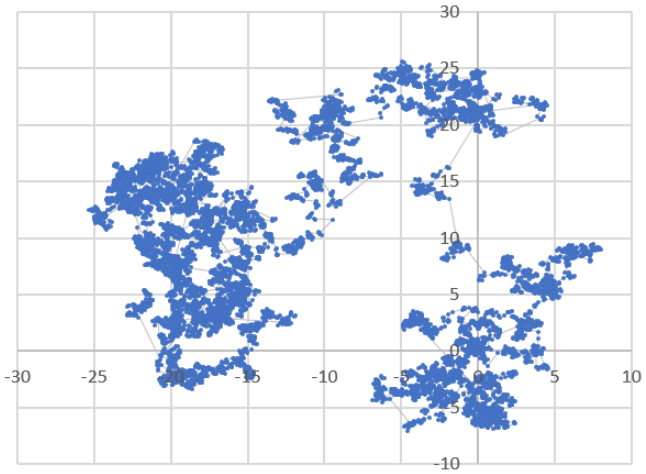

Figure 4: Clustered Brownian process

I generalize it to two dimensions, as follows. Start with $(S_0, S_0') = (0, 0)$. Then generate the points (S_n, S_n'), with $n = 1, 2$ and so on, using the recursion

$$S_n = S_{n-1} + R_n \cos(2\pi\theta_n) \tag{4}$$
$$S_n' = S_{n-1}' + R_n \sin(2\pi\theta_n) \tag{5}$$

where θ_n is uniform on $[0, 1]$, and the radius R_n is generated using the formula

$$R_n = \frac{1}{\lambda}\Big(-\log(1-U_n)\Big)^{\gamma}, \tag{6}$$

where U_n is uniform on $[0, 1]$. Also, $\lambda > 0$, and the random variables U_n, θ_n are all independently distributed. If $\gamma > -1$, then $\mathrm{E}[R_n] = \frac{1}{\lambda}\Gamma(1+\gamma)$ where Γ is the gamma function [Wiki]. In order to standardize the process, I use $\lambda = \Gamma(1+\gamma)$. Thus, $\mathrm{E}[R_n] = 1$ and if $\gamma > -\frac{1}{2}$,

$$\mathrm{Var}[R_n] = \frac{\Gamma(1+2\gamma)}{\Gamma^2(1+\gamma)} - 1.$$

We have the following cases:

- If $\gamma = 1$, then R_n has an exponential distribution.
- If $-1 < \gamma < 0$, then R_n has a Fréchet distribution. If in addition, $\gamma > -\frac{1}{2}$, then its variance is finite.
- If $\gamma > 0$, then R_n has a Weibull distribution, with finite variance.

Interestingly, the Fréchet and Weibull distributions are two of the three attractor distributions in extreme value theory.

The two-dimensional process consisting of the points (S_n, S_n') is a particular type of random walk. The random variables R_n represent the (variable) lengths of the successive increments. Under proper re-scaling, assuming the variance of R_n is finite, it tends to a time-continuous two-dimensional Brownian motion. However, if $\mathrm{Var}[R_n] = \infty$, it may not converge to a Brownian motion. Instead, it is very similar to a Lévy flight [Wiki], and produces a strong cluster structure, with well separated clusters. See Figure 4, based on $\gamma = -\frac{1}{2}$, $\lambda = 8$, and featuring the first 10,000 points of the bivariate sequence $\{(S_n, S_n')\}$.

The Lévy flight uses a Lévy distribution [Wiki] for R_n, which also has infinite expectation and variance. Along with the Cauchy distribution (also with infinite expectation and variance), it is one of the few stable distributions [Wiki]. Such distributions are attractors for an adapted version of the Central Limit Theorem (CLT), just like the Gaussian distribution is the attractor for the CLT. A well written, seminal book on the topic, is "Limit Distributions for Sums of Independent Random Variables", by Gnedenko and Kolmogorov [5].

For a simple introduction to Brownian and related processes, see the website RandomServices.org by Kyle Siegrist, especially the chapter on standard Brownian motions, here. The processes discussed in section 2.1 are further investigated in my book "Stochastic Processes and Simulations: A Machine Learning Perspective" [10].

2.2 Integrated Brownian motions and special auto-regressive processes

The Brownian motions pictured in Figure 5 are generated by simple time-discrete auto-regressive time series [Wiki]. Thus, the base process is auto-correlated, but the limit (after rescaling) is still a standard Brownian motion and thus perfectly random, if the auto-correlation structure is weak enough.

This autoregressive (AR) model is driven initial conditions $S_1, \ldots, S_p$ and the recursion

$$S_n = a_1 S_{n-1} + \cdots + a_p S_{n-p} + e_n,$$

where $a_1, \ldots, a_p$ are real coefficients satisfying some conditions to guarantee stationarity. By choice, the sequence $\{e_n\}$ is a white noise: $\mathrm{E}[e_n] = 0$, $\mathrm{Var}[e_n] = \sigma^2$ is fixed (it does not depend on n), and the e_n's are independently and identically distributed.

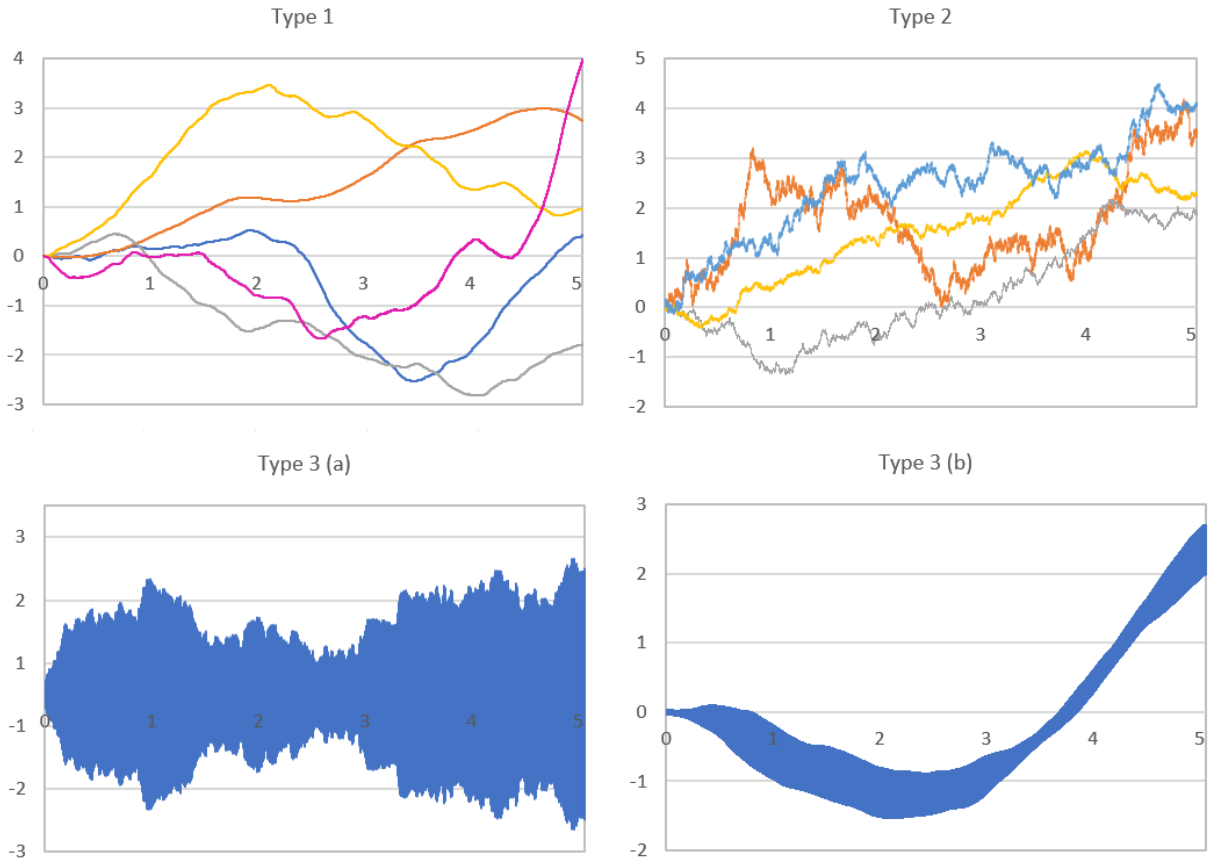

Figure 5: AR models, classified based on the types of roots of the characteristic polynomial

The behavior of this process, and thus of the resulting Brownian motions pictured in Figure 5, is determined by the roots of its characteristic polynomial of degree p:

$$x^p = a_1 x^{p-1} + a_2 x^{p-2} + \cdots + a_{p-1} x + a_p.$$

These roots can be real or complex, and simple or multiple. A borderline case, as far as stationarity is concerned, is when the root with highest modulus has a modulus equal to 1. If the modulus in question is above 1, the process is no longer stationary. The modulus of a real number a is its absolute value $|a|$, and for a complex number $a+bi$, it is defined as $\sqrt{a^2+b^2}$. If in addition the root with largest modulus is multiple, then something unusual happens: the resulting Brownian motion is very smooth and is not a Brownian motion anymore. It becomes is an integrated Brownian motion; its derivarive is a Brownian motion. See top left plot in Figure 5. The other plots in the same figure correspond to other akward situations, regarding the roots of the characteristic polynomials and the resulting behavior. This is discussed in detail in my article on linear algebra [8].

3 Python code

Section 3.1 covers the exact computation of the variances, while section 3.2 focuses on simulations: generating realizations of the sequence $\{S_n\}$, for various types of quasi-random walks described in section 1.

3.1 Computing probabilities and variances attached to S_n

This Python code is related to section 1.2.2, where you can find more details. It computes the variance of S_n for $n = 1, 2$ and so on, using two different methods: one based on the standard definition of the variance (denoted as `var1` in the code), and one based on Formula 3. The latter is denoted as `var2` in the code. Also, the variable `delta` represents δ_n. The output δ_n is featured in Figure 2. Finally the function `G` represents $g(n)$. The code below is also available on Github, here. Program name: `brownian_var.py`.

```
import math

epsilon=0.05
beta=0.45
alpha=1.00
nMax=5001

Prob={}
Exp={}
Var={}
Prob[(0,0)] =1
Prob[(0,-1)]=0
Prob[(0,1)] =0
Prob[(0,-2)]=0
Prob[(0,2)] =0

def G(n):
 return(alpha*(n**beta))

def psi(n,m):
  p=0.0
  if m>G(n):
    p=-1
  if m<-G(n):
    p=1
  return(p)

Exp[0]=0
Var[0]=0
OUT=open("rndproba.txt","w")
for n in range(1,nMax):
  Exp[n]=0
  Var[n]=0
  delta=0
  for m in range(-n-2,n+3,1):
    Prob[(n,m)]=0
  for m in range(-n,n+1,1):
    Prob[(n,m)]=(0.5+epsilon*psi(n-1,m-1))*Prob[(n-1,m-1)]\
       +(0.5-epsilon*psi(n-1,m+1))*Prob[(n-1,m+1)]
    Exp[n]=Exp[n]+m*Prob[(n,m)]
    Var[n]=Var[n]+m*m*Prob[(n,m)]
    if m>G(n-1) and m<n:
      delta=delta+8*epsilon*m*Prob[(n-1,m)]
  var1=Var[n]
  var2=Var[n-1]+1-delta
  string1=("%5d %.6f %.6f %.6f" % (n,var1,var2,delta))
  string2=("%5d\t%.6f\t%.6f\t%.6f\n" % (n,var1,var2,delta))
  print(string1)
  OUT.write(string2)
OUT.close()
```

3.2 Path simulations

This Python code performs all the simulations discussed in sections 1.2.2 and 1.3 and shown in Figure 1. The option `deviations='Small'` is discussed in detail in section 1.2.2, while `deviations='Large'` is explained in section 1.3. The function `G` in the code corresponds to $g(n)$. Also, if you want to simulate a perfectly random

walk, set ϵ (the parameter `eps` in the code) to zero. Finally, the code generates multiple realizations for any type of random walk. The number of realizations is determined by the parameter `Nsample`. The code below is also available on Github, here. Program name: `brownian_path.py`.

```
import random
import math
random.seed(1)

n=50000
Nsample=1
deviations='Large'
mode='Power'

if deviations=='Large':
  eps=0.01
  beta=0.54
  alpha=0.3
elif deviations=='Small':
  eps=0.05
  beta=0.35 #beta = 1 for log
  alpha=1

def G(n):
  if mode=='Power':
    return(alpha*(n**beta))
  elif mode=='Log' and n>0:
    return(alpha*(math.log(n)**beta))
  else:
    return(0)

OUT=open("rndtest.txt","w")
for sample in range(Nsample):
  print("Sample: ",sample)
  S=0
  for k in range(1,n):
    x=1
    rnd=random.random()
    M=G(k)
    if deviations=='Large':
      if ((S>=-M and S<0 and rnd<0.5+eps) or (S<=M and S>0 and rnd<0.5-eps) or
        (abs(S)>=M and rnd<0.5) or (S==0 and rnd<0.5)):
        x=-1
    elif deviations=='Small':
      if (S<-M and rnd<0.5-eps) or (S>M and rnd<0.5+eps) or (abs(S)<=M and rnd<0.5):
        x=-1
    print(k,M,S,x)
    S=S+x
    line=str(sample)+"\t"+str(k)+"\t"+str(S)+"\t"+str(x)+"\n"
    OUT.write(line)
OUT.close()
```

References

[1] Rabi Bhattacharya and Edward Waymire. *Random Walk, Brownian Motion, and Martingales*. Springer, 2021. 1

[2] D.J. Daley and D. Vere-Jones. *An Introduction to the Theory of Point Processes*. Springer, second edition, 2002. Volume 1 – Elementary Theory and Methods. 6

[3] D.J. Daley and D. Vere-Jones. *An Introduction to the Theory of Point Processes*. Springer, second edition, 2014. Volume 2 – General Theory and Structure. 6

[4] P. A. Van Der Geest. The binomial distribution with dependent Bernoulli trials. *Journal of Statistical Computation and Simulation*, pages 141–154, 2004. [Link]. 3

[5] B.V. Gnedenko and A. N. Kolmogorov. *Limit Distributions for Sums of Independent Random Variables*. Addison-Wesley, 1954. 6

[6] Manuel González-Navarrete and Rodrigo Lambert. Non-markovian random walks with memory lapses. *Preprint*, pages 1–14, 2018. arXiv [Link]. 1

[7] Vincent Granville. Detecting subtle departures from randomness. *Preprint*, pages 1–14, 2022. MLTechniques.com [Link]. 2, 3

[8] Vincent Granville. Gentle introduction to linear algebra, with spectacular applications. *Preprint*, pages 1–9, 2022. MLTechniques.com [Link]. 7

[9] Vincent Granville. Little known secrets about interpretable machine learning on synthetic data. *Preprint*, pages 1–14, 2022. MLTechniques.com [Link]. 3

[10] Vincent Granville. *Stochastic Processes and Simulations: A Machine Learning Perspective*. MLTechniques.com, 2022. [Link]. 6

[11] Peter Mörters and Yuval Peres. *Brownian Motion*. Cambridge University Press, 2010. Cambridge Series in Statistical and Probabilistic Mathematics, Volume 30 [Link]. 1, 2, 5

[12] Lan Wua, Yongcheng Qi, and Jingping Yang. Asymptotics for dependent Bernoulli random variables. *Statistics and Probability Letters*, pages 455–463, 2012. [Link]. 1