

Extended Introduction to Computer Science

CS1001.py

Lecture 6B: Randomness in Computing

Instructors: Daniel Deutch, Amir Rubinstein

Teaching Assistants: Michal Kleinbort, Noam Parzanchevsky,
Ben Bogin

School of Computer Science

Tel-Aviv University

Winter Semester 2019-20

<http://tau-cs1001-py.wikidot.com>

Randomness in Computing

- What is **randomness**? According to **Wikipedia**:

*Randomness is the **lack of pattern or predictability** in events. A random sequence of events, symbols or steps has **no order** and does not follow an **intelligible pattern** or combination. Individual random events are by definition unpredictable, but in many cases the frequency of different outcomes over a large number of events (or "trials") is predictable.*

- The programs we have written so far were almost always **deterministic**: given an input, the output is uniquely determined.
- In many cases it is useful to include randomness in computations.
- Such algorithms are termed **randomized** or **probabilistic** or **coin flipping** algorithms.
- Unlike deterministic algorithms, their execution path cannot be (easily) reproduced.

A Piece of History: Obtaining Random Sequences

From Wikipedia:

A Million Random Digits with 100,000 Normal Deviates

is a [random number book](#) by the [RAND Corporation](#), originally published in 1955.

The book, consisting primarily of a [random number table](#), was an important 20th century work in the field of [statistics](#) and [random numbers](#).

It was produced starting in 1947 by an electronic simulation of a [roulette wheel](#) attached to a [computer](#), the results of which were then carefully filtered and tested before being used to generate the table.

The RAND table was an important breakthrough in delivering random numbers, because such a large and carefully prepared table had never before been available. In addition to being available in book form, one could also order the digits on a series of [punched cards](#).

73735	45963	78134	63873
02965	58303	90708	20025
98859	23851	27965	62394
33666	62570	64775	78428
81666	26440	20422	05720

15838	47174	76866	14330
89793	34378	08730	56522
78155	22466	81978	57323
16381	66207	11698	99314
75002	80827	53867	37797

99982	27601	62686	44711
84543	87442	50033	14021
77757	54043	46176	42391
80871	32792	87989	72248
30500	28220	12444	71840

A random sampling of 300 random digits from *A Million Random Digits with 100,000 Normal Deviates*

Obtaining Random Sequences (cont.)

- **True** Random Number Generators (TRNG)
Extract randomness from **physical phenomena** such as cosmic radiation, radioactive decay, etc.
- A **philosophical** / **meta-physical** question for you to ponder about:
Are there any events in nature that are **truly random**?
- Recall Einstein's quote: "**God does not play dice with the universe.**"
referring to his views about quantum mechanics
(which is often misinterpreted:
<http://www.techinsider.io/god-does-not-play-dice-quote-meaning-2015-11>)
- **Pseudo-random** Number Generators (PRNG):
An algorithm to generate a long sequence of numbers, deterministically, from a short, truly random value called "**seed**". The long sequence **appears** random ("random enough") w.r.t different tests.

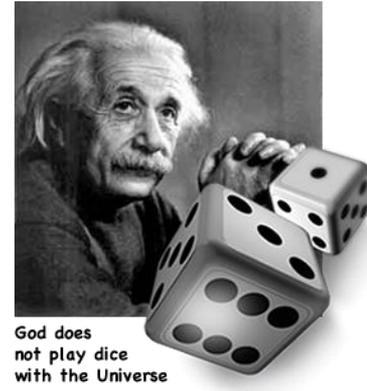


Image from:
<http://www.smallplanet.us>

Example for PRNG: Linear Congruential Generator

$$x_{i+1} = (a \cdot x_i + c) \bmod m$$

$$a = 1, c = 7, m = 12$$

$$x_0 = 0$$

$$x_1 = 7$$

$$x_2 = 2$$

$$x_3 = 9$$

...

The "seed".

Used to initialize the random number generator.

Picked at random (are we in a loop here?)

Can be picked e.g. by using the system's clock.

```
>>> a, c, m = 1, 7, 12
```

```
>>> r = 0 # x0
```

```
>>> for i in range(20):
```

```
    print(r, end=" ") # end each print with a space
```

```
    r = (a*r + c) % m
```

```
0 7 2 9 4 11 6 1 8 3 10 5 0 7 2 9 4 11 6 1 8
```

- The numbers in the sequence must eventually enter a cycle. The length of the cycle is called the **period** of the random number generator.
- The choice of the parameters affects the period.
Try, for example, $c=8$ instead of 7.

Randomness in Python

- Python employs a more sophisticated pseudo random generator, called **Mersenne Twister** (you are not expected to know its details)
- The name derives from the fact that the period of the generator is chosen to be a **Mersenne prime**: a prime number that is one less than a power of two. That is, it is a prime number that can be written in the form $2^n - 1$ for some positive integer n .

```
>>> import random
```

```
>>> random.random() #generate a random number  $\in [0, 1)$ 
```

```
0.9724062711684623
```

```
>>> random.random()
```

```
0.9793789492766168
```

```
>>> random.random()
```

```
0.2880152915931866
```

Randomness in Python – Some Useful Functions

```
>>> random.uniform(3.2, 12.01) # random number ∈[3.2, 12.01)
9.311113665186017
```

```
>>> random.randrange(0,1000,10) # random int from range(start, stop, step)
920
```

```
>>> random.choice([1,5,6,-44,9]) # random element from the input collection
-44
```

```
>>> lst = [random.choice("a"*5 + "bcdef") for i in range(1000)]
```

```
>>> lst[:10]
```

```
['c', 'a', 'a', 'a', 'a', 'e', 'b', 'a', 'a', 'a']
```

```
>>> lst.count("a")/len(lst)
```

```
0.497
```

```
>>> lst.count("b")/len(lst)
```

```
0.101
```

```
>>> L = [1,2,3,4,5]
```

```
>>> random.shuffle(L) #in place permutation
```

```
>>> L
```

```
[2, 5, 1, 4, 3]
```

Does Randomness Have any **Uses** in Computing?

- Absolutely:

- **Sampling** (usually a large data set, *e.g.* testing a program)

- **Simulation**



The next 2 examples

- **Cryptography** (soon, **Diffie-Hellman** protocol)

- Improving **efficiency** (soon, **primality testing**)

- Defense against **worst case** scenarios with high probability (soon, **Quicksort**)

Randomness in Action – Example 1: Sampling

Estimating π by a Monte Carlo Method

- From Wikipedia:

Monte Carlo methods (or Monte Carlo experiments) are a broad class of computational algorithms that rely on repeated random **sampling** to obtain **numerical results**.

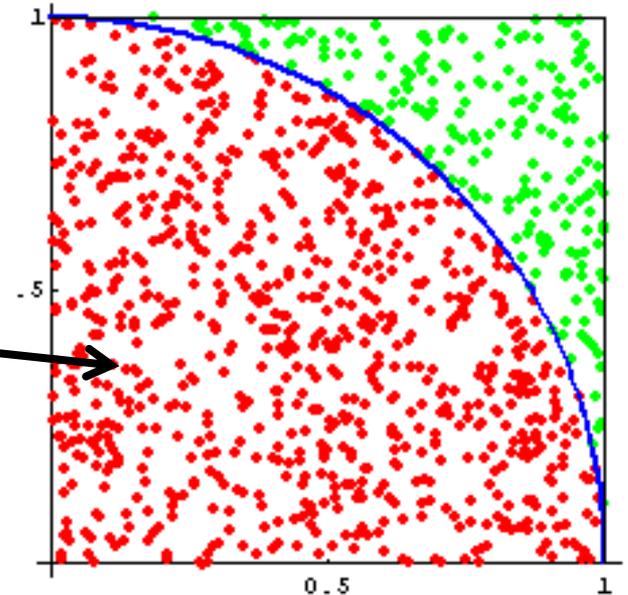
They are often used in physical and mathematical problems and are most useful when it is difficult or impossible to use other mathematical methods.

- The name "Monte Carlo" was coined by Nicholas Constantine Metropolis (1915-1999) and inspired by Stanislaw Ulam (1909-1986), because of the similarity of statistical simulation to games of chance, and because Monte Carlo is a center for gambling and games of chance.
(<http://mathfaculty.fullerton.edu/mathews/n2003/montecarlopimod.html>)

Randomness in Action – Example 1: Sampling

Estimating π

- Randomly choose points (x, y) in the unit square ($0 \leq x, y \leq 1$)
- Count how many of them are located inside the quarter circle of radius 1 centered in the origin.
- The ratio is the area of the quarter circle ($\pi/4$) divided by the area of the square (1).



Estimating π in Python

```
import random

def estimate_Pi(num=1000):
    """ estimate pi by a monte carlo experiment """
    count=0
    for n in range(num):
        x = random.random()
        y = random.random()
        if x**2 + y**2 <= 1.0:
            count += 1
    return 4*count/num
```

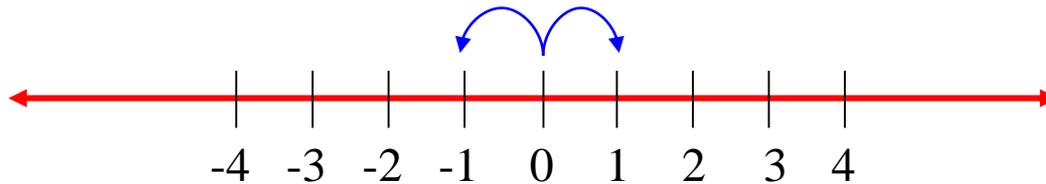
```
>>> estimatePi()
3.156
>>> estimatePi(100000)
3.12864
>>> estimatePi(10**8)
3.14175412 #took about a minute
```

```
>>> import math
>>> math.pi
3.141592653589793
```

Randomness in Action – Example 2: Simulation

Random Walk

- A **random walk** is a mathematical formalization of a path that consists of a succession of random steps.
- For example, a marker is placed at zero on the number line, and at each step moves $+1$ or -1 with equal probability. This is a **simple unbiased 1-dimensional** random walk.



- There are numerous other versions, for example grids of higher dimensions, graphs, biased, etc.

Random walk in two dimensions.

https://upload.wikimedia.org/wikipedia/commons/f/f3/Random_walk_2500_animated.svg

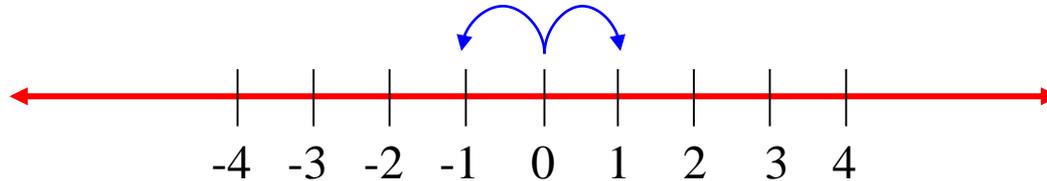


Randomness in Action – Example 2: Simulation

Random Walk

- Random walks are used to model various types of phenomena from a diverse range of fields, such as:
 - Economics: shares prices
 - Genetics: genetic drift (frequency of gene variants (allele) in a population)
 - Physics: Brownian motion and diffusion
 - Ecology: population dynamics
 - www : Twitter's WTF ("Who to Follow")
 - Google's Pagerank algorithm to rank internet pages
 - ...

Properties of Simple Unbiased 1D Random Walk



- Formally, a simple symmetric 1D random walk is defined as a series $\{S_1, S_2, \dots\}$ where $S_n = \sum_{j=1}^n Z_j$ and $Z_j \in \{-1, 1\}$.
- $|S_n|$ is the distance from 0 after n steps.
- It is known that:
$$\lim_{n \rightarrow \infty} \frac{E(|S_n|)}{\sqrt{n}} = \sqrt{\frac{2}{\pi}}$$
- Let's examine this property **empirically**.

Note that $E(|S_n|)$ is the **expectation** of $|S_n|$, so we need to **average** the distance over several executions.

Properties of Simple Unbiased 1D Random Walk – in Python

```
def rand_walk(steps, show=False):
    pos = 0 #initial position
    for i in range(steps):
        walk = random.choice([+1,-1])
        pos += walk
        if show:
            print("step", i, ":", pos)
    return pos

def test_rand_walk_dist(steps, num_simulations):
    dist = 0
    for i in range(num_simulations):
        dist += abs(rand_walk(steps))
    avg = dist/num_simulations
    return avg/(steps**0.5) #tends to (2/math.pi)**0.5
```

Simple Unbiased 1D Random Walk – Executions

```
>>> rand_walk(10, True)
```

```
step 0 : 1
step 1 : 2
step 2 : 1
step 3 : 0
step 4 : 1
step 5 : 0
step 6 : 1
step 7 : 2
step 8 : 3
step 9 : 4
4
```

```
>>> rand_walk(10, True)
```

```
step 0 : 1
step 1 : 2
step 2 : 3
step 3 : 2
step 4 : 1
step 5 : 0
step 6 : -1
step 7 : -2
step 8 : -3
step 9 : -2
-2
```

```
>>> import math
```

```
>>> (2/math.pi)**0.5
0.7978845608028654
```

```
>>> avg_rand_walk_dist(10, 100)
0.7779203044014212
```

```
>>> avg_rand_walk_dist(10, 1000)
0.8089106254710713
```

```
>>> avg_rand_walk_dist(10, 10000)
0.7869011729562995
```

```
>>> avg_rand_walk_dist(10, 100000)
0.7789638560292768
```

**#hmm... larger sample doesn't guarantee
better results**

Simple Unbiased 1D Random Walk – Additional Properties

- What is the average number of steps needed to reach a distance of b from 0?
- What is the probability that a walk of length n will return to the origin.
- What is the average distance from origin for walks of length n ?
- What about $>1D$?