Random Processes

Simulation and Modeling (CSCI 3010U)

Faisal Z. Qureshi

http://vclab.science.ontariotechu.ca

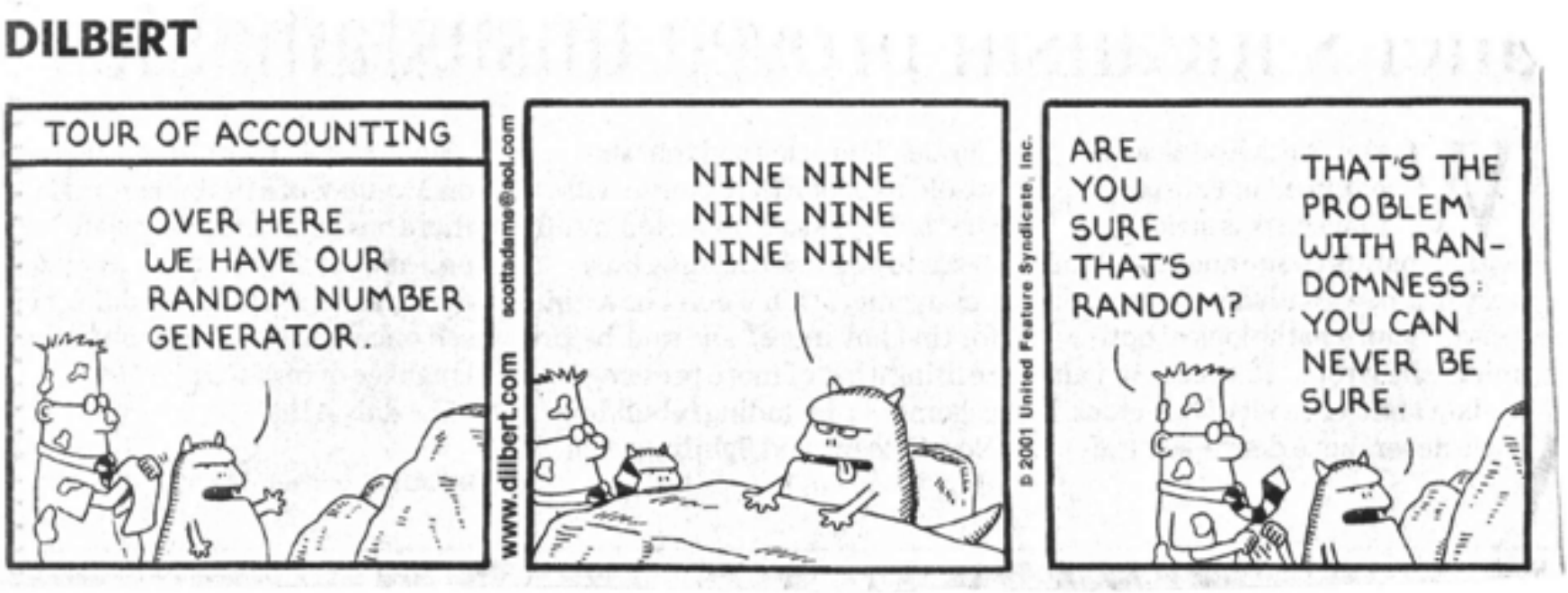


Figure 1: Dilbert and randomness

Random Processes

- Simulate random processes or activities
- Approximate phenomena that are too hard to describe deterministically, or a deterministic simulation is too costly
- Provide variation in the simulation data, so we can estimate errors or compute the range of an output

Monte Carlo Simulations

- Using repeated sampling to determine the properties of some phenomenon

- Principle of *inferential statistics*
  - A random sample tends to exhibit the same properties as the population from which it is drawn from
  - This principle doesn't always hold, which can lead to inaccurate results

Example

What is the probability of **not** getting a single *head* after 10 coin flips?

- The probability of not getting a head after a single toss is 0.5 (*assuming a fair coin*)
- So the probability of not getting a head after 10 tosses is $(0.5)^{10}$

Example

In the game of craps, "boxcars" refers to rolling a total of 12 with two dice. What is the probability of getting a boxcars in a game of craps after 24 throws?

Solve analytically

- Each die has 6 sides, so there are a total of $6 \times 6 = 36$ possible outcomes when rolling two dice.
- There is only 1 combination of outcomes that results in a total of 12: rolling a 6 on both dice. So, the probability of rolling boxcars on a single roll is $\frac{1}{36}$
- Probability of **not** rolling a boxcar in one throw $\left(1 - \frac{1}{36}\right)$
- Probability of **not** rolling a boxcar in 24 throws $\left(1 - \frac{1}{36}\right)^{24}$
- Probability of rolling a boxcar in 24 throws $\left(1 - \left(1 - \frac{1}{36}\right)^{24}\right)$

Monte Carlo Simulation

The concept of Monte Carlo simulations is attributed to the work of scientists Stanislaw Ulam, a Polish mathematician, and John von Neumann, a Hungarian-American mathematician, during the 1940s.

During World War II, Ulam was working on the Manhattan Project, the project that developed the atomic bomb. He was seeking solutions to mathematical problems related to the neutron diffusion equation, which described how neutrons moved in a fissile material. To overcome the limitations of analytical methods, Ulam conceived of using random sampling techniques to approximate solutions to complex mathematical problems.

Together with von Neumann, Ulam developed the concept further, and they applied it to a wide range of scientific and engineering problems. The term "Monte Carlo" was coined as a reference to the famous casino in Monaco known for its games of chance, reflecting the probabilistic nature of the method.

The first documented use of the Monte Carlo method was in 1947 in the Los Alamos Scientific Laboratory, where Ulam and von Neumann used it to simulate neutron diffusion. Since then, Monte Carlo simulations have become a fundamental tool in various fields such as physics, engineering, finance, and computer science.

Monte Carlo Simulation - Boxcars in 24 throws

```
def throw_dice():
    return random.choice([1,2,3,4,5,6])


def throw_two_dice(num_times):
    dice_throws = []
    for i in range(num_times):
        dice_throws.append((throw_dice(), throw_dice()))
    return dice_throws


def crap_boxcar(num_dice_throws):
    success = 0.
    outcomes = throw_two_dice(num_times = num_dice_throws)
    for i in outcomes:
        if i == (6,6):
            success += 1
            break
    return success
```

Monte Carlo Simulation - Boxcars in 24 throws

```
def simulate_crap_boxcar(num_trials):
    success = 0.
    for i in range(num_trials):
        if crap_boxcar(24) >= 1:     # We got at least one (6,6)
            success += 1             # 24 two-dice throws
    return success / num_trials


num_trials = 100000
p_monte_carlo = simulate_crap_boxcar(num_trials = num_trials)
```

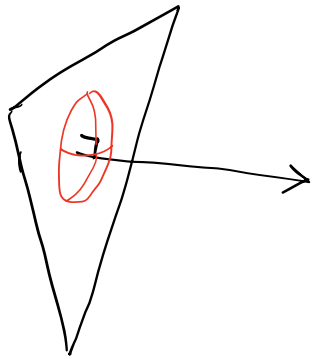
Prob. of getting at least one (6,6) in 24 tries

- Analytical: 0.4914038761309034
- Monte Carlo Simulation (100000 trials): 0.49359
- Monte Carlo Simulation (10 trials): 0.1

Monte Carlo integration

Monte Carlo integration is a technique for computing the value of a multidimensional definite integral

$$I = \int_\Omega f(\mathbf{x}) d\mathbf{x},$$



where Ω, as subset of $\mathbb{R}^m$, has volume

$$V = \int_\Omega d\mathbf{x}.$$

In naive Monte Carlo approach, points

$$\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n \in \Omega$$

are sampled uniformly on Ω. Then, I can be approximated by

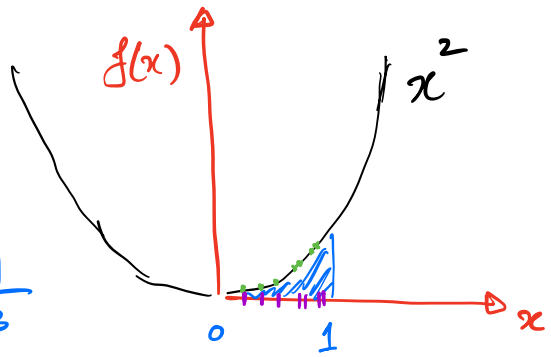
$$I \approx V \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{x}_i) = V \langle f \rangle$$

mean value of f at sampled locations

Monte Carlo integration

Example: Compute $I = \int_0^1 x^2 dx$?

$= \frac{x^3}{3}\Big|_0^1 = \frac{1}{3} - \frac{0}{3} = \frac{1}{3}$



Steps:

- $V = 1 - 0 = 1$
- Sample N values from uniform distribution between 0 and 1

$$\{u_1, u_2, u_3, \cdots, u_N\}$$

- The answer is

$$I = V \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{x}_i) = (1) \left(\frac{1}{N} \sum_{i=1}^{N} u_i^2 \right) = \frac{1}{N} \sum_{i=1}^{N} u_i^2$$

How many trials (or samples)?

Law of Large Numbers

In repeated independent tests with some actual probability p of an outcome for each test, the chance that the fraction of time that outcome occurs converges to p as the number of trials goes to *infinity*

Gambler's Fallacy

- Say you flip a coin 10 times, and get a string of *heads*. Are we more likely to get a *head* the 11th time?
  - No

How many trials?

How many samples are needed to have confidence in results?

Notion of Variance

How much spread there is in the possible outcomes?

Example

Say we want to compute the mean cgpa for students at a particular university. Obviously, the best option is to get cgpa for every student and then compute the mean. Oftentimes it is not possible to sample the entire population, so we look at the cgpa of a subset of students (called a sample) and estimate the mean cgpa. In this case, the estimated values fluctuates between different subsets of students (or samples). If estimated mean values vary greatly between the various subset of students then it lowers our confidence in the estimated value.

Mean, Variance, and Standard Deviation

Mean for a sample $\{x_1, x_2, x_3, \cdots, x_N\}$ is

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i,$$

where N refers to sample size.

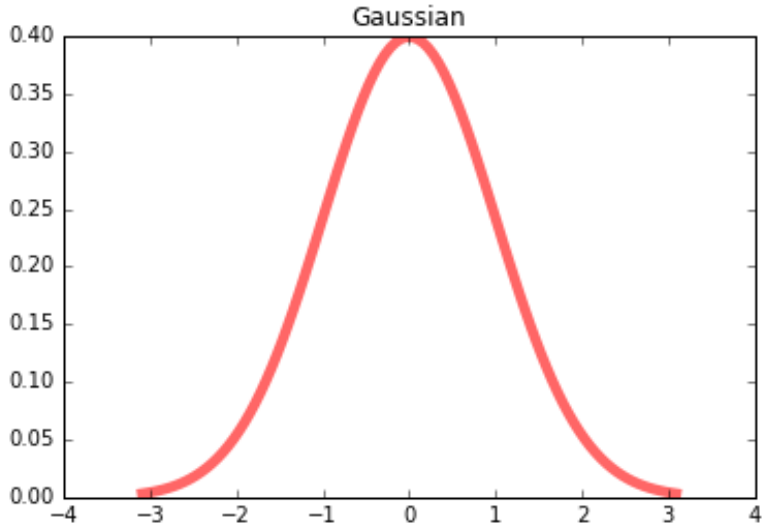
Variance is

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)^2.$$

Standard deviation is simply

$$\sigma = \sqrt{\sigma^2}$$

Normal (Gaussian) Distribution



- Nice mathematical properties
- Models many naturally occuring phenomenons really well
- Many random variables are roughly normally distributed
- Many experimental setups have normally distributed errors

Normal Distribution and Confidence Intervals

- Normal distribution is characterized by mean and variance
- Mean and variance can be used to construct confidence intervals
  - 68% of data within 1 standard deviation of mean
  - 95% of data within 2 standard deviation of mean
  - 99.7% of data within 3 standard deviation of mean

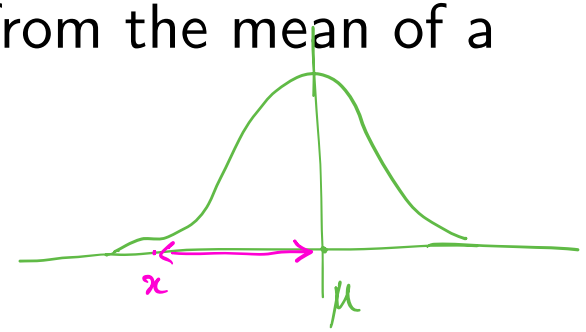
Central Limit Theorem

When working with large sample sizes ($N \geq 30$), the distribution of sample means tends to approximate a Normal (Gaussian) distribution, regardless of the distributions where the samples were drawn from. Therefore, Z-scores are based on standard Normal distribution.

Z-score

Z-score, also called standard score, is a measure of how many standard deviations a data point is away from the mean of a distribution. It is computed as

$$z = \frac{x - \mu}{\sigma},$$



where x is a data point, μ is the mean, and σ is the standard deviation.

Z-score is often used to compare data points from different distributions or to determine the likelihood of a particular value occurring within a distribution.

If a data point has a z-score of 1 then this data point is 1 standard deviation away from the mean.

Using Z-score for comparison across populations

Z-score allows comparisons across different populations or samples with varying means and standard deviations; therefore, Z-score is used to compute confidence intervals for the estimated mean (for a given sample). This is so because Z-score provides a standardized measure of how far an estimated mean is from the true mean (i.e., mean computed over the entire student population.).

Confidence intervals

Computing confidence interval

- Collect a sample
- Calculate sample mean μ and sample standard deviation σ
- Choose a confidence level, e.g., 90%, 95% (this is often problem specific)
- Find the Z-score corresponding to the chosen confidence level (*see Central Limit Theorem*)
- Compute

$$\text{Margin of Error} = z \times \frac{\sigma}{\sqrt{N}}$$

- Confidence internal is then

$$\text{Confidence internal} = \mu \pm \text{Margin of Error}$$

Coefficient of variation

A statistical measure that expresses the relative variability of a dataset compared to its mean

$$CV = \frac{\sigma}{\mu} \times 100,$$

here σ and μ denote the standard deviation and the mean, respectively. Here CV is expressed as a *percentage*.

The coefficient of variation is particularly useful when comparing the variability of datasets with different units or scales. A higher coefficient of variation indicates greater relative variability, while a lower coefficient of variation suggests less relative variability.

Problems with coefficient of variation

- Not well behaved when mean μ is near zero
- It cannot be used to compute *confidence intervals*

Examples: Nuclear Decay

Nuclear Decay

Nuclear decay, also known as radioactive decay, refers to the process by which unstable atomic nuclei lose energy by emitting radiation. This process results in the transformation of the nucleus into a more stable configuration. There are several types of nuclear decay, including alpha decay, beta decay, gamma decay, and electron capture. Each type involves the emission of different particles or radiation from the nucleus, leading to the formation of a different element or isotope. Nuclear decay is a fundamental concept in nuclear physics and plays a crucial role in various natural processes, such as the aging of rocks, the generation of heat in Earth's interior, and the functioning of nuclear reactors.

Nuclear Decay

Consider a large number of radioactive nuclei. According to the law of radioactive decay, the rate of decay is proportional to the number of nuclei. We can express this as the following differential equation:

$$\frac{dN}{dt} = -\lambda N.$$

Here, N is the number of nuclei and λ is the decay constant.

Nuclear Decay - Analytical Solution

We can actually solve this equation analytically and determine the number of nuclei remaining after time t:

$$N(t) = N(0)e^{-\lambda t}.$$

- $N(0)$ is the initial number of nuclei.
- $N(t)$ is the number of nuclei remaining after time t.
- λ is a measure of the probability of a radioactive nucleus decaying per unit time.
- e is the base of the natural logarithm (approximately equal to 2.71828)
- t is the time elapsed

Nuclear Decay

```
l = 0.05       # lambda
n0 = 100000    # number of nuclei at time 0
tmax = 60      # run till time is 60
steps = 10000 # how many samples between time 0 and 60
```

Nuclear Decay - Analytical solution

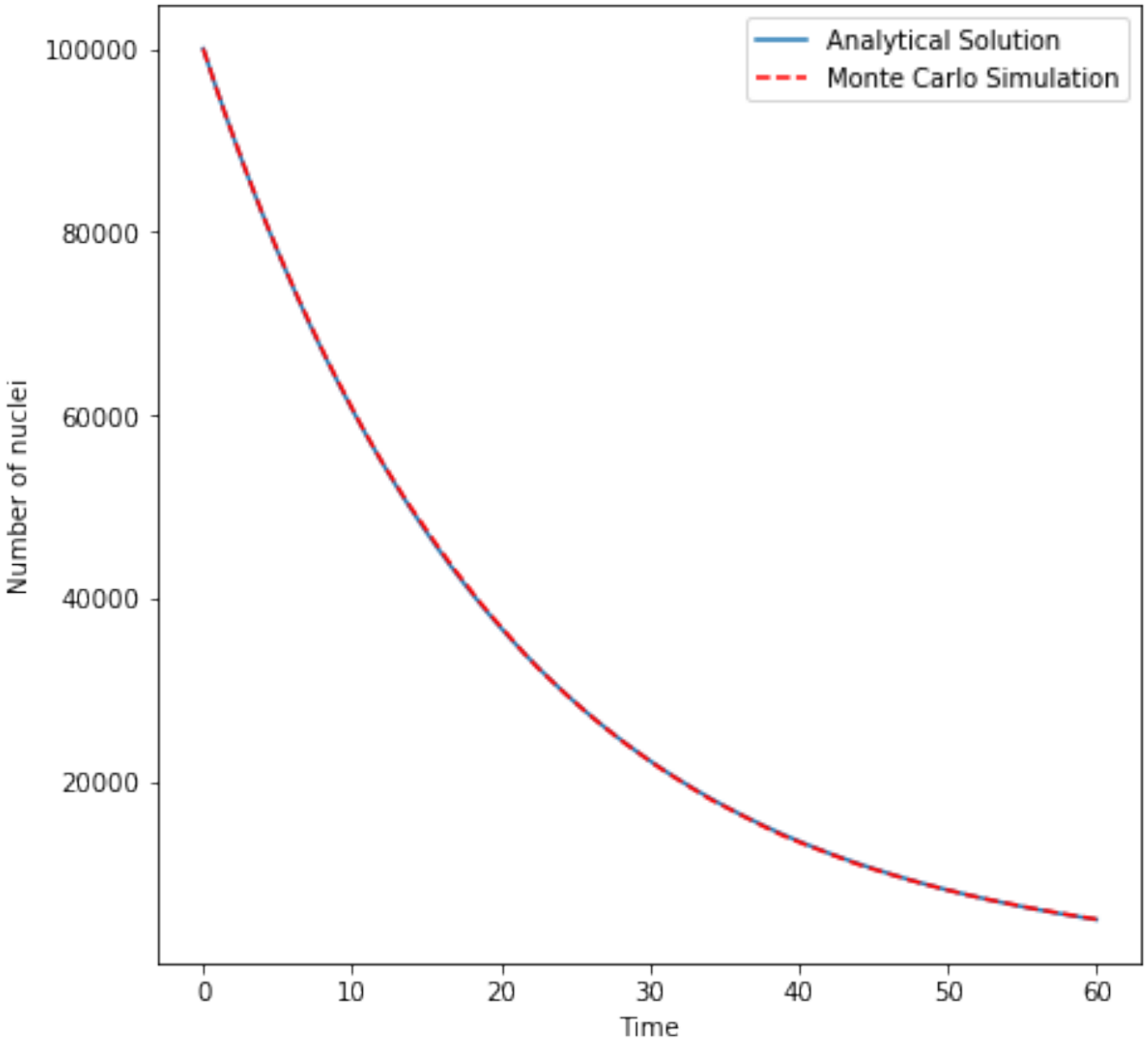
```
times = np.linspace(0, tmax, steps)
nta = n0 * np.exp(-l * times)
```

Nuclear Decay - Monte Carlo simulation

Each nucleus is identical, and we assume that during any given time interval Δt, each nucleus has the probability p of decaying.

```
prob_of_decay_per_time_step = l * tmax / steps


def nuclear_decay_monte_carlo(n0, p, steps):
    nt = np.empty([steps, 1])
    nt[0] = n0

    for i in range(1,steps):
        decay_or_not = np.random.random((1,int(nt[i-1])))
        num_not_decayed = np.sum(decay_or_not > p)
        nt[i] =  num_not_decayed

    return nt

nt = nuclear_decay_monte_carlo(n0, prob_of_decay_per_time_step, steps)
```

Nuclear Decay



Random Walks

Random Walks

In 1D random walk, the walker starts at origin and takes a right step with probability p and left step with probability $1 - p$.

```
n = 1000
x(0) = 0
do:
    n -= n
    c = draw_a_random_number()
    if c < p:
        x(t+1) = x(t) + 1
    else:
        x(t+1) = x(t) - 1
while n > 0
```

The walk takes one step in each time step of the simulation.

Multi-dimensional random walks

It is straightforward to extend a random walk to higher dimensions.

Example: 2D random walk

A walker takes one of *up*, *down*, *left* or *right* steps based upon the associated probabilities: p_u, p_d, p_l and p_r. These probabilities all sum to 1.

Collecting statistics from random walks

To get anything meaningful out of such simulations, we need to repeat them many times and collect statistics.

- Mean distance, what is the mean position of the walker after N steps.
- Maximum distance, what is the maximum distance that a walker reached during one of the runs.
- Average maximum distance, what is the average maximum distance that walker reached during all the runs.
- Histogram of positions

Random step lengths

- Changing step size at each time step
- Step size can also be a function of the length of the walk
  - Either growing or shrinking as the walks get longer
- Different step sizes in different directions

Boundary conditions 1

What if the space is limited and the random walk hits a boundary?

- Absorbing boundary, walker is consumed upon hitting the boundary.
- Reflecting boundary, walker changes direction upon hitting the boundary.
- Periodic boundary, walker appears at the positive boundary after passing through the negative boundary and vice versa.
  - Often used to simulated circular domains.
- Random boundary, walker is placed at a random location each time it hits the boundary.

Boundary conditions 2

Absorbing boundary conditions

- With absorbing boundary conditions the walks can end early
- This creates significant challenges when collecting statistics

Reflecting boundary conditions

- Walker tends to stay at the edges of the region

Boundary conditions 3

Periodic boundary conditions

- Periodic boundary conditions produce a more uniform distribution
- This is easily explained, since walkers do not stick to the edges

Random boundary conditions

- Creates uniform trails (distribution over locations)

Random walks: are they truly that random?

Some aspects of random works are surprising in the sense that these are not random at all.

- E.g., even length random walks (with step size of 1) all end up at even locations.
- E.g., odd length random walks (with step size of 1) all end up at odd locations.

We can introduce randomness by exploiting boundary conditions.

Variations on random walks 1

Traps

Trap is an extension of absorbing boundary condition. Here traps are randomly placed all over the region. A walker is consumed upon reaching a trap. Optionally it is possible to randomly place the walker at a different location in the region.
Absorbing boundary conditions and traps can often lead to very *short* walks.

- One option is to create a new walker (usually at a random location) whenever a walker dies.
- Second option is to have more than one walkers in parallel. If there are large enough walkers, the chances are high that some of these will survive longer than others.

Variations on random walks 2

When multiple walkers are active at the same time, it raises the spectre of collisions between walkers.

- Ignore the collisions
- Kill off one of the walker
- Walkers transform each other upon collision
  - This is often used when simulating chemical reactions

Solving the diffusion equation

Consider the diffusion equation

$$\frac{\partial P(x,t)}{\partial t} = D\frac{\partial^2 P(x,t)}{\partial x^2}$$

It is used to model a variety of physical processes, including fluid flow. It is also used in finance. Analytical solutions to diffusion equation do not exist except for the simplest cases. A suite of numerical solutions have been proposed in the literature. Numerical solutions, however, cannot be easily parallelized. Furthermore, these can only be used for a small set of boundary conditions.

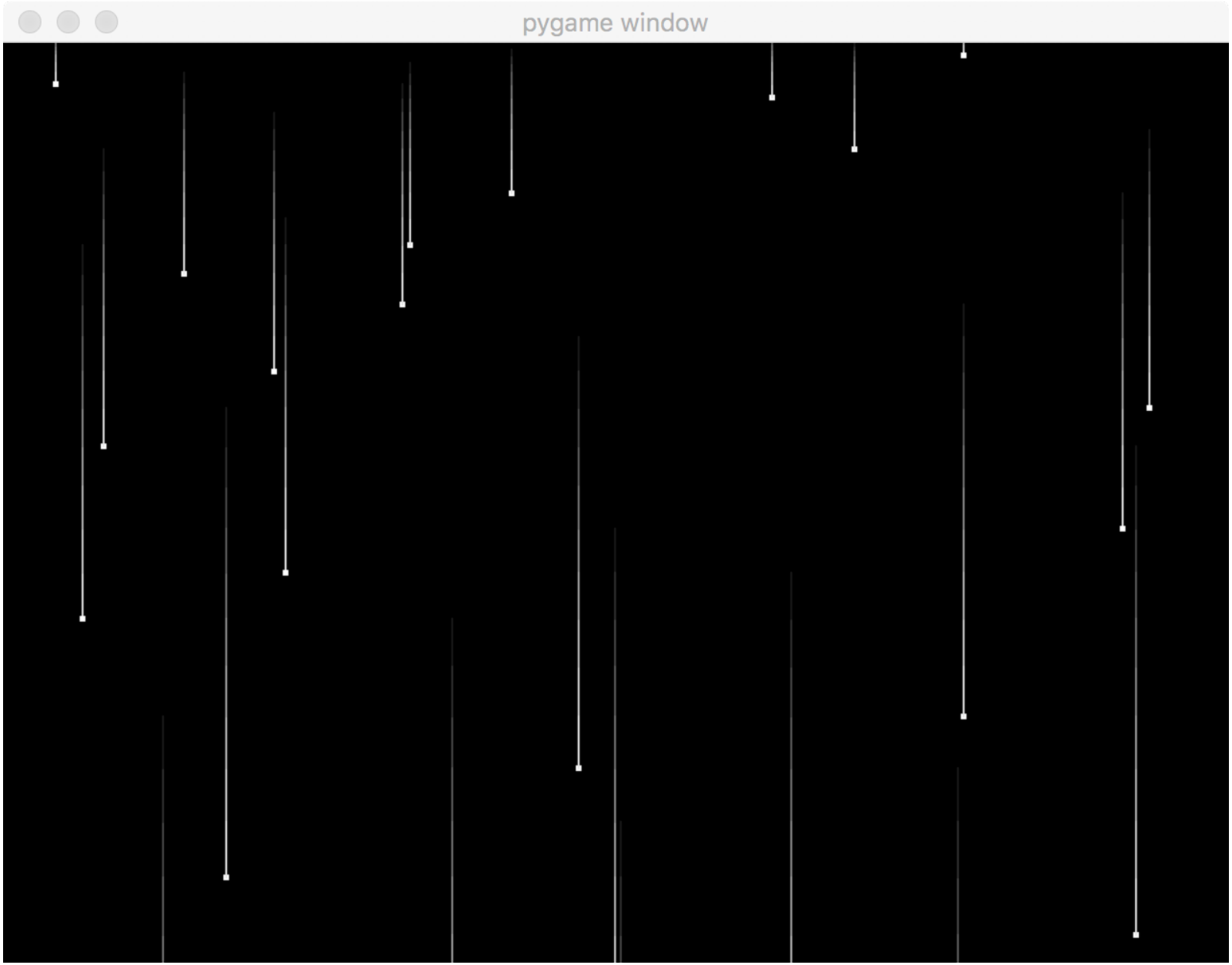
Solving the diffusion equation

It turns out that we can use random walks to solve the diffusion equation. Asymptotically as the number of walks approach infinity, the average properties of a random walk approach the solution to a diffusion equation.

Insight

- We are essentially replacing *human time* with *computer time*. Instead of coming up with a numerical or analytical solution to the problem, we will let computer figure it by running millions and millions of random walks.
- It is easy to model boundary conditions in random walks.
- Random walks can easily exploit multiple processors.

Modeling a rain drop falling in strong wind



Variations on random walks

Persistent random walks

Probabilities for the current step depends upon the previous step.

Multi-state random walks

The random walk can be in one of multiple states, which determine the probabilities for the current step.
This is akin to having a finite state machine with random transitions.

An application of multi-state random walks

Chromatographic columns

One application of multi-state walks is diffusion in a chromatographic column. In this application we are interested in how far a particular compound will move along a chromatographic column in a certain length of time. For each type of compound we have a probability α that it will move v units in each time step and a probability $1 - \alpha$ that it will not move.

Self avoiding random walks

- A SAW is a random walk on a 2D or 3D lattice that cannot return to one of the lattice points that it has already visited
- SAWs are of interest in several areas of physics and mathematics, and surprisingly a number of their properties have resisted rigorous mathematical analysis, so they must be studied in simulation
- SAWs terminate quickly
  - Consider a 2D lattice, a random walk with up, down, left, right step has a 3.7% chance of finishing just after 4 steps
  - How do we generate walks of lengths more than 1000?

Self avoiding random walks

- Adjust statistics to account for a less random selection
  - If one of the three moves is blocked, we can still randomly select one of the two open moves, but giving less weight to the statistics computed for this walk

Computing weight $w(N)$ for a walk of length N

- $w(1) = 1$, since we can always take the first step
- If all three moves are blocked $w(N) = 0$
- If all three moves are possible $w(N) = w(N-1)$
- If only m moves are possible, $0 < m < 3$, then $w(N) = \frac{m}{3} w(N-1)$, select one of the m possible moves at random

Combining statistics over many walks

$$\left\langle R^2 \right\rangle = \frac{\sum_i w_i(N) R_i^2}{\sum_i w_i(N)}$$

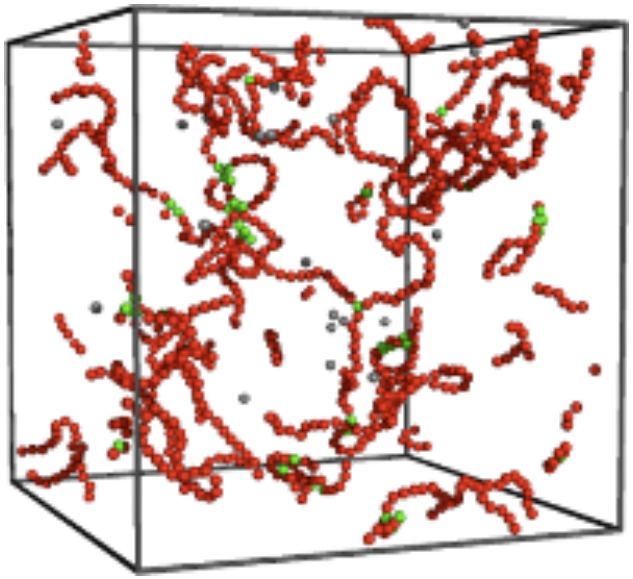
Self avoiding random walks

- The strategy described in the previous slide gives us longer walks, since we are able to avoid some terminations
- Walks with heigher weights are in a way *healthier*, we can make more copies of these
- Compute $r_i = \frac{w(N)}{\langle w(N) \rangle}$. For a healthy walk, $r_i > 1$.
- If $r_i > 1$, make c copies of the walker and assign each of the walkers weight $\frac{w(N)}{c}$
- The number of copies is given by $c = \min(r_i, m)$, where m is the possible number of moves
- If $r_i < 1$ then the walker is removed with probability $1 - r_i$

Applications of self avoiding random walks

Polymer physics

- A polymer consists of N repeated units, called *monomers*.
- Each monomer consists of a small number of tightly bound atoms. Typically N is in the 103 to 105 range.
- For polymers of length N one of the important physical properties is $\langle R_N^2 \rangle$, the mean squared end-to-end length of the polymer



(From U. Reading, UK)

Generating Random Numbers

Random numbers

There are ways to generate *truly* random numbers, but these require physical processes, those that we believe to be truly random.

- Physical processes are costly and there isn't an easy way to interface these with a computer.
- Physical process are *not repeatable*, so we can't use the same set of random numbers in several simulations for testing and comparison. (*Of course we can always save the random numbers*.)

Pseudo random numbers

Pseudo random numbers are generated using deterministic techniques, but they appear to be random

We can always reproduce the same set of random numbers by using the same starting conditions

We can also analyze them mathematically i.e. these numbers have the same statistical properties as “truly” random numbers

Generating pseudo random numbers

All techniques produce random integers in the range 0 to m. We can convert this number to a real number between 0 and 1 through division by m.

1. Start with a number x_0, often called the seed.
2. Use a function $f(x)$ that generates the next number

$$x_{i+1} = f(x_i)$$

Caveat

As soon as f returns a previously generated number, the sequence repeats itself. This results in short sequences with *bad* statistical properties.

Generating uniformly distributed random numbers

- Simplest type of random number generator
- Easy to analyze
- Good random number generators exist

It is possible to generate random numbers from *other distributions* given uniformly distributed random numbers.

Linear Congruential Method (LCM)

One of the best random number generator is also the simplest

$$x_{i+1} = (ax_i + c) \mod m,$$

where

- m, the modulus, $m > 0$.
- a, the multiplier, $0 < a < m$.
- c, the increment, $0 \leq c < m$.
- x_0, the starting value, $0 \leq x_0 < m$.

Python code for LCM

```
def lcm(m, a, c, seed):
    while True:
        seed = (a * seed + c) % m
        yield seed
```

Linear congruential method

- The quality of the random number generator depends heavily on the choice of m, a and c
- The length of the sequence cannot exceed m, since there are only m values less than m, so we want m to be as large as possible
- Good values of a and c depend upon the value of m
- Rubric:
  - c is relatively prime to m
  - $b = a - 1$ is a multiple of p, for prime p dividing m
  - b is a multiple of 4, if m is a multiple of 4

Linear congruential method

LCM has been studied extensively for 50 years or so, and good values for a, c and m are available.

m	a	c
134456	8121	28411
243000	4561	51349
259200	7141	54773
233280	9301	49297
714025	4096	150889

Generating longer random number sequences

In many simulations we often need billions of random numbers. So how can we generate longer sequences. Recall LCM can only produce sequences of length less than or equal to m.

The length of the sequence (after which the sequence repeats itself) is called the *period* of the random number generator.

Shuffling technique

The range of an LCM can be increased using a *shuffling* technique.

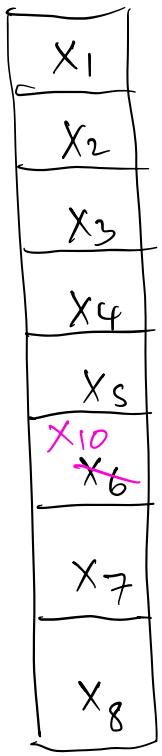
Shuffling technique

Procedure

1. Initialize k entries of array v with random values between 0 and 1.
2. Generate a random number y between 0 and m
3. Compute index $j = k\frac{y}{m}$
4. Set $r = v[j]$
5. Generate a random number y between 0 and m
6. Set $v[j] = y$
7. Returns r

Properties

- If m is the sequence length produced by the base random number generator then shuffling will produce a sequence that is several times m in length.
- Shuffling also reduces any correlations that might exist in the original sequence.



lcm (X_0)

$X_9 \leftarrow \text{lcm}(X_8)$

$0 \leq X_9 \leq m$

$j = \left(\frac{X_9}{m}\right)(8)$

↑ # of slots

$r = X_6$

$X_{10} \leftarrow \text{lcm}(X_9)$

generate : r

Tests for random numbers

General tests for randomness that can be used for any distributions

- Chi-Square test
- Kolmogorov-Smirnov test

Chi-Square test

- A statistical test commonly used to compare observed data with data one would expect to obtain according to a specific hypothesis
- Chi-Square test accepts or rejects the *Null Hypothesis*, which states that there is no statistically significant difference between the observed and the expected frequencies

Chi-Square Test

Consider the following table that shows the "expected" and "observed" counts for some event

	Elephants	Zebras	Antelopes
Expected (E)	10	20	15
Observed (O)	13	18	14

We want to confirm the *Null Hypothesis*, which states that there is no statistically significant difference between the count of animals that we observed and the number of animals that we were expecting to observe.

Chi-Square Test

Compute test statistic χ^2

$$\chi^2 = \sum_i \frac{(O_i - E_i)^2}{E_i},$$

where O_i is the observed count in bin i and E_i is the expected count in bin i

Compute degrees of freedom $d = \#bins - 1$.

Find the critical value at the chosen significance level, often 0.05, and the degrees of freedom.

If the χ^2 is less than the critical value then the Null Hypothesis stands.

Chi-square Test

Use the Chi-square table to find critical value at the chosen significance level and the degrees of freedom.

	$P(X \leq x)$							
	0.010	0.025	0.050	0.100	0.900	0.950	0.975	0.990
r	$\chi^2_{0.99}(r)$	$\chi^2_{0.975}(r)$	$\chi^2_{0.95}(r)$	$\chi^2_{0.90}(r)$	$\chi^2_{0.10}(r)$	$\chi^2_{0.05}(r)$	$\chi^2_{0.025}(r)$	$\chi^2_{0.01}(r)$
1	0.000	0.001	0.004	0.016	2.706	3.841	5.024	6.635
2	0.020	0.051	0.103	0.211	4.605	5.991	7.378	9.210
3	0.115	0.216	0.352	0.584	6.251	7.815	9.348	11.34
4	0.297	0.484	0.711	1.064	7.779	9.488	11.14	13.28
5	0.554	0.831	1.145	1.610	9.236	11.07	12.83	15.09
6	0.872	1.237	1.635	2.204	10.64	12.59	14.45	16.81
7	1.239	1.690	2.167	2.833	12.02	14.07	16.01	18.48
8	1.646	2.180	2.733	3.490	13.36	15.51	17.54	20.09
9	2.088	2.700	3.325	4.168	14.68	16.92	19.02	21.67
10	2.558	3.247	3.940	4.865	15.99	18.31	20.48	23.21

Example

For our example: $\chi^2(2) = 1.67$ and $d = 2$. The critical value at significance level 0.05 for degrees of freedom 2 is 5.991. Since $\chi^2(2)$ is less than the critical value, the Null Hypothesis stands.

Taken from https://onlinecourses.science.psu.edu/stat414/node/147

Chi-Square Test

Another way to use Chi-Square test is to compute p-value as follows

$$\text{p-value} = P(X > \chi^2(d))$$

If p-value is less than then the level of significance (usually 0.05), reject the Null Hypothesis.

Chi square test: p-values and α-values

A p-value is used in hypothesis testing. The smaller the p-value, the stronger the evidence that you should reject the null hypothesis.

- p-value > 0.10: No evidence against the null hypothesis. The data appear to be consistent with the null hypothesis.
- $0.05 <$ p-value < 0.10: Weak evidence against the null hypothesis in favor of the alternative.
- $0.01 <$ p-value < 0.05: Moderate evidence against the null hypothesis in favor of the alternative.
- $0.001 <$ p-value < 0.01: Strong evidence against the null hypothesis in favor of the alternative.
- p-value < 0.001 Very strong evidence against the null hypothesis in favour of the alternative

Chi square test: p-values and α-values

- Significance level is a measure of how certain you want to be about your results: low significance values correspond to a low probability that the experimental results happened by chance.
- Scientists usually set the significance level at 0.05, or 5 percent. This means that experimental results have, at most, a 5% chance of being reproduced in a random sampling process.

Apply Chi-square test to see if a coin is biased

Say we flipped a coin 51 times, and we got 28 heads and 23 tails. These are observed frequencies. The expected number of heads and tails for an unbiased coin, after 51 flips, is 25.5.

$$\begin{aligned}\chi^2 &= \frac{(28-25.5)^2}{25.5} + \frac{(23-25.5)^2}{25.5} \\ &= 0.516\end{aligned}$$

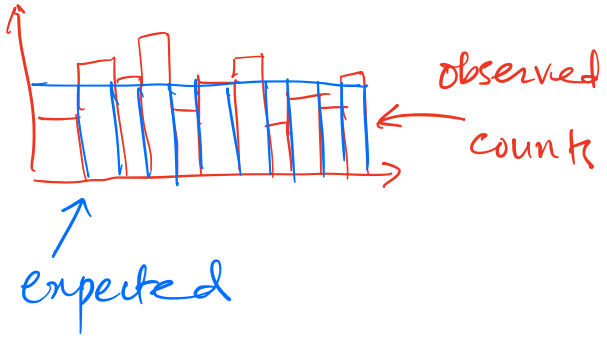
Since for this experiment the number of outcomes is 2 (head or tail), the degree of freedom is $2-1=1$. Using the table above we see that the p-value is between 0.9 and 0.1 (corresponding the $\chi^2_{expected}$) values of 0.016 and 2.706. Consequently p-value is greater than the level of significance, which we choose to be 0.05. We accept the Null Hypothesis.

The coin is not biased.

Chi-square test for uniform random number generator

Given Y_i the number of items in bin i, and p_i the probability that an item is placed in bin i, we compute Chi-Square statistics as follows:

$$\chi^2 = \sum_{i=1}^{k} \frac{(Y_i - np_i)^2}{np_i},$$



where n is the number of items and k is the number of bins.

In the case of uniform random number generator, we expect that the probability of falling in a bin is equally likely, i.e., $p_i = 1/k$.

Apply the Chi square test with degree of freedom $= k - 1$

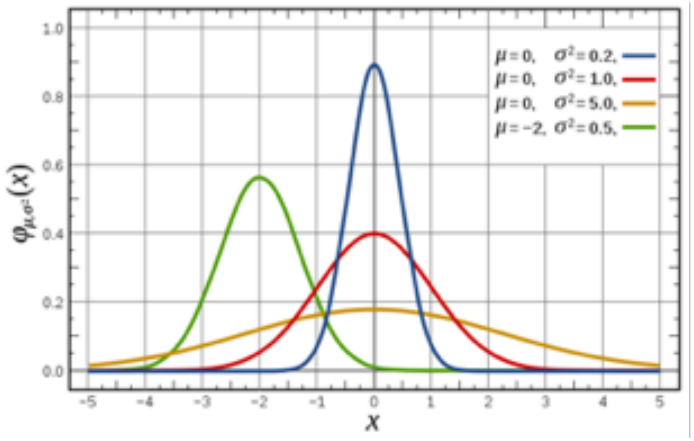
Kolmogorov-Smirnov (KS) test

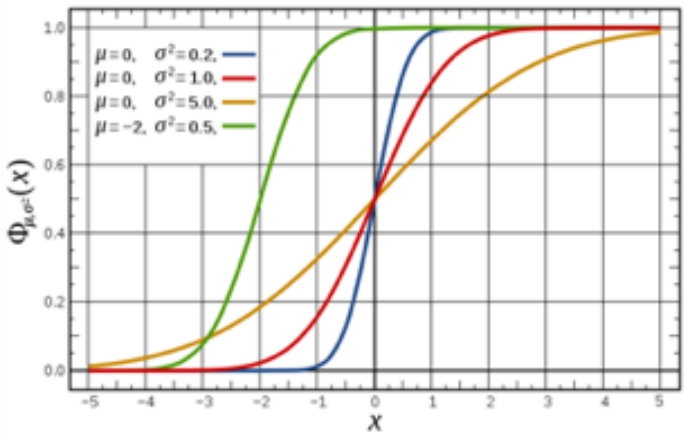
SELF STUDY NOT ON MIDTERM

- The Kolmogorov-Smirnov (KS) test is a non-parametric test used to determine whether a sample comes from a specific distribution.
- It's particularly useful when the distribution of the population is unknown or when the data may not follow a normal distribution.
- The KS test compares the empirical cumulative distribution function (ECDF) of the sample with the cumulative distribution function (CDF) of the specified distribution.
  - The test statistic is the maximum absolute difference between these two functions.

Cumulative Distribution Function

$$F_X(S) = Pr(X \leq x)$$





Calculate the empirical cumulative distribution function (ECDF)

Given a set of n samples as $\{x_1, x_2, \ldots, x_n\}$

1. Sort in ascending order: $x_{(1)} \leq x_{(2)} \leq \ldots \leq x_{(n)}$.
2. For each sample $x_{(i)}$, the cumulative probability $F(x_{(i)})$ is the proportion of samples that are less than or equal to $x_{(i)}$:

$$F(x_{(i)}) = \frac{\text{Number of samples less than or equal to } x_{(i)}}{n} = \frac{i}{n},$$

where i ranges from 1 to n.

This function gives the proportion of samples that are less than or equal to a given value x.

Empirical CDF

The empirical CDF $F_n(x)$ is defined as:

$$F_n(x) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}_{\{x_i \leq x\}},$$

where $\mathbf{1}_{\{x_i \leq x\}}$ is an indicator function that equals 1 if $x_i \leq x$ and 0 otherwise.

Empirical CDF

Example: {1,23,5,54,3,2,46}

Sort samples in ascending order

$$\{1, 2, 3, 5, 23, 46, 54\}$$

$F(23) = 5/7$

Compute empirical CDF:

$F(1) = 1/7, F(2) = 2/7, F(3) = 3/7, F(5) = 4/7, F(23) = 5/7, F(46) = 6/7,$ and $F(54) = 7/7$

Kolmogorov-Smirnov test

- We are given the cumulative distribution function F_{CDF}
- Compute empirical cumulative probability distribution F_{data} from the sample data
- Compute KS statistic—the maximum absolute difference between the empirical cumulative distribution function (ECDF) and the specified cumulative distribution function (CDF):

$$D = \max(|F_{\text{data}}(x) - F_{\text{CDF}}(x)|),$$

  where x ranges over all observed points
- Compute D_{critical} (see table in the next slides), and if $D < D_{\text{critical}}$ then the *null hypothesis* holds
  - Find D_{critical} by using the row for number of samples and column for the significance level.

Kolmogorov-Smirnov test

Table 1: Critical values of the Kolmogorov-Smirnov Test Statistic

SAMPLE SIZE (N)	LEVEL OF SIGNIFICANCE FOR D = MAXIMUM [$F_0(X)$ - $S_n(X)$]				
	.20	.15	.10	.05	.01
1	.900	.925	.950	.975	.995
2	.684	.726	.776	.842	.929
3	.565	.597	.642	.708	.828
4	.494	.525	.564	.624	.733
5	.446	.474	.510	.565	.669
6	.410	.436	.470	.521	.618
7	.381	.405	.438	.486	.577
8	.358	.381	.411	.457	.543
9	.339	.360	.388	.432	.514
10	.322	.342	.368	.410	.490
11	.307	.326	.352	.391	.468
12	.295	.313	.338	.375	.450
13	.284	.302	.325	.361	.433
14	.274	.292	.314	.349	.418
15	.266	.283	.304	.338	.404
16	.258	.274	.295	.328	.392
17	.250	.266	.286	.318	.381
18	.244	.259	.278	.309	.371
19	.237	.252	.272	.301	.363
20	.231	.246	.264	.294	.356
25	.210	.220	.240	.270	.320
30	.190	.200	.220	.240	.290
35	.180	.190	.210	.230	.270
OVER 35	$\frac{1.07}{\sqrt{N}}$	$\frac{1.14}{\sqrt{N}}$	$\frac{1.22}{\sqrt{N}}$	$\frac{1.36}{\sqrt{N}}$	$\frac{1.63}{\sqrt{N}}$

Kolmogorov-Smirnov test

Example

D_{critical} for 5 samples at 0.05 significance level is 0.565

Kolmogorov-Smirnov test

We can use KS test for testing uniform random number generators

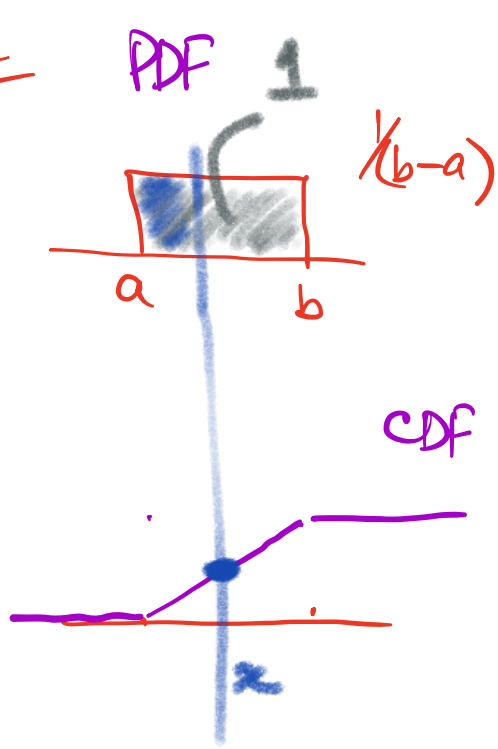
- Generate N samples and estimate empirical CDF
- Use CDF for uniform distribution to compute KS statistic D
- Find D_{critical} value using N bins and the desired significance level
- If $D < D_{\text{critical}}$, the Null Hypothesis stands

CDF for uniform distribution

Given $X \sim U(a, b)$

CDF on support $a < x < b$ is

$$F(x) = \int_a^x \frac{1}{b-a} dw$$
$$= \frac{w}{b-a}\Big|_a^x$$



Solving this we get

$$F(x) = \begin{cases} 0 & x \leq a \\ \frac{x-a}{b-a} & a < x < b \\ 1 & x \geq b \end{cases}$$

Tests for uniformly distributed random numbers

- Equidistribution test
- Serial test
- Coupon collecter's test
- Serial correlation test

NOT ON MIDTERM

Equidistribution test

The equidistribution test is based on the fact that uniform random integers should be evenly distributed amongst the possible integer values

1. Chose some convenient d (not too large) and then generate a sequence of random integers between 0 and $d-1$
2. Assign them to d bins based on their integer values (since the numbers are uniformly distributed we would expect to find an equal number of integers in each bin)
3. With a sequence of n numbers, we would expect to find $\frac{n}{d}$ integers in each bin, so we can use a Chi-square test. In this case we have $p_i = \frac{1}{d}$, and the number of degrees of freedom is $d-1$
4. Use Chi-square test

Serial test

Can be considered an extension of equidistribution test that considers pairs of adjacent numbers.

Key idea

Given a sequence of random integers between 0 and $d-1$. Consider a pair of adjacent numbers:

$$(Y_{2i}, Y_{2i+1})\,.$$

Since each pair is equally probable, so the pairs should be evenly distributed over d^2 bins.

Serial test

1. Generate a sequence of n random integers
2. Divide the sequence into $\frac{n}{2}$ pairs.
3. Each pair is assigned to one of the d^2 bins.
4. Apply Chi-square test. Probability for each bin is $\frac{1}{d^2}$ and the degrees of freedom is d^2.

Caveat

- For this to work the sequence should be reasonable long (the length should be at least $5d^2$).
- This approach can be extended to higher dimensions. However, it quickly becomes infeasible since the number of bins grow *very* rapidly.
- Not practical beyond quadruples of random numbers.

Coupon collector's test

- This test is also based on a sequence of integer random numbers between 0 and $d-1$, but now we are interested in collecting a complete set of integers between 0 and $d-1$, have at least one of each
- In particular we are interested in the length of sequence r that it takes to get the complete set

Procedure

- We start at the beginning of the sequence and keeping moving down the sequence until we find a complete set of d integers
  - The position where this occurs is called the sequence length
- We then move to the next item in the sequence and repeat the process
- We are interested in the distribution of sequence lengths

Coupon collector's test

- The following experession gives probabilities for sequences of lengths $n \geq 10$

$$p_n = \frac{1}{10^{n-1}} \sum_{j=0}^{q} (-1)^j \left(\begin{array}{c} q \\ j \end{array} \right) (q-j)^{n-1}$$

- Use Chi-Square test to see if the observed sequence lengths match those returned by the above expression

Spectral test

- One of the best tests for random number generators is the Spectral test
- This test can only be applied to linear congruential random number generators
- So far all random number generators that are known to be good pass this test, and all that are known to be bad fail it
- This test is quite complicated, so we won't examine the details, only the basic ideas
- The test operates on the real random numbers sequence and it examines all m numbers in the sequence
- The test starts by constructing t dimensional points out of the sequence of random numbers. We construct m points as follows:

$$(U_n, U_{n+1}, U_{n+2}, \cdots, U_{n+t-1})$$

- A new point is constructed starting from each number, so each number in the original sequence appears in t points.

Generally speaking t is between 2 and 6.

Spectral test

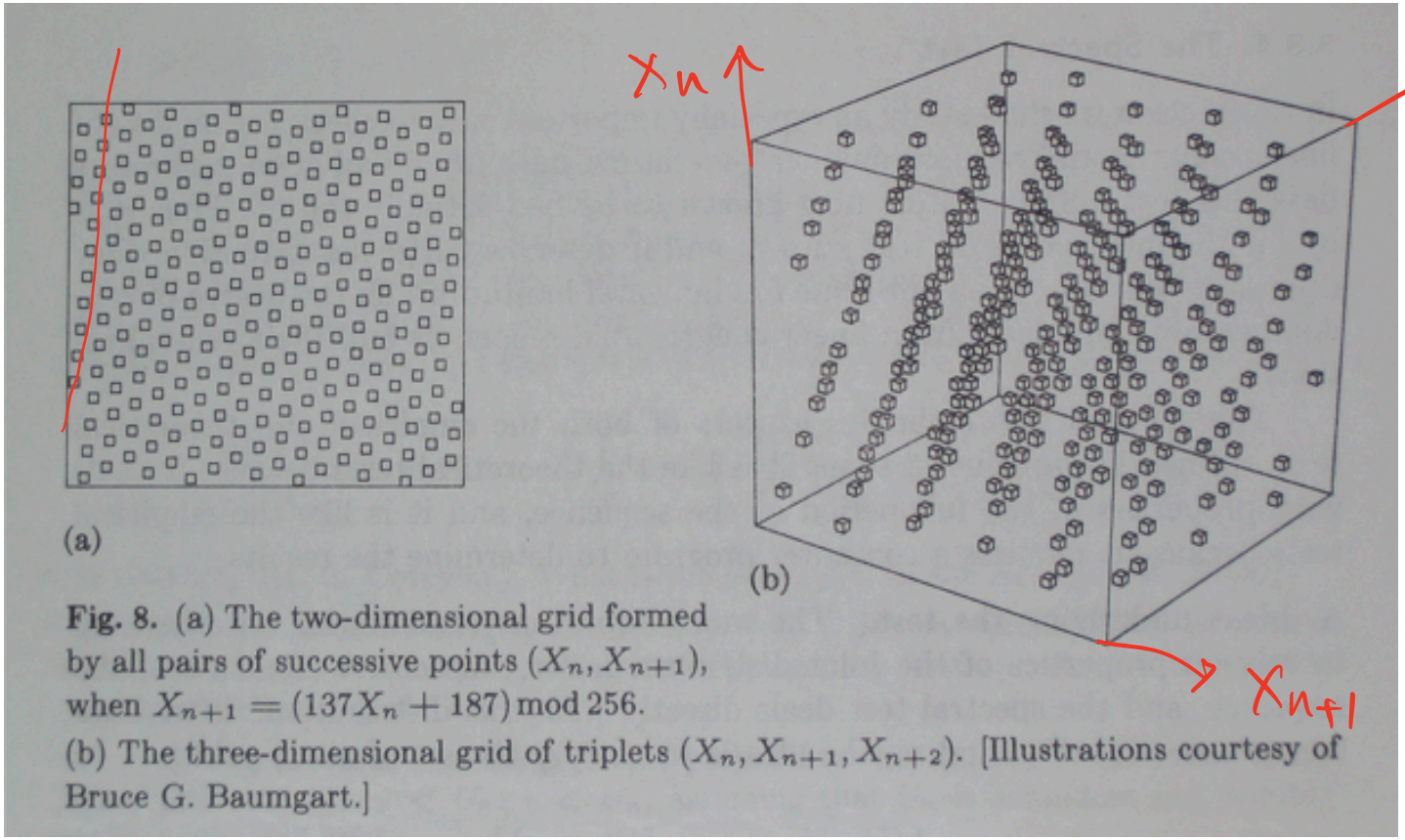


Fig. 8. (a) The two-dimensional grid formed by all pairs of successive points (X_n, X_{n+1}), when $X_{n+1} = (137X_n + 187) \bmod 256$. (b) The three-dimensional grid of triplets (X_n, X_{n+1}, X_{n+2}). [Illustrations courtesy of Bruce G. Baumgart.]

- Notice the pattern observed in the above figure
- "Real" random numbers when truncted exhibit the same pattern
- *Grain* of the random number generator

A note about the quality of random number generators

- These tests are applied to different sequences generated by a random number generator
- All sequences from a “good” generator will pass these tests
- In practice, however, some sequences even from a “good” generator will fail some tests
  - This behavior is also observed for “true” random number sequences

Generating random numbers from uniform distributions

Given: uniform random number generator that return an integer U between 0 and d.

Recipe to get a uniform random number r between a and b.

$$r = a + (b - a)\frac{U}{d}$$

Recipe to get a uniform random number r between 0 and 1.

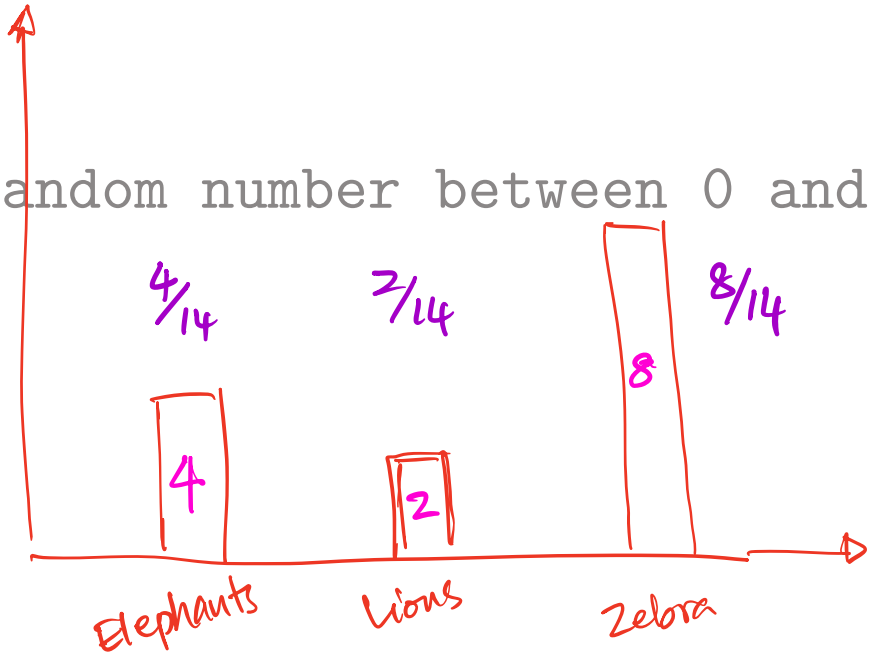
$$r = \frac{U}{d}$$

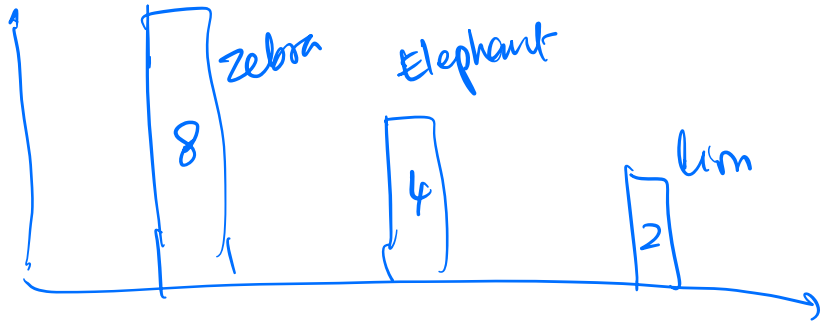
Generating random numbers from non-uniform discrete distributions

Say we want to generate samples form a following discrete distribution: x_i with probability p_i, where $i \in [0, n]$ and $\sum_i p_i = 1$.

Recipe

```
U = rand(0,1) # uniform random number between 0 and 1
if U < p[0]:
    return x0
elif U < p[0]+p[1]:
    return x1
...
else:
    return xn
```





Generating random numbers from non-uniform discrete distributions

We can also store the cumulative probabilities, so we don't have to compute probability sums eachtime as follows:

$$c_j = \sum_{i=0}^{j} p_i,$$

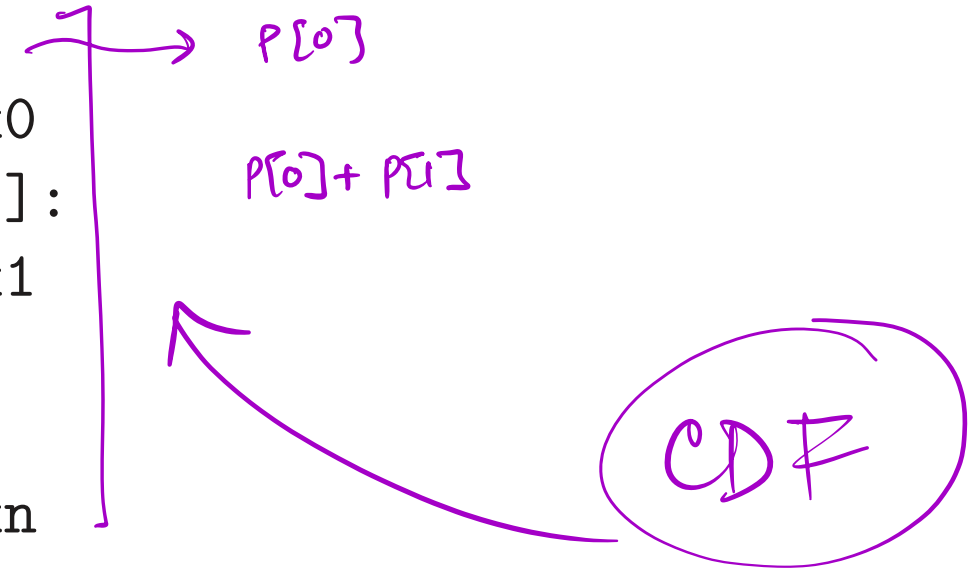
where $j \in [0, n]$.

Always keep the most likely choices (those with higher probabilities) at the top. This saves a lot of "if" comparisons. Also allows us to use binary search to find the correct bin.

Generating random numbers from non-uniform discrete distributions

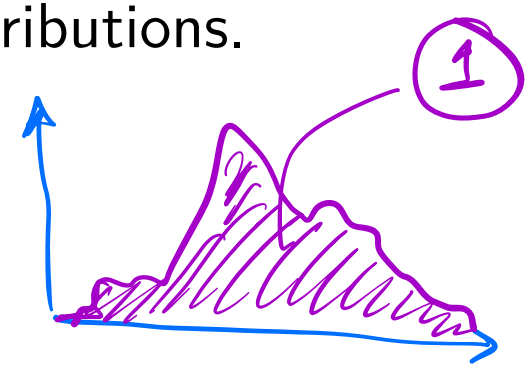
Recipe Using cumulative probabilities

```
U = rand(0,1) # uniform random number between 0 and 1
if U < c[0]:
    return x0
elif U < c[1]:
    return x1
...
else:
    return xn
```

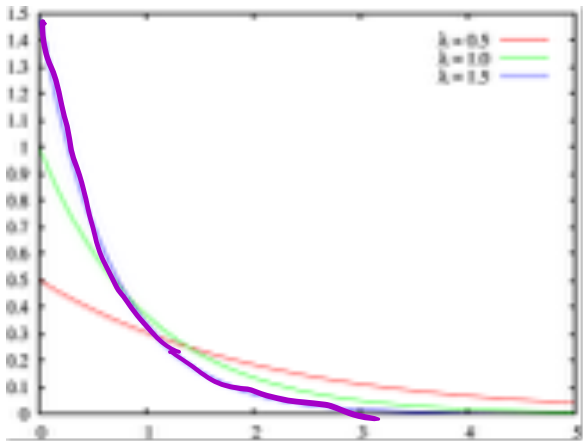


Generating random numbers from non-uniform continuous distributions

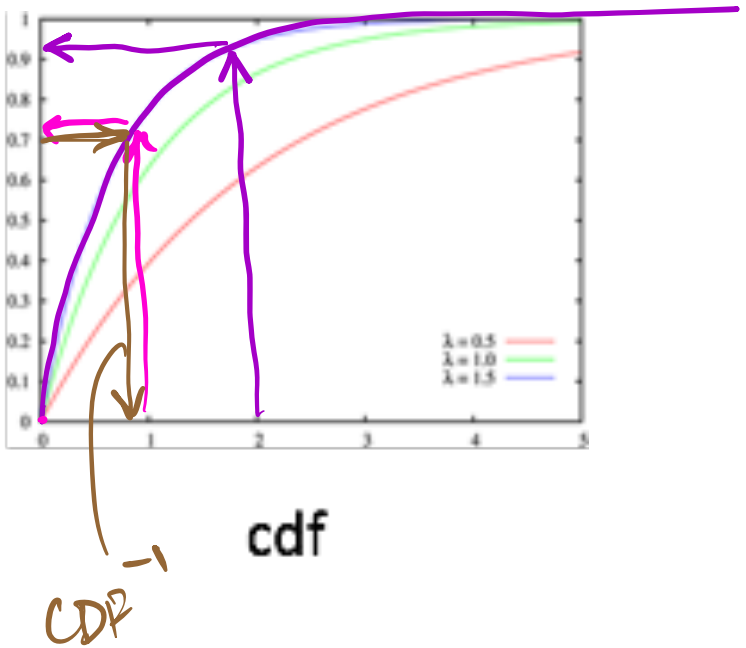
Cumulative probability distribution $F(X)$ is used to generate samples from non-uniform continuous distributions.



- $F(-\infty) = 0$
- $F(\infty) = 1$
- $0 \leq F(x) \leq 1$
- $F(x_1) \leq F(x_2)$ if $x1 \leq x2$



pdf



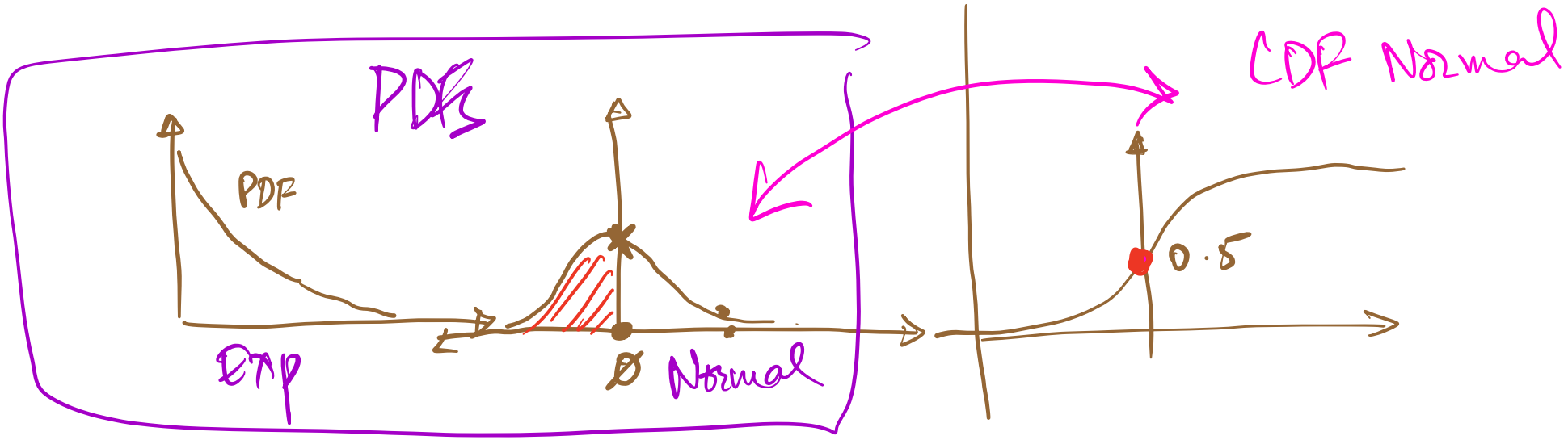
cdf

Generating random numbers from non-uniform continuous distributions

Consider the cumulative probability distribution $F(X) = Pr(x \leq X)$ (associated with probability distribution $f(x)$, where x is the random variable with distribution $f(x)$).

Insight

A uniform random number generator produces a random probability and the value of $F(X)$ is a probability, so the inverse of $F(X)$ takes the probability and returns a random number with probability $f(X)$.



Generating random numbers from non-uniform continuous distributions

Given $F(X) = Pr(x \leq X)$ (associated with probability distribution $f(x)$, where x is the random variable with distribution $f(x)$).

If $F^{-1}(X)$ then it is straightforward to get a random number from $f(x)$ using the following recipe.

```
u = rand(0,1) # a uniform random number between 0 and 1

return F_inv(u)
```

inverted CDF function

The key challenge is that in many cases F^{-1} doesn't exist.

Inverse of a function

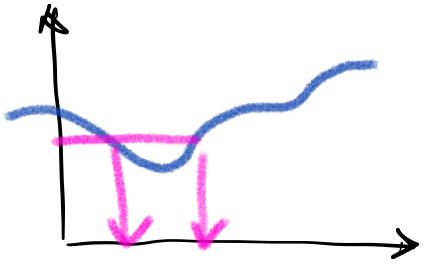
Given a function $f(x)$, you want to find the function $f^{-1}(x)$ such that $f^{-1}(f(x)) = x$ for all x in the domain of f.

Here are the general steps to find the inverse of a function:

- Replace $f(x)$ with y, so $y = f(x)$
- Swap x and y, so we get $x = f^{-1}(y)$
- Solve for y
- Replace y with $f^{-1}(x)$
- Make sure that the domain of the original function matches the range of the inverse function, and vice versa.

$y = f(x)$ Say $g(x) = f^{-1}(x)$

$g(f(x)) = x$



$f(x) = 3x + 7$

$g(f(x)) = x \Rightarrow f(g(x)) = x$

$\downarrow$

$3g(x) + 7 = x$

$\Rightarrow 3g(x) = x - 7$

$\Rightarrow g(x) = \frac{x-7}{3}$

$y = 3x + 7$

$\Rightarrow x = 3y + 7$

$\Rightarrow 3y = x - 7$

$\Rightarrow \boxed{y} = \frac{x-7}{3}$

$f^{-1}(x)$

Inverse of a function

- Not all functions have inverses.
- The inverse of a function may need to be restricted to a certain domain in order to be a function. This restriction ensures that the inverse function is one-to-one.
- For some functions, finding the inverse may not be straightforward or even possible using elementary functions. In such cases, numerical methods or special techniques may be required to approximate or find the inverse.

Exponential distribution - PDF

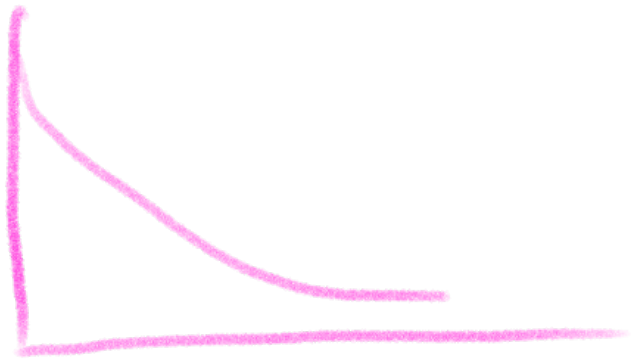
Exponential distributions, for example is used in nuclear decay — if a substance emits a particle every μ seconds on average, then the times between two emissions will be exponentially distributed with mean μ.

The probability density function (PDF) of the exponential distribution is

$$f(x) = \lambda e^{-\lambda x} \quad \text{for } x \geq 0,$$

where λ is the rate parameter. μ and λ are related as follows

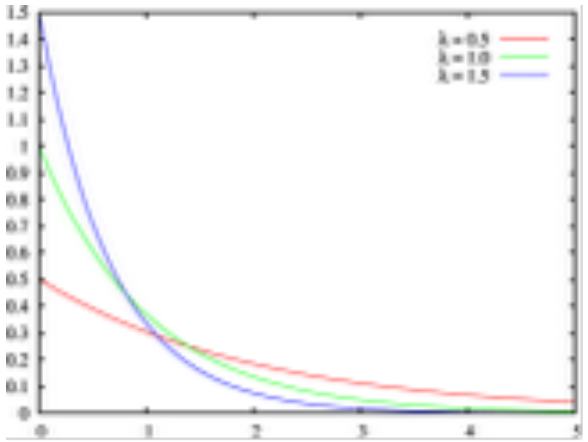
$$\lambda = \frac{1}{\mu}$$



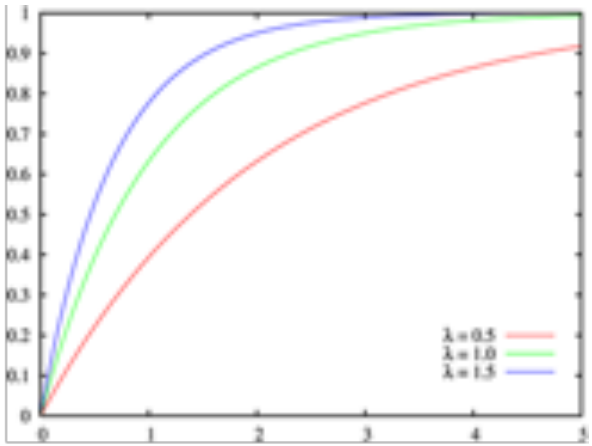
Exponential distribution - CDF

To find the cumulative distribution function (CDF), $F(x)$, we integrate the PDF from 0 to x:

$$F(x) = \int_0^x \lambda e^{-\lambda t}\, dt$$
$$= \left[-e^{-\lambda t}\right]_0^x$$
$$= 1 - e^{-\lambda x}$$



pdf



cdf

Generating random numbers from *exponential distributions*

$F^{-1}(x)$ for cumulative probability distribution for *exponential distribution* is

$$F^{-1}(x) = -\frac{1}{\lambda}\ln(1 - U)$$

ln is slow; however, this approach is acceptable unless we plan to generate a very large number of samples. U is a uniform random number between 0 and 1.

$f(x) = 1 - e^{-\lambda x}$

$f(g(u)) = u$

$1 - e^{-\lambda g(u)} = u$

$e^{-\lambda g(u)} = 1 - u$

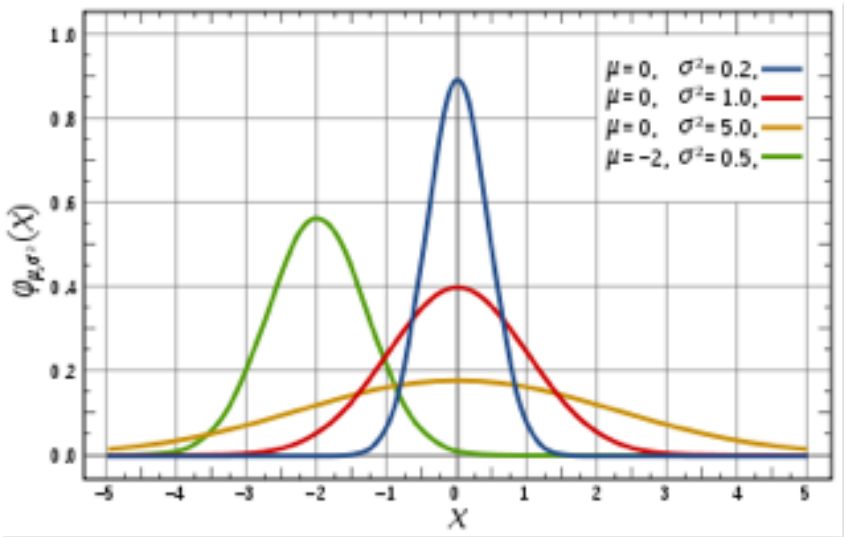
$-\lambda g(u) = \ln(1-u)$

$g(u) = -\frac{1}{\lambda}\ln(1-u)$

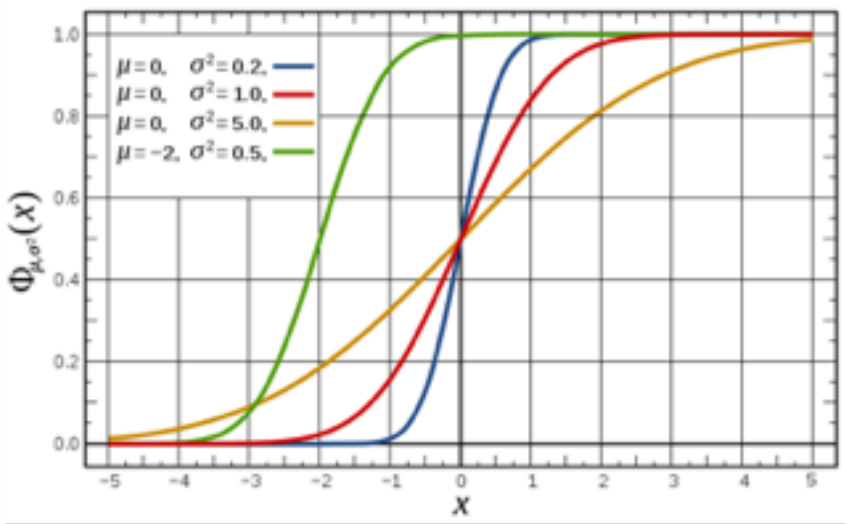
$f(x) =$

Generating samples from normal distribution

One of the most used probability distribution



pdf



cdf

Generating samples from normal distribution

We will focus on generating samples form normal distribution of mean (μ) 0 and variance (σ^2) 1.

It is easy to generate samples from normal distribution $N(\mu, \sigma)$ as follows:

1. $n \sim N(0, 1)$ (a random number generated from a uniform distribution with mean 0 and variance 1)
2. $\mu + \sigma n \sim N(\mu, \sigma)$

Generating samples from normal distribution

Cumulative probability distribution for $N(0,1)$ is

$$F(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{t^2}{2}} dt$$

In this case it is $F(x)$ is invertible; however, it is not easy to compute.

We will look at two other approaches (for generating numbers from normal distribution) that give acceptable results.

Generating samples from normal distribution

We will use the rejection method to generate samples from a normal distribution

Rejection method

1. Generate a sample and check to see if it is from the *desired* distribution.
2. If yes, return the sample.
3. Else, try again.

The trick is to guess the correct samples more frequently, to avoid having to generate many samples.

Polar method for generating samples from normal distribution

1. Generate two uniform random numbers U_1 and U_2
2. Set $V_1 = 2U_1 - 1$ and $V_2 = 2U_2 - 1$
3. Compute $S = V_1^2 + V_2^2$
4. If $S \geq 1$ return to step 1
5. Else compute

$$X1 = V_1 \sqrt{\frac{-2 \ln S}{S}}$$

and

$$X2 = V_2 \sqrt{\frac{-2 \ln S}{S}}$$

6. Return X_1 and X_2

Polar method for generating samples from normal distribution

- This method produces two samples
- The downside is that the method uses logarithm and square root operations, which are expensive
- The polar method needs a random point (V_1, V_2) distributed on a circle with radius 1. This is hard to generate, but it is easy to generate a random point within a square. So we use the smallest square containing the circle and generate a random point within this square. If this point is also within the circle we continue, otherwise we generate another point.

Ratio method for generating samples from normal distribution

1. Generate two uniform random numbers U_1 and U_2
2. Set $X = \sqrt{\frac{8}{2e}} \frac{(U_2 - \frac{1}{2})}{U_1}$
3. If $X^2 \leq 5 - 4e^{\frac{1}{4}} U_1$ return X
4. If $X^2 \geq 5 - 4e^{-1.35} U_1 + 1.4$ go back to step 1
5. If $X^2 \leq -\frac{4}{\ln U_1}$ return X
6. Go back to step 1

Steps 2 and 3 are optional, but they increase the efficiency of the algorithm considerably. In this case we only produce one random number, but we avoid using logarithm most of the time, so it could be more efficient than the polar method. Again we generate a pair of random numbers, and then check to see if they produce the right result, otherwise we try again.

Summary

- Monte Carlo techniques
- Random walks
- Techniques for generating and testing sequences of random numbers
- Applications of random walks