Generating Random and Pseudorandom Numbers

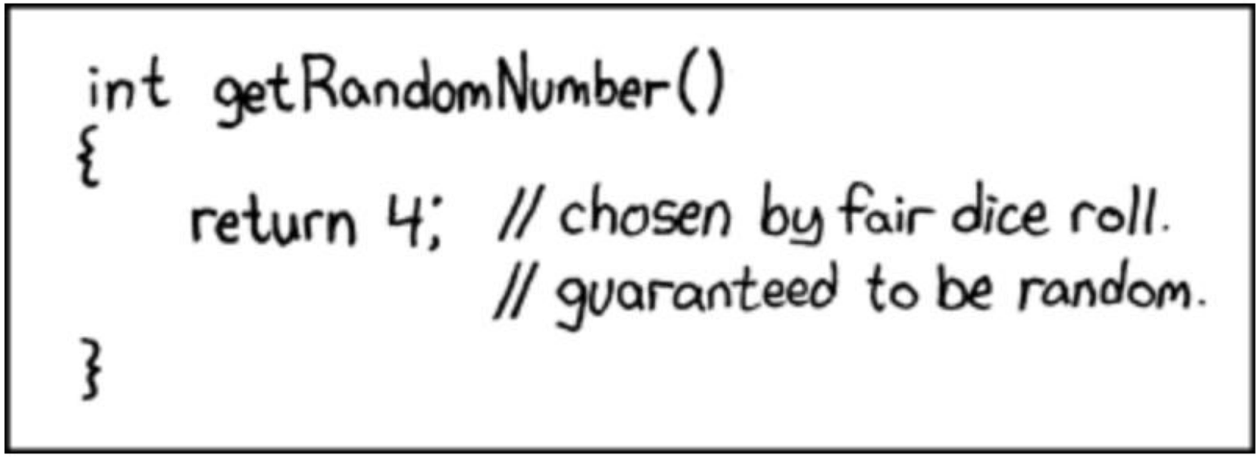
Michael Goodrich

CS 165

Some slides from CS 15-853:Algorithms in the Real World, Carnegie Mellon University

Random Numbers in the Real World

https://fitforrandomness.files.wordpress.com/2010/11/dilbert-does-randomness.jpg



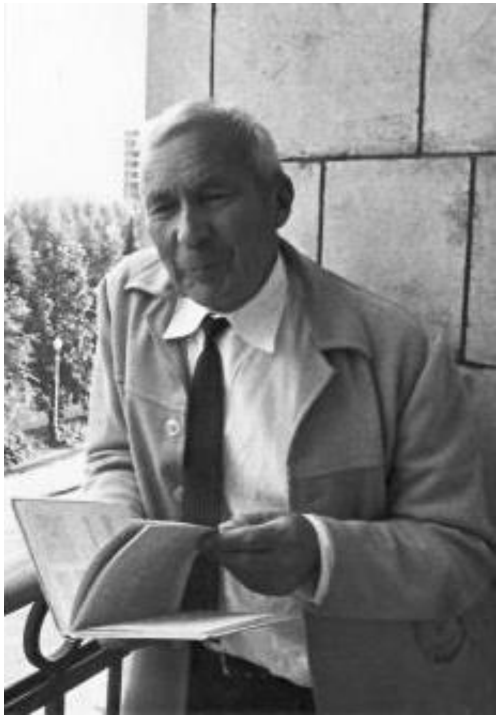
```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

https://xkcd.com/221/

Random number sequence definitions

Randomness of a sequence is the Kolmogorov complexity of the sequence (size of smallest Turing machine that generates the sequence) – infinite sequence should require infinite size Turing machine.

This definition is useful for proving computational complexity results, but it is not as useful for algorithm experiments.



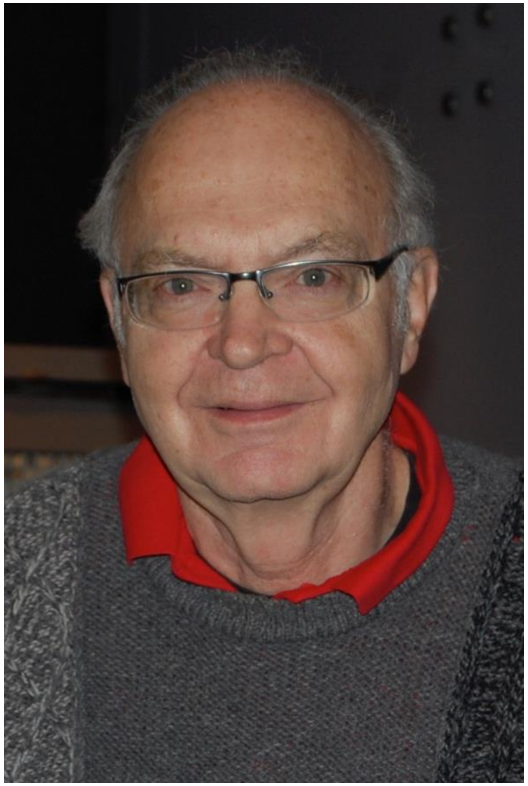
Andrey Kolmogorov

Random number sequence definitions

Each element is chosen independently from a probability distribution [Donald Knuth].

This definition is more usable for algorithm experiments.

A typical distribution is the **uniform** distribution, where every number in a range of numbers is equally likely.



Donald Knuth

Environmental Sources of Randomness

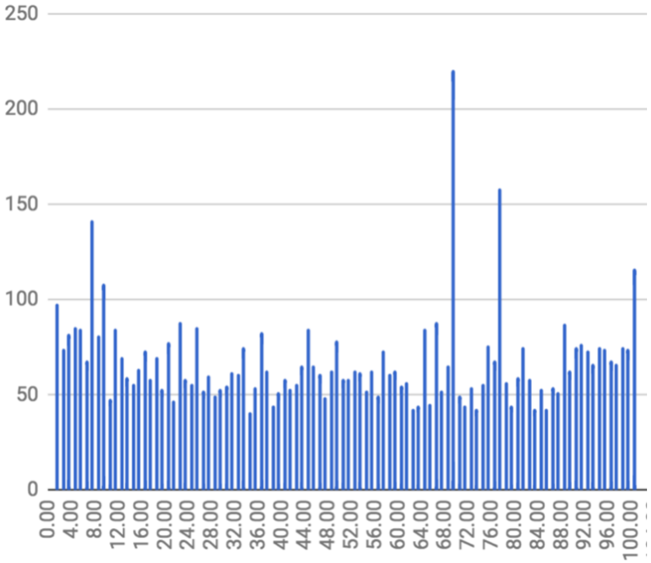
Radioactive decay http://www.fourmilab.ch/hotbits/

Radio frequency noise http://www.random.org

Noise generated by a resistor or diode.

Inter-keyboard timings from a human user (watch out for buffering)

Not a good source: Asking a human for a random number between 0 and 100:



Combining Sources of Randomness

Suppose $r_1, r_2, \ldots, r_k$ are random numbers from different sources. E.g.,

r_1 = from JPEG file

r_2 = sample of hip-hop music on radio

r_3 = clock on computer

r_4 = lower order bits in time it takes a human to click

$$b = r_1 \oplus r_2 \oplus \cdots \oplus r_k$$

If any one of $r_1, r_2, \ldots, r_k$ is truly random, then so is b.

Skew Correction

Von Neumann's algorithm – converts biased random bits to unbiased random bits:

Collect two random bits.

Discard if they are identical.

Otherwise, use first bit.

Efficiency?

John von Neumann

Uniform Random Numbers

- The skew correction method gives us uniformly random bits from possibly biased random bits.
- We can concatenate i random bits as $b_1 b_2 \ldots b_i$
- This gives us a number, k, uniformly distributed in the range from 0 to $2^i - 1$.
- How can we get a number uniformly distributed from 0 to n - 1 when n is not a power of 2?
  - Generate $k = b_1 b_2 \ldots b_i$ from uniform random bits
  - Two choices:
    - Compute r = k mod n
    - Repeatedly generate k until k < n

  - Which is best?

Pseudorandom Number Generators

- A pseudorandom number generator (PRNG) is an algorithm for generating a sequence of numbers whose properties approximate the properties of sequences of random numbers.
- The PRNG-generated sequence is not truly random, because it is completely determined by an initial value, called the PRNG's seed (which may include truly random values).
- Although sequences that are closer to truly random can be generated using hardware random number generators, pseudorandom number generators are important in practice for their speed and reproducibility.

Pseudorandom Number Generators

- PRNGs are central in applications such as simulations (e.g. for the Monte Carlo method), electronic games (e.g. for procedural generation), and cryptography.
- Cryptographic applications require the output not to be predictable from earlier outputs.

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."

- John Von Neumann, 1951

Linear Congruential Generator (LCG)

x_0 = given, $x_{n+1} = P_1 x_n + P_2 \pmod{N}$ n = 0,1,2,... (*)

$x_0 = 79$, $N = 100$, $P_1 = 263$, and $P_2 = 71$

x_1 = 79*263 + 71 (mod 100) = 20848 (mod 100) = 48,
x_2 = 48*263 + 71 (mod 100) = 12695 (mod 100) = 95,
x_3 = 95*263 + 71 (mod 100) = 25056 (mod 100) = 56,
x_4 = 56*263 + 71 (mod 100) = 14799 (mod 100) = 99,

Sequence: 79, 48, 95, 56, 99, 8, 75, 96, 68, 36, 39, 28, 35, 76, 59, 88, 15, 16, 79, 48, 95

Park and Miller:
$P_1 = 16807$, $P_2 = 0$, $N = 2^{31}-1 = 2147483647$, $x_0 = 1$.

ANSI C rand():
$P_1 = 1103515245$, $P_2 = 12345$, $N = 2^{31}$, $x_0 = 12345$

Matsumoto’s Marsenne Twister

Considered one of the best linear congruential generators.

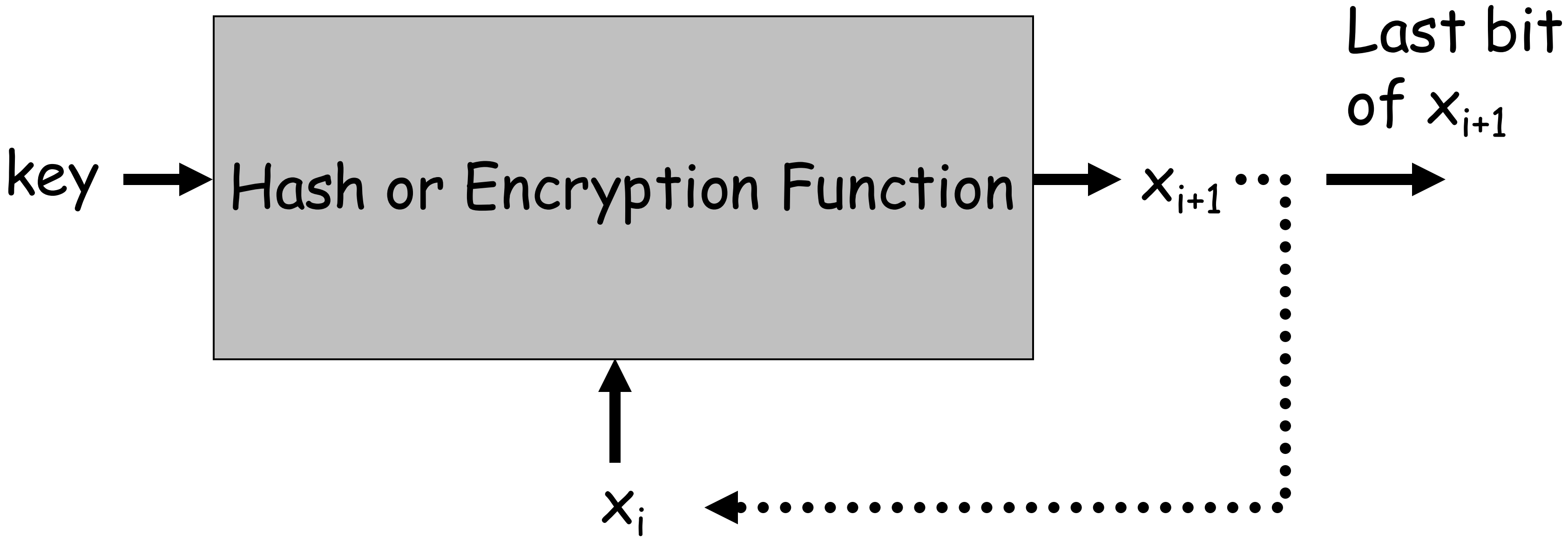
http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html

Cryptographically Strong Pseudorandom Number Generator

Next-bit test: Given a sequence of bits x_1, x_2, …, x_k, there is no polynomial time algorithm to generate x_{k+1}.

Yao [1982]: A sequence that passes the next-bit test passes all other polynomial-time statistical tests for randomness.

Hash/Encryption Chains



(need a random seed x_0 or key value)

Some Cryptographic Hash Functions

- SHA-1 Hash function https://en.wikipedia.org/wiki/SHA-1
- MD5 Hash function https://en.wikipedia.org/wiki/MD5
- These functions are good pseudo-random number generators and when seeded with a random number generator, they provide good sequences for use in algorithm experiments.

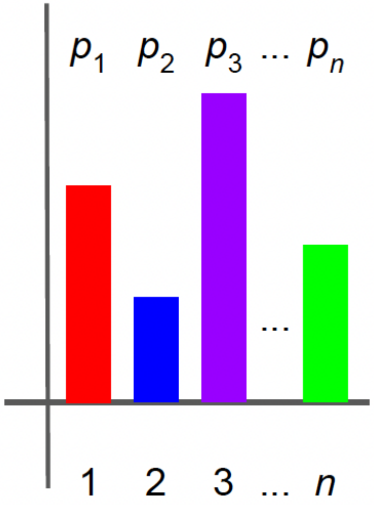
Random Numbers in Python

https://docs.python.org/3/library/random.html

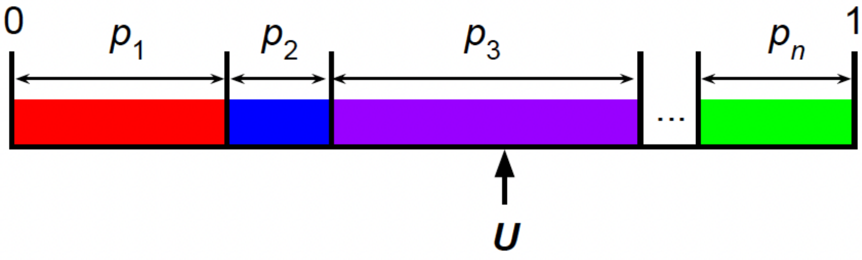
[Review this website]

Sampling from a Discrete Distribution

Let **p** := (p_1, ..., p_n) be a discrete probability distribution ($0 < p_i < 1$, $\sum_i p_i = 1$).



Step 1: Use **p** to make n bins of unit interval [0,1].



Step 2: Simulate $U \sim$ Uniform([0,1]).

Step 3: Return integer j such that U is in bin j.

"Throw a dart and choose the bin it lands in"

Discrete Inverse Transformation Method

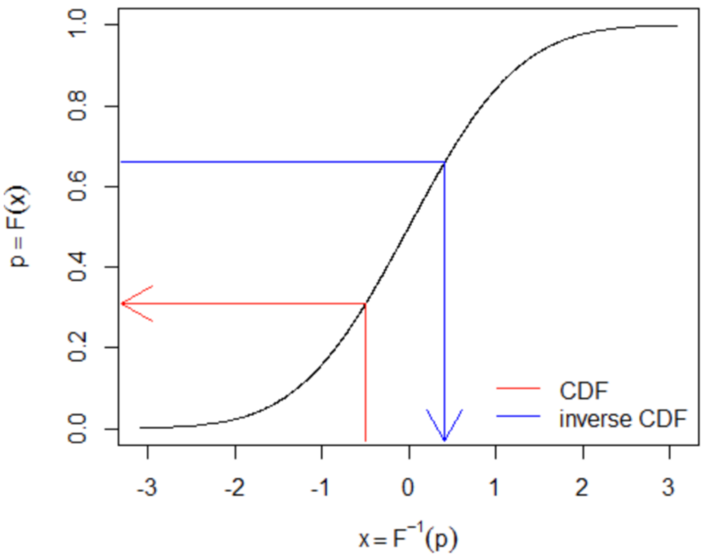
- The **cumulative distribution function** (CDF) gives the probability that the random variable X is less than or equal to x and is usually denoted F(x). The cumulative distribution function of a random variable X is the function given by

$$F(x) = \mathrm{P}[X \leq x].$$

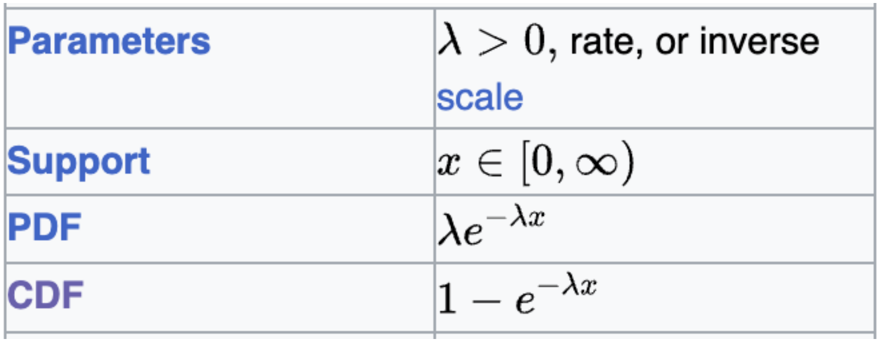
  - Compute the CDF F(x) for x = 0,1, 2,…, n ,and store in an array.
  - Generate a U(0,1) variate u and search the array to find x so that $F(x) \leq u < F(x + 1)$.
  - return x.

Continuous Inverse Transformation Method

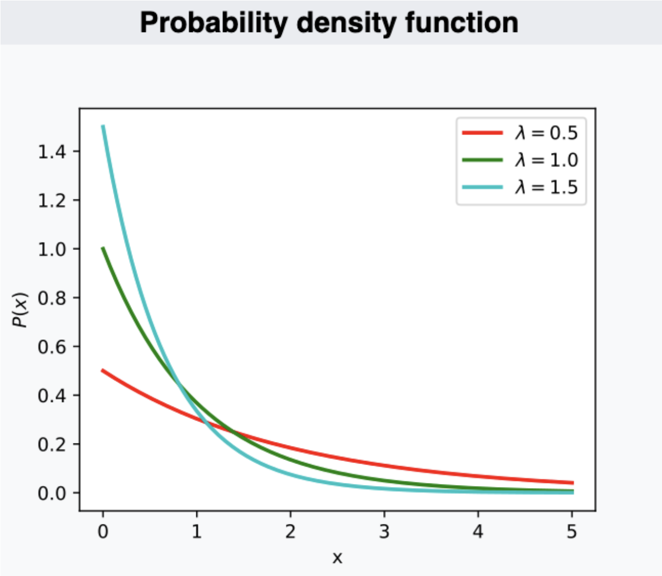
- Suppose we have a closed form for the inverse, $F^{-1}(u)$, of a CDF, F(x), for a given distribution.
- Then we can approximately sample as follows:
  - Generate a random real number, u, uniformly between 0 and 1.
  - Return $F^{-1}(u)$.



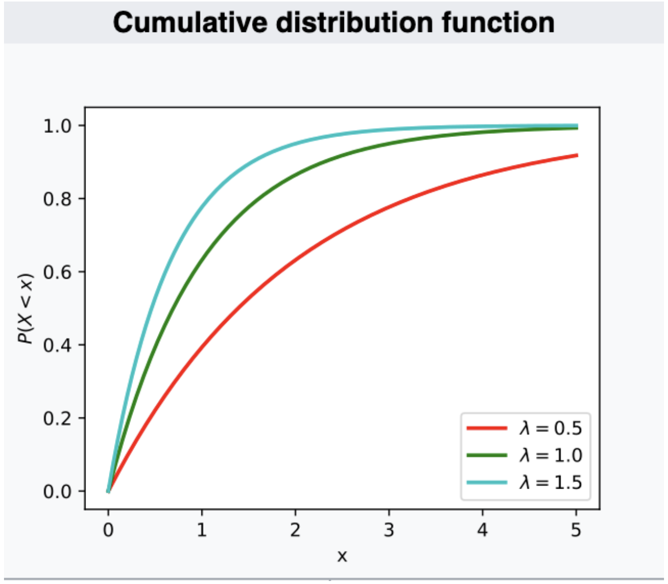
Example: Exponential distribution



Parameters	$\lambda > 0$, rate, or inverse scale
Support	$x \in [0, \infty)$
PDF	$\lambda e^{-\lambda x}$
CDF	$1 - e^{-\lambda x}$



Probability density function



Cumulative distribution function

- Generation: Generate a U(0,1) random number u and return -λ ln(u) as Exp(a).