Simple Random Sampling: Not So Simple

Kellie Ottoboni
with Philip B. Stark and Ron Rivest

Department of Statistics, UC Berkeley
Berkeley Institute for Data Science

BSTARS
March 23, 2017

University of California, Berkeley
Department of Statistics

BIDS
BERKELEY INSTITUTE
FOR DATA SCIENCE

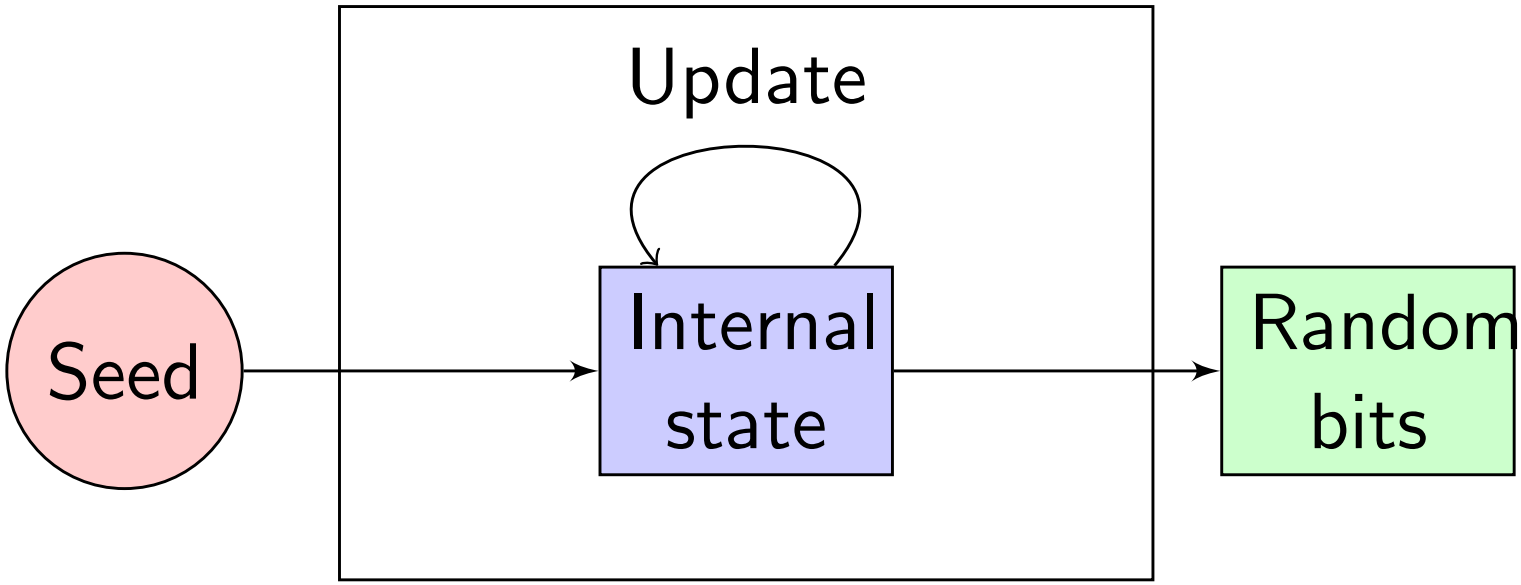
Simple random sampling lies at the heart of many statistical methods.

In practice, it is difficult to draw truly random samples.

Instead, people tend to draw samples using

1. A **pseudorandom number generator** (PRNG) that produces sequences of bits, plus
2. A sampling algorithm that maps a sequence of pseudorandom numbers into a subset of the population

Pseudorandom number generator: a deterministic algorithm that produces sequences that are (ideally) computationally indistinguishable from the uniform distribution



(Wikipedia)

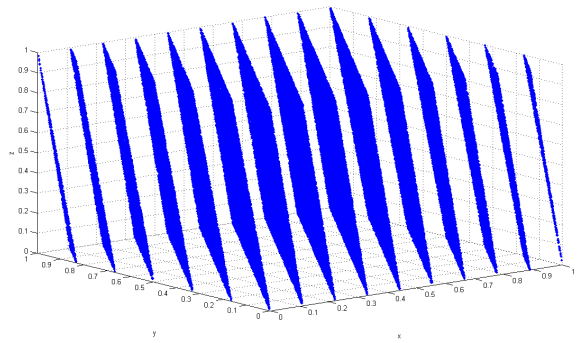
Corollary (Fewer pigeons than pigeonholes)

If the number of possible samples is greater than the size of a PRNG's state space, then the PRNG cannot possibly generate all samples.

Does it matter in practice?

PRNG	# Internal states	# Possibilities	Proportion of attainable possibilities
32-bit linear congruential generators	4 billion	Samples of 10 out of 50 items ≈ 10 billion	0.4
Mersenne Twister	$\approx 2 \times 10^{6010}$	Permutations of 2084 items $\approx 3 \times 10^{6013}$	0.0001

Even if a PRNG can generate all possible samples, it may not be sufficiently random.

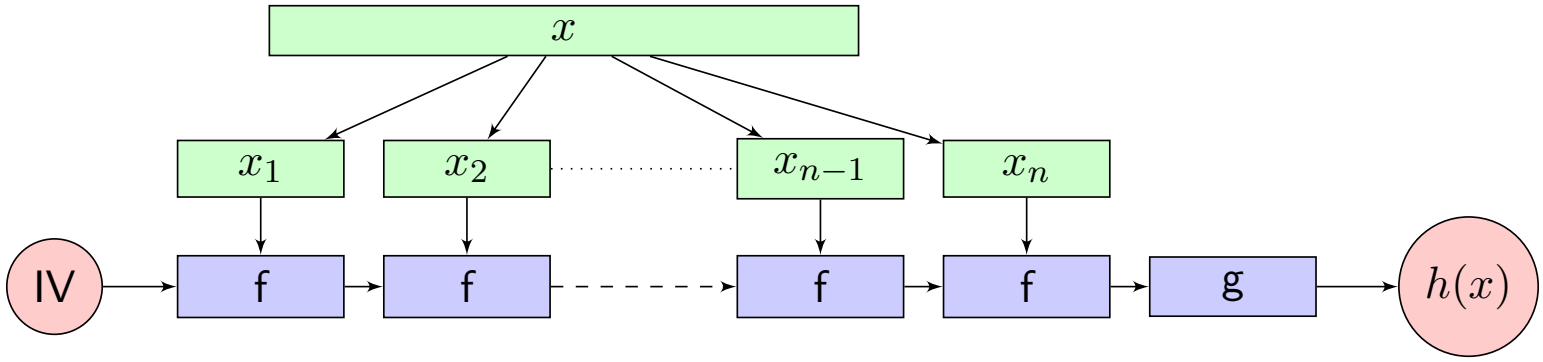


(Wikipedia)

$$x_{n+1} = (65539 x_n) \mod 2^{31}$$

Triples of RANDU lie on 15 planes in 3D space.

One solution: Find a class of PRNGs with infinite state space **and** good pseudorandom behavior



Cryptographic hash functions:

- computationally infeasible to invert
- difficult to find two inputs that map to the same output
- small input changes produce large, unpredictable changes to output
- resulting bits are uniformly distributed

- Preliminary results: SHA-256 hash function PRNG produces samples with equal probabilities while other common PRNGs don't
- Replace the default PRNGs in Python `https://www.github.com/statlab/cryptorandom`
- Results apply more broadly to computer simulations: permutation tests, bootstrapping, MCMC, etc.

Thanks!

https://github.com/kellieotto/prng-slides