15-110 Principles of Computing – S19

Lecture 25:
Simulation, Random numbers, Monte Carlo - 2

Teacher:
Gianni A. Di Caro

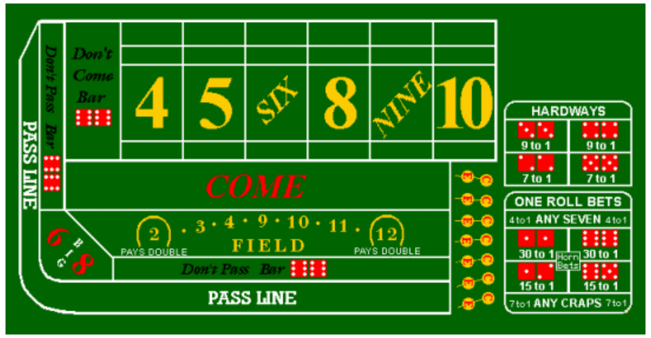
Recap: random module, graphical visualization, numpy

- `seed(seed_val = Null)` initializes the state of the random generator
- `uniform(from, to)` generates random float numbers uniformly in the real range `[from, to]`
- `randint(from, to)` generates random integer numbers uniformly in the range `{from, ..., to}`
- `choice(seq)` generates elements drawn uniformly from the sequence seq

- `matplotlib` + random): let's plot a set of horizontal lines at different heights (y coordinates) and of different colors (using RGB encoding)
- Use of methods from `numpy` (numerical python) module

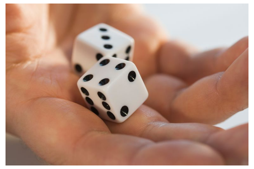
Check the code examples in the notebook!

A more difficult problem, that *needs* MC: craps game



- Pascal's problem was fairly easy (nowadays!) to be solved analytically

➢ The shooter (of the dice) decides to play either to **Pass the line** or **Don't pass the line**



➢ Conditions for the shooter to win when Pass the line:
- If the first roll is a **natural** (7 or 11), while it loses if it is **craps** (2,3,12).
- If some other number is rolled, this number becomes a ***point***
  - If the point number is rolled before number 7, the shooter wins, otherwise loses

➢ Conditions for the shooter to win when Don't Pass the line:
- If the first roll is 2 or 3, while it loses if it 7 or 11, and ties for 12
- If some other number is rolled, this number becomes a ***point***
  - If the point number is rolled before number 7, the shooter wins, otherwise loses

Which play is the best for the shooter? What are the probabilities for winning in either cases?

They seem to be difficult questions: MC simulation can help to figure out the odds!

A more difficult problem, that *needs* MC: craps game

```
def play_craps_game():
    pass_wins = 0
    pass_losses = 0
    dont_pass_wins = 0
    dont_pass_losses = 0
    dont_pass_ties = 0

    throw = roll_die() + roll_die()
    if throw == 7 or throw == 11:
        pass_wins +=1
        dont_pass_losses += 1
    elif throw ==2 or throw == 3 or throw == 12:
        pass_losses += 1
        if throw == 12:
            dont_pass_ties += 1
        else:
            dont_pass_wins += 1
    else:
        point = throw
        while True:
            throw = roll_die() + roll_die()
            if throw == point:
                pass_wins += 1
                dont_pass_losses += 1
                break
            elif throw == 7:
                pass_losses += 1
                dont_pass_wins += 1
                break
    return (pass_wins, pass_losses, dont_pass_wins,
            dont_pass_losses, dont_pass_ties)
```

```
def craps_simulation(num_trials):
    res = [0]*5
    stats = []
    for t in range(num_trials):
        hand = play_craps_game()
        stats += hand
        #stats.append(hand)
    for i in range(5):
        res[i] = sum(stats[i::5])
        print(res[i])
        res[i] /= num_trials
    return res
```

```
res = craps_simulation(100000)

print('Pass, estimated prob. of winning {:4.3f}'.format(res[0]))
print('Pass, estimated prob. of losing {:4.3f}'.format(res[1]))
print("Don't Pass, estimated prob. of winning {:4.3f}".format(res[2]))
print("Dont' Pass, estimated prob. of losing {:4.3f}".format(res[3]))
print("Dont' Pass, estimated prob. of a tie {:4.3f}".format(res[4]))
```

```
Pass, estimated probability of winning 0.489
Pass, estimated probability of losing 0.511
Don't Pass, estimated probability of winning 0.483
Dont' Pass, estimated probability of losing 0.489
Dont' Pass, estimated probability of a tie 0.028
```

Methodological observations on the use of MC simulation

- MC simulation is clearly useful for tackling problems in which **predictive nondeterminism** plays a role
  - **Causal determinism:** cause-and-effect**,** the future can be exactly predicted from the past
  - **Predictive nondeterminism:** the future could in principle be predicted from the past, but we don't have enough information; this means there's uncertainty
    - E.g. we need probabilities to model our ignorance or inherent limits (e.g., Heisenberg's principle), but there's no true randomness in the universe
  - **Causal nondeterminism**: the future cannot be predicted exactly from the past, some things are uncaused, such that there is true randomness in the universe

Methodological observations on the use of MC simulation

- In the previous gambling examples, dice throwing brings an inherent uncertainty into the game, but in principle, *if* we could instantaneously access all the environment and dice information, we could exactly predict each dice outcome

- In practice, we have assumed that there's an *underlying causal determinism,* but we don't have enough information, such that we had to deal with a scenario of *predictive nondeterminism.* Based on these assumptions we have:

  1. Modeled the uncertainty with a probabilistic model of a fair dice

  2. Used the model in our MC simulation

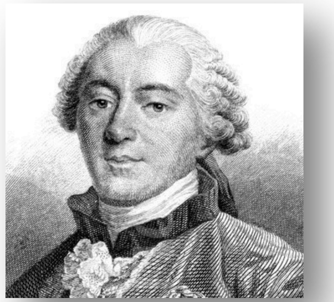
  3. Used the results of the simulation to estimate probabilistic expectations of the game model

✓ This MC approach would work every time we have to deal with a scenario of predictive nondeterminism

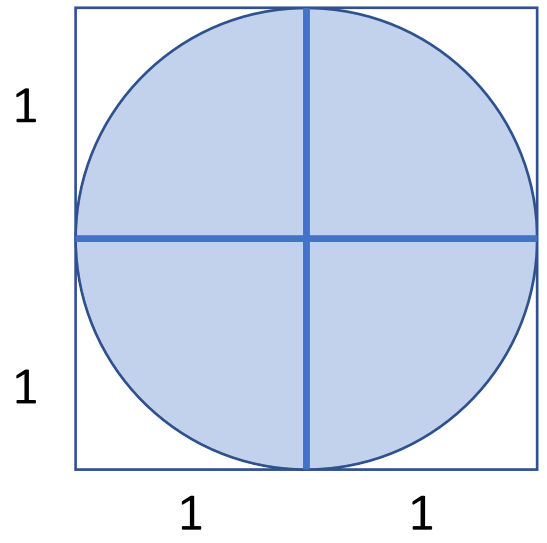
Can we use MC simulation when there's no uncertainty involved in the quantities to compute?

Estimating π: Buffon's needles

- ✓ π is a constant such that for any circle, $2\pi r$ is the *circumference* and πr^2 is the **area** of the circle of radius r
- ✓ We "know" that π is 3.14159265359… https://www.piday.org/million/
- ➢ How those digits are computed? … Earliest estimate dates about 1650 B.C. ….
  - ❖ Long before computers were invented, *Louis Leclerc, Comte de* **Buffon (1707-1788)** proposed a stochastic simulation method, **a Monte Carlo, to estimate the value of π**



- ✓ Buffon used a *randomized algorithm* to estimate a *purely deterministic quantity*!



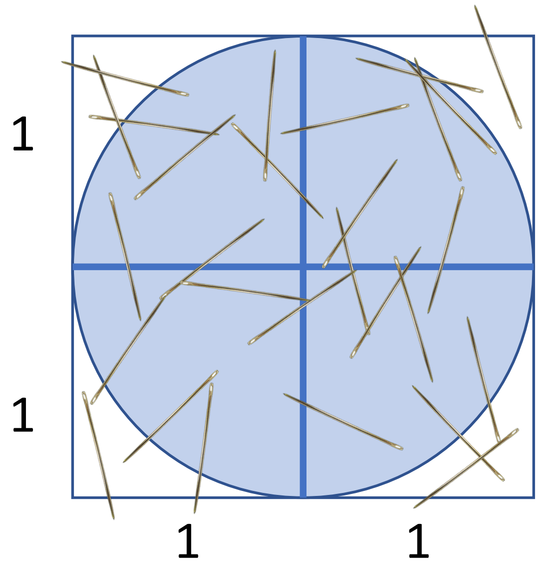
$A = \pi r^2, \quad r = 1 \quad \Longrightarrow A = \pi$ $\quad \Longrightarrow$ Estimating A is equivalent to estimate π

→ *Equivalent problem*: estimating A

Buffon's idea: First, inscribe the circle in a square of side $r = 1$ and estimate the relative occupancy between square and circle areas

- ○ **randomly drop a large number of needles** in the vicinity of the square
- ○ the ratio between the **number of needles with tips lying within the square** to the **number of needles with tips lying within the circle** would provide an estimate of the ratio between the areas of the square and that of the circle

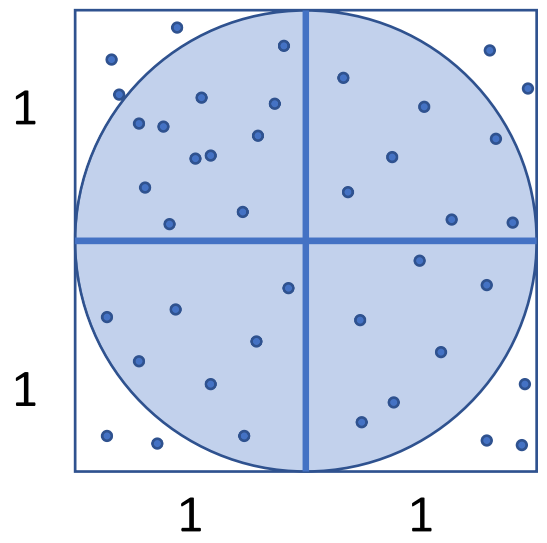
Estimating π: Buffon’s needles



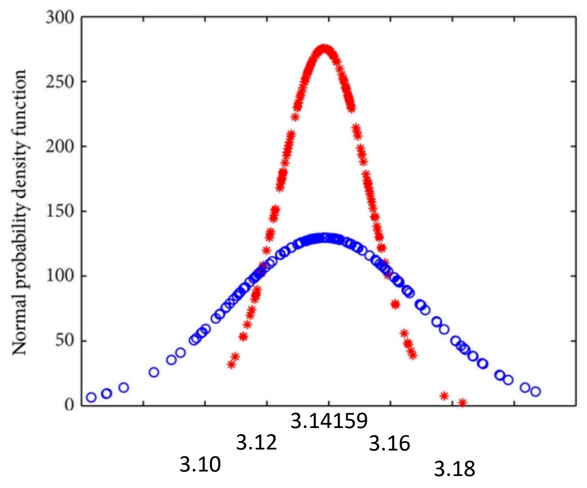
$$\frac{needles\ in\ circle}{needles\ in\ square} \approx \frac{area\ of\ circle}{area\ of\ square}$$

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

- We can just generate **points**, <u>randomly and uniformly distributed within the square</u>
- Count the number of points that fall within the circle vs. the number of points that fall within the square
- $\pi \approx 4 \times \frac{\#\ points\ in\ circle}{\#\ points\ in\ square}$



Estimating π: Buffon’s Monte Carlo



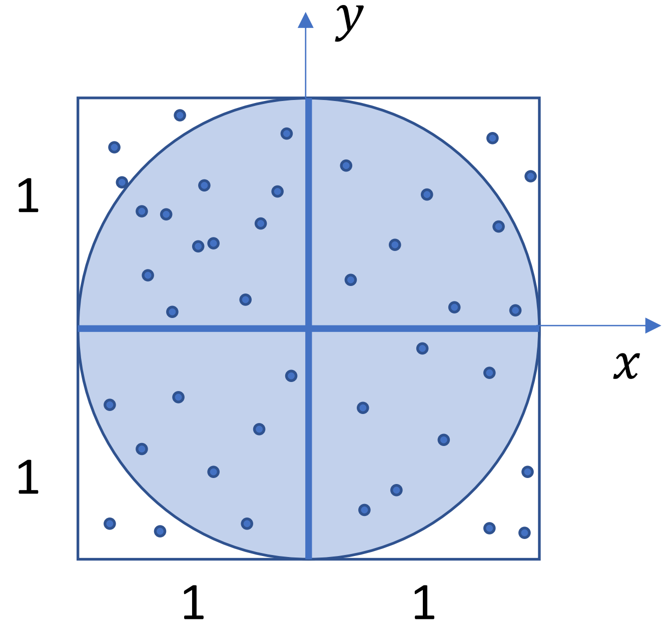
Standard deviation σ_n of a set $\{v_1, v_2, \cdots, v_n\}$, of n measurements, affected by uncertainty:

- $\bar{v} = \frac{1}{n}\sum_{i=1}^{n} v_i$ (mean)
- $\sigma_n = \sqrt{\frac{\sum_{i=1}^{n}(v_i - \bar{v})^2}{n-1}}$

➢ Experimental methodology:

- **Goal**: estimate the value of π with a given precision ε (error) using MC simulation for the Buffon’s model
- **Experiment design:**
  - T experiment *trials*
  - Each trial is a MC simulated Buffon’s measure, that returns a numeric estimate for the value of π based on n sampling points
  - π’s estimate for the experiment is computed as the arithmetic average (the *mean*) of the estimates from the single trials
  - At the end of an experiment, check whether the desired *precision* ε for estimating π has been achieved. If not, increase the number of sampling points n and perform a new experiment
  - Measure the precision as the standard deviation of the mean estimate of π over the experiment’s trials

Estimating π: Buffon's Monte Carlo



```
def pi_estimate_by_monte_carlo_trial(num_points):
    in_circle = 0
    for p in range(num_points):
        x = random.uniform(-1,1)
        y = random.uniform(-1,1)
        if math.sqrt(x*x + y*y) <= 1:
            in_circle += 1
    return 4 * (in_circle / num_points)

def pi_estimate(num_trials, num_points):
    estimates = []
    for t in range(num_trials):
        pi_estimate = get_pi_est_by_monte_carlo(num_points)
        estimates.append(pi_estimate)
    mean_estimates = sum(estimates) / num_trials
    stddev_estimates = std_dev(estimates)
    return mean_estimates, stddev_estimates
```

Estimating π: Buffon's Monte Carlo

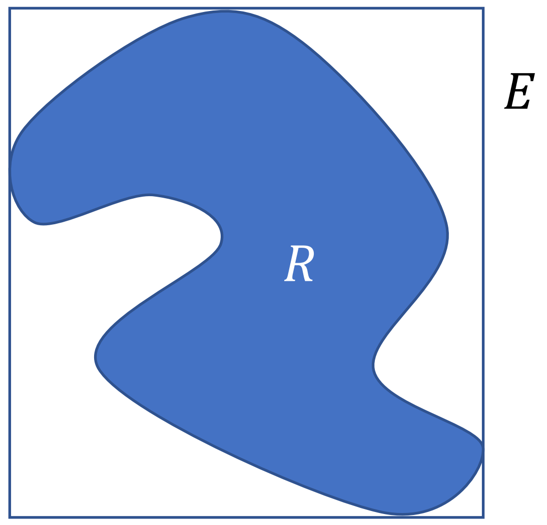
```
def std_dev(values):
    mean = sum(values) / len(values)
    std = 0.0
    for v in values:
        std += (v - mean) ** 2
    return math.sqrt(std / (len(values)-1))

def pi_estimate_with_given_precision(precision, num_trials = 50, num_points = 100):
    stddev_estimate = precision
    while stddev_estimate  >= precision:
        estimate, stddev_estimate = pi_estimate(num_trials, num_points)
        num_points *= 2
    print('Exact pi value: 3.141592653')
    print('Estimate of pi: {:10.8f} (estimated error: {:3.4f}, #points: {})'.format(pi_estimate,
                                                                                    stddev_estimate,
                                                                                    num_points))
pi_estimate_with_given_precision(0.01, 40)
```

```
Exact pi value: 3.141592653
Estimate of pi: 3.14283984 (estimated error: 0.0073, #points: 102400)
```

Estimating areas with Monte Carlo simulation

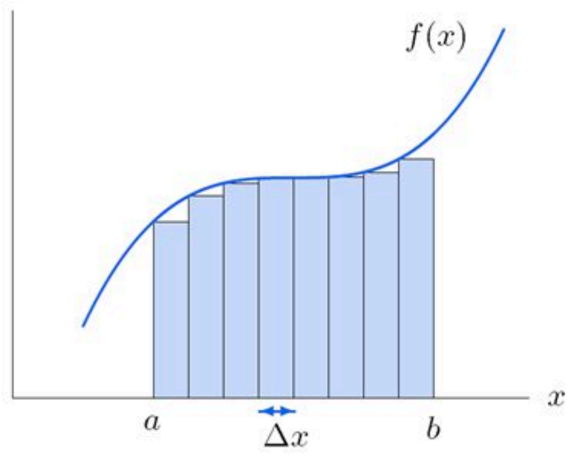
✓ General Monte Carlo procedure to estimate the area R of some region:

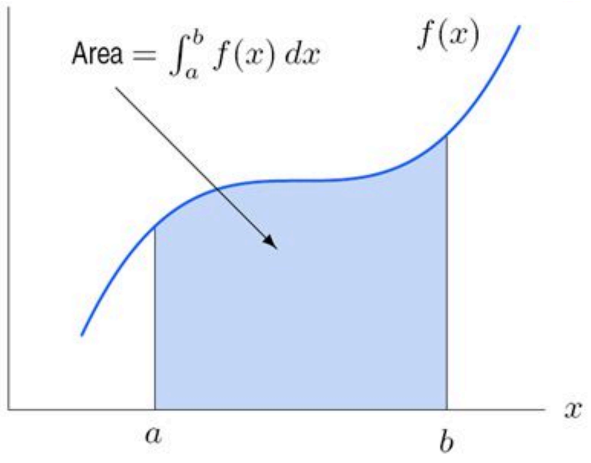
1. Select a region E whose area is easy to calculate and that totally encloses the region R
2. The smaller the difference between the two areas the best the approximation will be
3. Generate a set of N random points within the region E
4. Compute the fraction F of points that fall within R
5. $R = F \times E$

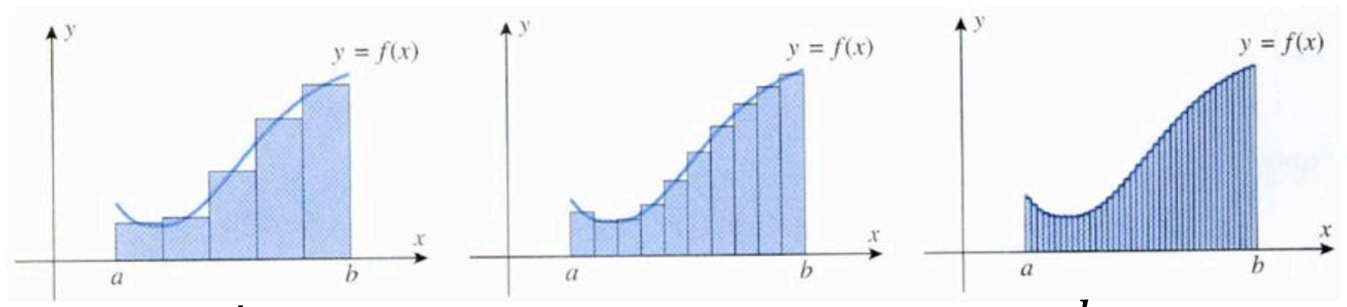


Example: Estimating integrals (areas) with Monte Carlo simulation

Total <u>area</u> under / below a function $f(x) = \int_a^b f(x)dx$





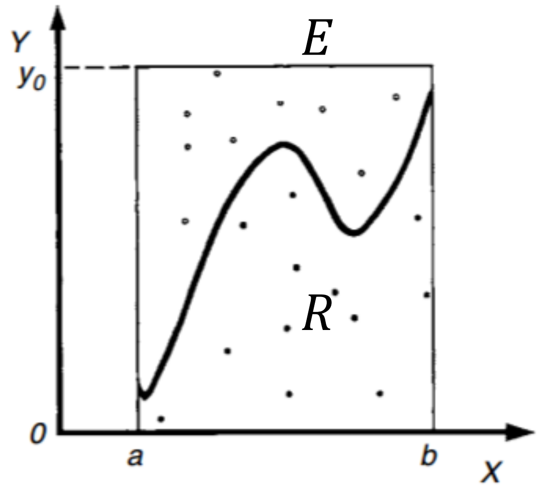


$$\Delta x \longrightarrow dx$$

Estimating integrals (areas) with Monte Carlo simulation

- **Hit-or-miss** (acceptance-rejection) Monte Carlo integration of a function $f(x)$

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

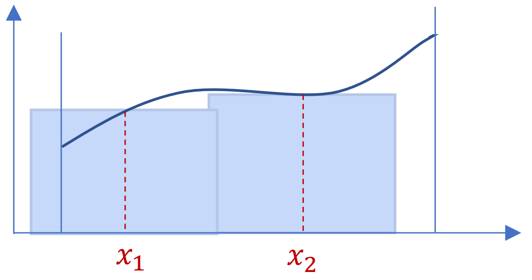


- Draw N random points from a uniform distribution in $[a,b]\times[0,y_0]$
- Count the number n of points falling below $f(x)$ to estimate the area below the function, $F = \left(\frac{n}{N}\right)$
- $\hat{R} = F \times E = \left(\frac{n}{N}\right) \times [y_0(b-a)]$

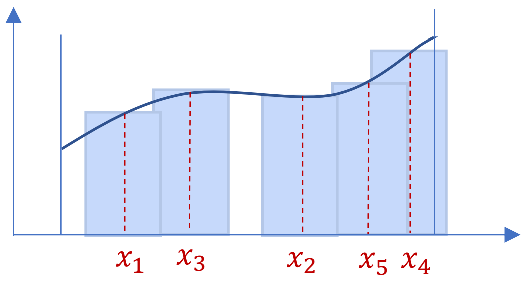
- **Crude / Simple Monte Carlo**: generate N random x points from a uniform distribution in $[a,b]$ and compute:

$$\hat{R} = \frac{[b-a]}{N}\sum_i f(x_i) \qquad \hat{R} \to R \quad \text{as } N \to \infty$$

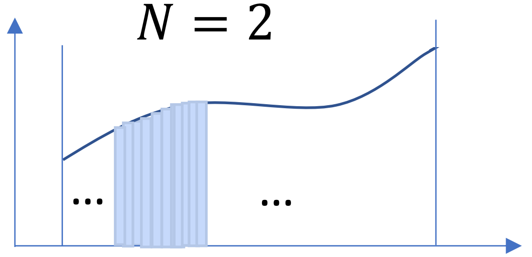
- How good is the estimate? $\varepsilon_N = \|R - \hat{R}\|$
  - $\varepsilon_N \approx [b-a]\frac{\sigma_N}{\sqrt{N}} \qquad \sigma_N = \sqrt{\frac{1}{N-1}\sum_{i=1}^N\left(f(x_i) - \hat{R}\right)^2}$



$N = 2$



$N = 5$



$N \to \infty$