History of Random Number Generators

Seed \( x_0 \),
\[ x_i = f(x_{i-1}), \]
\[ u_i = g(x_i) \]

Pierre L’Ecuyer
Once upon a time, ... 5000 years ago

Dice, coins, and other devices were used long ago to make “random” selections and produce random numbers in games of chance. 5000-year old dice have been found in Iran and Irak. Dice were popular in India, China, Egypt, ..., some 4000 years ago.

People believed the outcomes were not truly random, but decided by god!

Source: quatr.us/west-asia/dice-invented-history-dice.htm, awesomedice.com/blog/551/new-candidate-for-oldest-dice/
2000 years ago in the Roman Empire

Sometimes a single bit would decide between life and death.
Tables of random sampling numbers

For statisticians, throwing dice to take random samples was inconvenient. Following a suggestion of Karl Pearson, Tippett (1927) published a table of 41,600 random digits taken from a 1925 census report.


Kendall and Babington-Smith (1938): first “machine” to produce the numbers. Cardboard disk divided in 10 sectors, rotating at about 250 turns per minute. Light beam flashed at random time, about every 2 seconds. Flashed sector number was recorded by a human. They made a table of 100,000 sampling numbers.
# Appendix

Random Sampling Numbers Produced by the Machine

<table>
<thead>
<tr>
<th>1st Thousand</th>
</tr>
</thead>
<tbody>
<tr>
<td>23157</td>
</tr>
<tr>
<td>05545</td>
</tr>
<tr>
<td>14871</td>
</tr>
<tr>
<td>38976</td>
</tr>
<tr>
<td>97312</td>
</tr>
<tr>
<td>11742</td>
</tr>
<tr>
<td>43361</td>
</tr>
<tr>
<td>93806</td>
</tr>
<tr>
<td>49540</td>
</tr>
<tr>
<td>36768</td>
</tr>
<tr>
<td>07092</td>
</tr>
<tr>
<td>43310</td>
</tr>
<tr>
<td>61570</td>
</tr>
<tr>
<td>31352</td>
</tr>
<tr>
<td>57048</td>
</tr>
<tr>
<td>09243</td>
</tr>
<tr>
<td>79757</td>
</tr>
<tr>
<td>93732</td>
</tr>
<tr>
<td>72621</td>
</tr>
<tr>
<td>61020</td>
</tr>
<tr>
<td>97839</td>
</tr>
<tr>
<td>89160</td>
</tr>
<tr>
<td>25966</td>
</tr>
<tr>
<td>81443</td>
</tr>
<tr>
<td>11322</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2nd Thousand</th>
</tr>
</thead>
<tbody>
<tr>
<td>64755</td>
</tr>
<tr>
<td>10302</td>
</tr>
<tr>
<td>71017</td>
</tr>
<tr>
<td>60012</td>
</tr>
<tr>
<td>37380</td>
</tr>
<tr>
<td>47869</td>
</tr>
<tr>
<td>38040</td>
</tr>
<tr>
<td>73508</td>
</tr>
<tr>
<td>32823</td>
</tr>
<tr>
<td>97591</td>
</tr>
</tbody>
</table>
How to measure the quality of a table?

Probability theory: for independent random digits, any possible table of $n$ digits has the same probability, so no table is better than another!
How to measure the quality of a table?

Probability theory: for independent random digits, any possible table of \( n \) digits has the same probability, so no table is better than another!

KBS introduced a notion of locally random sequence: every reasonably long subsequence should appear random and pass simple statistical tests. They proposed a small set of tests that measure:

1. the frequency of each digit;
2. the frequency of each pair in successive values (serial test);
3. the frequency of certain blocks of five digits (poker);
4. lengths of gaps between the occurrences of a given digit.

Frequencies are compared with expectations via a chi-square.

They replicated the tests with disjoint parts of their table. Their table passed the tests.
Tables in books were not convenient for random sampling with computers. 

**Project:** produce a million random digits in a fully automated way. 

An electronic device emits pulses randomly, about 100,000 per second. Every second, we count how many pulses, modulo 32. 20 of the 32 possible values are mapped to decimal digits, the others are discarded.

The digits were saved on **20,000 punched cards**, 50 per card.
1947: RAND Corporation project of a million random digits

Tables in books were not convenient for random sampling with computers.

**Project**: produce a million random digits in a fully automated way. An electronic device emits pulses randomly, about 100,000 per second. Every second, we count how many pulses, modulo 32. 20 of the 32 possible values are mapped to decimal digits, the others are discarded.

The digits were saved on 20,000 punched cards, 50 per card.

Odd digits were slightly more frequent! Then each digit was transformed by adding the corresponding digit of the previous card, mod 10. The transformed digits passed statistical tests.

**Horton and Smith III (1949)** proved that adding mod \( b \) random digits in base \( b \) reduces the bias.

A copy of the punched cards (large box) could be purchased in 1950.
1955: The million-digit book

Book with 50 rows of 50 digits per page for 400 pages. Reissued in paperback in 2001. Available at Amazon.

This book has maximum suspense: when reading digits in succession, at any step, one has no clue of what comes after!!!
Generating random numbers on the fly

For physicists doing Monte Carlo simulation, reading random digits from punched cards or tables was too slow and memory was also limited.

Two main approaches:
1. A fast physical device (electronic noise, etc.);
2. A purely deterministic algorithm that imitates randomness in software.

At the end of the RAND report of Brown (1949):
“... for the future ... it may not be asking too much to hope that ... some numerical process will permit us to produce our random numbers as we need them. The advantages of such a method are fairly obvious in large-scale computation where extensive tabling operations are relatively clumsy.”
Physical devices

Coins, dice, roulette, picking balls from an urn, shuffling cards, etc., have been used for centuries.

With computers, electronic devices such as counters of random events and periodic sampling of electric noise, are faster and more convenient.

Thousands of articles and around 2000 patents for physical RNGs in the last 70 years! Thermal and electric noise, photoelectric effect, devices based on quantum physics phenomena such as light beam splitters, shot noise in electronic circuits, radioactive decay detected by a Geiger counter, etc.

An early example: Cobine and Curry (1947): electric noise in a gas tube in a magnetic field is amplified and sampled to produce random bits.

Transformation techniques to reduce biais and correlations.

Fastest commercial devices today produce about 3 Gbits per second.
1957: ERNIE

The Electronic Random Number Indicator Equipment (ERNIE) could produce 50 random digits per second, used to determine winning numbers in the British Savings Bonds Lottery.

High voltage applied at each end of glass tubes filled with neon gas, produces current inside the tube, creating noise, which was amplified and collected.
ERNIE 4

Original ERNIE was used until 1972, then upgraded. ERNIE 4, still in use since 2004, extracts random bits from thermal noise in transistors.
Our ERNIE, 2017
Our ERNIE, 2017
**Algorithmic generators**

$S$, finite **state** space; $s_0$, **seed** (initial state);

$f : S \rightarrow S$, **transition** function;

$g : S \rightarrow [0, 1]$, **output** function.

$S_0$
Algorithmic generators

$\mathcal{S}$, finite state space; $s_0$, seed (initial state);
$f : \mathcal{S} \rightarrow \mathcal{S}$, transition function;
$g : \mathcal{S} \rightarrow [0,1]$, output function.

\[ \begin{array}{c}
S_0 \\
g \\
\downarrow \\
U_0
\end{array} \]
Algorithmic generators

$S$, finite state space; $s_0$, seed (initial state);

$f : S \rightarrow S$, transition function;

$g : S \rightarrow [0, 1]$, output function.

\[
\begin{array}{c}
s_0 \xrightarrow{f} s_1 \\
g \downarrow \\
u_0
\end{array}
\]

Period of \{s_n, n \geq 0\}:

$\rho \leq \text{cardinality of } S$.

Key feature: Can reproduce exactly the same sequence at will, without storing it.
Algorithmic generators

$S$, finite state space; $s_0$, seed (initial state);
$f : S \rightarrow S$, transition function;
$g : S \rightarrow [0, 1]$, output function.

\[ s_0 \xrightarrow{f} s_1 \]
\[ g \downarrow \quad g \downarrow \]
\[ u_0 \quad u_1 \]

Period of $\{s_n, n \geq 0\}$: $\rho \leq$ cardinality of $S$.

Key feature: Can reproduce exactly the same sequence at will, without storing it.
Algorithmic generators

$S$, finite state space; $s_0$, seed (initial state);
$f : S \rightarrow S$, transition function;
$g : S \rightarrow [0, 1]$, output function.
**Algorithmic generators**

\( S \), finite state space; \( s_0 \), seed (initial state);
\( f : S \to S \), transition function;
\( g : S \to [0, 1] \), output function.

\[
\begin{array}{cccccccc}
\cdots & f & \to & s_{\rho-1} & f & \to & s_0 & f & \to & s_1 & f & \to & \cdots & f & \to & s_n & f & \to & s_{n+1} \\
& g & \downarrow & & g & \downarrow & & g & \downarrow & & g & \downarrow & & g & \downarrow & &
\end{array}
\]

\[
\begin{array}{cccccccc}
\cdots & u_{\rho-1} & & u_0 & & u_1 & & \cdots & u_n & & u_{n+1} & &
\end{array}
\]

**Period** of \( \{s_n, \ n \geq 0\} \): \( \rho \leq \) cardinality of \( S \).

**Key feature:** Can reproduce exactly the same sequence at will, without storing it.
1946: von Neumann Middle-square method

Paper published in 1951. Start with 2\(a\)-digit number, square it to get 4\(a\) digits and take the middle 2\(a\) digits. Repeat.

With most seeds, ends in a very short cycle. Many heuristic “improvements” have been proposed, even recently. Not much theoretical support.
Two-digit numbers in base 10

(Source: Wikipedia)
Using successive decimals of $\pi$, $e$, etc., to imitate a random sequence was suggested long ago.

Metropolis et al. (1950) computed 2,000 decimals of $\pi$ and $e$ for this purpose and the sequences passed statistical tests.

Pathria (1962) did it for 10,000 decimals of $\pi$, then Esmenjaud-Bonnardel (1965) for 100,000 decimals. No statistical problem was found.

The world record in 2016 was 22,459,157,718,361 decimal digits of $\pi$.

But using these digits is much too slow, and it takes too much space to store them.
Notions of uniformly distributed infinite sequence

A more fundamental question: what can we prove about the uniformity and independence of these successive digits, in the long run?

Borel (1909): A sequence of digits in base $b$ is $k$-distributed if any of the $b^k$ possible blocks of $k$ successive digits appears with the same frequency in the long run. It is $\infty$-distributed if this holds for any $k \geq 1$. 

Borel proved that almost all real numbers (w.r.t. uniform measure) have an $\infty$-distributed digital expansion in base $b$ for any $b$ (i.e., they are normal numbers). We do not know if $\pi$ or $e$ is a normal number.

Similar notions of uniformity for sequences of real numbers in $[0, 1)$ were studied by Weyl (1916), Korobov (1948), Franklin (1963, 1965), etc. See Kuipers and Niederreiter (1974).

Sequences that provably satisfy these definitions can be constructed, but they are impractical for simulation, and often do not look random.
Notions of uniformly distributed infinite sequence

A more fundamental question: what can we prove about the uniformity and independence of these successive digits, in the long run?

Borel (1909): A sequence of digits in base $b$ is $k$-distributed if any of the $b^k$ possible blocks of $k$ successive digits appears with the same frequency in the long run. It is $\infty$-distributed if this holds for any $k \geq 1$.

Borel proved that almost all real numbers (w.r.t. uniform measure) have an $\infty$-distributed digital expansion in base $b$ for any $b$ (i.e., they are normal numbers). We do not know if $\pi$ or $e$ is a normal number.
Notions of uniformly distributed infinite sequence

A more fundamental question: what can we prove about the uniformity and independence of these successive digits, in the long run?

Borel (1909): A sequence of digits in base $b$ is $k$-distributed if any of the $b^k$ possible blocks of $k$ successive digits appears with the same frequency in the long run. It is $\infty$-distributed if this holds for any $k \geq 1$.

Borel proved that almost all real numbers (w.r.t. uniform measure) have an $\infty$-distributed digital expansion in base $b$ for any $b$ (i.e., they are normal numbers). We do not know if $\pi$ or $e$ is a normal number.

Similar notions of uniformity for sequences of real numbers in $[0, 1)$ were studied by Weyl (1916), Korobov (1948), Franklin (1963, 1965), etc. See Kuipers and Niederreiter (1974).

Sequences that provably satisfy these definitions can be constructed, but they are impractical for simulation, and often do not look random.
RNGs with long known finite period

1945: Mathematician Derrick Lehmer started using ENIAC computers for his research in number theory. He developed the vision that abstract mathematics could contribute to this new emerging field named “large-scale computing” (and vice-versa).
1945: Mathematician **Derrick Lehmer** started using ENIAC computers for his research in number theory. He developed the vision that abstract mathematics could contribute to this **new emerging field named “large-scale computing”** (and vice-versa).

In **1949**, at a symposium at Harvard, he suggested the idea of an algorithmic RNG with finite state space, based on a recurrence with long and known period. To realize this, he proposed the **linear congruential generator** (LCG):

\[ x_i = ax_{i-1} \mod m \]  
and output \[ u_i = x_i / m, \]

for \( 0 < a < m \).
RNGs with long known finite period

1945: Mathematician Derrick Lehmer started using ENIAC computers for his research in number theory. He developed the vision that abstract mathematics could contribute to this new emerging field named “large-scale computing” (and vice-versa).

In 1949, at a symposium at Harvard, he suggested the idea of an algorithmic RNG with finite state space, based on a recurrence with long and known period. To realize this, he proposed the linear congruential generator (LCG):

\[ x_i = ax_{i-1} \mod m \text{ and output } u_i = x_i / m, \]

for \( 0 < a < m \). He proposed two specific ones:

For decimal arithmetic: \( m = 10^8 + 1 \). Generated 1,250 8-digit numbers per hour on an IBM calculating punch 602A.

For binary arithmetic: \( m = 2^{31} - 1 \) and \( a = 23 \), with period \( 2^{31} - 2 \). Could generate 625 8-digit numbers per second on the ENIAC.
Many more LCGs

Already in 1950, people started to use multiplicative LCGs with $m = 2^e$ (for binary computers) or $m = 10^e$ (for decimal computers), with $e$ equal to the computer word length, to gain speed. This limited the period to $\rho = m/4$ and $\rho = m/200$, respectively.

They also started to take $a$ as a sum of 2 or 3 powers of 2 or 10, e.g., $a = 2^d + 1$, $2^d + 2 + 1$, $2^d + 4 + 1$, etc., so the product $ax \mod m$ could be computed by just a few shifts and adds.
Many more LCGs

Already in 1950, people started to use multiplicative LCGs with $m = 2^e$ (for binary computers) or $m = 10^e$ (for decimal computers), with $e$ equal to the computer word length, to gain speed. This limited the period to $\rho = m/4$ and $\rho = m/200$, respectively.

They also started to take $a$ as a sum of 2 or 3 powers of 2 or 10, e.g., $a = 2^d + 1$, $2^d + 2 + 1$, $2^d + 4 + 1$, etc., so the product $ax \mod m$ could be computed by just a few shifts and adds.

Taussky and Todd (1954) report instances with $m = 2^{43}$, $2^{42}$, $2^{39}$, $2^{36}$, $10^{11}$, $10^{10}$. They also pointed out that with $m = 2^e$, the period of the $j$th least significant bit cannot exceed $\max(0, 2^{j-2})$, so one should not rely on the least significant bits.
Rothenberg (1960) proposed adding a constant $c$:

$$x_i = (ax_{i-1} + c) \mod m$$

and proved that with $m = 2^{35}$, $a = 2^d + 1$, and $c$ odd, the period is $2^{35}$. He also suggested $m = 2^{35}$, $a = 7$, $c = 1$ for the IBM 704 35-bit computer.

Hull and Dobell (1962) proved sufficient conditions for maximal period for general $m$, $a$, and $c > 0$. In some cases, they were already known.
Statistical testing

Large collection of empirical statistical tests developed over the years, since around 1940 and even earlier.

Frequency, disjoint serial, overlapping serial, gap tests, runs up and down, runs of given digit, poker, coupons collector, collisions, nearest points, birthday spacings, iterated spacings, random walks, binary matrix rank, linear complexity, . . .

In many cases, the theoretical distribution used initially for the test statistic was incorrect! Examples: overlapping serial tests and run tests.
Statistical testing

Large collection of empirical statistical tests developed over the years, since around 1940 and even earlier.

Frequency, disjoint serial, overlapping serial, gap tests, runs up and down, runs of given digit, poker, coupons collector, collisions, nearest points, birthday spacings, iterated spacings, random walks, binary matrix rank, linear complexity, . . .

In many cases, the theoretical distribution used initially for the test statistic was incorrect! Examples: overlapping serial tests and run tests.

Marsaglia (1996): DIEHARD.
L’Ecuyer and Simard (2007): TestU01 (early versions since before 1990.)
**The spectral test**

**Coveyou and MacPherson (1967)** (ORNL report in 1965) exposed the **lattice structure** of LCGs and proposed the **spectral test** to measure their multivariate uniformity.

They showed that all vectors $\mathbf{u} = (u_i, \ldots, u_{i+s-1})$ belongs to a lattice; all lie in equidistant parallel hyperplanes with **wavelength** $d_s = 1/\ell_s$ where $\ell_s$ (the frequency) is the length of the shortest nonzero vector in the dual lattice. They suggested comparing $\ell_s$ with its theoretical upper bound for any given $m$. 

Examples: RANDU, with $m = 2^{31}$ and $a = 2^{16}+3$; RANNO, with $m = 2^{35}$ and $a = 2^{18}+3$; Lehmer’s LCG with $m = 2^{31}-1$ and $a = 23$, etc.
The spectral test

Coveyou and MacPherson (1967) (ORNL report in 1965) exposed the lattice structure of LCGs and proposed the spectral test to measure their multivariate uniformity.

They showed that all vectors \( \mathbf{u} = (u_i, \ldots, u_{i+s-1}) \) belongs to a lattice; all lie in equidistant parallel hyperplanes with wavelength \( d_s = 1/\ell_s \) where \( \ell_s \) (the frequency) is the length of the shortest nonzero vector in the dual lattice. They suggested comparing \( \ell_s \) with its theoretical upper bound for any given \( m \).

They proposed an algorithm to compute \( \ell_s \) and gave several examples for \( s \) up to 10 and \( m \) near \( 10^{10}, 2^{31}, 2^{35}, \) and \( 2^{47} \). They showed for example that \( d_s \) is always large when \( a \) is small or when \( a = 2^d + 3 \) and \( m = 2^e \).

Examples: RANDU, with \( m = 2^{31} \) and \( a = 2^{16} + 3 \); RANNO, with \( m = 2^{35} \) and \( a = 2^{18} + 3 \); Lehmer’s LCG with \( m = 2^{31} - 1 \) and \( a = 23 \), etc.
Marsaglia (1968) examined the lattice structure in terms of the minimal number of hyperplanes points that contain all the points.

Spectral test algorithm was improved by Knuth (1969), Dieter (1975), Fincke and Pohst (1985), etc.

Should we discard LCGs altogether because of their lattice structure?
Marsaglia (1968) examined the lattice structure in terms of the minimal number of hyperplanes points that contain all the points.

Spectral test algorithm was improved by Knuth (1969), Dieter (1975), Fincke and Pohst (1985), etc.

Should we discard LCGs altogether because of their lattice structure? **No.** This structure also has a positive side.
Multiple recursive generator (MRG)

Duparc et al. (1953): Fibonacci recurrence $x_i = (x_{i-1} + x_{i-2}) \mod m$.

Mitchell and Moore (1958): Lagged-Fibonacci, $x_i = (x_{i-r} + x_{i-k}) \mod m$. Bad structure, but still several versions in C and C++ libraries.

Zierler (1959) studied the general case:

$$x_i = (a_1x_{i-1} + \cdots + a_kx_{i-k}) \mod m.$$  

Max period $m^k - 1$, possible only if $m$ is prime.

Alanen and Knuth (1964): Algorithm to verify max period conditions.

Good parameters based on spectral test:

Deng and co-authors (2000 to 2012): very large order $k$. Bad structures.
Combined recurrences

Forsythe (1951): described an early method to construct an RNG with long known period: Take several short sequences with relatively prime (known) periods, run them in parallel, and combine their states to produce the output at each step.

For example, with sequences of periods 31, 33, 34, and 35, we can obtain a period of 1,217,370.
Combined LCGs and MRGs

Whichmann and Hill (1982) proposed an RNG that combines three 16-bit LCGs of prime periods near $2^{15}$. Still used in Excel!
Zeisel (1986) pointed out that their RNG is equivalent to another LCG, with period near $2^{43}$.


They are equivalent to an LCG or MRG with large modulus, equal to the product of moduli of the components. Was proved by L’Ecuyer and Tezuka (1991) for LCGs, and by L’Ecuyer (1996) for MRGs. Popular examples: MRG32k3a, MRG31k3p.
Linear recurrences modulo 2

Tausworthe (1965): linear feedback shift register (LFSR):

\[ x_i = (a_1 x_{i-1} + \cdots + a_k x_{i-k}) \mod 2, \quad u_i = \sum_{\ell=1}^{\ell} x_i s + \ell - 1 2^{-\ell}. \]

Takes a block of \( w \) bits starting at every multiple of \( s \). Fast implementation in hardware via a shift register.

With primitive polynomial, it is \( t \)-distributed to \( \ell \) bits of accuracy for all \( \ell \leq s \) and \( t \leq \lfloor k/s \rfloor \).

Tootill et al. (1973): introduced stronger property of asymptotically random (to \( w \) bits), also called maximally equidistributed (ME). Means \( t \)-distributed to \( \ell \) bits of accuracy whenever \( \ell \leq w \) and \( t \leq \lfloor k/\ell \rfloor \).

They found a specific ME Tausworthe generator with \( x_i = (x_{i-607} + x_{i-273}) \mod 2, s = 512, \) and \( w = 23 \).

Lewis et al. (1973): generalized feedback shift register (GFSR):

\[ y_i = (y_{i-r} \oplus y_{i-q}), \] with \( qw \)-bit state. \( w \) copies of same recurrence. Period limited to \( 2^q - 1 \) despite \( qw \)-bit state.
These RNGs are special cases of the general $\mathbb{F}_2$-linear family:

\[
\begin{align*}
    x_i &= A x_{i-1}, \\
    y_i &= B x_i, \\
    u_i &= \sum_{\ell=1}^{w} y_{i,\ell-1} 2^{-\ell} = y_{i,0} y_{i,1} y_{i,2} \cdots,
\end{align*}
\]

Matsumoto and Kurita (1992): Twisted GFSR, with period $2^k - 1$ for $k$-bit state.

Matsumoto and Kurita (1994): tempered TGFSR.


Panneton, L’Ecuyer, and Matsumoto (2006): The WELL generator, nearly as fast as MT19937, but they are ME, and mix the bits much better at each step. Period lengths: $2^{512} - 1$ to $2^{44497} - 1$. 
Lindholm (1968): a primitive polynomial of $A$ with few nonzero coefficients always leads to bad statistical behavior. There should be nearly 50% 0 and 50% 1 in the characteristic polynomial.

Tezuka and L’Ecuyer (1991) and Wang and Compagner (1993) proposed combined Tausworthe generators to achieve this. Period is the product of the periods of components.

L’Ecuyer (1996, 1999) showed how to construct maximally equidistributed combined Tausworthe generators and provided several specific implementations.

Note: All $\mathbb{F}_2$-linear RNGs fail statistical tests on the linear complexity of the binary sequence of any given bit.
Nonlinear generators

Linear RNGs are not cryptographically secure and most fail some tests that exploit their linearity. This can motivate nonlinear RNGs.

Coveyou (1969): studied polynomial recurrence \( x_i = p(x_{i-1}) \mod m \).

Knuth (1981): quadratic recurrence modulo \( m \).


RNGs coming from cryptology:

Blum, Blum, and Schub (1986): BBS generator.


Then SHA, TEA, ChaCha, Threefish, etc.
Multiple streams and substreams

Bratley, Fox, and Shrage (1983): Table of seeds spaced 100,000 steps apart for LCG with \( m = 2^{31} - 1 \) and \( a = 16807 \).

L’Ecuyer and Coté (1991) introduced facilities with multiple disjoint streams of length \( 2^{50} \) partitioned into substreams of length \( 2^{30} \), based on a combined LCG of period near \( 2^{61} \).

L’Ecuyer, Simard, Chan, Kelton (2002) proposed RngStreams, which offers many more streams and substreams, based on MRG32k3a, with period near \( 2^{191} \).

Available in many programming languages, and now used in various software environments such as MATLAB, SAS, Arena, R, SSJ, etc.
Some key references

**Main paper:** L’Ecuyer, P. 2017.
“History of Uniform Random Number Generation”.

“Fourier Analysis of Uniform Random Number Generators”.
*Journal of the ACM* 14:100–119.

Reading, MA: Addison-Wesley.

L’Ecuyer, P. 2012.
“Random Number Generation”.

“Random Numbers for Parallel Computers: Requirements and Methods, with Emphasis on GPUs”.
Open access at [http://dx.doi.org/10.1016/j.matcom.2016.05.005](http://dx.doi.org/10.1016/j.matcom.2016.05.005).

L’Ecuyer, P., and F. Panneton. 2009.
“$F_2$-Linear Random Number Generators”.

“TestU01: A C Library for Empirical Testing of Random Number Generators”.
*ACM Transactions on Mathematical Software* 33 (4): Article 22.

“An Object-Oriented Random-Number Package with Many Long Streams and Substreams”.

“Mathematical Methods in Large Scale Computing Units”.