(Pseudo) Randomness

Klaus Sutner

Carnegie Mellon University

Spring 2024

Random Numbers

Random numbers (or random bits) are a crucial ingredient in many algorithms.

For example, all industrial strength primality testing algorithms rely on the availability of random bits. Modern cryptography is unimaginable without randomness.

> Anyone attempting to produce random numbers by purely arithmetic means is, of course, in a state of sin.
>
> John von Neumann

Neumann is referring to the inescapable fact that computers are deterministic devices (more or less), it is actually not all that easy to produce random bits using a computer: whatever program we run will produce the same bits if we run it again.

Pre-History

In the olden days, the RAND Corporation used a kind of electronic roulette wheel to generate a million random digits (rate: one per second).

In 1955 the data were published under the title:

A Million Random Digits With 100,000 Normal Deviates

"Normal deviates" simply means that the distribution of the random numbers is bell-shaped rather than uniform. But the New York Public Library shelved the book in the psychology section.

The RAND guys were surprised to find that their original sequence had several defects and required quite a bit of post-processing before it could pass muster as a random sequence. This took years to do.

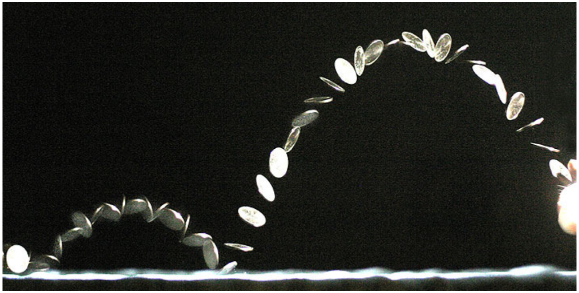
Available at http://www.rand.org/publications/classics/randomdigits.

Predictions

To make matters worse, it is rather difficult to even say exactly what is meant by a sequence of random bits. Of course, intuitively we all know what randomness means, right? It is closely connected to a massive inability to predict outcommes.

In particular, if you roll a die, all 6 outcomes are *equally possible*.

Flipping Coins



Another often used random bit generator: flipping fair coins.

Nasty Question: What is a fair coin? How do you flip it?

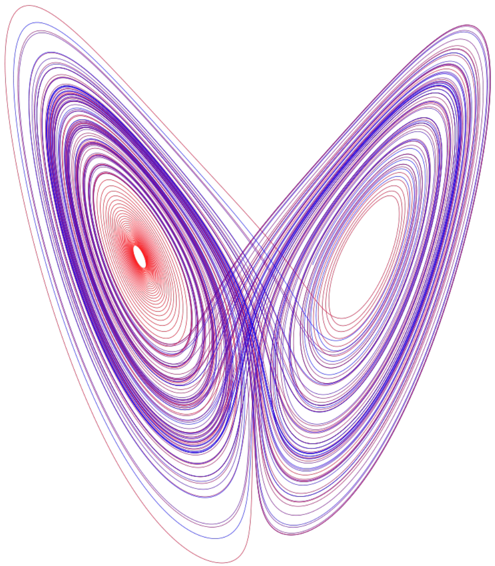
Lorenz Attractor

Here is a famous example discovered by Lorenz in the 1963, in an attempt to study a hugely simplified model of heat convection in the atmosphere. In terms of differential equations the model looks like so:

$$
\begin{aligned}
x' &= \sigma(y - x) \\
y' &= rx - y - xz \\
z' &= xy - bz
\end{aligned}
$$

These are not spatial coordinates, x stands for the amplitude of convective motion, y for temperature difference between rising and falling air currents, and z between temperature in the model and a simple linear approximation.

For appropriate values of the parameters we get the following, entirely deterministic, but random-looking, behavior.



Lava Lamps

Getting Serious

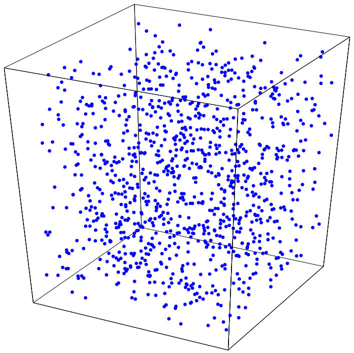
The previous systems seem to produce randomness, but do not escape the clutches of determinism (well, where does classical physics end and quantum physics start?). Instead, they exploit high sensitivity to initial conditions, which produces apparently chaotic behavior.

That may be perfectly good enough for some applications, but leaves an unpleasant aftertaste. Can we do better and go truly random?

Can we set up a random generator that would not yield to any attempts at analysis, even in the presence of unlimited compute power?

Krypton-85

Quantum physics in the form of radioactivity is a bold attack on randomness (except that no one likes to keep a lump of radioactive material and a Geiger-Müller counter on their desk). Solution: keep the radioactive stuff someplace else and get the random bits over the web.



Apparently true random bits from www.fourmilab.ch.

Huge Difference

The last system (and really also the lava lamps, see below) is very different from the others: if our current understanding of physics is halfway correct, there is no way to predict certain events in quantum physics, like radioactive decay. It is fundamentally impossible to predict future behavior, even if we could establish initial conditions correctly, which we cannot thanks to Herr Heisenberg. This is utterly weird, since Schrödinger's wave equation is perfectly deterministic, it is the measurement process that somehow produces randomness.

The other, purely mechanical systems such as dice and coins, we encounter deterministic chaos: given sufficiently precise descriptions of the initial conditions, and sufficient compute power, one could in principle compute the outcomes (if we think of them as classical systems). In actual reality, the inability to establish initial conditions with sufficient accuracy, and computational issues, make prediction quite impossible here, too.

Cloudflare

Weird systems are actually used in the RealWorld$^{\text{TM}}$ to generate random bits.

Cloudflare e.g. needs lots of high-quality randomness for their internet services. They harvest their random bits from various physical sources:

- A wall of lava lamps in San Francisco.
- A chaotic pendulum in London.
- A lump of radioactive material in Singapore.

The raw random bits are then fed into a cryptographically secure pseudo-random number generator (CSPRNG) that generates more good random bits.

Fiat Lux

Incidentally, Noll and Cooper at Silicon Graphics were the first to use lava lamps as random number generators. They even have a US Patent, number 5,732,138: "Method for seeding a pseudo-random number generator with a cryptographic hash of a digitization of a chaotic system."

They discovered one day that the pretty lava lamps were completely irrelevant: they could get even better random bits with the lens cap on: there is enough noise in the circuits to get good randomness. So quanum physics strikes again.

The "Entropy Wall" at Cloudflare, by contrast, takes pictures of the whole wall, and then uses a clever hash function to turn the huge collection of pixels into a large random number.

Getting Serious

A more ambitious way to use light, is to exploit an elementary quantum optical process: a photon hitting a semi-transparent mirror, and either passes through or is reflected. Schrödinger already got great mileage out of this idea.

The Quantis system was developed at the University of Geneva, the first practical model was released in 1998, http://www.idquantique.com/

The Magic Device

http://www.idquantique.com/random-number-generation/quantis-random-number-generator/

Quantis TRNG

Features

- True quantum randomness
- High bit rate of 4Mbits/sec (up to 16Mbits/sec for PCI card)
- Low-cost device (1000+ Euros)
- Compact and reliable
- USB or PCI, drivers for Windows and Linux

Applications

- Numerical Simulations
- Statistical Research
- Lotteries and gambling
- Cryptography

The Central Issue

Anyone who lives in this physical universe of ours is keenly aware of the effects of randomness–in the sense of unpredictability, a total lack of computational shortcuts. Just roll a die a couple of times.

But that means absolutely nothing mathematically, in particular since we currently do not have a satisfactory mathematical model of all of physics. It requires quite a bit of work to make sense out of randomness in a strictly mathematical sense, without appeals to physical intuition.

And, as one might suspect, computability falls by the wayside. Recall Kolmogorov-Chaitin complexity: it provides a definition of randomness for finite and infinite objects, but fails to be computable.

Hilbert's 6th Problem

Mathematical Treatment of the Axioms of Physics.

The investigations on the foundations of geometry suggest the problem: To treat in the same manner, by means of axioms, those physical sciences in which already today mathematics plays an important part; in the first rank are the theory of probabilities and mechanics.

This is in reference to Hilbert's seminal "Grundlagen der Geometrie" from 1899, together with Dedekind-Peano arithmetic the first modern axiomatization of a mathematical area. Well worth reading even today.

Success Stories

To be sure, there are excellent physical theories available, in particular relativity theory and quantum theory, but things get dicey when one tries to prove strict mathematical results. In essence, we can only argue relative to some axiomatization, and no one knows how to axiomatize **all** of physics—or whether a particular axiomatization correctly describes actual physics.

Still, we can home in on a few central properties of randomness—properties that we feel intuitively to be associated with randomness (strongly motivated by our experience of physics) rather than being able to derive them from first principles. But one has to be careful, intuition is not always reliable.

Unpredictability

Unpredictability is the key idea: suppose we are at time 0, and we want to understand the state of some physical system S at time $t > 0$. We understand $S(0)$, but the only way to determine its state $S(t)$ is to "run" the system till time t. But we cannot determine $S(t)$ at time $0 < t' \ll t$ no matter what we try (in particular execute some clever computation). There is no computational shortcut, we have computational incompressibility.

This is exactly why Ω is a truly random sequence, there is no way to predict bit number $n{+}1$, even if we somehow have figured out the first n bits.

Obstructions to Randomness

This may sound counterintuitive, but it is actually easier to try to define a random infinite binary sequence $\alpha \in \mathbf{2}^{\omega}$ instead of a single random bit, or a random finite sequence.

Here is one fruitful line of attack: what would disqualify a sequence α from being random in any intuitive sense of the word? For example, two obvious potential problems are:

- Bias (or skew): the limiting density of a 0 is not $1/2$.
- Correlation: the $i+1$st bit in α is not independent from the ith bit.

Kolmogorov-Randomness

Kolmogorov suggested to use incompressibility as a measure of randomness.

Definition

An infinite sequence $\alpha \in \mathbf{2}^\omega$ is Kolmogorov-random if for some constant c and all n: $C(\alpha[n]) \geq n - c$.

So the prefixes $\alpha[n]$ are algorithmically c-incompressible with the same constant c; $\alpha[n]$ is essentially the shortest description of itself. Chaitin's Ω is the perfect example of a sequence that is random in this sense.

Again, incompressibility is very similar in spirit to the notion of randomness: there is no rhyme nor reason, one has to have a full record to reconstruct the sequence.

The Importance of Prefix Complexity

It is important that the notion of incompressibility is based on Chaitin's prefix complexity, not the more intuitive plain Kolmogorov-Chaitin complexity.

Theorem (Martin-Löf)

Let f be computable such that $\sum 2^{-f(n)}$ diverges. Then for any $\alpha \in \mathbf{2}^\omega$ there are infinitely many n such that $K(\alpha[n]) < n - f(n)$.

The proof is quite involved. Note that $f(n) = n$ fails to work, but $f(n) = \log n$ satisfies the hypothesis. Hence we can infinitely often shorten the description in plain Kolmogorov-Chaitin complexity by a logarithmic amount; we don't have c-incompressibility.

Martin-Löf Randomness

Here is the main idea: We want to define randomness in terms of a collection of tests, a sequence is random if it survives all the tests.

As we will see, computability plays a major role in describing the tests, without it, our tests would rule out all sequences.

It is helpful to think of an infinite sequence $\alpha \in \mathbf{2}^\omega$ as an infinite branch in the full infinite binary tree $\mathbf{2}^\star$.

The initial segments are ordinary finite binary words and we can try to express conditions on α by placing conditions on prefixes $\alpha[n]$.

Example: Bias

Suppose weed out sequences with limiting density at least $2/3$.

For $x \in \mathbf{2}^\star$, define the density to be

$$D(x) = \frac{\#_1 x}{|x|}$$

Then for a bad string $\alpha \in \mathbf{2}^\omega$ there must be infinitely many n such that

$$\alpha[n] \in B_{n,2/3} = \{\, x \in \mathbf{2}^n \mid D(x) \geq 2/3 \,\}$$

Note that it is perfectly fine if this happens only for finitely many n, in this setting we don't care about bad initial behavior. In practice things are a bit more complicated, but let's not worry about this yet.

Cylinders and Measure

We will need to measure the size of sets $S \subseteq \mathbf{2}^\omega$.

To this end let x be a finite word and define the cylinder generated by x (essentially the subtree anchored at x) and its measure by

$$\begin{aligned} \mathsf{cyl}(x) &= \{\, \alpha \in \mathbf{2}^\omega \mid x \sqsubset \alpha \,\} \\ \mu(\mathsf{cyl}(x)) &= 2^{-|x|} \end{aligned}$$

For the bias example, each string $x \in \mathbf{2}^n$ such that $D(x) \geq 2/3$ contributes a cylinder of size 2^{-n} to $B_{n,2/3}$.

Extending the Measure

These cylinders form basic open sets in $\mathbf{2}^\omega$ and any open set S can be written as the disjoint union of countably many of them. For the math guys: these are Borel sets.

Hence we can measure any such set S:

$$S = \bigcup_i \mathsf{cyl}(x_i)$$

$$\mu(S) = \sum_i 2^{-|x_i|}$$

In the bias example, we have $\mu(B_{n,2/3}) = d/2^n$ where d is the number of strings $x \in \mathbf{2}^n$ such that $D(x) \geq 2/3$.

Intuition

Think of $\mathbf{2}^\omega$ as the real interval $[0,1] \subseteq \mathbb{R}$: interpret $\alpha \in \mathbf{2}^\omega$ as the binary expansion of some real in $[0,1]$.

Given a binary word $x = x_1 x_2 \dots x_n$, the cylinder cyl(x) then corresponds to a real interval comprising all numbers with expansion

$$0.x_1 x_2 \dots x_n \; z_0 z_1 z_2 \dots$$

The length of this interval is 2^{-n}, the measure of cyl(x).

We will try to cover a given set of reals by a collection of intervals of shrinking size (a set of constructive measure zero).

Example: Bias, Contd

For the bias example, to weed out sequences with limiting density at least $2/3$ we can use the following test sets:

$$K_n = \bigcup_{m \geq n} B_{m,2/3} = \bigcup \{ \,\mathsf{cyl}(x) \mid |x| \geq n \wedge D(x) \geq 2/3 \,\}$$

Being in K_n is not a problem per se, but being in $K = \bigcap_n K_n$ is: any such sequence has limiting density at least $2/3$ and should be eliminated from our pool of potential random sequences.

Of course, we need to perform countably many such tests to make sure that the limiting density is not larger that $1/2 + \varepsilon$ for any $\varepsilon > 0$. And we need to do the same for a lower bound.

General Case

More generally, we consider tests analogous to the density test: we want a descending chain of open sets

$$K_0 \supseteq K_1 \supseteq K_2 \supseteq \ldots \supseteq K_n \supseteq \ldots$$

where $\mu(K_n) \leq 2^{-n}$ so that these sets are becoming "small" as n increases and their intersection has constructive measure zero.

We eliminate all sequences in this set $K = \bigcap_n K_n$.

For the math guys: K is a G_δ null set.

One needs to tinker a bit with the K_n sets from the bias example to make sure they are sufficiently small.

The Catch

We want to declare a sequence α to be random if it survives this kind of test for various choices of test sequences (K_n). For example, we would want to apply all possible density tests, and not just for single bits, but also for blocks of arbitrary lengths.

Unfortunately, we cannot simply allow arbitrary tests (K_n): if we do, then all sequences are eliminated.

To see why, let $\alpha \in \mathbf{2}^\omega$ arbitrary and define a special test $K_n^\alpha = \text{cyl}(\alpha[n])$. Then $K^\alpha = \bigcap K_n^\alpha = \{\alpha\}$.

So, if we want to get anything useful out of this, we need to limit the permissible tests (K_n).

Computability to the Rescue

The key in designing a usable randomness test lies in imposing a computability constraint.

Definition

A sequential test has the additional property that

$$\mathcal{K} = \{\, (n, x) \mid \mathsf{cyl}(x) \sqsubset K_n \,\} \subseteq \mathbb{N} \times \mathbf{2}^\star$$

is semidecidable.

Think of $\mathcal{K}$ as being recursively enumerable: we can compute the pairs (n, x) one after the other, approximating the intersection that is the actual test.

Clearly, there are only countably many sequential tests.

And, we certainly can no longer design a test that eliminates an arbitrary sequence α (unless α is computable).

Applying a Sequential Test

By definition, we can effectively enumerate all the pairs (n, x) in $\mathcal{K} = \{\, (n, x) \mid \forall\, \alpha\, (x \sqsubset \alpha \Rightarrow \alpha \in K_n) \,\}$.

What does it mean for a particular sequence α to fail this test?

We need $\alpha \in \bigcap K_n$ where

$$K_n = \bigcup_{(n,x) \in \mathcal{K}} \mathsf{cyl}(x)$$

So for every n, we have to find an x such that $(n, x) \in \mathcal{K}$ and $x \sqsubset \alpha$.

Of course this is not effective, even assuming that we have α as an oracle. But it's not terribly far off.

Universal Tests

Taking our definitions at face value, we would have to check all possible sequential tests to check for randomness.

Here is an amazing result that shows that in essence we only need to deal with a single test. The proof uses the existence of a universal Turing machine and is not particularly difficult.

Theorem (Universal Test)

There is a universal sequential test U such that for any sequential test K we have $K_{n+c} \subseteq U_n$ for some constant c.

Martin-Löf Randomness

Definition

An infinite sequence α is Martin-Löf random if it passes a universal sequential test.

This definition is very strongly supported by the empirical fact that any practical test of randomness in ordinary probability theory can be translated into a sequential test. So, we are just dealing will all of these tests at once (plus all conceivable others).

As it turns out, the last approach produces the same notion of randomness as Kolmogorov-Chaitin style program-size complexity.

Theorem

A sequence is Martin-Löf random iff it is Kolmogorov-random.

How Bad is Randomness?

The definition may be elegant and right, but, sadly, it does not yield any methods to construct a random sequence. Aux contraire:

Theorem

Any Martin-Löf random sequence fails to be computable.

A typical example of a low-complexity random sequence is Chaitin's Ω (more later) which turns out to be Δ_2 in the arithmetical hierarchy. Close, but not computable.

Truth in Advertising

In the RealWorld$^{\text{TM}}$, one can often get away dealing with randomness by employing combinatorics and a little measure theory, without ever thinking about what randomness really means.

For applications in complexity, it is standard to assume the relevant properties of randomness, essentially in an axiomatic fashion. As long as our source of randomness is sufficiently well-behaved this will work just fine.

Exercises

Exercise

Give a detailed proof that limiting density can be checked by a suitable test set. In particular, make sure that semi-decidability holds.

Exercise

Show how to check for the frequencies of blocks of arbitrary finite length in a sequential test.

Exercise

Show that every Martin-Löf random sequence fails to be computable. Assume a random sequence is computable and show how to construct a test that rejects it.

Exercise

Show that there are uncountably many Martin-Löf random sequences (in fact, they form a set of measure 1).

RealWorld^{TM}

Recall von Neumann’s dictum:

Anyone attempting to produce random numbers by purely arithmetic means is, of course, in a state of sin.

Of course, formally speaking, he is absolutely correct. However, in many places it suffices to have bits that just appear random to the unaided eye, and the effort to get real random ones is just not warranted. “Purely arithmetic means” are often good enough.

What is surprising in this context is that often one can get away with murder. But beware applications in cryptography.

Pseudo-Randomness

In the real world, one often makes do with a pseudo-random number generator (PRNG) based on iteration: the pseudo-random sequence is the orbit of a some initial element under some function.

$$\begin{aligned} x_0 &= \text{ the seed, chosen somehow} \\ x_{n+1} &= f(x_n) \end{aligned}$$

where f is easily computable.

An excellent choice for the seed x_0 is to pick it at random. ☠

The Update Map

There are two ways to think about the maps f:

$$f : \mathbb{N} \longrightarrow \mathbb{N}$$
$$f : \mathbf{2}^k \longrightarrow \mathbf{2}^k$$

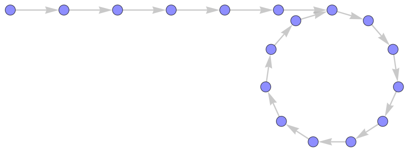
In the first case, we want an arithmetical function that somehow manages to produce bizarre outputs, even though it may seem like a reasonable algebraic operation.

In the second case we think about messing around with the bits directly, shifting, masking, xoring, rotating . . .

And, of course, we can combine both, just think of a natural number as a binary string.

The Lasso

Of course, we are taking a huge step away from real randomness here, this sequence would perish miserably when exposed to a Martin-Löf test. The function f typically operates on some finite domain such as 64-bit words. Every orbit necessarily looks like so:



So we can only hope to make f fast and guarantee long periods.

Still, there are many applications where this type of pseudo-randomness is sufficient.

One has to be very careful with cryptography, though!

The Seed

Needless to say, running a PRNG twice with the same seed x_0 is going to produce exactly the same “random” sequence.

As a practical matter, this is a huge advantage: it makes computations that are based on the random numbers reproducible, which is important for debugging and verification.

More generally, if we are willing to pay for a truly random seed, we would hope that the iterative PRNG would amplify the randomness: we provide m truly random bits and get back n high-quality pseudo-random bits where $n \gg m$.

Hence PRNGs reduce the need for truly random bits, but do not entirely eliminate them.

The Real Challenge

As von Neumann kindly pointed out, as a matter of principle we're hosed, no matter what we do.

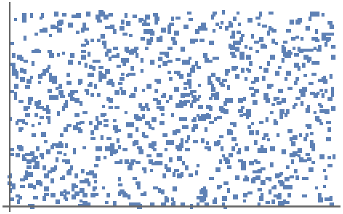
So the real question is this: how simple and easy-to-compute can we make our sinful function f and still get pseudo-random numbers that are sufficiently good to drive certain algorithms?

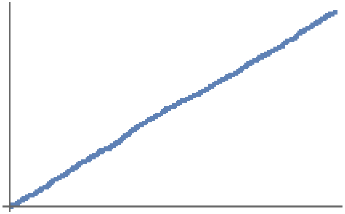
In the worst case, we could resort to actual quantum randomness, but that is expensive in many ways.

As it turns out, often we can get away with PRNGs that seem to be too simple-minded to actually work.

Von Neumann's RNG

Start with a 10-digit number x. Then iterate the following operation: square x, and extract the central 10 digits.





The first 1000 numbers, starting at 1234567890.

Linear Congruential Generator

A typical example: a simple affine map modulo m.

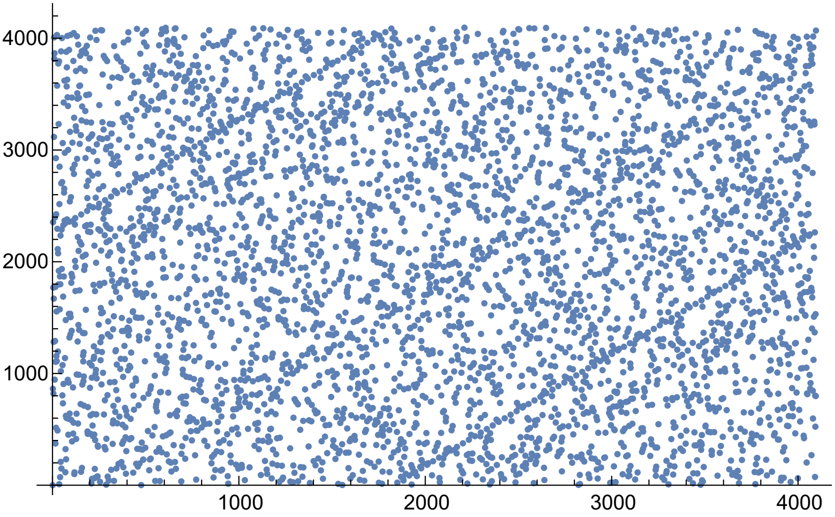
$$x_{n+1} = a\,x_n + b \bmod m$$

The trick here is to choose the proper values for the coefficients. Don't try this at home, look at the literature or the web. A choice that works reasonably well is

$$a = 1664525 \qquad b = 1013904223 \qquad m = 2^{32}$$

Note that a modulus of 2^{32} amounts to unsigned integer arithmetic on a 32-bit architecture, so this is implementation-friendly.

In fact, this LCG produces a full-length cycle for all moduli 2^k, $k \leq 32$.



4000
3000
2000
1000
1000
2000
3000
4000

Multiplicative Congruential Generator

Omit the additive offset and use multiplicative constants only.

If need be, use a higher order recurrence.

$$x_n = a_1 x_{n-1} + a_2 x_{n-2} + \ldots a_k x_{n-k} \bmod m$$

For prime moduli one can achieve period length $m^k - 1$.

This is almost as fast and easy to implement as LCG (though there is of course more work involved in calculating modulo a prime).

Note, though, that there is more state: we need to store all of the previous values $x_{n-1}, x_{n-2}, \ldots, x_{n-k}$.

Inverse Congruential Generator

Choose the modulus m to be a prime number and define the patched inverse of x to be

$$\overline{x} = \begin{cases} 0 & \text{if } x = 0, \\ x^{-1} & \text{otherwise.} \end{cases}$$

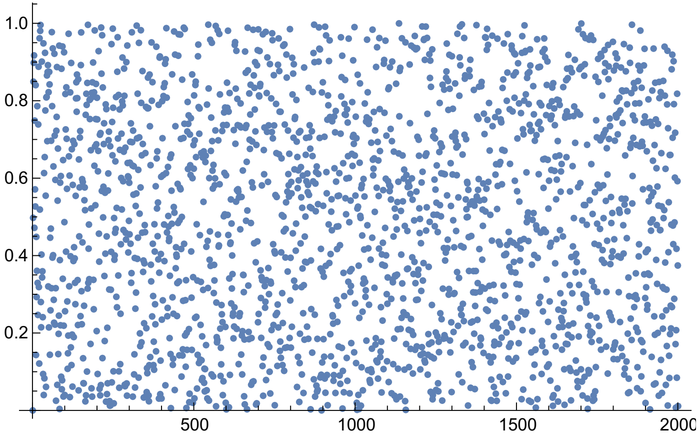
Then we can define a pseudo-random sequence by

$$x_{n+1} = a\,\overline{x_n} + b \bmod m$$

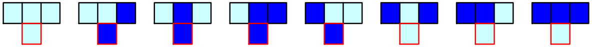
Computing the inverse can be handled by the extended Euclidean algorithm.

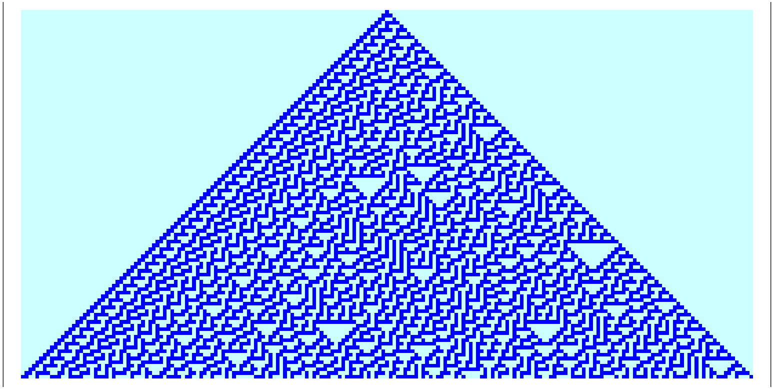
Again, it is crucial to choose the proper values for the coefficients.

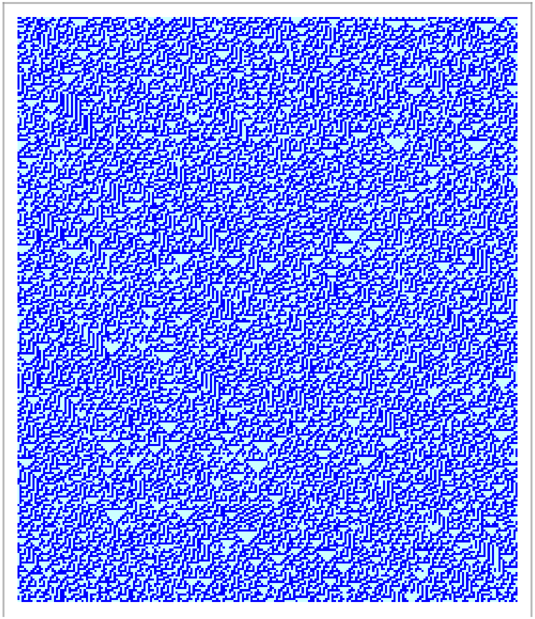
Modulus 2038074743



My Favorite PRNG: ECA 30







Mersenne Twister

Fairly recent (1998) method by Matsumoto and Nishimura, seems to be the tool of choice at this point (used in Sage, Maple, Matlab, GMP . . .).

- Has huge period of $2^{19937} - 1$, a Mersenne prime.
- Is statistically random in all the bits of its output (after a bit of post-processing).
- Has negligible serial correlation between successive values.
- Only statistically unsound generators are much faster.

The technique used is very clever and not exactly obvious.

The algorithm works on bit-vectors of length w (typically 32 or 64).

Let k be the degree of the recursion, and choose $1 \leq m < k$ and $0 \leq r < w$.

The MT Recurrence

So we are trying to generate a sequence of bit-vectors $x_i \in \mathbf{2}^w$.

Define the join $\text{join}(x, y)$ of $x, y \in \mathbf{2}^w$ to be the the first $w - r$ bits of x followed by the last r bits of y.

$$\text{join}(x, y) = (x_1, x_2, \ldots, x_{w-r}, y_{w-r+1}, \ldots, y_w)$$

We can use the join operation to define the following recurrence:

$$x_{n+k} = x_{n+m} \oplus \text{join}(x_n, x_{n+1}) \cdot A$$

Here A is a sparse companion-type matrix that makes it easy to perform the vector-matrix multiplication.

Companion Matrix

The $w \times w$ matrix A has the following form:

$$A = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ & & \ddots & & \\ 0 & 0 & 0 & \dots & 1 \\ a_{w-1} & a_{w-2} & a_{w-3} & \dots & a_0 \end{pmatrix}$$

Note that $z \cdot A$ is not really a vector-matrix operation and can be handled in $O(w)$ steps.

This is just a convenient way to describe the necessary manipulations.

Good Parameters

Here is one excellent choice for the parameters:

$$w = 32 \qquad k = 624 \qquad m = 397 \qquad r = 31$$

and the A matrix is given by

$$\boldsymbol{a} = \text{0x9908B0DF}$$

which hex-number represents the entries in the last row of A.

Recall that the recurrence determines x_{n+k} in terms of x_{n+m}, x_n and x_{n+1}, so we need to store a bit of state: $x_n, \ldots, x_{n+k-1} = x_{n+623}$.

So we need to store 624 words, not too bad.

Initialization

Initialization is non-trivial here, we need k words before the actual algorithm can get started.

A standard method is to pick $x_0 \in \mathbf{2}^w$ (at random) and then define

$$x_i = c \cdot (x_{i-1} \oplus \textsf{rshft}(x_{i-1}, w-2)) + i$$

where c is again cleverly chosen, typically $c = 1812433253$.

Initial conditions with lots of 0s should be avoided, it takes a while before good randomness appears in the generated sequence.

A Miracle

This particular choice of parameters achieves the theoretical upper bound for the period:

$$2^{wk-r} = 2^{19937} \approx 4.315 \times 10^{6001}.$$

After a little bit of post-processing of the sequence $(x_i)_{i \geq 0}$, this method produces very high quality pseudo-random numbers, and is not overly costly.