Lecture 12: Stochastic programs, Monte Carlo sampling

(download slides and .py files to follow along)

Tim Kraska

MIT Department of Electrical Engineering and Computer Science

Announcements

The Simple World of Newtonian Mechanics

- Want to build computational models of physical world
- Every effect has a cause, so physical world can be understood causally
  - E.g., Newton's three laws of motion
  - Suggests that mathematical descriptions of physical effects are possible

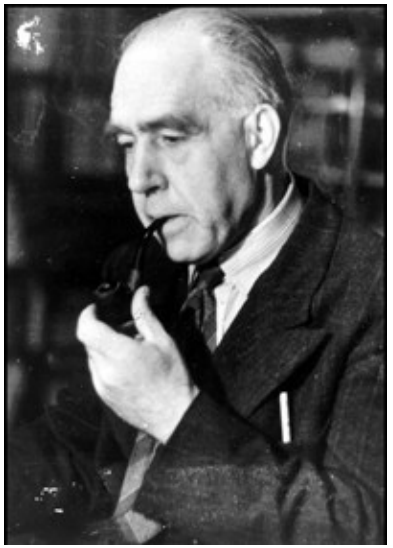
1643 – 1727

Two+ Centuries After Newton

- First “bug” : radioactive decay
- Then Heisenberg and Bohr argued that at its most fundamental level, the behavior of the physical world cannot be predicted
  - For example, cannot precisely measure position and momentum of a particle at the same time
- Fine to make statements of the form “x is highly likely to occur,” but not of the form “x is certain to occur”
- Introducing uncertainty into modeling physical world

“The more precise the measurement of position, the more imprecise the measurement of momentum, and vice versa.”

“Those who are not shocked when they first come across quantum theory cannot possibly have understood it. …If you think you can talk about quantum theory without feeling dizzy, you haven't understood the first thing about it.”



Not Everyone Bought It

- Einstein and Schrodinger objected
  - "I, at any rate, am convinced that He [God] does not throw dice."– Albert Einstein
  - Bohr, in response, said, "Einstein, don't tell God what to do."

Does It Really Matter?

- Suppose I flip two coins
- Can you correctly predict whether the flips will yield
  - 2 heads?
  - 2 tails?
  - 1 head and 1 tail?
- Need to know accurately:
  - weight distribution of coin
  - velocity and acceleration of thumb
  - orientation of coin on thumb before flip
  - air flow around coin
  - height above landing spot
  - elasticity of floor
  - …

The Moral

- The world might or might not be inherently unpredictable
- Even if not, lack of knowledge prevents precise predictions
- Therefore, can treat the world as inherently unpredictable

The Moral

- The world might or might not be inherently unpredictable
- Even if not, lack of knowledge prevents precise predictions
- Therefore, can treat the world as inherently unpredictable

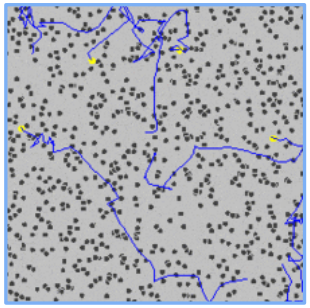
Causal nondeterminism – some events truly random

Predictive nondeterminism – in principle might be able to predict, but don't have enough information. There is chaos, but not randomness

Use a **stochastic** process to represent systems or to model phenomena that seem to change in a random way

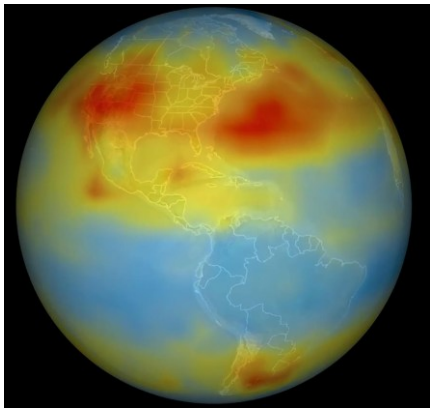
Some Places We Use Stochastic Models

- Systems with choices in state transitions, e.g.,
  - Motion of particles in a fluid (Brownian motion)
  - Outcomes of play in football



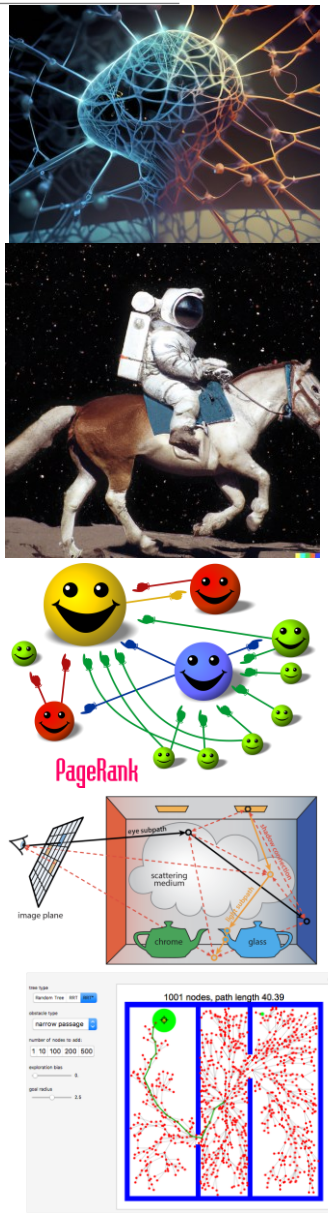
- Systems where measurement is uncertain or noisy, e.g.,
  - Polling data
  - Radio astronomy

- Systems where can't measure all factors, e.g.,
  - Weather systems



Stochasticity and Randomness in Computing

- Stochastic Gradient Descent is at the heart of deep learning and modern AI
- Markov Chain Monte Carlo methods are used throughout inference
- Simulated annealing uses randomness to find optimal solutions
- Google's Page Rank search algorithm
- Large language models
- Realistic 3D rendering uses Monte-Carlo random sampling to compute incoming light and to antialias pixels
- In robotics, path planning and control often use rapidly-exploring Random Tree
- Simulation of complex systems
- Perturbation analysis uses randomness to characterize the sensitivity of systems and methods
- Randomized algorithms



Today's Lecture

- Randomness, simulation, and stochastic thinking

SMART NEWS

Mathematician Who Made Sense of the Universe's Randomness Wins Math's Top Prize

Michel Talagrand took home the 2024 Abel Prize for his work on stochastic systems, randomness and a proof of a physics reaction that many experts thought was unsolvable

Top prizes in both Math and CS Awarded in 2024

Computer Theorist Wins $1 Million Turing Award

This year's honor will go to Avi Wigderson, an Israeli-born mathematician and theoretical computer scientist who specializes in randomness.

Stochastic Processes

- In general, a system often can be defined by a set of state variables, and processes that determine the transition to a next set of values for those variables
  - Causal processes
    - Example: a car's state at a point in time is describe by a position, heading and velocity; its state at a subsequent time is determined by those parameters and the actions of the driver
- In a **stochastic** system, the process for determining next state might depend both on the previous states of the process **and on some random element**
  - Including for predictive nondeterminism
    - Example: a car's future position may not be exactly determined by its kinematics and the actions of the driver because of road conditions, topography, etc.
- This will require a change in how we think about building computational models of the world

Two Specifications for a Process

```
def roll_die():
    """Return an int between 1 and 6"""
```

Two Specifications for a Process

```
def roll_die():
    """Return an int between 1 and 6"""
    return 2
```

Two Specifications for a Process

```
def roll_die():
    """Return an int between 1 and 6"""
    return 2
```

`random.choice` selects an element from a list with equal likelihood

```
def roll_die():
    """Return a random int between 1 and 6"""
    return random.choice([1, 2, 3, 4, 5, 6])
```

Two Specifications for a Process

```
def roll_die():
    """Return an int between 1 and 6"""
    return 2
```

`random.choice` selects an element from a list with equal likelihood

```
def roll_die():
    """Return a random int between 1 and 6"""
    return random.choice([1, 2, 3, 4, 5, 6])
```

Any implementation that satisfies the second specification would also satisfy the first.

But one that satisfies the first specification might or might not satisfy the second

Simulate Five Rolls

```
def roll_die():
    """Return a random int between 1 and 6"""
    return random.choice([1, 2, 3, 4, 5, 6])

def test_roll(n):
    result = ''
    for _ in range(n):
        result += ' ' + str(roll_die())
    return result

for _ in range(10):
    print(test_roll(5))
```

How probable is the output 11111?

Discrete Probability is About Counting

- Count the total number of possible events (often called the universe of events)
- Count the number of events that have the property of interest
- Divide second number by the first
- Probability of rolling 11111?
  - All events: 11111, 11112, 11113, …, 11121, 11122, …, 66666
  - Ratio: 1/(6**5)
  - ~0.0001286
- Probability of 12345? Same probability

Some Basic Facts about Probability

- (Discrete) Probabilities are always in the range **0 to 1**. 0 if impossible, and 1 if guaranteed
- If the probability of an event occurring is **p**, the probability of it not occurring must be **1-p**
- When estimating probabilities that involve multiple events, must start with the question of whether the events are **independent** of each other

Independence

- Two events are **independent** if the outcome of one event has no influence on the outcome of the other
- Independence should not be taken for granted

How to create (Pseudo) Randomness

How do we create a random number?

Simple Linear Congruential Generator (LCG)

```
seed = 12345

def set_seed(new_seed):
    global seed # tells Python to use the variable outside of the function
    seed = new_seed

def random_nb():
    global seed
    a = 1664525
    c = 1013904223
    m = 2**32
    seed = (a * seed + c) % m
    return seed / m
```

- **The modulus** $m = 2^{32}$
  - It ensures outputs fits in a 32-bit unsigned integer range.
- **The increment** $c = 1013904223$
  - The key rule: to achieve a *full period*, c must be **relatively prime** to m.
  - Because $m$is a power of 2, this means c just has to be **odd** — which it is.
  - Being large and odd helps distribute outputs better across the 32-bit range.
- **The multiplier** $a = 1664525$
  - Chosen carefully to give good statistical properties when combined with $m$and c.

More on pseudo random numbers

- How do we get always different random numbers?
- How do we ensure to always get the same (pseudo) random numbers in the same order? Why is this useful?
- How can we create other Random Distributions?

```
def random_int_range(start=0, end=1):
    r = random_nb()  # base random number between 0 and 1
    return int(start + r * (end - start))

def random_poisson(lmbda):
    """Generate a random integer following a Poisson distribution (Knuth algorithm)."""
    L = math.exp(-lmbda)
    k = 0
    p = 1.0

    while True:
        k += 1
        p *= random_nb()
        if p <= L:
            break
    return k - 1
```

Python random module

random.seed(a=None)

- Initializes the random number generator.
- If a is None, uses current system time or OS entropy.
- Using the same seed → same random sequence (useful for reproducibility)

Uniform Numbers

- random.random() → float in [0.0, 1.0)
- random.uniform(a, b) → float in [a, b]
- random.randint(a, b) → integer in [a, b] (inclusive)

Choice and Sampling

- random.choice(seq) → random element from a sequence
- random.choices(seq, weights=None, k=1) → list of k elements (with optional weighting)
- random.sample(population, k) → unique sample (no repeats)
- random.shuffle(seq) → shuffles list *in place*

Python random module

- random.normalvariate(mu, sigma) → normal (Gaussian)
- random.gauss(mu, sigma) → faster cached Gaussian (same output as above)
- random.expovariate(lambd) → exponential
- random.lognormvariate(mu, sigma) → log-normal
- random.paretovariate(alpha) → Pareto

And others

SIMULATING PROBABILITIES

Using Simulation to Estimate Probabilities

- Going to explore simulation of rolling a group of dice many times
- First of many simulations we will see
- **Simulation**
  - Run many trials in which we select one event from the universe of possible events
  - For each trial, compute some properties of event
  - Report some statistics about the properties over the set of trials

A Simulation of Die Rolling

Simulate trying to roll a specific sequence

```
def run_sim(goal, num_trials):
    total = 0
    for i in range(num_trials):


        result = ''
        for _ in range(len(goal)):
            result += str(roll_die())
        if result == goal:
            total += 1
```

A Simulation of Die Rolling

Simulate trying to roll a specific sequence

```
def run_sim(goal, num_trials):
    total = 0
    for i in range(num_trials):
        if i != 0 and i % 100_000 == 0:
            print(f"Starting trial {i}")
        result = ''
        for _ in range(len(goal)):
            result += str(roll_die())
        if result == goal:
            total += 1
    print(f"Actual probability of {goal} = "
          f"{1 / 6**len(goal):.8f}")
    print(f"Estimated Probability of {goal} = "
          f"{total / num_trials:.8f}")


run_sim('11111', 1000)
```

A Simulation of Die Rolling

Simulate trying to roll a specific sequence

```
def run_sim(goal, num_trials):
    total = 0
    for i in range(num_trials):
        if i != 0 and i % 100_000 == 0:
            print(f"Starting trial {i}")
        result = ''
        for _ in range(len(goal)):
            result += str(roll_die())
        if result == goal:
            total += 1
    print(f"Actual probability of {goal} = "
          f"{1 / 6**len(goal):.8f}")
    print(f"Estimated Probability of {goal} = "
          f"{total / num_trials:.8f}")

run_sim('11111', 1000)
```

Repeat for many trials

Roll appropriate # of dice

Count # of hits

Compute observed probability

Output of Simulation

- Actual probability = 0.0001286
- Estimated probability = 0.0

Output of Simulation

- Actual probability = 0.0001286
- Estimated probability = 0.0
- Actual probability comes directly from math, but estimated probability?
- Why did simulation give the **wrong** answer?

Output of Simulation

- Actual probability = 0.0001286
- Estimated probability = 0.0
- Actual probability comes directly from math, but estimated probability?
- Why did simulation give the **wrong** answer?
  - 6**5 = 7776 possibilities, so in 1000 trials we are unlikely to observe one event in that universe with enough frequency to estimate accurately
  - Even if observed once, estimate would then be 0.001

Output of Simulation

- Actual probability = 0.0001286
- Estimated probability = 0.0
- Actual probability comes directly from math, but estimated probability?
- Why did simulation give the **wrong** answer?
  - 6**5 = 7776 possibilities, so in 1000 trials we are unlikely to observe one event in that universe with enough frequency to estimate accurately
  - Even if observed once, estimate would then be 0.001

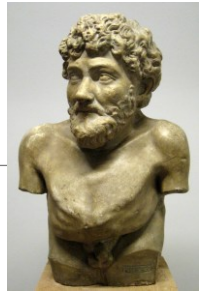
Let's try 1,000,000 trials

Actual probability of 11111 = 0.0001286
Estimated Probability of 11111 = 0.000131

How about 10M trials

Actual probability of 11111 = 0.0001286
Estimated Probability of 11111 = 0.0001286

Morals



Aesop
620 BC – 564 BC

- Moral 1: it takes a lot of trials to get a good estimate of the frequency of occurrence of a **rare** event. We'll talk lots more in later lectures about how to **know** when we have enough trials to trust the estimate
- Moral 2: one should not confuse the **sample probability** with the actual probability
- Moral 3: there was really no need to do this by simulation, since there is a perfectly good closed form answer. We will see many examples where this is not true, where only simulation can provide answers about processes and outcomes

What did we just do? Monte Carlo Simulation

- **Model uncertain inputs:**
  - Identify the variables in a model that are subject to uncertainty and define them using probability distributions.
- **Run trials:**
  - Generate a set of random samples from the input distributions for each variable. Each complete set of samples represents a single "trial" or "iteration".
- **Calculate results:**
  - Calculate the output of the model for each trial based on the specific random inputs for that iteration.
- **Aggregate results (MC of MC) → More on it in later lectures**
  - Repeat the process hundreds or thousands of times. The results from all the trials are collected to form a probability distribution of possible outcomes.
- **Analyze the distribution:**
  - Analyze the aggregated results to understand the range of possibilities and the likelihood of each outcome, rather than relying on a single average estimate.

Simulations Are Used a Lot

- To model systems that are mathematically intractable
  - Otherwise may not be possible to analyze a system
- To extract useful intermediate results
- To support iterative development by successive refinement
  - Start with basic system, then incrementally add features
- To support exploring of variations by asking/answering "what if" questions
- Let's illustrate with Monte Carlo Simulation

Another Example

- Assume you had a great boardgame idea, you want to commercialize
- You estimate that you can sell 10,000 games at a price of $15 and a cost of $10, and a fixed cost of $10k. So overall you expect
  10k*($15 - $10) - $10k = $40k profit
- However, you are also uncertain about all of these numbers and you estimate that the price might have to be between $10 and $20, whereas the cost might vary between $5 and $15 and demand might follow a normal distribution with a variance of 1000

Under these conditions, how likely is it that you will make a profit?

```python
random.seed(3333)
def run_simulation():
    price = np.random.uniform(10, 20)
    cost = np.random.uniform(5, 15)
    demand = max(np.random.normal(10000, 1000), 0)

    revenue = price * demand
    variable_profit = (price - cost) * demand
    total_profit = variable_profit - FIXED_COST

    return revenue, total_profit

def likelihood_of_loss(num_simulations=100_000):
    losses = 0
    for _ in range(num_simulations):
        _, profit = run_simulation()
        if profit < 0:
            losses += 1
    return losses / num_simulations

print("Likelihood of a loss", likelihood_of_loss())
```

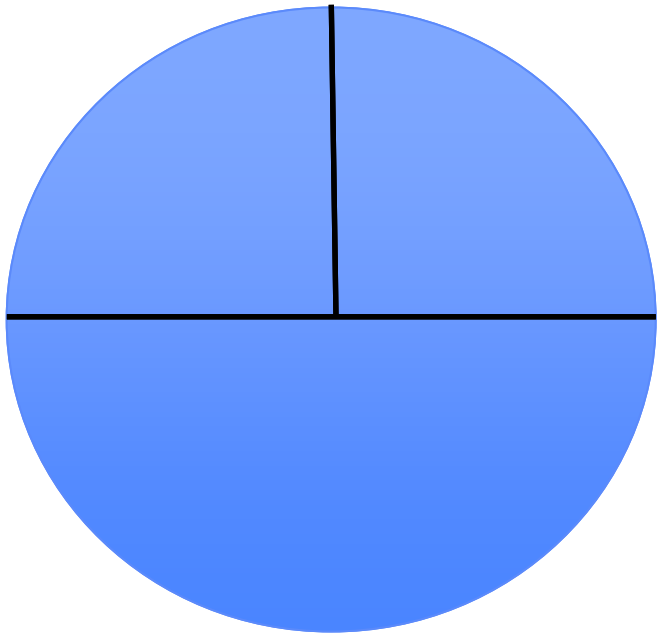
Exploiting Randomness

- **Using randomized computation to model stochastic situations**
- Using randomized computation to solve problems that are not inherently random
- E.g., what's π

Exploiting Randomness

- Using randomized computation to model stochastic situations
- Using randomized computation to solve problems that are not inherently random
- E.g., what's π

π



First called pi in 1706, by William Jones. Euler popularized the name in 1737

$$\frac{circumference}{diameter} = \pi \qquad area = \pi * radius^2$$

Rhind Papyrus (~1550 BCE)

$4*(8/9)^2 = 3.16$

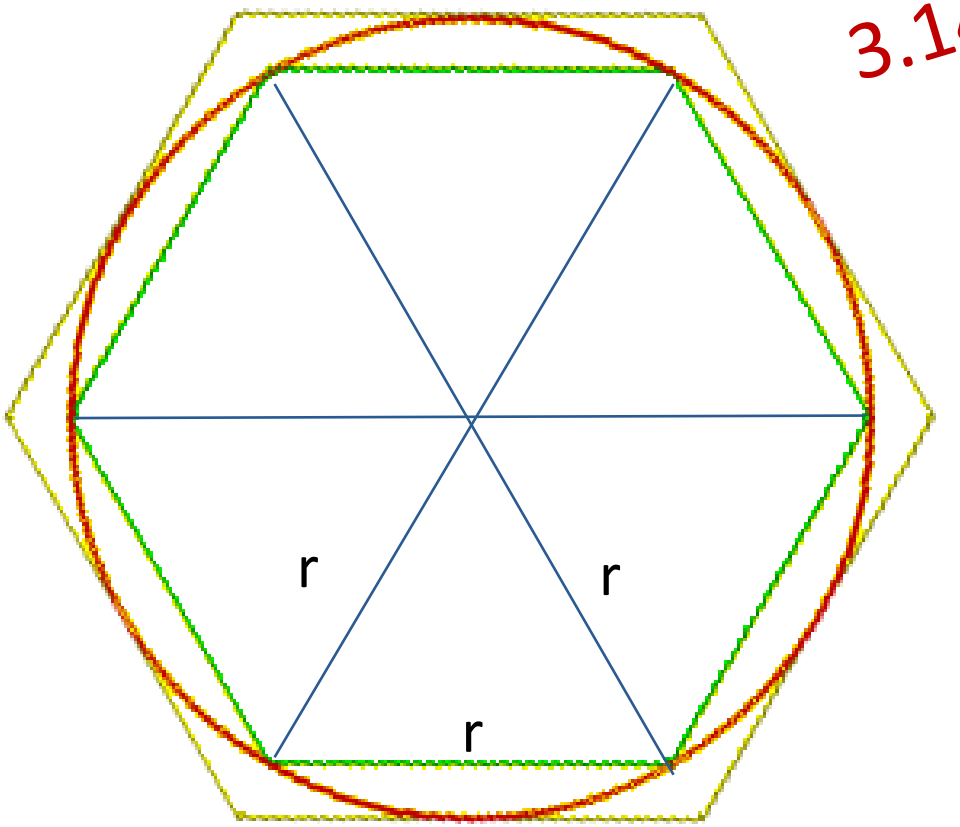
~1100 Years Later

3.0

"And he made a molten sea, ten cubits from the one brim to the other: it was round all about, and his height was five cubits: and a line of thirty cubits did compass it round about."

—1 Kings 7.23

~300 Years Later (Archimedes)



3.14185

Perimeter of interior hexagon is 6r
Circumference of circle is 2πr
So 3 is a lower bound on π

Similarly, the length of the sides of the outer hexagon provides an upper bound on π

Archimedes used a 96-sided polygon

$3 + 10/71 < \pi < 3 + 10/70$

$3.140845070422535 < \pi < 3.142857142857143$

700 Years later

- Zu Chongzhi used polygons with 24,576 sides!

$$3.1415926 < \pi < 3.1415927$$

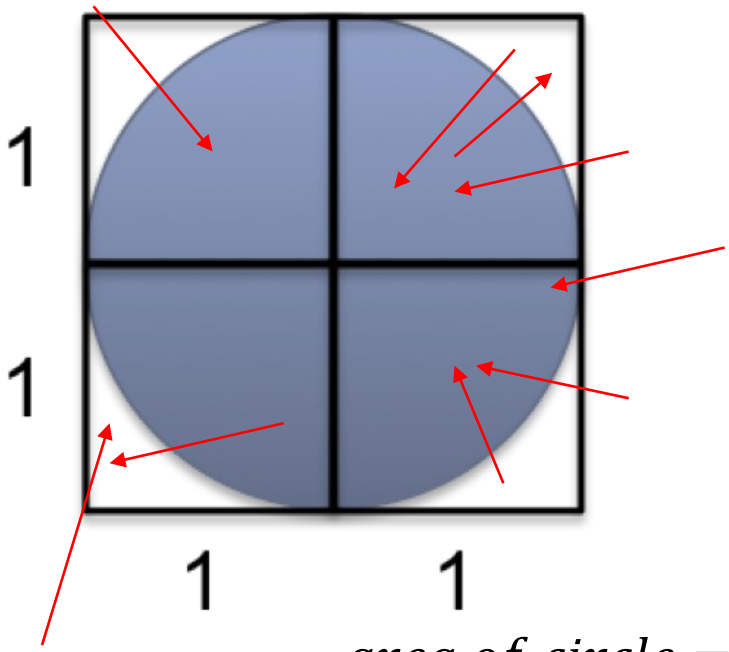
And ~1000 Years Later

- Adriaan Anthonisz (1527-1607) estimated it at 355/113 (roughly 3.141592920353982 5)

~300 Years Later (Buffon-Laplace)

~300 Years Later (Buffon-Laplace)

Drop needles at random



$A_s = 2*2 = 4$

$A_c = \pi r^2 = \pi$

$$\frac{needles\ in\ circle}{needles\ in\ square} = \frac{area\ of\ circle}{area\ of\ square}$$

$$area\ of\ circle = \frac{area\ of\ square * needles\ in\ circle}{needles\ in\ square}$$

$$\boxed{area\ of\ circle} = \frac{4 * needles\ in\ circle}{needles\ in\ square}$$

Simulating Monte-Carlo Method

```
def throw_needles(num_needles):
    in_circle = 0
    for Needles in range(1, num_needles + 1):
        x = random.random()
        y = random.random()
        if (x*x + y*y)**0.5 <= 1:
            in_circle += 1
    #Counting needles in one quadrant only, so multiply by 4
    return 4*(in_circle/num_needles)
```

Random float between 0 and 1

What are the minimum and maximum possible estimates?

Let's try 10, 100, 1000 and 10000 needles

Throwing needles

With 10 needles, estimate for pi: 2.4

With 100 needles, estimate for pi: 3.12

With 1000 needles, estimate for pi: 3.184

With 10000 needles, estimate for pi: 3.162

With 100000 needles, estimate for pi: 3.14352

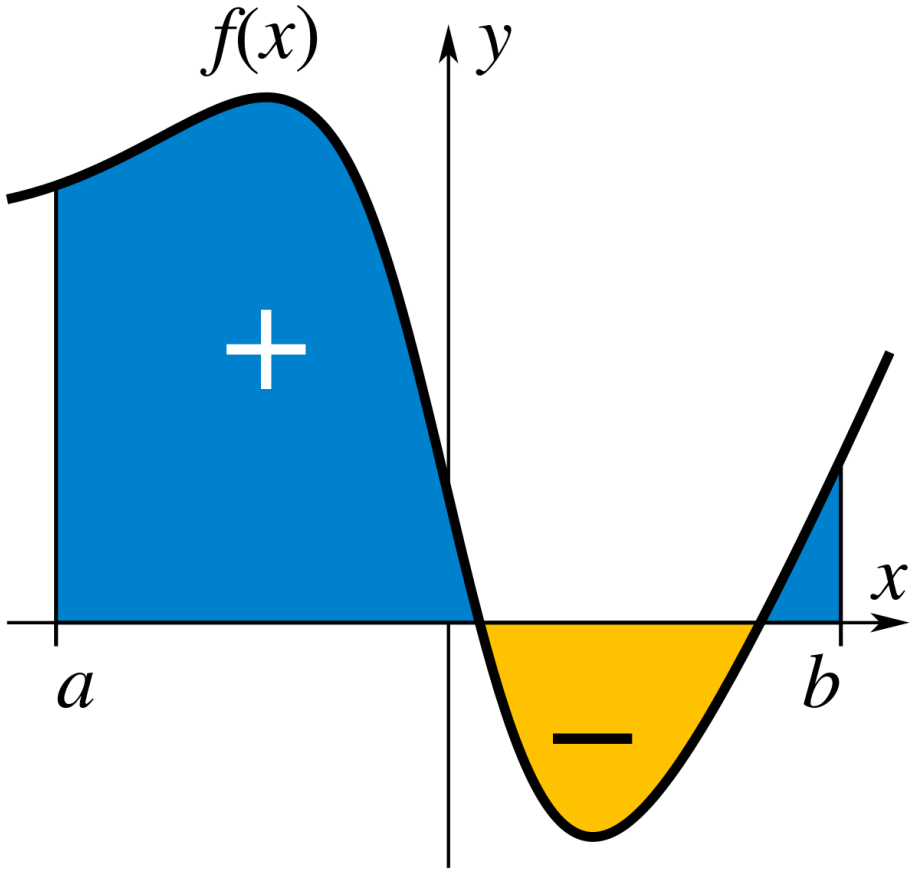
With 1000000 needles, estimate for pi: 3.144664

With 10000000 needles, estimate for pi: 3.1412016

With 100000000 needles, estimate for pi: 3.14208944

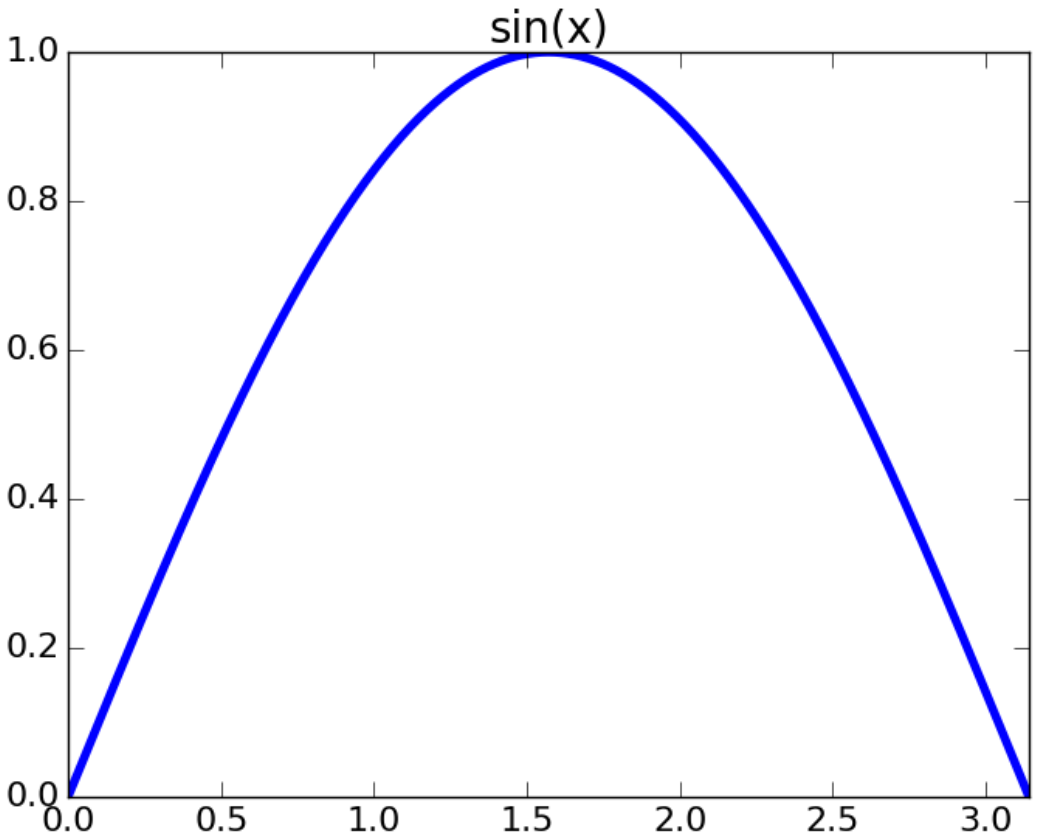
Definite Integrals: Geometric Intrepretation

$$\int_a^b f(x)dx$$



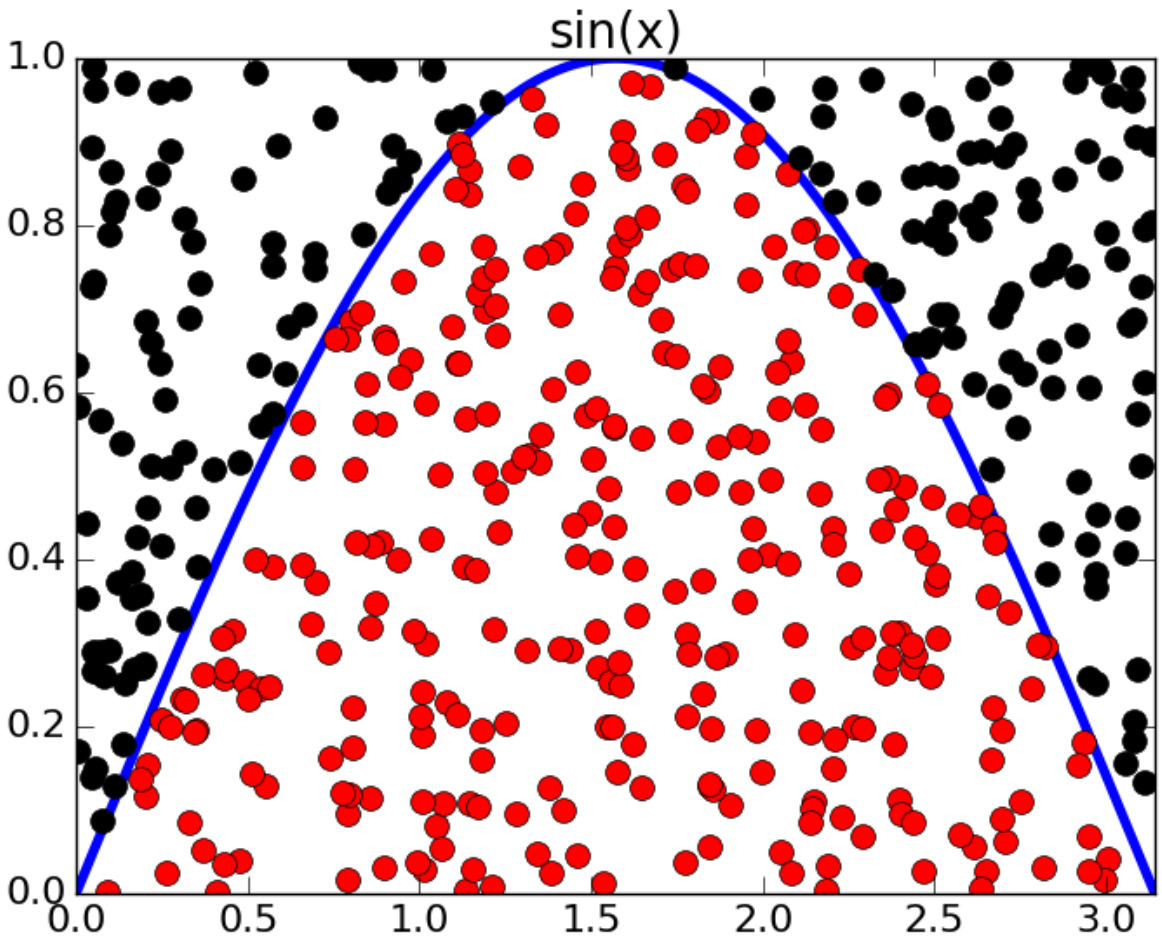
Wikipedia

Sin(x)



Integral of sin(x) from 0 to π is area under the curve

Random Points



Count number of points below curve, compared to total number of points

Integration by Sampling

```
def integrate(fcn, minX, maxX, num_samples = 1_000_000):
    under_curve = 0
    for _ in range(num_samples):
        x = random.uniform(minX, maxX)
        y = fcn(x)
        under_curve += y
    return under_curve / num_samples * (maxX - minX)


def integrate_and_print(fcn, minX, maxX, num_samples = 1_000_000):
    result = integrate(fcn, minX, maxX, num_samples)
    print(f'Integral of {fcn.__name__} from {minX:.0f} to {maxX:2f} =',
          f'{result:2f}')
```

```
integrate_and_print(np.sin, 0, np.pi)
integrate_and_print(np.sin, 0, 2*np.pi)
integrate_and_print(np.cos, 0, np.pi)
```

```
Integral of sin from 0 to 3.141593 = 1.999850
Integral of sin from 0 to 6.283185 = -0.003081
Integral of cos from 0 to 3.141593 = -0.001723
```

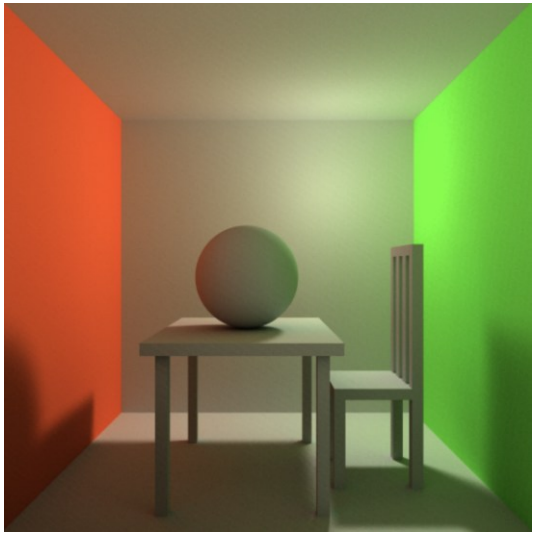
```
integrate_and_print(np.sin, 0, np.pi)
integrate_and_print(np.sin, 0, 2*np.pi)
integrate_and_print(np.cos, 0, np.pi)
```

```
Integral of sin from 0.000000 to 3.141593 = 2.015174
Integral of sin from 0.000000 to 6.283185 = 0.025890
Integral of cos from 0.000000 to 3.141593 = -0.020811
```

Word of caution!! Introducing a Bug

- Suppose I forgot to multiply by 4 in my throw_needles procedure
- Code would still converge as I increased the number of needles
- But just because it converges, doesn't mean the answer is right!
- It just means that I can say with some confidence that on any subsequent trial, the answer will lie within a particular range

Motivation and eye candy from graphics



http://madebyevan.com/webgl-path-tracing/ https://people.csail.mit.edu/tzumao/h2mc/

https://www.youtube.com/watch?v=frLwRLS_ZR0

https://graphics.pixar.com/library/PathTracedMovies/paper.pdf

Summary

- Monte Carlo simulation provides a method for estimating parameters of a model by simulating or sampling a subset of a population
- Need ways to determine confidence in estimate
  - Depends on size of sample
  - Depends on variance of samples
  - Will return to this more formally next lecture
- Buffon-Laplace is one example of a random sampling method