History of Random Number Generators

Seed $\mathbf{x}_0$,
$\mathbf{x}_i = f(\mathbf{x}_{i-1})$,
$u_i = g(\mathbf{x}_i)$

Pierre L'Ecuyer

Université de Montréal

Inria informatics mathematics

École Polytechnique, Palaiseau, Déc. 2017

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.



Source: quatr.us/west-asia/dice-invented-history-dice.htm, awesomedice.com/blog/551/new-candidate-for-oldest-dice/

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

2000 years ago in the Roman Empire

Only need one bit to 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.

Fisher and Yates (1928): table with digits picked from a table of logarithms.

Kermack and Kendrick (1937): table of digits taken from a telephone directory.
Proposed run tests and gap tests to detect periodic behavior.

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.

First page of table from Kendall and Babington-Smith (1938)

APPENDIX

Random Sampling Numbers Produced by the Machine

1st Thousand

23157	54859	01837	25993	76249	70886	95230	36744
05545	55043	10537	43508	90611	83744	10962	21343
14871	60350	32404	36223	50051	00322	11543	80834
38976	74951	94051	75853	78805	90194	32428	71695
97312	61718	99755	30870	94251	25841	54882	10513
11742	69381	44339	30872	32797	33118	22647	06850
43361	28859	11016	45623	93009	00499	43640	74036
93806	20478	38268	04491	55751	18932	58475	52571
49540	13181	08429	84187	69538	29661	77738	09527
36768	72633	37948	21569	41959	68670	45274	83880
07092	52392	24627	12067	06558	45344	67338	45320
43310	01081	44863	80307	52555	16148	89742	94647
61570	06360	06173	63775	63148	95123	35017	46993
31352	83799	10779	18941	31579	76448	62584	86919
57048	86526	27795	93692	90529	56546	35065	32254
09243	44200	68721	07137	30729	75756	09298	27650
97957	35018	40894	88329	52230	82521	22532	61587
93732	59570	43781	98885	56671	66826	95996	44569
72621	11225	00922	68264	35666	59434	71687	58167
61020	74418	45371	20794	95917	37866	99536	19378
97839	85474	33055	91718	45473	54144	22034	23000
89160	97192	22232	90637	35055	45489	88438	16361
25966	88220	62871	79265	02823	52862	84919	54883
81443	31719	05049	54806	74690	07567	65017	16543
11322	54931	42362	34386	08624	97687	46245	23245

2nd Thousand

64755	83885	84122	25920	17696	15655	95045	95947
10302	52289	77436	34430	38112	49067	07348	23328
71017	98495	51308	50374	66591	02887	53765	69149
60012	55605	88410	34879	79655	90169	78800	03666
37330	94656	49161	42802	48274	54755	44553	65090
47869	87001	31591	12273	60626	12822	34691	61212
38040	42737	64167	89578	39323	49324	88434	38706
73508	30908	83054	80078	86669	30295	56460	45336
32623	46474	84061	04324	20628	37319	32356	43969
97591	99549	36630	35106	62069	92975	95320	57734

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.

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.

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.

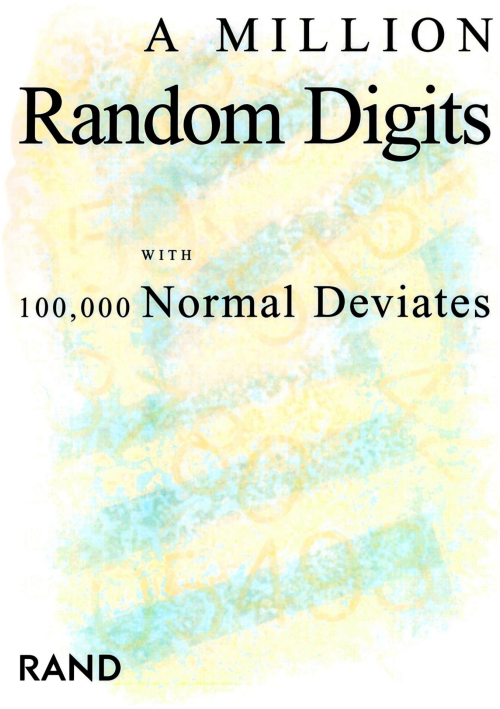
Horton and Smith III (1949) proved that adding mod b random digits in base b reduces the bias.

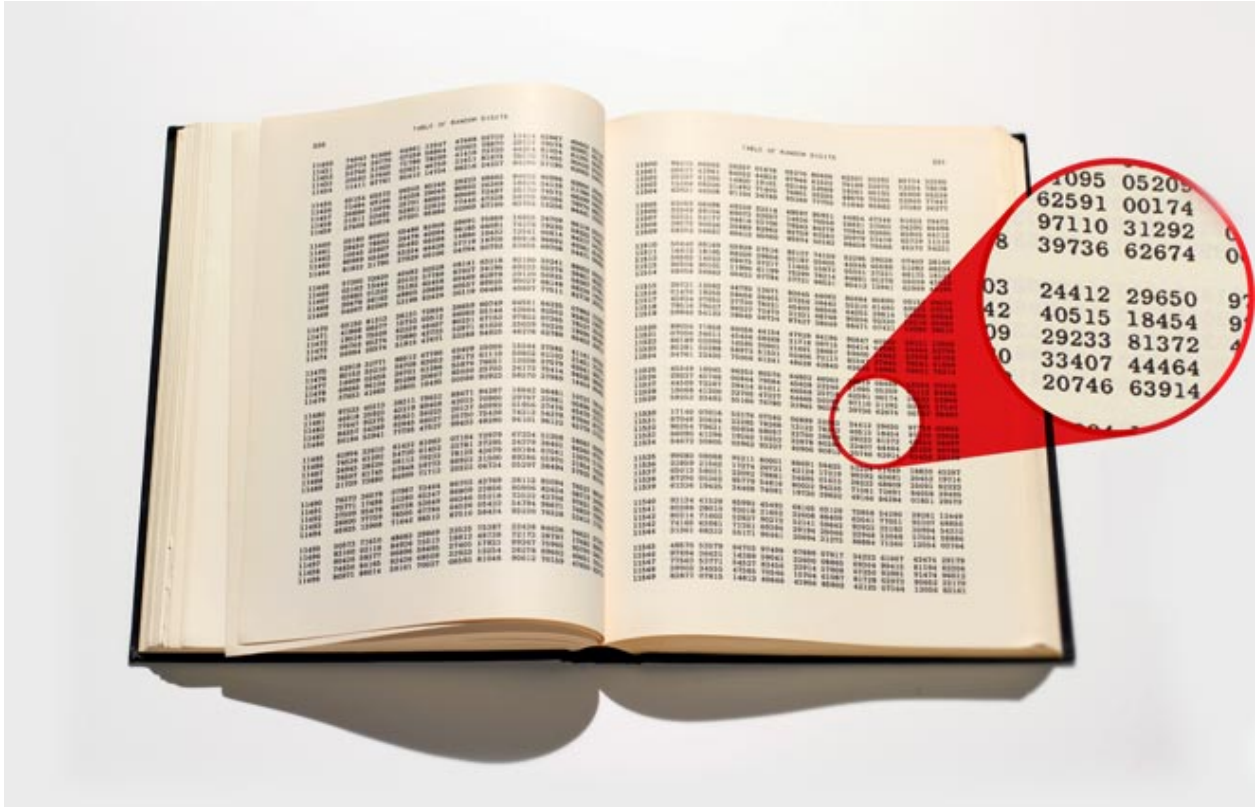
A copy of the punched cards (large box) could be purchased in 1950.

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.

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.

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.

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.

Algorithmic generators

$\mathcal{S}$, finite state space;
$f : \mathcal{S} \to \mathcal{S}$, transition function;
$g : \mathcal{S} \to [0, 1]$, output function.

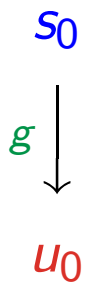
s_0, seed (initial state);

s_0

Algorithmic generators

$\mathcal{S}$, finite state space;
$f : \mathcal{S} \to \mathcal{S}$, transition function;
$g : \mathcal{S} \to [0, 1]$, output function.

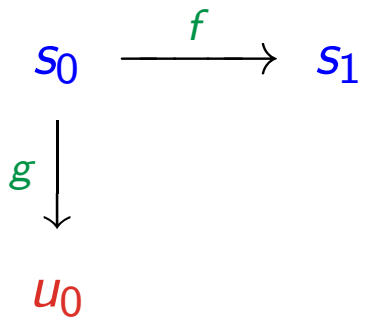
s_0, seed (initial state);



Algorithmic generators

$\mathcal{S}$, finite state space;
$f : \mathcal{S} \to \mathcal{S}$, transition function;
$g : \mathcal{S} \to [0, 1]$, output function.

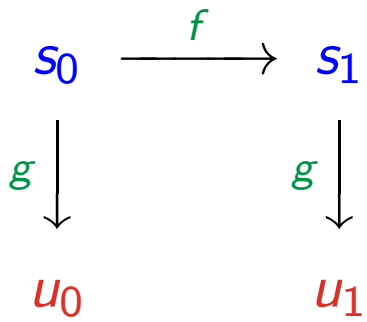
s_0, seed (initial state);



Algorithmic generators

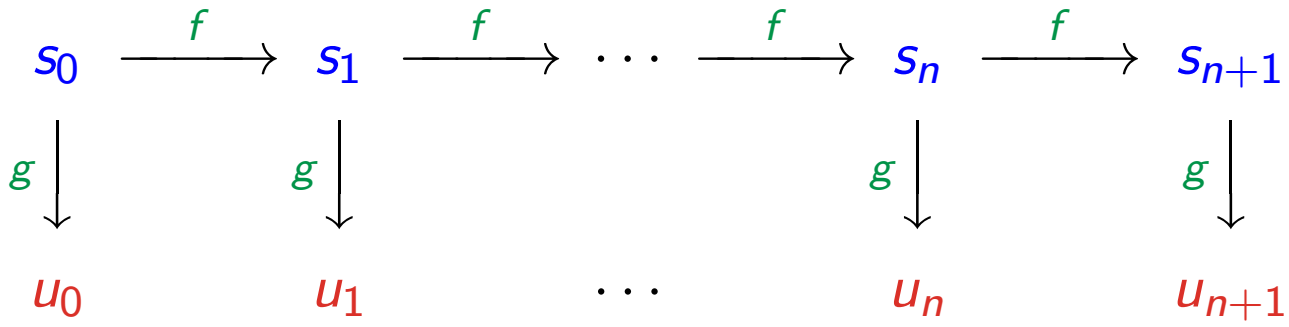
$\mathcal{S}$, finite state space;
$f : \mathcal{S} \to \mathcal{S}$, transition function;
$g : \mathcal{S} \to [0, 1]$, output function.

s_0, seed (initial state);



Algorithmic generators

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



Algorithmic generators

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

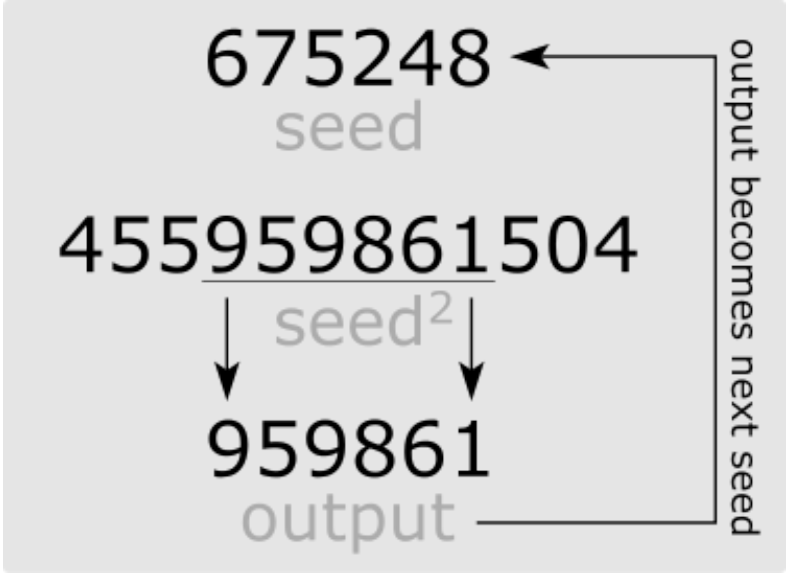
$$\cdots \xrightarrow{f} s_{\rho-1} \xrightarrow{f} s_0 \xrightarrow{f} s_1 \xrightarrow{f} \cdots \xrightarrow{f} s_n \xrightarrow{f} s_{n+1}$$
$$\downarrow g \qquad \downarrow g \qquad \downarrow g \qquad\qquad \downarrow g \qquad \downarrow g$$
$$\cdots \quad u_{\rho-1} \quad u_0 \quad u_1 \quad \cdots \quad u_n \quad u_{n+1}$$

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

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

1946: von Neumann Middle-square method

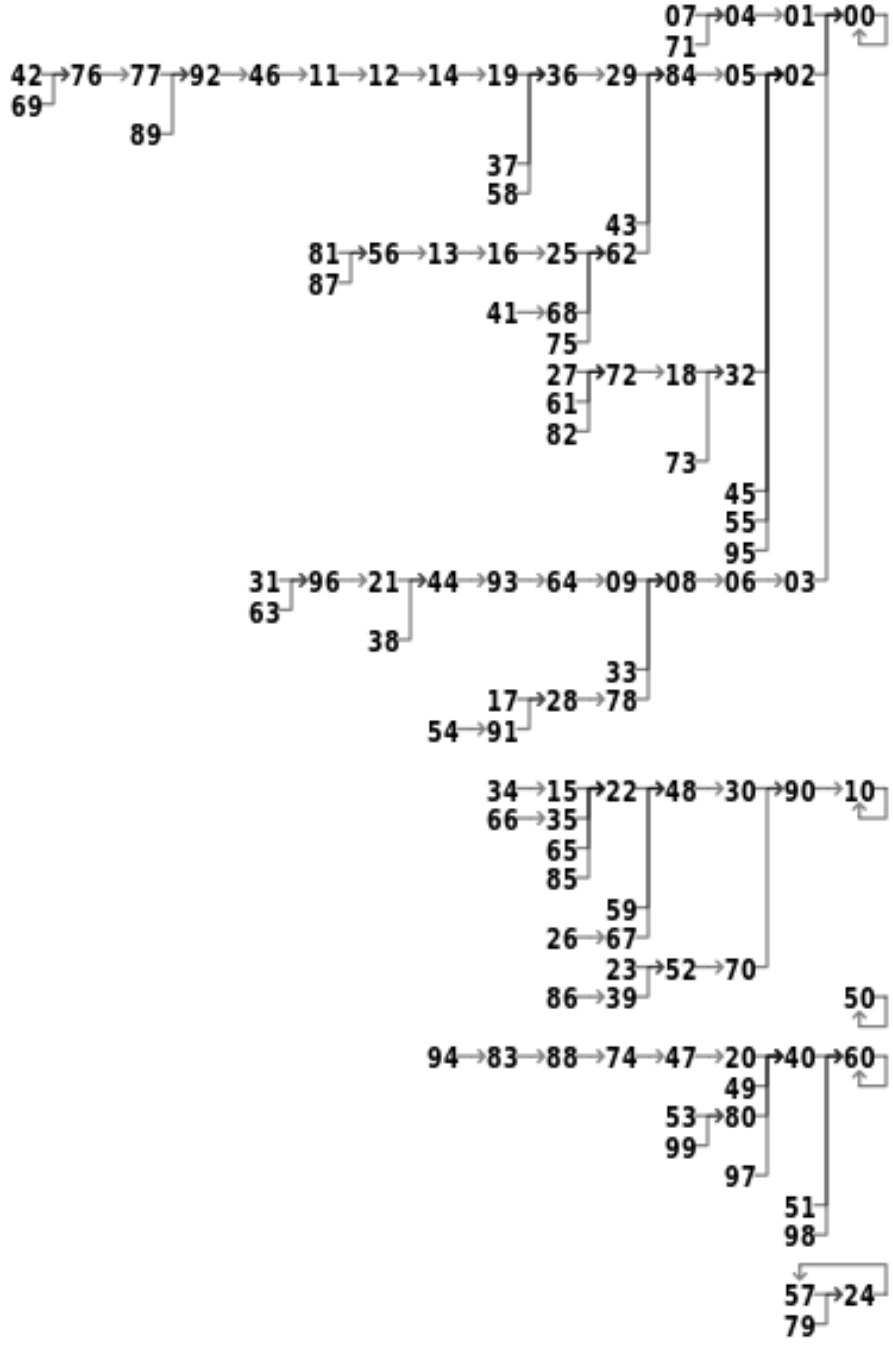
Paper published in 1951. Start with $2a$-digit number, square it to get $4a$ digits and take the middle $2a$ digits. Repeat.



Source: Wikipedia

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)

1950 or earlier: Decimals of π and e

Using successive decimals of π, e, etc., to imitate a random sequence was suggested long ago.

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

Pathria (1962) did it for 10,000 decimals of π, 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 π.

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 ∞-distributed if this holds for any $k \geq 1$.

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 ∞-distributed if this holds for any $k \geq 1$.

Borel proved that almost all real numbers (w.r.t. uniform measure) have an ∞-distributed digital expansion in base b for any b (i.e., they are normal numbers). We do not know if π 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 ∞-distributed if this holds for any $k \geq 1$.

Borel proved that almost all real numbers (w.r.t. uniform measure) have an ∞-distributed digital expansion in base b for any b (i.e., they are normal numbers). We do not know if π 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).

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).

Number theory to the rescue: In **1949**, at a symposium at Harvard, Lehmer 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):

$$\text{state} \quad x_i = a x_{i-1} \bmod m \quad \text{and output} \quad u_i = x_i / m,$$

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).

Number theory to the rescue: In **1949**, at a symposium at Harvard, Lehmer 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):

$$\text{state} \quad x_i = a x_{i-1} \bmod m \quad \text{and output} \quad u_i = x_i / m,$$

for $0 < a < m$. He suggested 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 jth least significant bit cannot exceed $\max(0, 2^{j-2})$, so one should not rely on the least significant bits.

Adding a constant term

Rothenberg (1960) proposed adding a constant c:

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

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.

Empirical statistical Tests

Hypothesis $\mathcal{H}_0$: "$\{u_0, u_1, u_2, \dots\}$ are i.i.d. $U(0,1)$ r.v.'s".
We know that $\mathcal{H}_0$ is false, but can we detect it ?

Empirical statistical Tests

Hypothesis $\mathcal{H}_0$: "$\{u_0, u_1, u_2, \ldots\}$ are i.i.d. $U(0,1)$ r.v.'s".
We know that $\mathcal{H}_0$ is false, but can we detect it ?

Test:
— Define a statistic T, function of the u_i, whose distribution under $\mathcal{H}_0$ is known (or approx.).
— Reject $\mathcal{H}_0$ if value of T is too extreme. If suspect, can repeat.

Different tests detect different deficiencies.

Empirical statistical Tests

Hypothesis $\mathcal{H}_0$: "$\{u_0, u_1, u_2, \ldots\}$ are i.i.d. $U(0,1)$ r.v.'s".
We know that $\mathcal{H}_0$ is false, but can we detect it ?

Test:
— Define a statistic T, function of the u_i, whose distribution under $\mathcal{H}_0$ is known (or approx.).
— Reject $\mathcal{H}_0$ if value of T is too extreme. If suspect, can repeat.

Different tests detect different deficiencies.

Utopian ideal: T mimics the r.v. of practical interest. Not easy.

Ultimate dream: Build an RNG that passes all the tests? Formally impossible.

Empirical statistical Tests

Hypothesis $\mathcal{H}_0$: "$\{u_0, u_1, u_2, \ldots\}$ are i.i.d. $U(0,1)$ r.v.'s".
We know that $\mathcal{H}_0$ is false, but can we detect it ?

Test:
— Define a statistic T, function of the u_i, whose distribution under $\mathcal{H}_0$ is known (or approx.).
— Reject $\mathcal{H}_0$ if value of T is too extreme. If suspect, can repeat.

Different tests detect different deficiencies.

Utopian ideal: T mimics the r.v. of practical interest. Not easy.

Ultimate dream: Build an RNG that passes all the tests? Formally impossible.

Compromise: Build an RNG that passes most reasonable tests.
Tests that fail are hard to find.
Formalization: computational complexity framework.

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.

Knuth (1969, 1981).

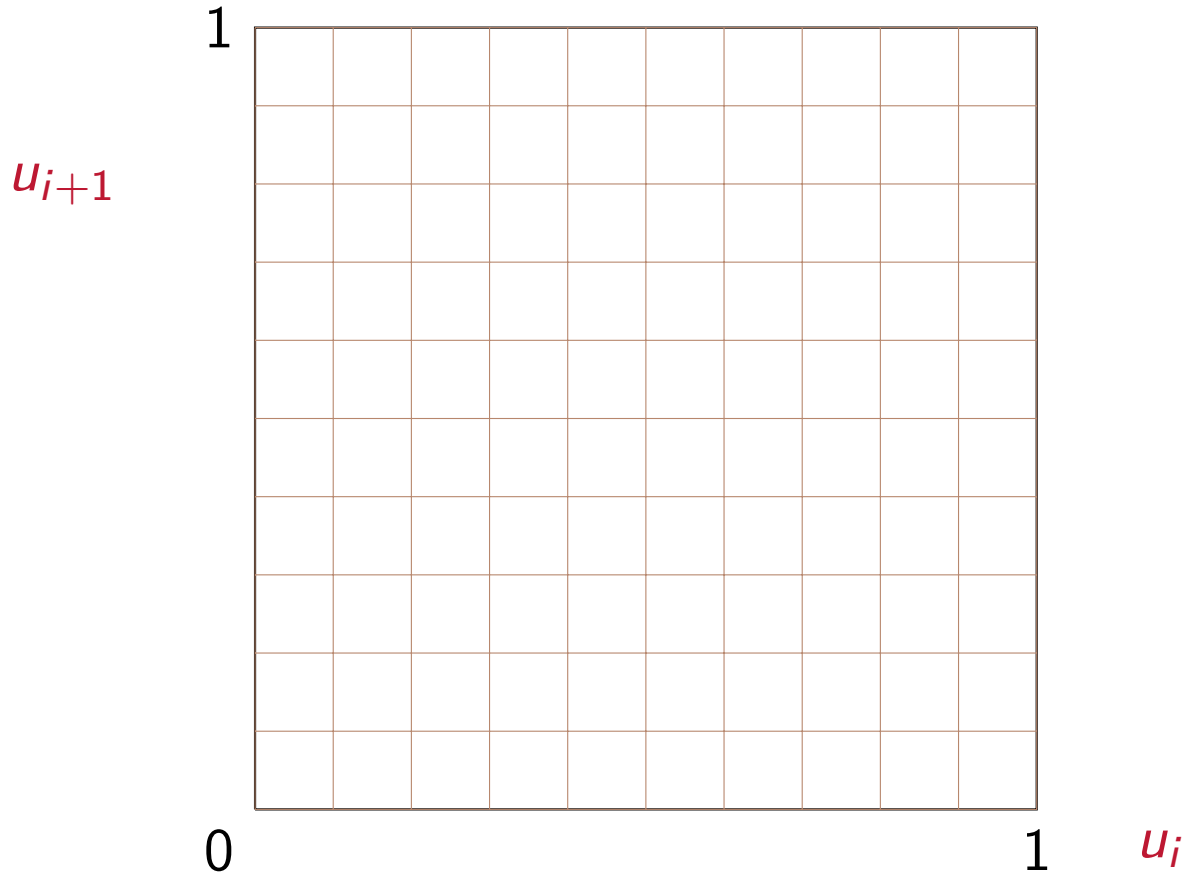
Dudewicz and Ralley (1981): TESTRAND (in FORTRAN).

Marsaglia (1996): DIEHARD.

L'Ecuyer and Simard (2007): TestU01 (early versions since before 1990.)

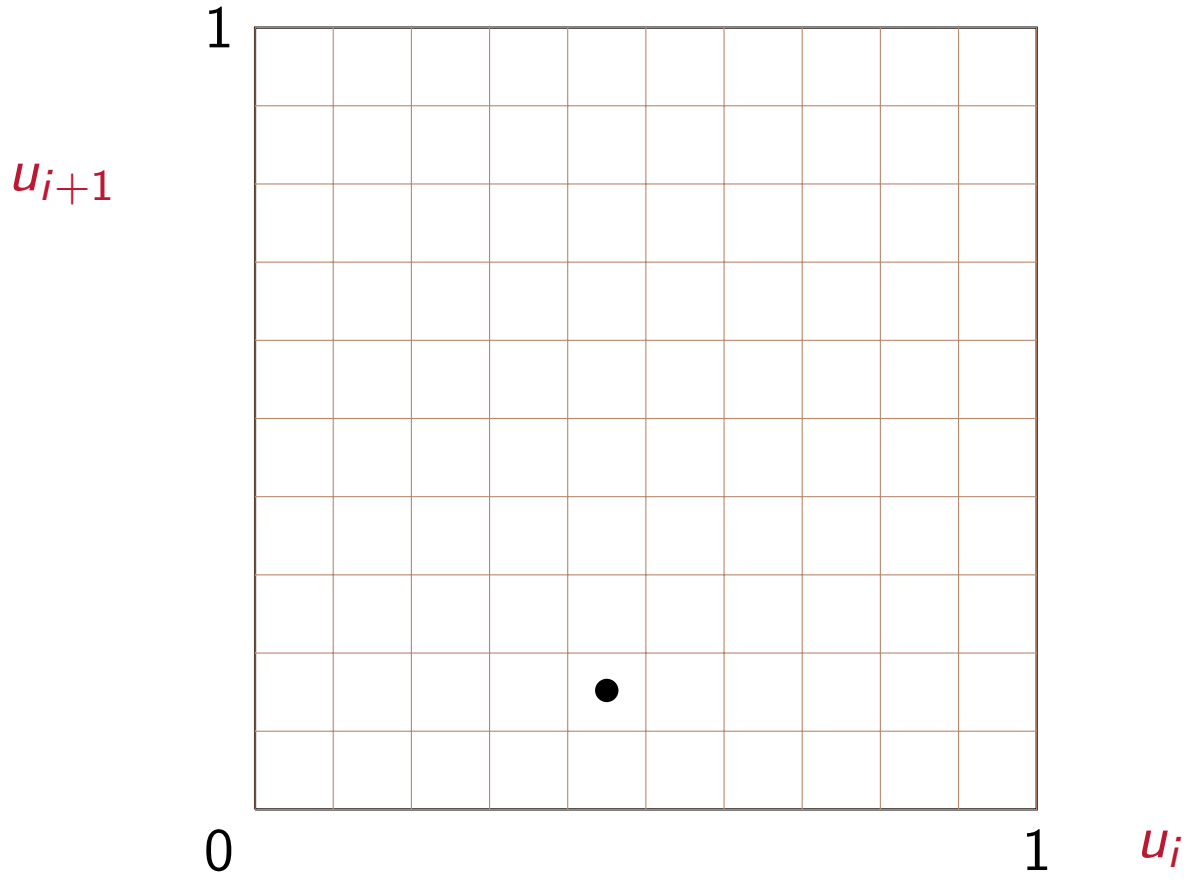
NIST suites (2001, 2010).

Examples: Serial and collision tests



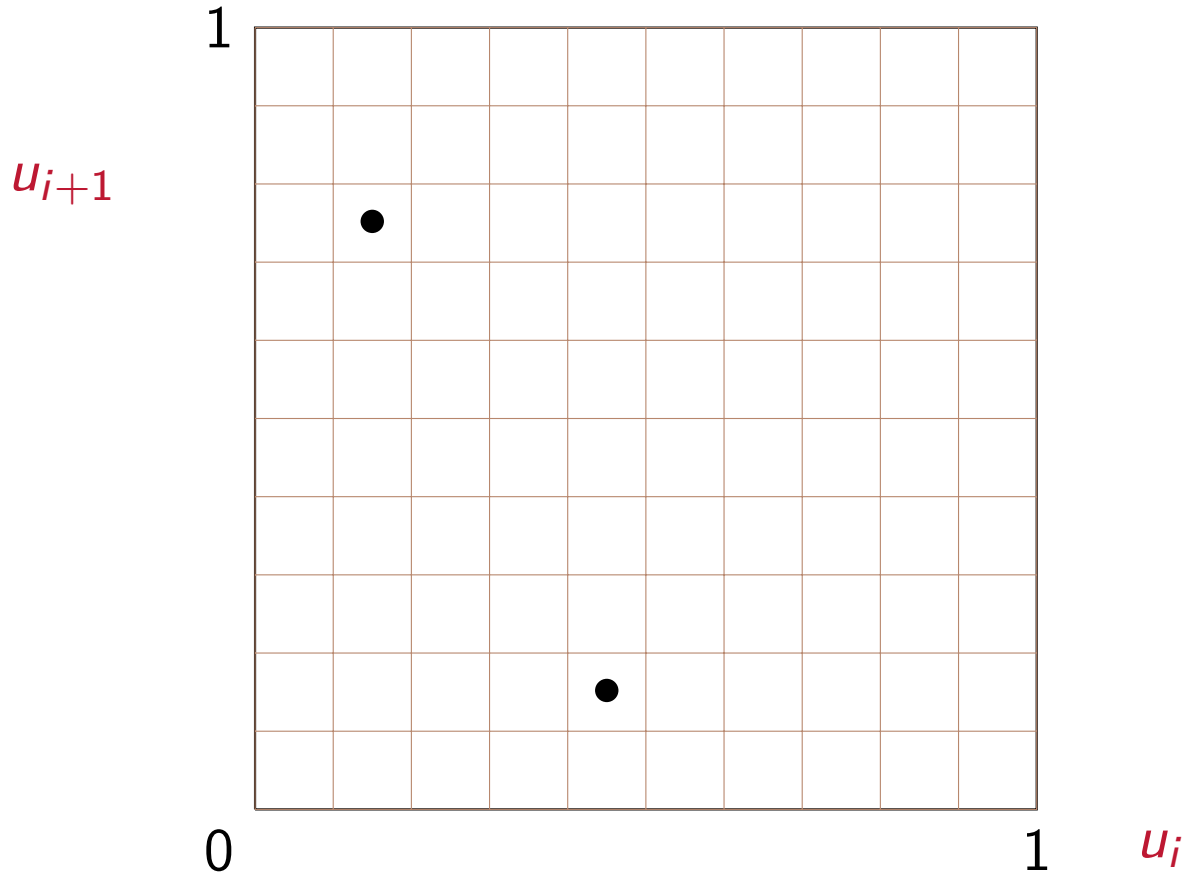
Throw $n = 10$ points in $k = 100$ boxes.

Examples: Serial and collision tests



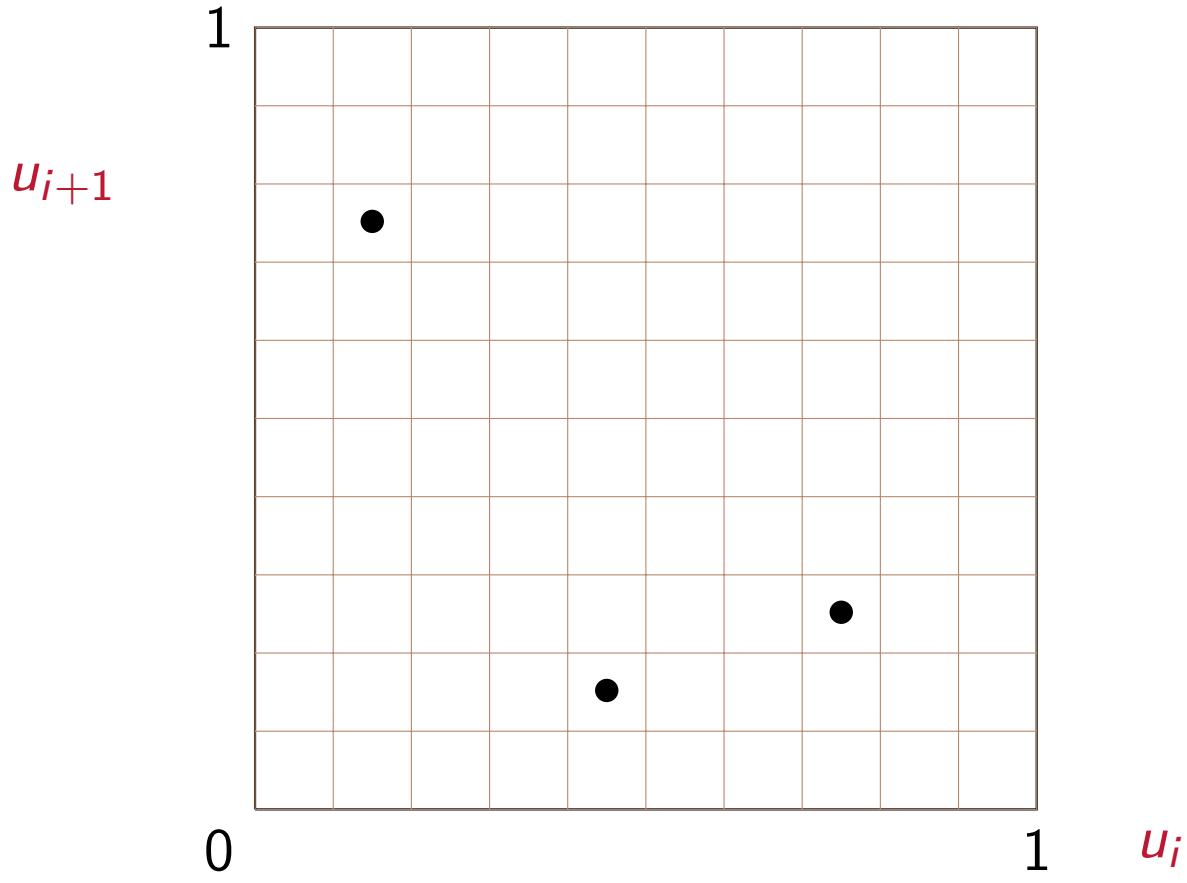
Throw $n = 10$ points in $k = 100$ boxes.

Examples: Serial and collision tests



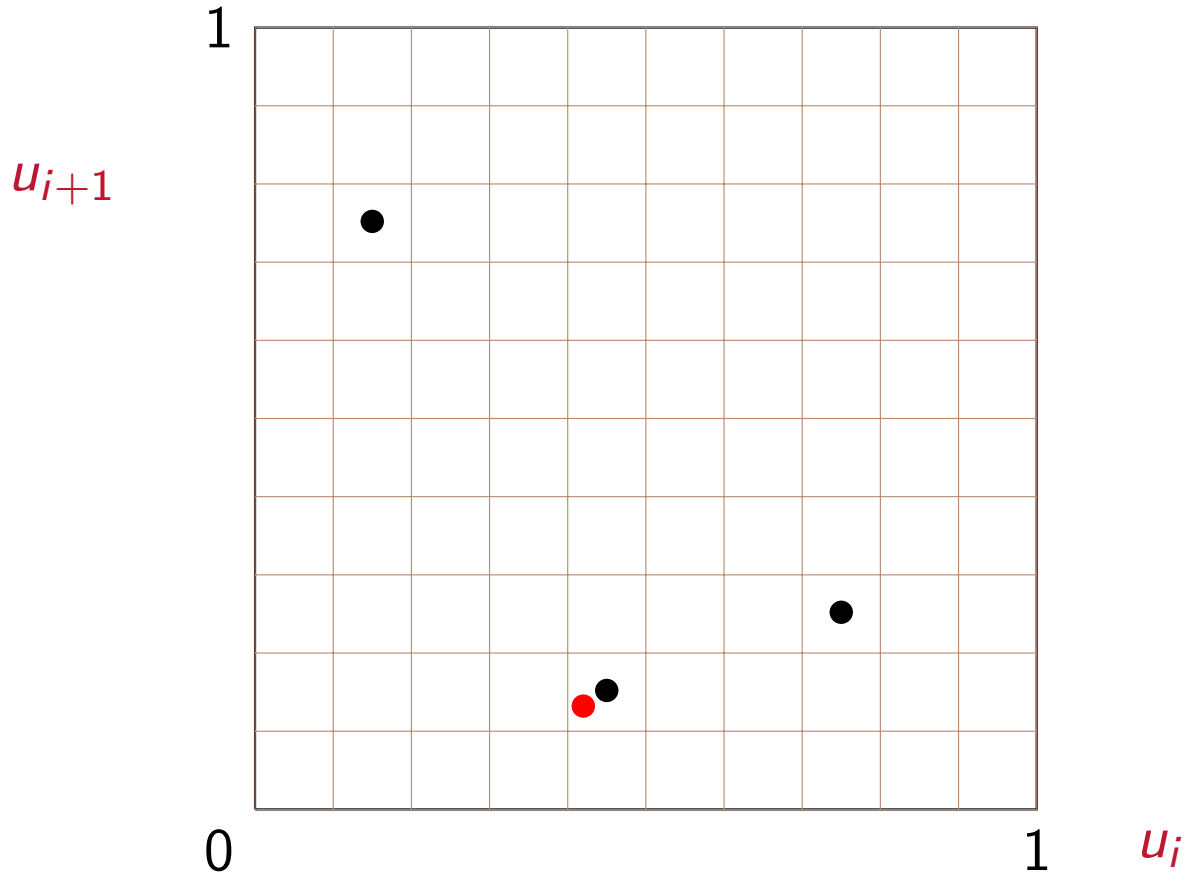
Throw $n = 10$ points in $k = 100$ boxes.

Examples: Serial and collision tests



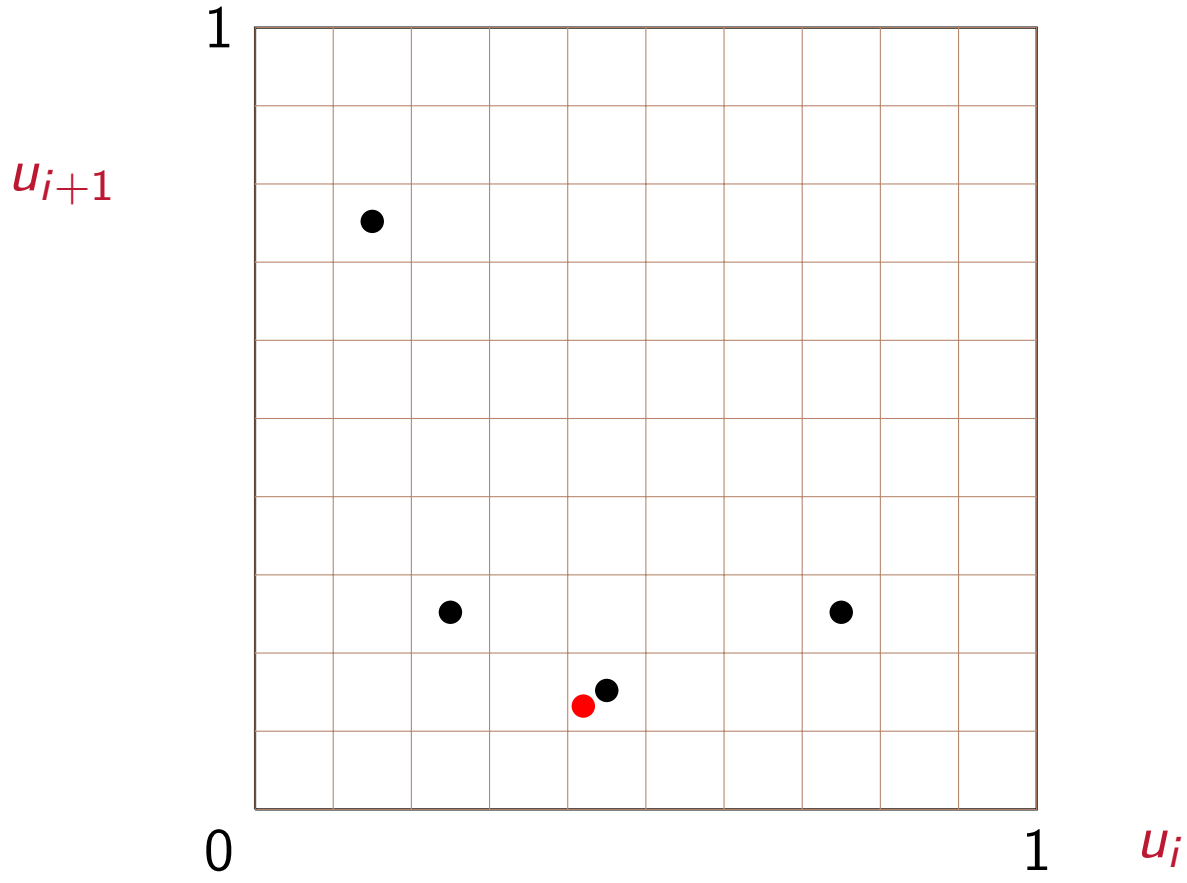
Throw $n = 10$ points in $k = 100$ boxes.

Examples: Serial and collision tests



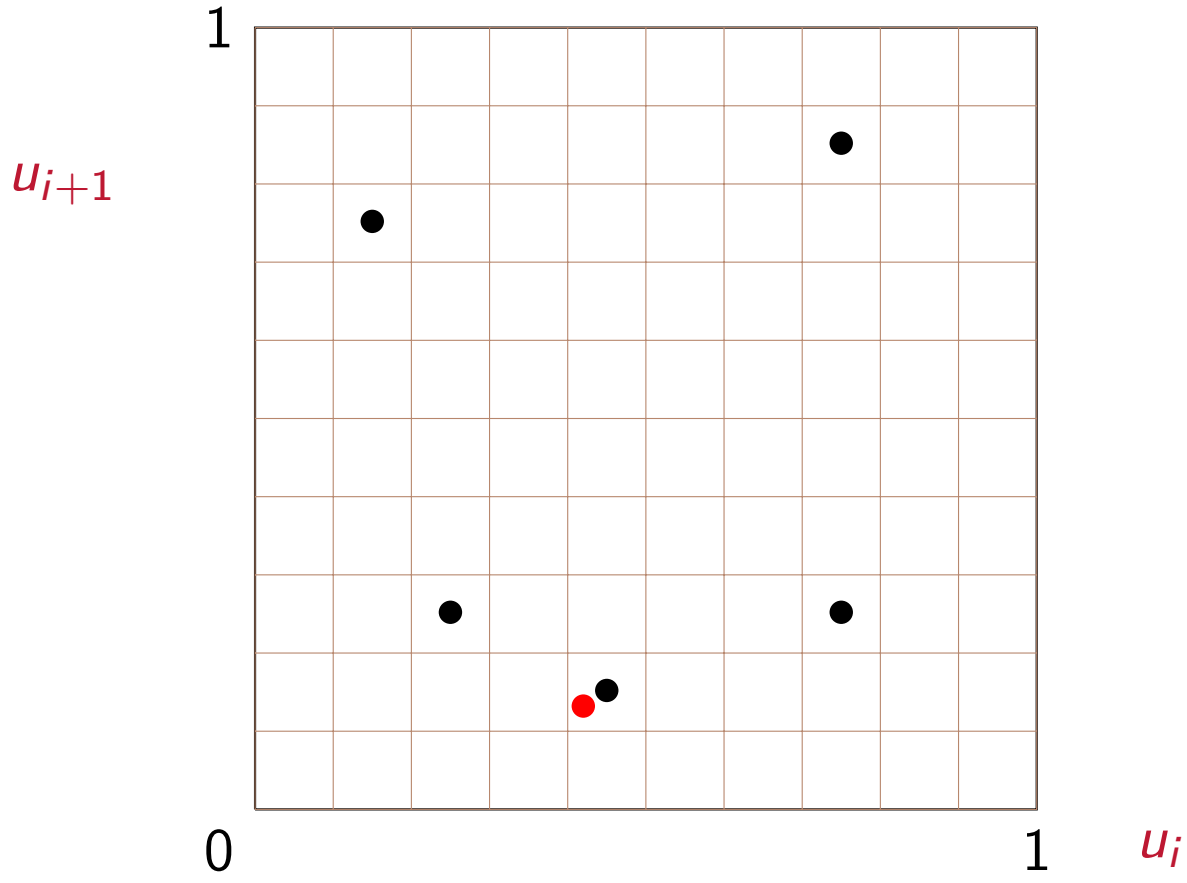
Throw $n = 10$ points in $k = 100$ boxes.

Examples: Serial and collision tests



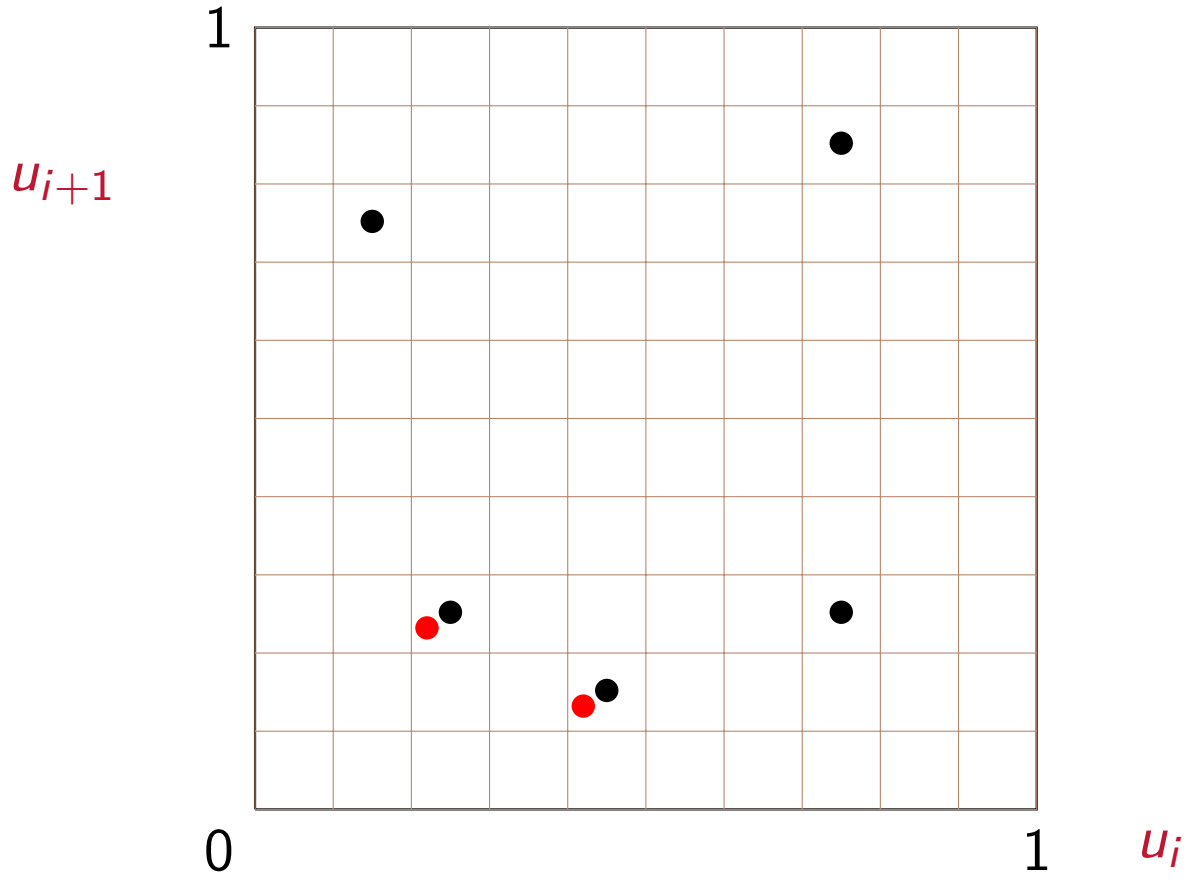
Throw $n = 10$ points in $k = 100$ boxes.

Examples: Serial and collision tests



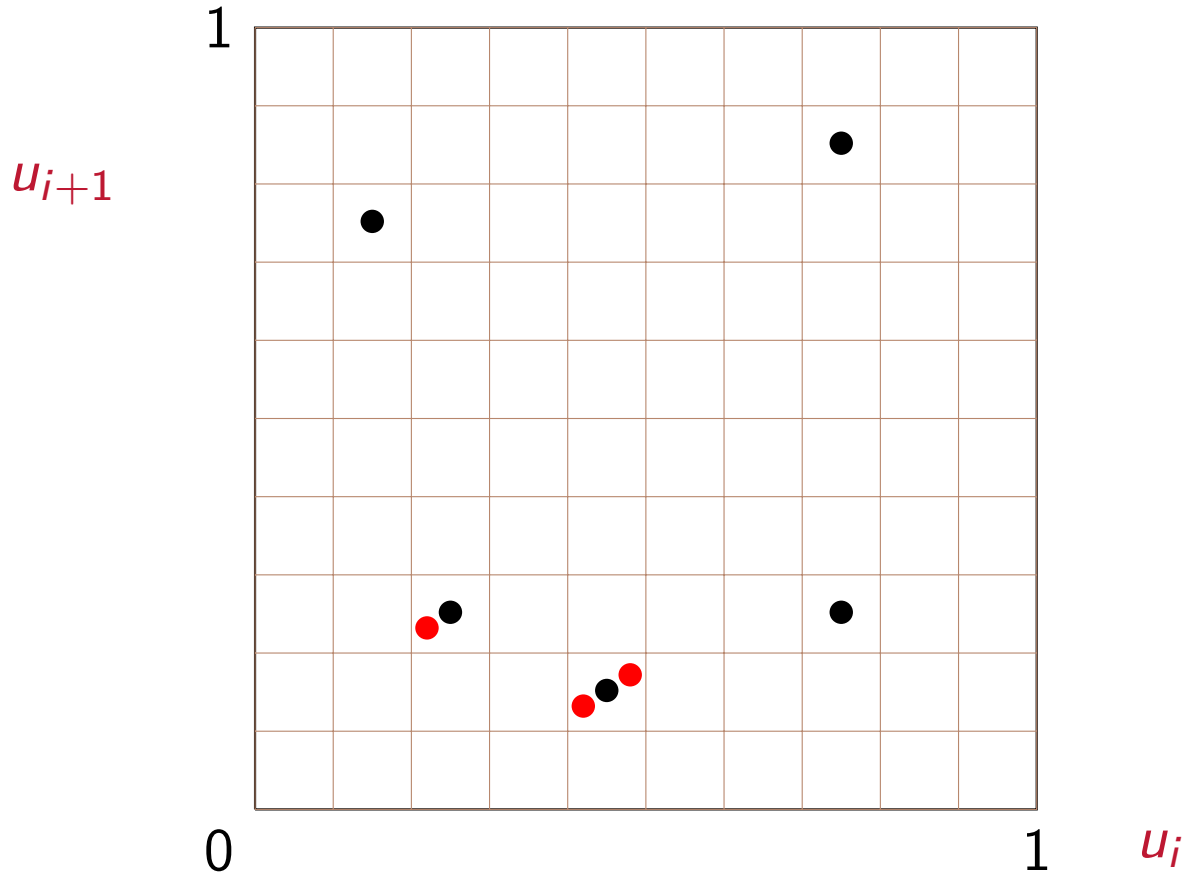
Throw $n = 10$ points in $k = 100$ boxes.

Examples: Serial and collision tests



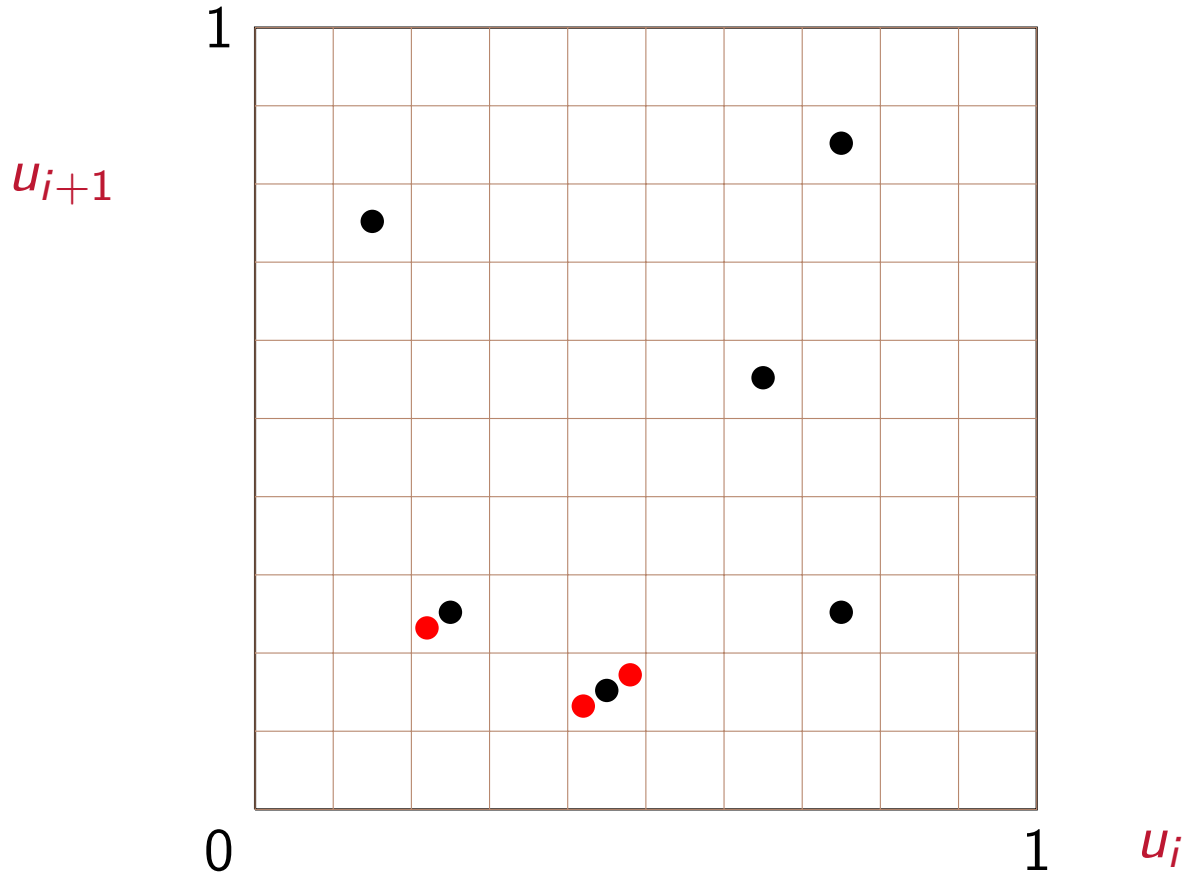
Throw $n = 10$ points in $k = 100$ boxes.

Examples: Serial and collision tests



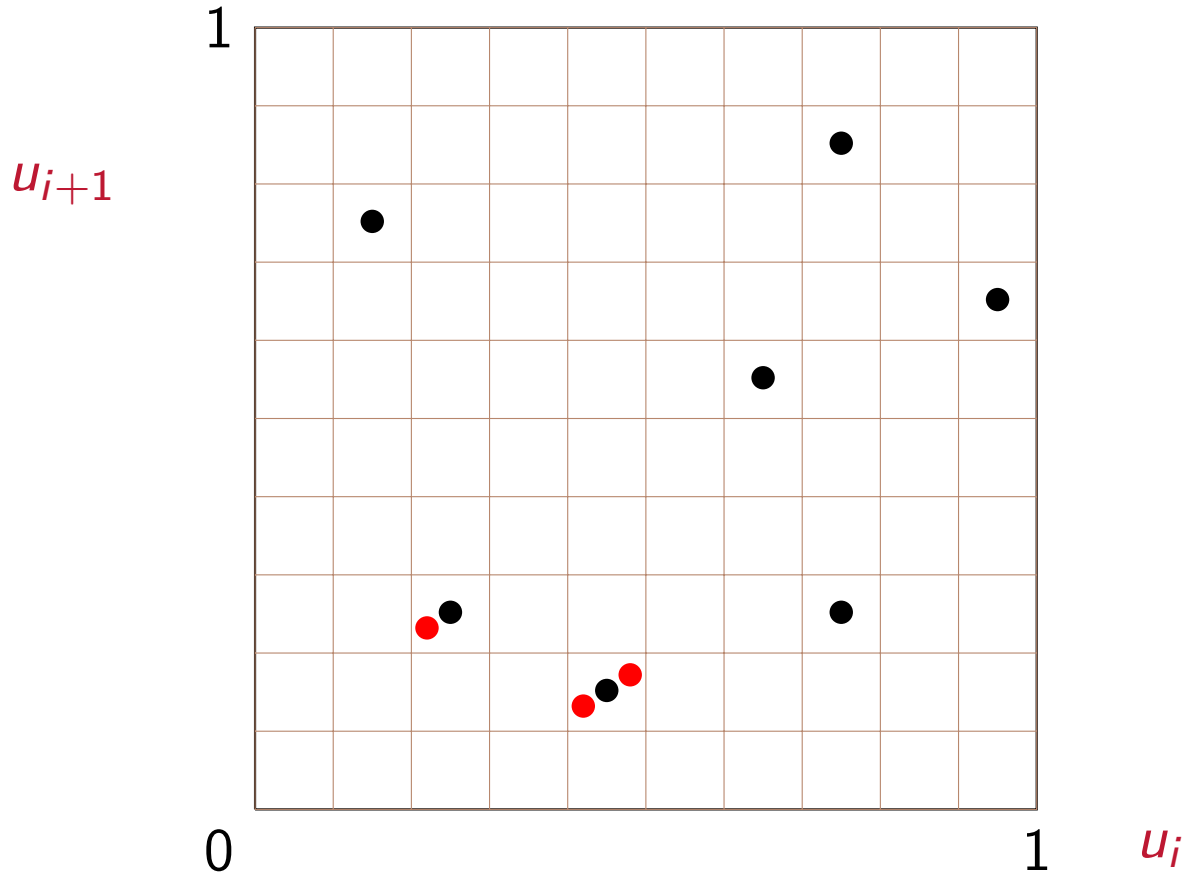
Throw $n = 10$ points in $k = 100$ boxes.

Examples: Serial and collision tests



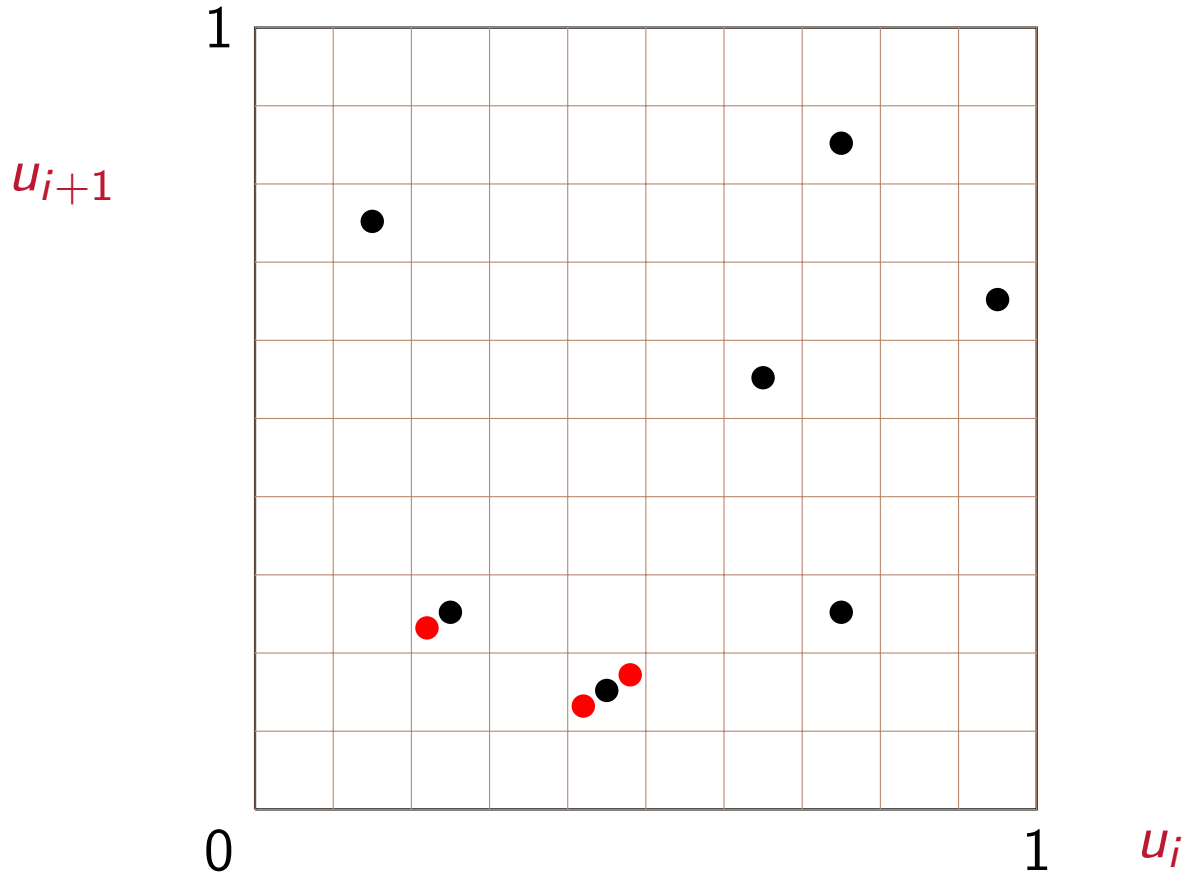
Throw $n = 10$ points in $k = 100$ boxes.

Examples: Serial and collision tests



Throw $n = 10$ points in $k = 100$ boxes.

Examples: Serial and collision tests



Throw $n = 10$ points in $k = 100$ boxes.

Here we observe 3 collisions. $\mathbb{P}[C \geq 3 \mid \mathcal{H}_0] \approx 0.144$.

Collision test

Partition $[0,1)^s$ in $k = d^s$ cubic boxes of equal size.
Generate n points $(u_{is}, \ldots, u_{is+s-1})$ in $[0,1)^s$.

$$C = \text{number of collisions.}$$

Collision test

Partition $[0,1)^s$ in $k = d^s$ cubic boxes of equal size.
Generate n points $(u_{is}, \ldots, u_{is+s-1})$ in $[0,1)^s$.

$$C = \text{number of collisions.}$$

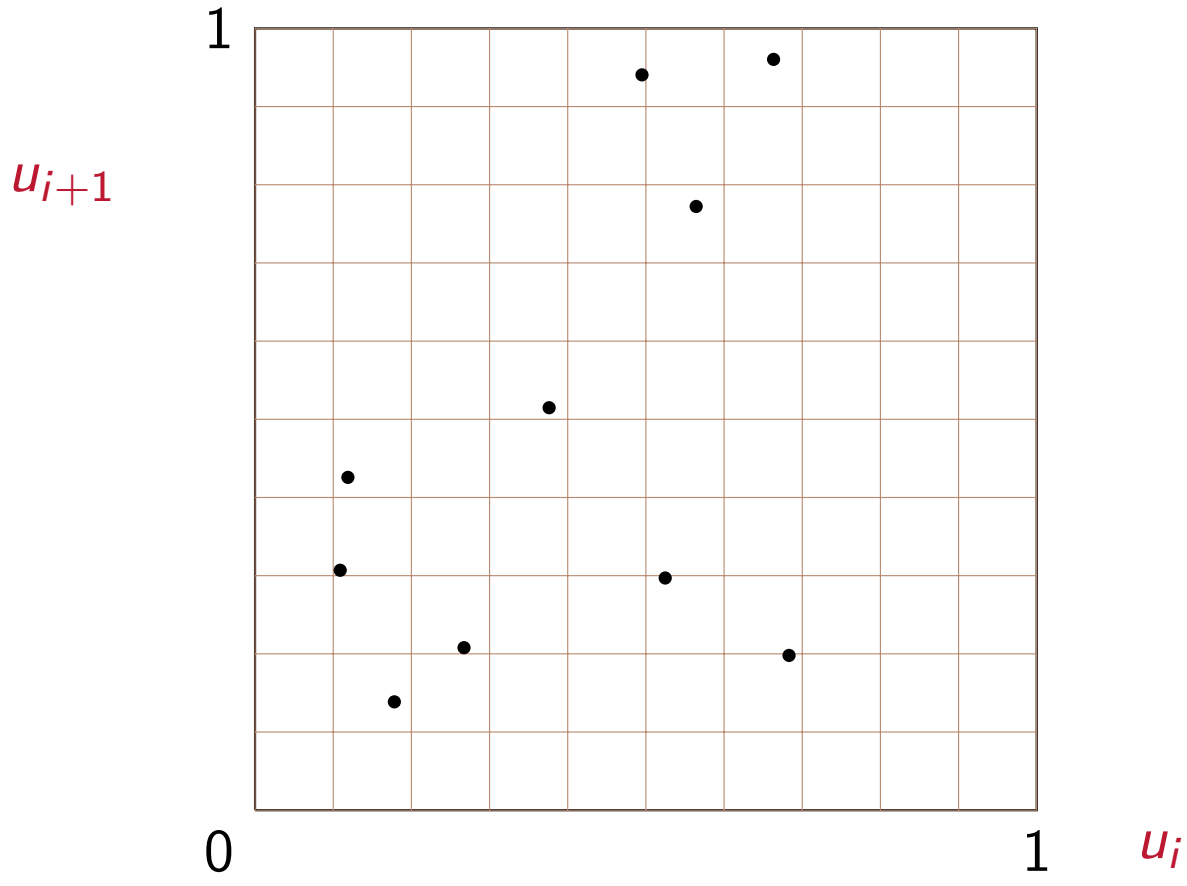
Under $\mathcal{H}_0$, $C \approx$ Poisson of mean $\lambda = n^2/(2k)$, if k is large and λ is small.

If we observe c collisions, we compute the p-values:

$$\begin{aligned} p^+(c) &= \mathbb{P}[X \geq c \mid X \sim \text{Poisson}(\lambda)], \\ p^-(c) &= \mathbb{P}[X \leq c \mid X \sim \text{Poisson}(\lambda)], \end{aligned}$$

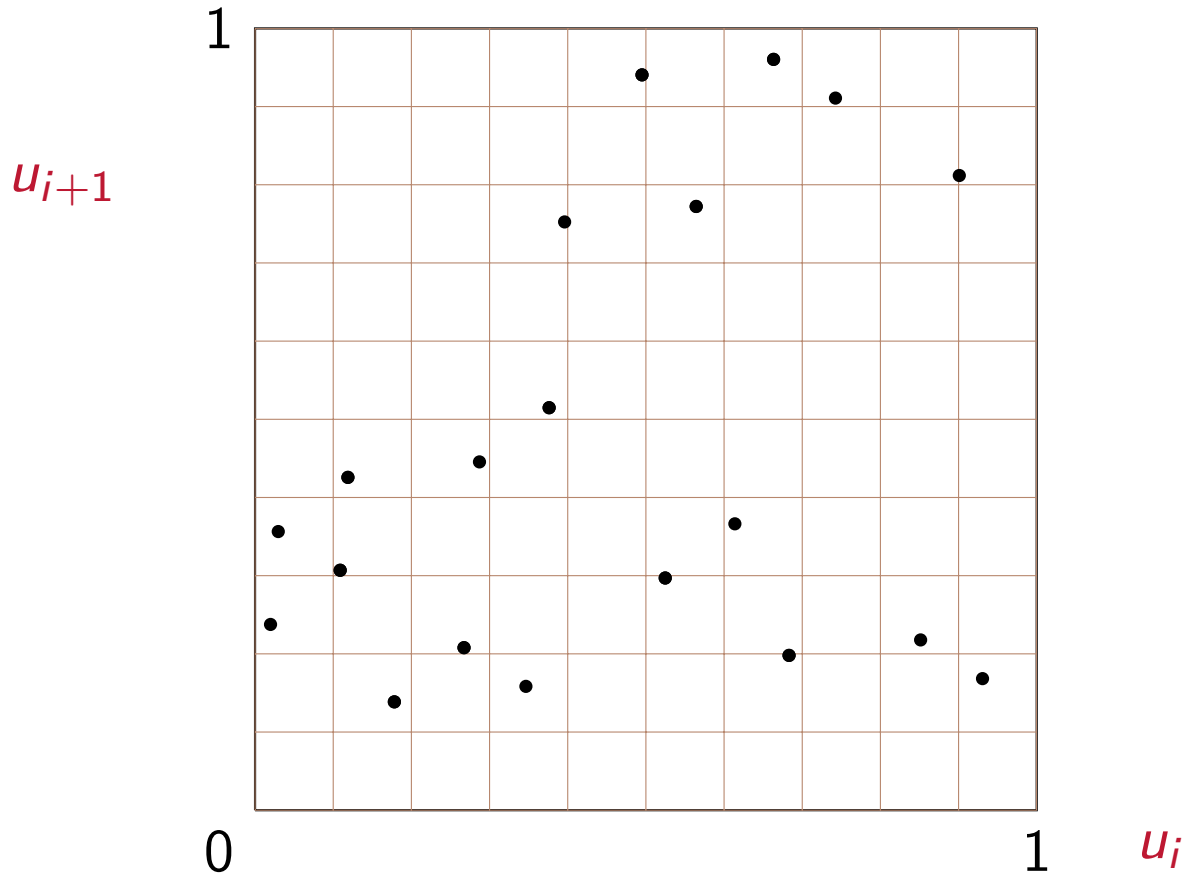
We reject $\mathcal{H}_0$ if $p^+(c)$ is too close to 0 (too many collisions)
or $p^-(c)$ is too close to 1 (too few collisions).

Example: LCG with $m = 101$ and $a = 12$:



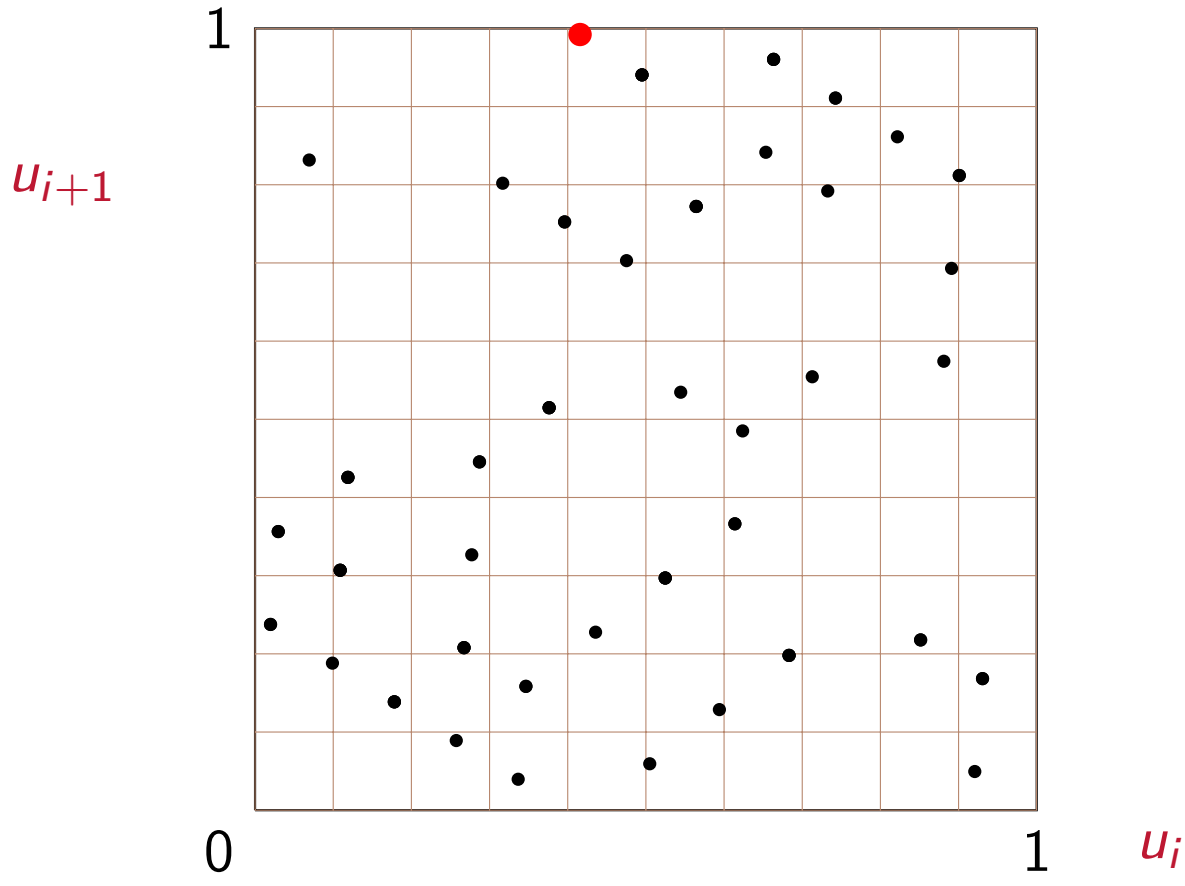
n	λ	C	$p^-(C)$
10	1/2	0	0.6281

Example: LCG with $m = 101$ and $a = 12$:



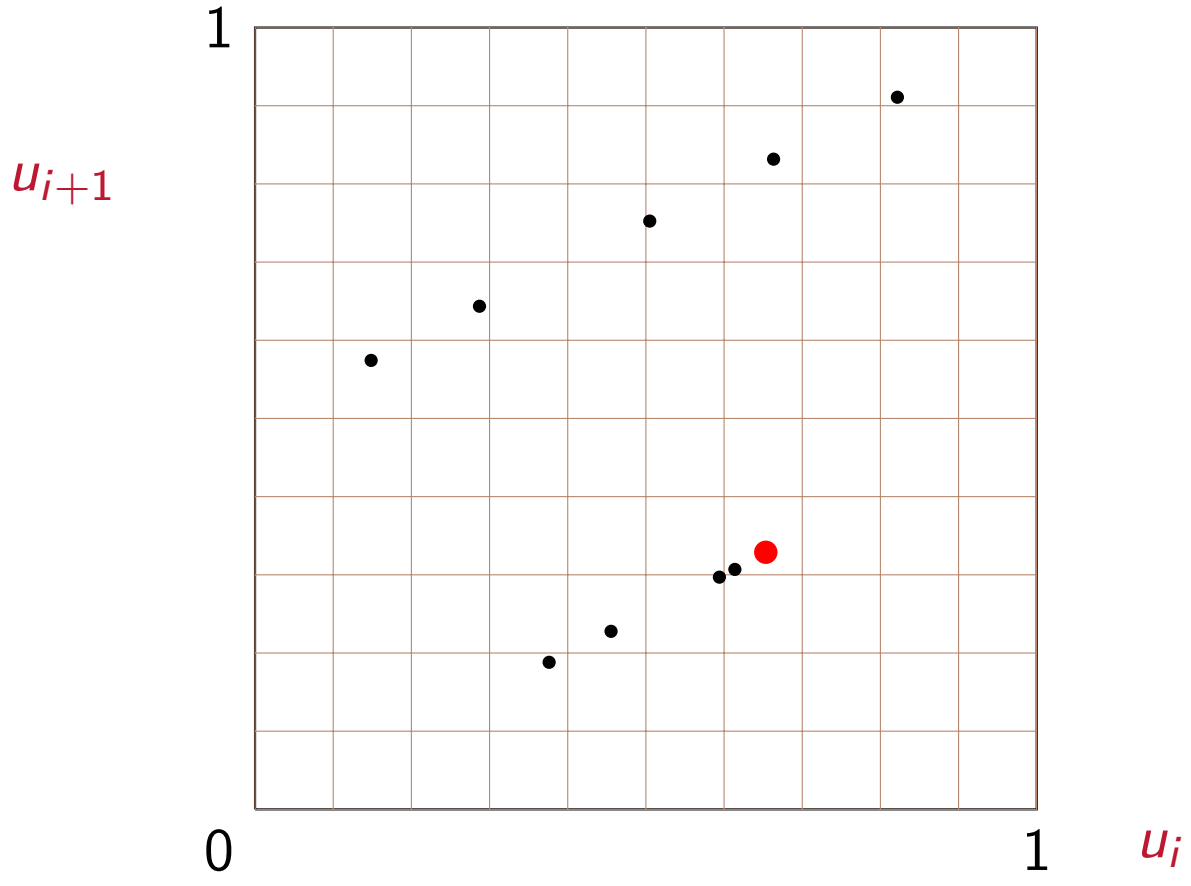
n	λ	C	$p^-(C)$
10	$1/2$	0	0.6281
20	2	0	0.1304

Example: LCG with $m = 101$ and $a = 12$:



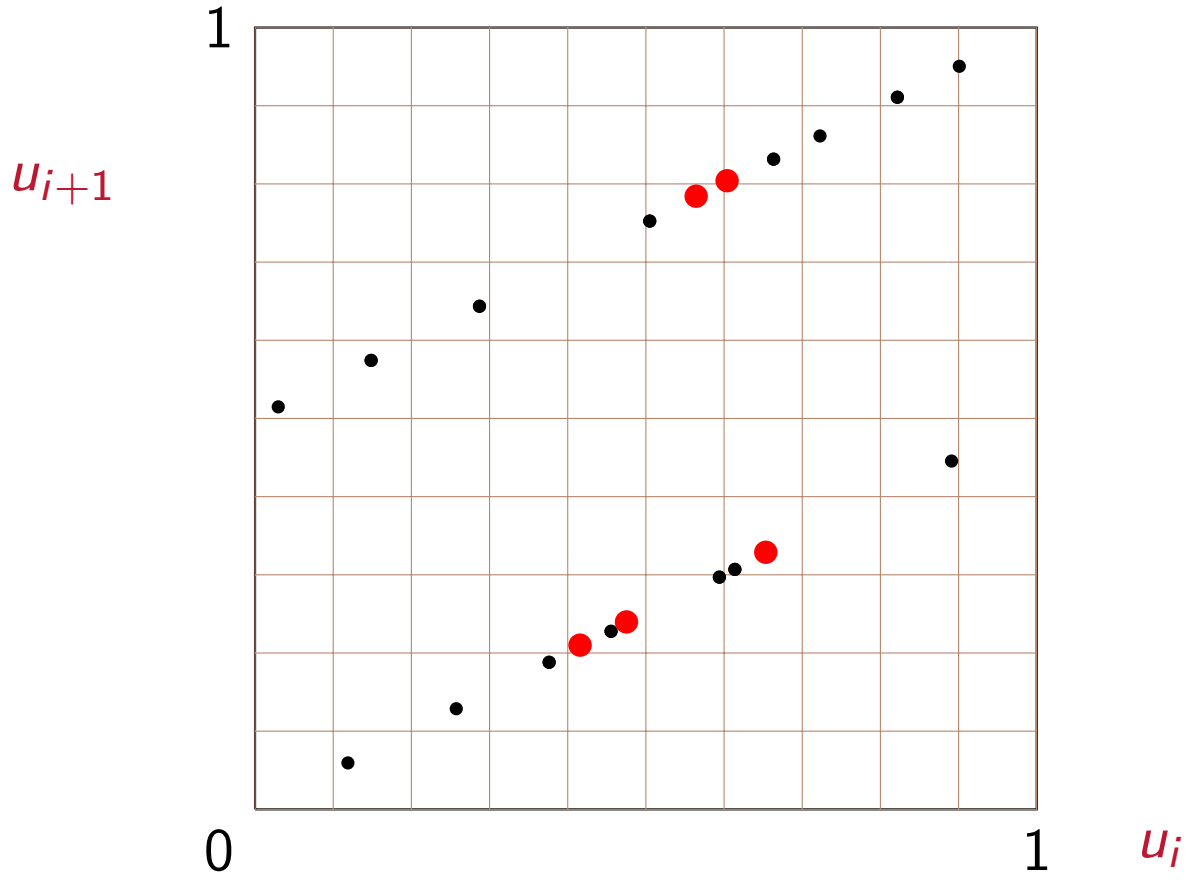
n	λ	C	$p^-(C)$
10	1/2	0	0.6281
20	2	0	0.1304
40	8	1	0.0015

LCG with $m = 101$ and $a = 51$:



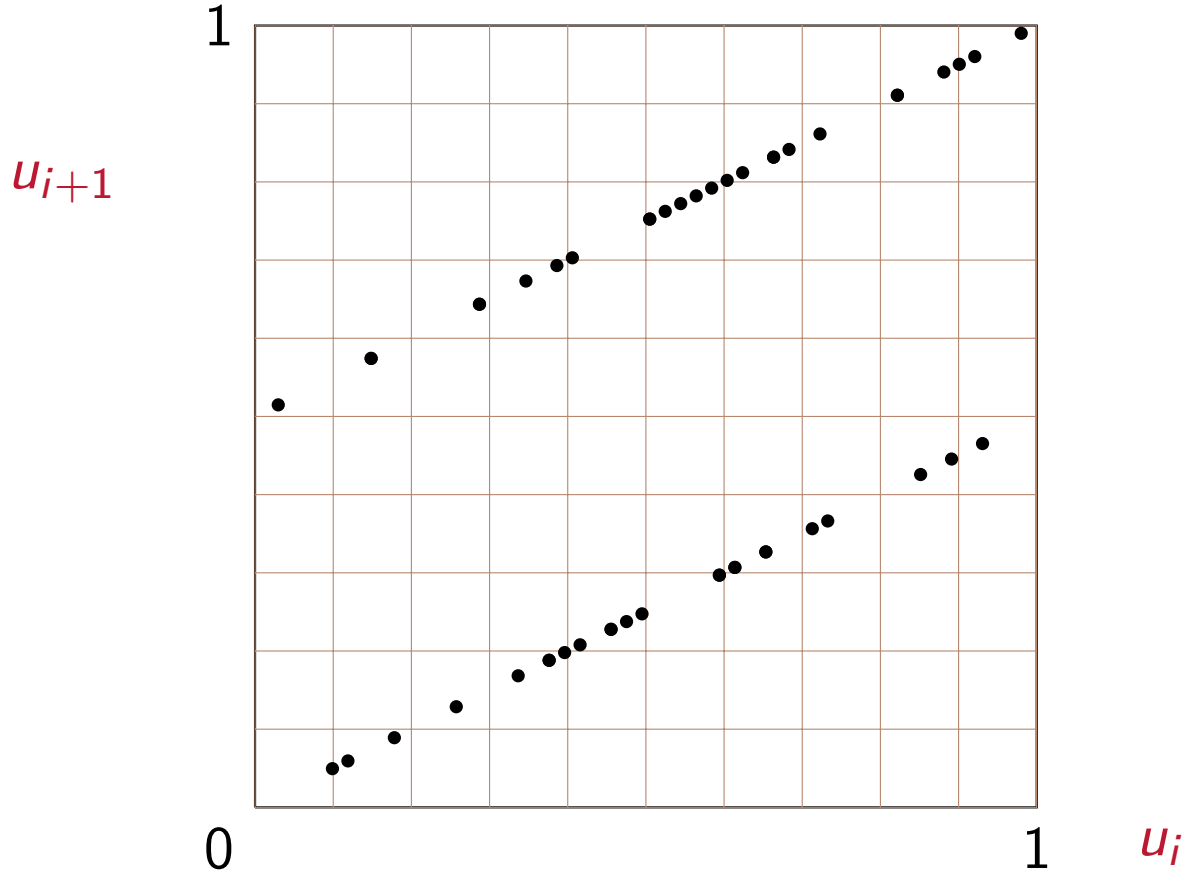
n	λ	C	$p^+(C)$
10	1/2	1	0.3718

LCG with $m = 101$ and $a = 51$:



n	λ	C	$p^+(C)$
10	1/2	1	0.3718
20	2	5	0.0177

LCG with $m = 101$ and $a = 51$:



n	λ	C	$p^+(C)$
10	$1/2$	1	0.3718
20	2	5	0.0177
40	8	20	2.2×10^{-9}

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 ℓ_s (the frequency) is the length of the shortest nonzero vector in the dual lattice. They suggested comparing ℓ_s with its theoretical upper bound for any given m.

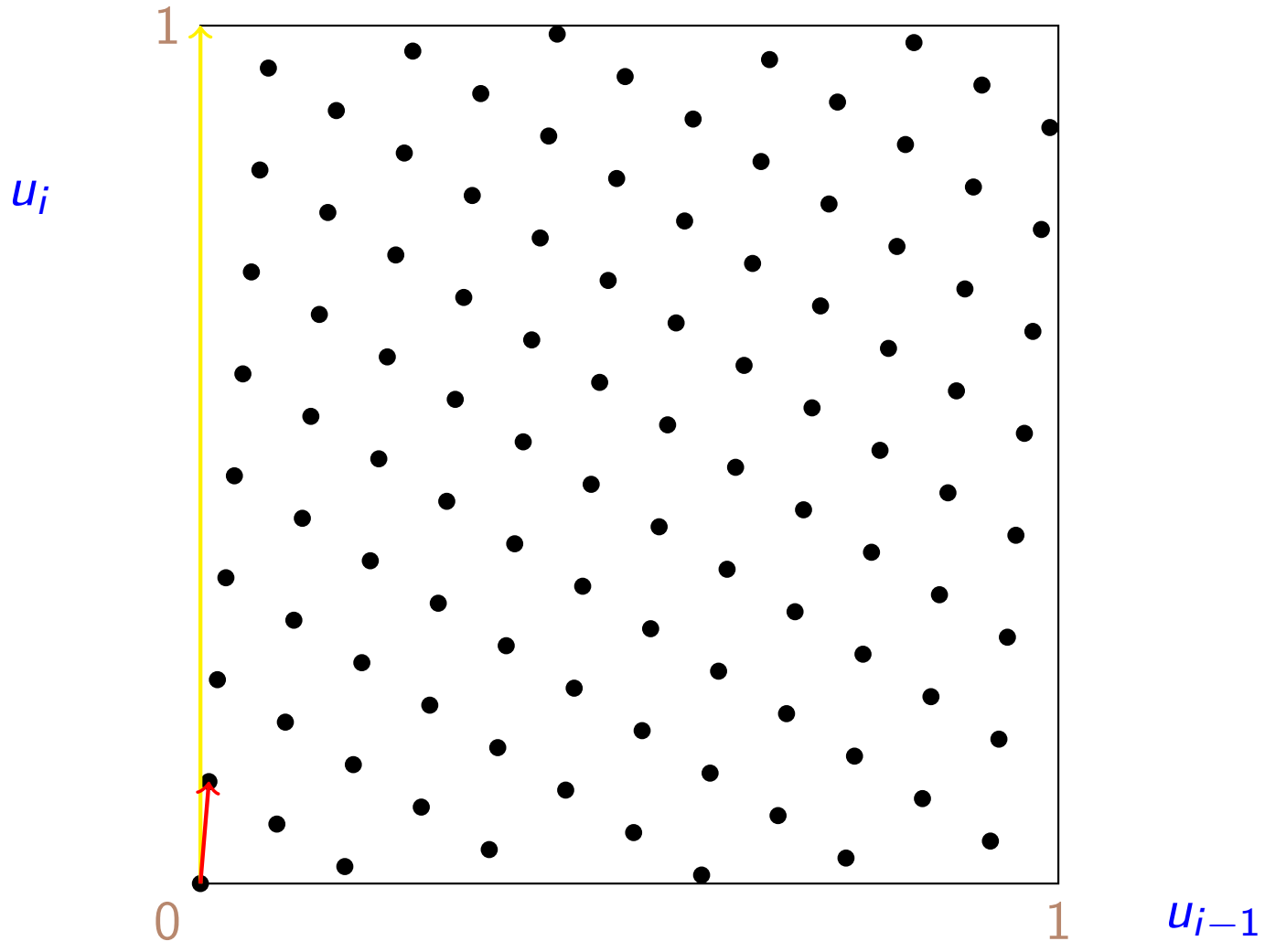
The spectral test

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 ℓ_s (the frequency) is the length of the shortest nonzero vector in the dual lattice. They suggested comparing ℓ_s with its theoretical upper bound for any given m.

They proposed an algorithm to compute ℓ_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.



$$x_i = 12\,x_{i-1} \bmod 101; \qquad u_i = x_i/101$$

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 $x_i = (x_{i-1} + x_{i-2}) \bmod m$.

Mitchell and Moore (1958): Lagged-Fibonacci, $x_i = (x_{i-r} + x_{i-k}) \bmod m$.
Bad structure: All points (u_{n-k}, u_{n-r}, u_n) belong to only two parallel planes in $[0,1)^3$. But still several versions in C and C++ libraries.

Zierler (1959) studied the general case:

$$x_i = (a_1 x_{i-1} + \cdots + a_k x_{i-k}) \bmod m.$$

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:
Grube (1973), L'Ecuyer, Blouin, Couture (1988).

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

Marsaglia and Zaman (1991): Add-with-carry and Subtract-with-borrow.
Modulo $m = 2^e$, huge period, near m^k.

Example: subtract-with-borrow (SWB)

State $(x_{n-48}, \ldots, x_{n-1}, c_{n-1})$ where $x_n \in \{0, \ldots, 2^{31} - 1\}$ and $c_n \in \{0, 1\}$:

$$
\begin{aligned}
x_n &= (x_{n-8} - x_{n-48} - c_{n-1}) \bmod 2^{31}, \\
c_n &= 1 \text{ if } x_{n-8} - x_{n-48} - c_{n-1} < 0, \qquad c_n = 0 \text{ otherwise}, \\
u_n &= x_n / 2^{31},
\end{aligned}
$$

Period $\rho \approx 2^{1479} \approx 1.67 \times 10^{445}$.

Example: subtract-with-borrow (SWB)

State $(x_{n-48}, \ldots, x_{n-1}, c_{n-1})$ where $x_n \in \{0, \ldots, 2^{31} - 1\}$ and $c_n \in \{0, 1\}$:

$$\begin{aligned} x_n &= (x_{n-8} - x_{n-48} - c_{n-1}) \bmod 2^{31}, \\ c_n &= 1 \text{ if } x_{n-8} - x_{n-48} - c_{n-1} < 0, \qquad c_n = 0 \text{ otherwise}, \\ u_n &= x_n / 2^{31}, \end{aligned}$$

Period $\rho \approx 2^{1479} \approx 1.67 \times 10^{445}$.

In Mathematica versions ≤ 5.2:
modified SWB with output $\tilde{u}_n = x_{2n}/2^{62} + x_{2n+1}/2^{31}$.

Great generator?

All points $(u_n, u_{n+40}, u_{n+48})$ belong to only two parallel planes in $[0,1)^3$.

Ferrenberg et Landau (1991). "Critical behavior of the three-dimensional Ising model: A high-resolution Monte Carlo study."

Ferrenberg, Landau et Wong (1992). "Monte Carlo simulations: Hidden errors from "good" random number generators."

All points $(u_n, u_{n+40}, u_{n+48})$ belong to only two parallel planes in $[0,1)^3$.

Ferrenberg et Landau (1991). "Critical behavior of the three-dimensional Ising model: A high-resolution Monte Carlo study."

Ferrenberg, Landau et Wong (1992). "Monte Carlo simulations: Hidden errors from "good" random number generators."

Tezuka, L'Ecuyer, and Couture (1993). "On the Add-with-Carry and Subtract-with-Borrow Random Number Generators."

Couture and L'Ecuyer (1994) "On the Lattice Structure of Certain Linear Congruential Sequences Related to AWC/SWB Generators."

Collision test of Subtract-with-Borrow in (old) Mathematica 5.2

For the unit cube $[0,1)^3$, divide each axis in $d = 100$ equal intervals. This gives $k = 100^3 = 1$ million boxes.

Generate $n = 10\,000$ vectors in 25 dimensions: $(U_0, \ldots, U_{24})$.
For each, note the box where (U_0, U_{20}, U_{24}) falls.
Here, expected number of collisions under $\mathcal{H}_0$ is $\lambda = 50$.

Collision test of Subtract-with-Borrow in (old) Mathematica 5.2

For the unit cube $[0,1)^3$, divide each axis in $d = 100$ equal intervals. This gives $k = 100^3 = 1$ million boxes.

Generate $n = 10\,000$ vectors in 25 dimensions: $(U_0, \ldots, U_{24})$.
For each, note the box where (U_0, U_{20}, U_{24}) falls.
Here, expected number of collisions under $\mathcal{H}_0$ is $\lambda = 50$.

Results: $C = 2070,\ 2137,\ 2100,\ 2104,\ 2127,\ \ldots.$

Collision test of Subtract-with-Borrow in (old) Mathematica 5.2

For the unit cube $[0,1)^3$, divide each axis in $d = 100$ equal intervals. This gives $k = 100^3 = 1$ million boxes.

Generate $n = 10\,000$ vectors in 25 dimensions: $(U_0, \ldots, U_{24})$.
For each, note the box where (U_0, U_{20}, U_{24}) falls.
Here, expected number of collisions under $\mathcal{H}_0$ is $\lambda = 50$.

Results: $C = 2070,\ 2137,\ 2100,\ 2104,\ 2127,\ \ldots.$

With MRG32k3a: $C = 41,\ 66,\ 53,\ 50,\ 54,\ \ldots.$

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}.

L'Ecuyer (WSC 1986, 1988): Combined LCGs.

L'Ecuyer (1996, 1999): Combined MRG.

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.

Combined MRGs.

Two [or more] MRGs in parallel:

$$\begin{aligned} x_{1,n} &= (a_{1,1}x_{1,n-1} + \cdots + a_{1,k}x_{1,n-k}) \bmod m_1, \\ x_{2,n} &= (a_{2,1}x_{2,n-1} + \cdots + a_{2,k}x_{2,n-k}) \bmod m_2. \end{aligned}$$

One possible combinaison:

$$z_n := (x_{1,n} - x_{2,n}) \bmod m_1; \qquad u_n := z_n/m_1;$$

L'Ecuyer (1996): the sequence $\{u_n,\ n \geq 0\}$ is also the output of an MRG of modulus $m = m_1 m_2$, with small added "noise". The period can reach $(m_1^k - 1)(m_2^k - 1)/2$.

Permits one to implement efficiently an MRG with large m and several large nonzero coefficients.

Parameters: L'Ecuyer (1999); L'Ecuyer et Touzin (2000).
Implementations with multiple streams.

A recommendable generator: MRG32k3a

Choose six 32-bit integers:
x_{-2}, x_{-1}, x_0 in $\{0, 1, \ldots, 4294967086\}$ (not all 0) and
y_{-2}, y_{-1}, y_0 in $\{0, 1, \ldots, 4294944442\}$ (not all 0). For $n = 1, 2, \ldots$, let

$$
\begin{aligned}
x_n &= (1403580 x_{n-2} - 810728 x_{n-3}) \bmod 4294967087, \\
y_n &= (527612 y_{n-1} - 1370589 y_{n-3}) \bmod 4294944443, \\
u_n &= [(x_n - y_n) \bmod 4294967087]/4294967087.
\end{aligned}
$$

A recommendable generator: MRG32k3a

Choose six 32-bit integers:
x_{-2}, x_{-1}, x_0 in $\{0, 1, \ldots, 4294967086\}$ (not all 0) and
y_{-2}, y_{-1}, y_0 in $\{0, 1, \ldots, 4294944442\}$ (not all 0). For $n = 1, 2, \ldots$, let

$$\begin{aligned} x_n &= (1403580 x_{n-2} - 810728 x_{n-3}) \bmod 4294967087, \\ y_n &= (527612 y_{n-1} - 1370589 y_{n-3}) \bmod 4294944443, \\ u_n &= [(x_n - y_n) \bmod 4294967087]/4294967087. \end{aligned}$$

(x_{n-2}, x_{n-1}, x_n) visits each of the $4294967087^3 - 1$ possible values.
(y_{n-2}, y_{n-1}, y_n) visits each of the $4294944443^3 - 1$ possible values.

The sequence $u_0, u_1, u_2, \ldots$ is periodic, with 2 cycles of period

$$\rho \approx 2^{191} \approx 3.1 \times 10^{57}.$$

A recommendable generator: MRG32k3a

Choose six 32-bit integers:
x_{-2}, x_{-1}, x_0 in $\{0, 1, \ldots, 4294967086\}$ (not all 0) and
y_{-2}, y_{-1}, y_0 in $\{0, 1, \ldots, 4294944442\}$ (not all 0). For $n = 1, 2, \ldots$, let

$$\begin{aligned} x_n &= (1403580 x_{n-2} - 810728 x_{n-3}) \bmod 4294967087, \\ y_n &= (527612 y_{n-1} - 1370589 y_{n-3}) \bmod 4294944443, \\ u_n &= [(x_n - y_n) \bmod 4294967087]/4294967087. \end{aligned}$$

(x_{n-2}, x_{n-1}, x_n) visits each of the $4294967087^3 - 1$ possible values.
(y_{n-2}, y_{n-1}, y_n) visits each of the $4294944443^3 - 1$ possible values.

The sequence $u_0, u_1, u_2, \ldots$ is periodic, with 2 cycles of period

$$\rho \approx 2^{191} \approx 3.1 \times 10^{57}.$$

Robust and reliable for simulation.
Used by SAS, R, MATLAB, Arena, Automod, Witness, Spielo gaming, ...

Linear recurrences modulo 2

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

$$x_i = (a_1 x_{i-1} + \cdots + a_k x_{i-k}) \bmod 2, \qquad u_i = \sum_{\ell=1}^{w} x_{is+\ell-1} 2^{-\ell}.$$

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 ℓ bits of accuracy for all $\ell \le s$ and $t \le \lfloor k/s \rfloor$.

Tootill et al. (1973): introduced stronger property of asymptotically random (to w bits), also called maximally equidistributed (ME).
Means t-distributed to ℓ bits of accuracy whenever $\ell \le w$ and $t \le \lfloor k/\ell \rfloor$.
They found a specific ME Tausworthe generator with
$x_i = (x_{i-607} + x_{i-273}) \bmod 2$, $s = 512$, and $w = 23$.

Lewis et al. (1973): generalized feedback shift register (GFSR):
$\mathbf{y}_i = (\mathbf{y}_{i-r} \oplus \mathbf{y}_{i-q})$, 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:

$$\begin{aligned} \mathbf{x}_i &= \mathbf{A}\mathbf{x}_{i-1}, \\ \mathbf{y}_i &= \mathbf{B}\mathbf{x}_i, \\ u_i &= \sum_{\ell=1}^{w} y_{i,\ell-1} 2^{-\ell} = .y_{i,0}\ y_{i,1}\ y_{i,2}\ \cdots, \end{aligned}$$

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

Matsumoto and Kurita (1994): tempered TGFSR.

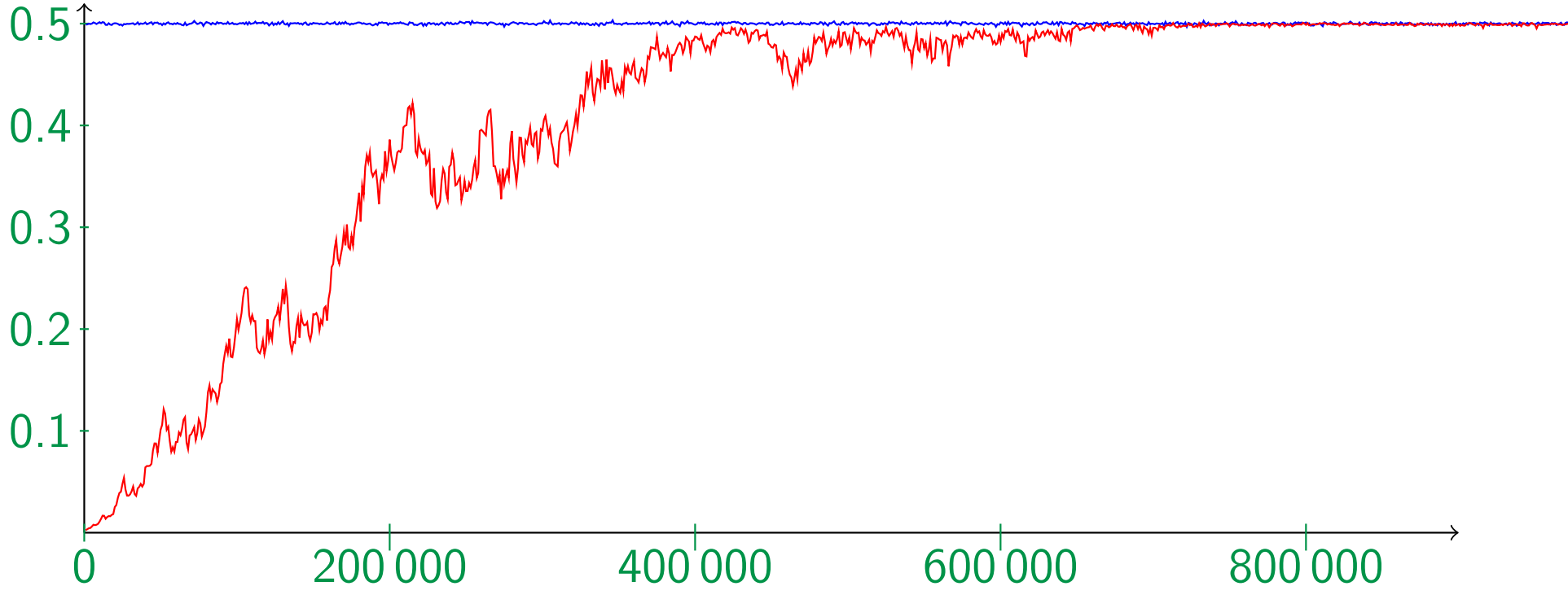
Matsumoto and Nishimura (1998): Mersenne Twister. `MT19937`.

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 $2^{44497} - 1$.

Impact of a matrix A that changes the state too slowly.

Experiment: take an initial state with a single bit at 1.
Try all k possibilities and take the average of the k values of u_n obtained for each n.

WELL19937 vs MT19937; moving average over 1000 iterations.



Combined linear recurrences modulo 2

Lindholm (1968): a primitive polynomial of $\mathbf{A}$ 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 $x_i = p(x_{i-1}) \bmod m$.

Knuth (1981): quadratic recurrence modulo m.

Marsaglia (1985): multiplicative Lagged Fibonacci.

L'Ecuyer and Granger-Piché (2003): combination of MRG with $\mathbb{F}_2$-linear. Proof of multivariate uniformity.

RNGs coming from cryptology:

Blum, Blum, and Schub (1986): BBS generator.
NIST (2001): AES 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".
In *Proceedings of the 2017 Winter Simulation Conference*: IEEE Press.

Coveyou, R. R., and R. D. MacPherson. 1967.
"Fourier Analysis of Uniform Random Number Generators".
Journal of the ACM 14:100–119.

Knuth, D. E. 1998.
The Art of Computer Programming, Volume 2: Seminumerical Algorithms. Third ed.
Reading, MA: Addison-Wesley.

L'Ecuyer, P. 2012.
"Random Number Generation".
In *Handbook of Computational Statistics* (second ed.)., edited by J. E. Gentle, W. Haerdle, and Y. Mori, 35–71. Berlin: Springer-Verlag.

L'Ecuyer, P., D. Munger, B. Oreshkin, and R. Simard. 2017.
"Random Numbers for Parallel Computers: Requirements and Methods, with Emphasis on GPUs".
Mathematics and Computers in Simulation 135:3–17.
Open access at `http://dx.doi.org/10.1016/j.matcom.2016.05.005`.


L'Ecuyer, P., and F. Panneton. 2009.
"$\mathbf{F}_2$-Linear Random Number Generators".
In *Advancing the Frontiers of Simulation: A Festschrift in Honor of George Samuel Fishman*, edited by C. Alexopoulos, D. Goldsman, and J. R. Wilson, 169–193. New York: Springer-Verlag.


L'Ecuyer, P., and R. Simard. 2007, August.
"TestU01: A C Library for Empirical Testing of Random Number Generators".
ACM Transactions on Mathematical Software 33 (4): Article 22.


L'Ecuyer, P., R. Simard, E. J. Chen, and W. D. Kelton. 2002.
"An Object-Oriented Random-Number Package with Many Long Streams and Substreams".
Operations Research 50 (6): 1073–1075.


Lehmer, D. H. 1951.
"Mathematical Methods in Large Scale Computing Units".
The Annals of the Computation Laboratory of Harvard University 26:141–146.