How to Create and Use Random Numbers

Christiane Lemieux

Department of Statistics and Actuarial Science

Faculty of Mathematics

University of Waterloo

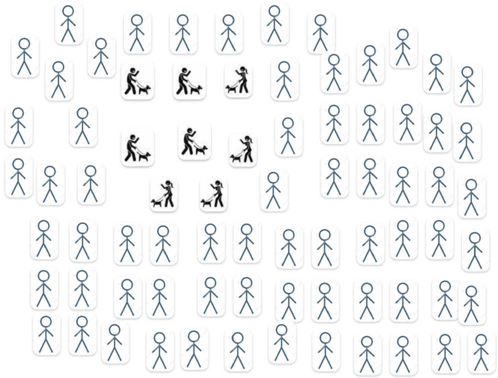
WiMWiM Series
April 4, 2024

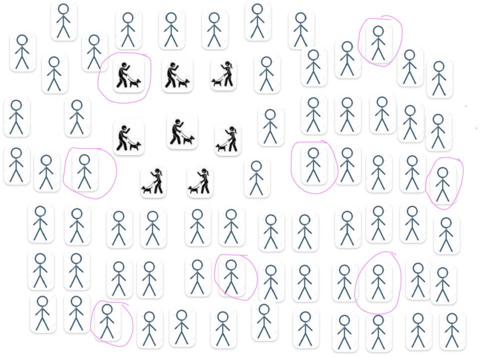
Why/When Are Random Numbers Useful?

- Games (to avoid repeated patterns and make it more fun!)
- Polls/Surveys
  - Cannot ask WHOLE population to answer question
  - Use a SAMPLE from which we'll INFER how the whole population would have responded
  - Can only do this if sample is truly RANDOM
- AI: randomness used in exploration, training, and estimation phases of many AI algorithms

Randomness as Key Ingredient for Survey/Poll

Q: What proportion of ABC Village residents prefer dogs over cats?





Randomly select people in ABC Village

Need a way to sample randomly

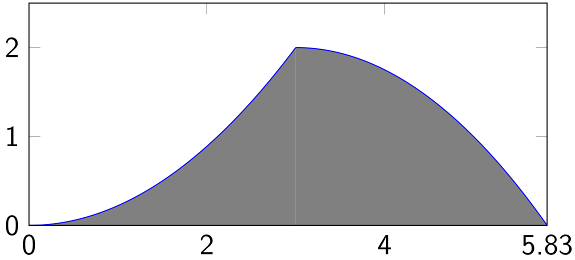
PLAN FOR TODAY

1. Methods that use random numbers to solve problems
2. How to generate random numbers on a computer
3. Quasi-random numbers

A mix of mathematics, statistics and computer science!

Random numbers to compute surface area

- By now you've learned how to easily compute the surface areas of **regular shapes** such as a square, circle, triangle, etc, as long as you have the required measurements
- What if you get an irregular shape?

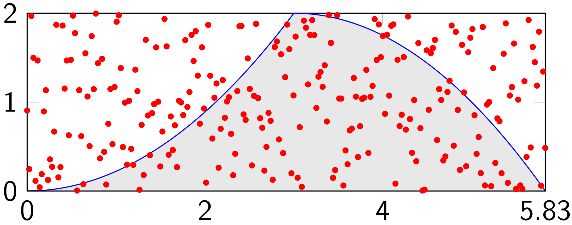


- What if you need to compute the volume of an irregular solid?

Random numbers to compute surface area

Idea I: Find a regular shape to put around the irregular one; randomly choose N points inside the regular shape and count how many are inside your irregular shape (call this n); estimate the surface area by

$$R \times \frac{n}{N} \text{ where } R = \text{surface area of regular shape}$$



Called the **"Hit-and-Miss" Method**

Hit-and-Miss: Why does it work?

Need a key ingredient: **expected value of a random variable**

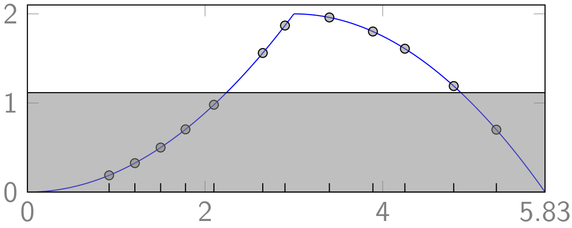
Let X be the result when you roll a balanced die.
The **expected value** of X is the weighted average of all possible values X can take, where the weight is given by the probability that X takes this value. We write it as $\mathrm{E}(X)$.
What is the value of $\mathrm{E}(X)$?
Let Y be the maximum value you get when you roll two dice. What is the value of $\mathrm{E}(Y)$?

Hit-and-Miss: Why does it work?

Random numbers to compute surface area

Idea II: Randomly choose N points on the x axis; measure the height of the shape at those points (get N measurements $h_1, h_2, \ldots, h_N$); estimate the surface area by a rectangle whose height is the AVERAGE height measured over the random points, given by

$$\text{base} \times \frac{1}{N}(h_1 + \ldots + h_N)$$



Called the **"Monte Carlo Method"**

Playing with these ideas on a computer

	$N = 10$	$N = 100$	$N = 1000$	$N = 10000$	$N = 100000$
Hit-and-Miss	4.66	5.48	5.77	5.79	5.77
Monte Carlo	5.65	5.69	5.59	5.73	5.77

...and the true answer is $\ldots 5.77124$

$$\int_0^3 \frac{2}{9}x^2 dx + \int_3^{3+\sqrt{8}} (2 - 0.25(x-3)^2) dx = \frac{2}{9} \times 2 + 2\sqrt{8} - \frac{1}{12}(\sqrt{8})^3 = 5.77124$$

Let's look at some Python code implementing these ideas.

Q: What is behind the **random.uniform** function in python?

How to generate numbers on a computer

How can one generate "true" randomness?

- Dice, balls in an urn, spinner, etc.
- But what if we need millions of them very quickly?
- Could use physical devices (e.g., based on principles of quantum mechanics) $\Rightarrow$ not ideal (measurement errors, reproducibility, speed, . . .)

Instead, we generate **pseudo-random numbers (PRNs)** using **pseudo-random number generators (PRNGs)**.

- "pseudo" because they look random but are in fact "deterministic" (not random)
- Means that eventually, the same sequence of numbers starts appearing again (periodic behavior)

Pseudo-random number generators (PRNG)

A **good PRNG** should

- produce random variates $u_1, \ldots, u_n$ (PRNs) that look random (can use **theoretical and statistical tests** to support this assumption)
- allow to set a seed for reproducibility
- have a large period
- be fast
- should be easy to understand and implement.

Middle-Square Method

One of the first pseudorandom number generators that was used for simulation was the **"middle square method"** by John von Neumann in 1949, which works as follows:

1. Start with a 4-digit positive integer Z_0 and square it to obtain an integer with up to 8 digits; if necessary, append zeros to the left to make it exactly eight digits.
2. Take the middle four digits of this eight-digit number as the next four-digit number, Z_1.
3. Place a decimal point at the left of Z_1 to obtain the first "$U(0,1)$ number," U_1.
4. Then let Z_2 be the middle four digits of Z_1^2 and let U_2 be Z_2 with a decimal point to the left, and so on.

$Z_0 = 2372 \Rightarrow Z_0^2 = 05626384 \Rightarrow Z_1 = 6263 \Rightarrow U_1 = 0.6263$
$\Rightarrow Z_1^2 = 39225169, Z_2 = 2251, U_2 = 0.2251, \ldots$

Practice

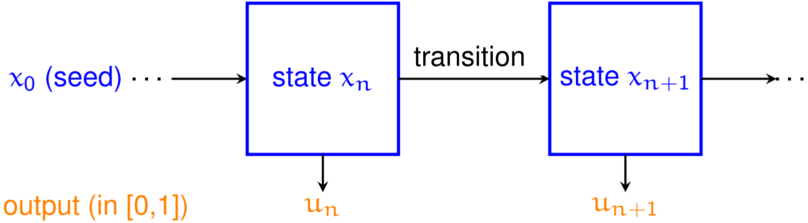
Starting with $Z_0 = 6543$, determine the first four numbers U_1 to U_4 output by this PRNG.

Problems with Middle-Square Method

1. Period is no larger than 10^4. Why?
2. If the middle 4 digits are all zeroes, the generator gets stuck and output 0 forever.
3. If the first half of a number in the sequence is zeroes, the subsequent numbers will be decreasing to zero. (Try $Z_0 = 3001$)
4. Can also get stuck on certain values: (Try $Z_0 = 2500$).
5. Other bad choices with very short cycles: (Try $Z_0 = 0540$)

Basic Principles for Pseudorandom Number Generator (PRNG)

- PRNG usually output (pseudorandom) numbers between 0 and 1
- PRNG works by applying a **transition** function to a **state**, and then an **output transformation** from the state to a (pseudorandom)number between 0 and 1



- state is typically a whole number (or a list of whole numbers) between 0 and $m-1$ where m is a **large** whole number
- if state x_i returns to x_0, the sequence starts repeating itself (period length of i)
- to make sure we stay in the range $\{0, 1, \ldots, m-1\}$ we need **modular arithmetic**

Tool for PRNG: Modular Arithmetic

Linear Congruential Generators

- Lehmer in 1951 introduced **linear congruential generators (LCGs)** which are PRNGs recursively defined by

$$x_n = a x_{n-1} \mod m, \quad n \geqslant 0,$$

  with multiplier a, modulus $m \geqslant 0$ and seed x_0.
- Maximum period of an LCG is $m - 1$. Why?
- Maxium period is reached if a is a **primitive element mod** m... Means smallest positive integer r such that $a^r \mod m = 1$ is $r = m - 1$.
- Q: find a primitive element mod 7

LCGs

$$x_n = a x_{n-1} \mod m,$$

To obtain PRNs, simplest output function is $u_n = \frac{x_n}{m} \in [0, 1)$.

Toy Example: $m = 11$, $a = 6$, $x_0 = 1 \Rightarrow x_n = 6x_{n-1}$ mod 11,
What sequence $u_0, u_1, u_2, \ldots$ do you get? What is the period?

Multiple Recursive Generator

Idea: look back more than one state, e.g., use

$$x_n = ax_{n-1} + bx_{n-2} + cx_{n-3} \bmod m$$

MRG32k3a

Combined MRG from P. L'Ecuyer (Montreal) with 2 components and for which

$$\begin{aligned}
x_{1,n} &= (1403580 x_{1,n-2} - 810728 x_{1,n-3}) \bmod (2^{32} - 209), \\
x_{2,n} &= (527612 x_{2,n-1} - 1370589 x_{2,n-3}) \bmod (2^{32} - 22853), \\
z_n &= (x_{1,n} - x_{2,n}) \bmod (2^{32} - 209), \\
u_n &= z_n/(2^{32} - 209).
\end{aligned}$$

- The parameters of this generator were found through extensive searches based on theoretical and statistical tests.
- Period of about 2^{191}. This is 3138550867693340381917894711603833208051177722232017256448
- Code available online at `http://simul.iro.umontreal.ca/rng/MRG32k3a.c`

Quasi-Random Numbers

- Random samples can be irregular (clusters of points, large gaps with no points)
- Since computer is already creating “fake” numbers, could we not make them be less irregular, more uniformly distributed?
- This is the idea behind **quasi-random numbers** also referred to as low-discrepancy point sets or sequences

Low-discrepancy point sets

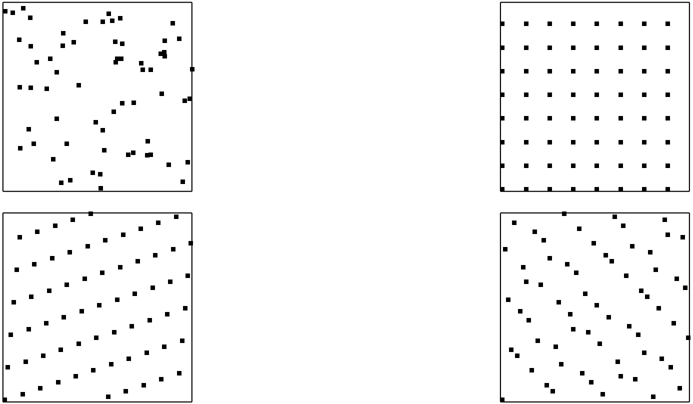


Figure: Four different point sets with $n = 64$: pseudorandom (top left), rectangular grid (top right), Korobov lattice (bottom left), and Sobol' (bottom right).

Low-discrepancy sequences: a first example

In one dimension, we can construct a sequence of points $u_0, u_1, \ldots$ with a low discrepancy as follows:

1. Choose a base b
2. To define u_i:
   - expand i in base b, i.e., write $i = a_0 + a_1 b + a_2 b^2 + a_3 b^3 + \ldots$:

     e.g., for $i = 5$ and $b = 2$ write 5='101', i.e., $5 = (2^0 + 2^2)$ so $a_0 = a_2 = 1$ and all other a_l's are 0.

   - apply *radical-inverse function*:
     $u_i = S_b(i) := a_0 \frac{1}{b} + a_1 \frac{1}{b^2} + a_2 \frac{1}{b^3} + \ldots,$

     e.g., for $i = 5$ and $b = 2$ we get $u_5 = S_2(5) = 1 \times 2^{-1} + 1 \times 2^{-3} = 5/8$

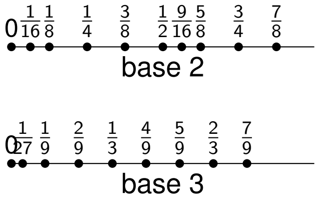
     Try it: What is $S_3(5)$?

This yields the *van der Corput sequence in base* b*, denoted* S_b *(goes back to 1935)*

van der Corput Sequence in base 2

Practice: write out the first 10 terms of the sequence S_2 (van der Corput sequence in base 2)

van der Corput Sequences



Extending the van der Corput sequence to more than one dimension

Why? Recall for hit-and-miss we need points in two dimensions.

How do we do this? Possible approach:

- use a different base for each dimension (Halton sequence, 1960).
- That is, let S_b denote the van der Corput sequence in base b, and $S_b(n)$ be the nth term of this sequence.
- The Halton sequence in s dimensions is given by $(S_{b_1}, \ldots, S_{b_s})$ where the b_j's are pairwise co-primes.
- Typically, take b_j to be the jth prime number.

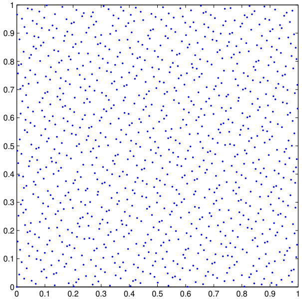
Halton sequence in three dimensions

$$\mathbf{u}_1 = (0, 0, 0) \qquad \mathbf{u}_2 = (1/2, 1/3, 1/5)$$

$$\mathbf{u}_3 = (1/4, 2/3, 2/5) \qquad \mathbf{u}_4 = (3/4, 1/9, 3/5)$$

$$\mathbf{u}_5 = (1/8, 4/9, 4/5)$$

First two dimensions:



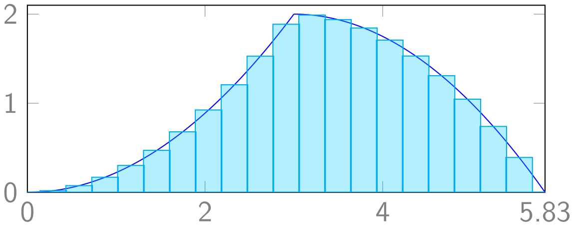
Go back to our computer program and test this

Key takeaways

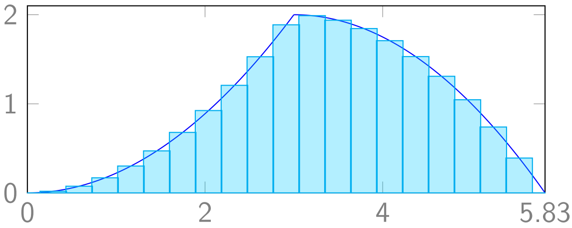
1. **Random numbers** are used in numerous computing tasks for all kinds of problems
2. Computers rely on **pseudorandom number generators** to generate pseudo-random numbers very quickly
3. **Quasi-random numbers** are more uniform than pseudorandom numbers so they can often provide better approximations
4. Need a mix of **mathematics, statistics and computer science** to play with this and understand how it all works

MC: Why does it work?

- Want to show that $\mathsf{E}(B\frac{1}{N}(h_1 + h_2 + \ldots + h_N)) = A$ (where B is the base)
- Sufficient to show that $\mathsf{E}(h_1) = A/B$
- The h_i's could take any value between 0 and 2 because x can take any value between 0 and B...
- Approximation: x can only take, say, $M = 20$ values equally spaced between 0 and B, each with probability $1/M$



MC: Why does it work



Approximation: x can take M values (mid-point of rectangles) with probability $1/M$
$\Rightarrow$ each rectangle corresponds to one of those M cases, with base equal to $B \times 1/M$ and height is the value taken by h_1
$\Rightarrow$ total surface of M rectangles $\approx$ $B\mathsf{E}(h_1)$ $\underbrace{\text{total surface of } M \text{ rectangles}}_{\approx A}$ $\approx B\mathsf{E}(h_1)$