Lecture 6: Monte Carlo Simulation

(download slides and .py files from Stellar to follow along)

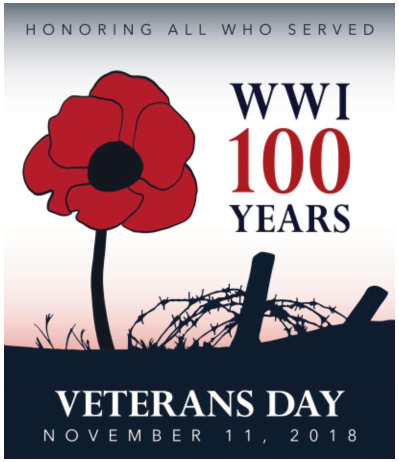
John Guttag

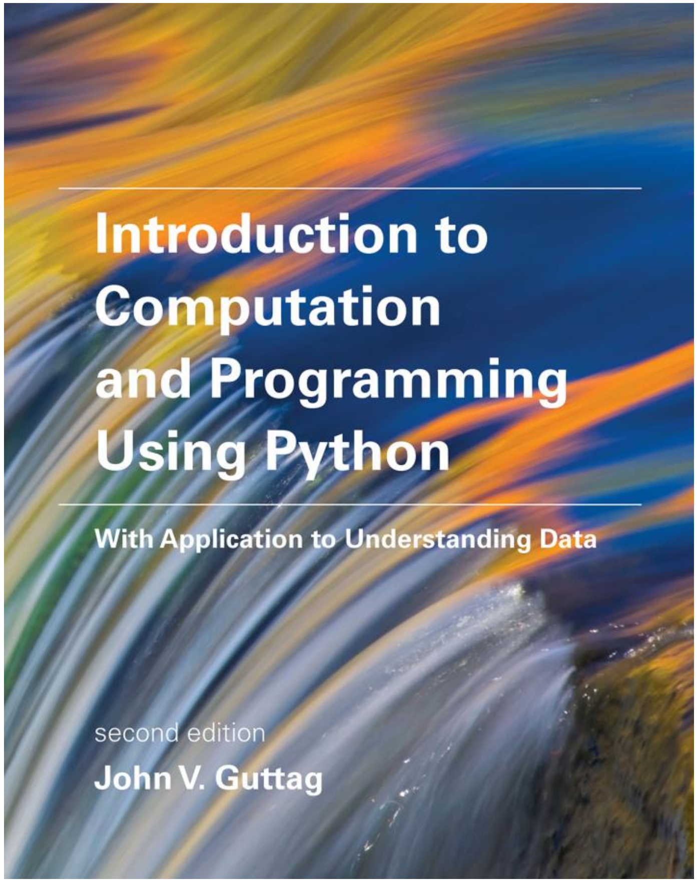
MIT Department of Electrical Engineering and Computer Science

Relevant Reading

- Today
  - Chapter 16
- Next week
  - Sections 15.3-15.4

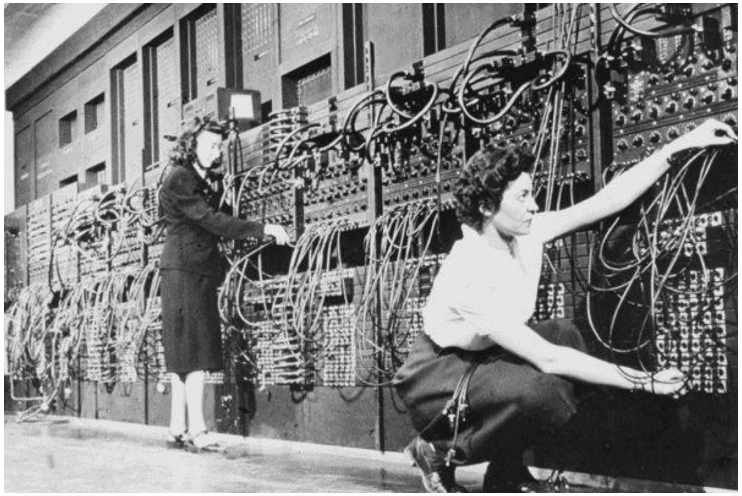
No class or office hours on Monday





A Little History

- Stanilaw Ulam, recovering from an illness, was playing a lot of solitaire
- Tried to figure out probability of winning, and failed
- Thought about playing lots of hands and counting number of wins, but decided it would take years
- Asked Von Neumann if he could build a program to simulate many hands on ENIAC

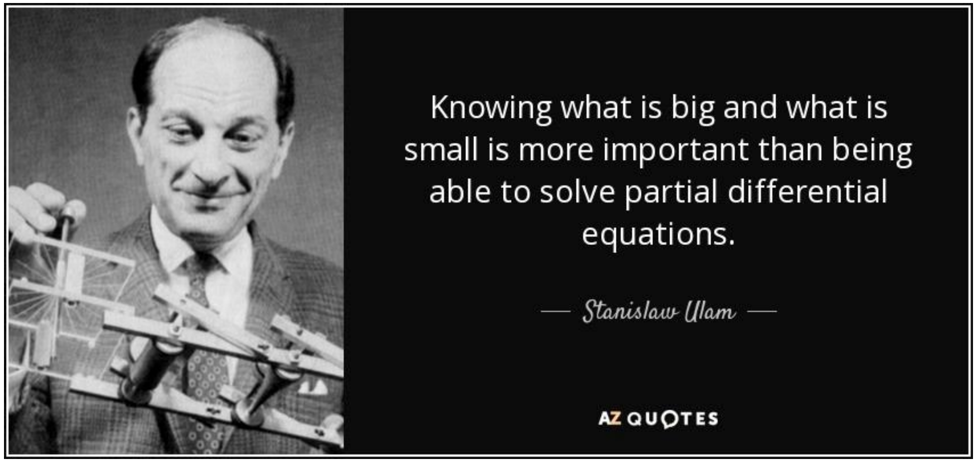


Who Was Stanislaw Ulam?

- Polish-American mathematician, many significant contributions to mathematics and physics
- Ulam's Conjecture:

$$f(n) = \begin{cases} \dfrac{n}{2} & if\ n\ even \\ 3 * n + 1 & if\ n\ odd \end{cases}$$

$$\forall n > 0\ \exists i\ f^{i}(\mathrm{n}) = 1$$



“Checking” Ulam’s Conjecture

```python
def UlamConjecture(n, toPrint = False):
    """assumes n a positive int"""
    result = [n]
    while n != 1:
        if n%2 == 0:
            n = n//2
        else:
            n = 3*n + 1
        if toPrint:
            result.append(n)
    if toPrint:
        print(result)
        print('Maximum value =', max(result))
```

```python
import random, sys
for i in range(10000):
    UlamConjecture(random.randint(1, sys.maxsize))
print('So far, it seems to be true')
```

Monte Carlo Simulation

- A method of estimating the value of an unknown quantity using the principles of inferential statistics
- Inferential statistics
  - *Population*: a set of examples
  - *Sample*: a proper subset of a population
  - Key fact: a *random sample* tends to exhibit the same properties as the population from which it is drawn
- Exactly what we did with random walks

An Example

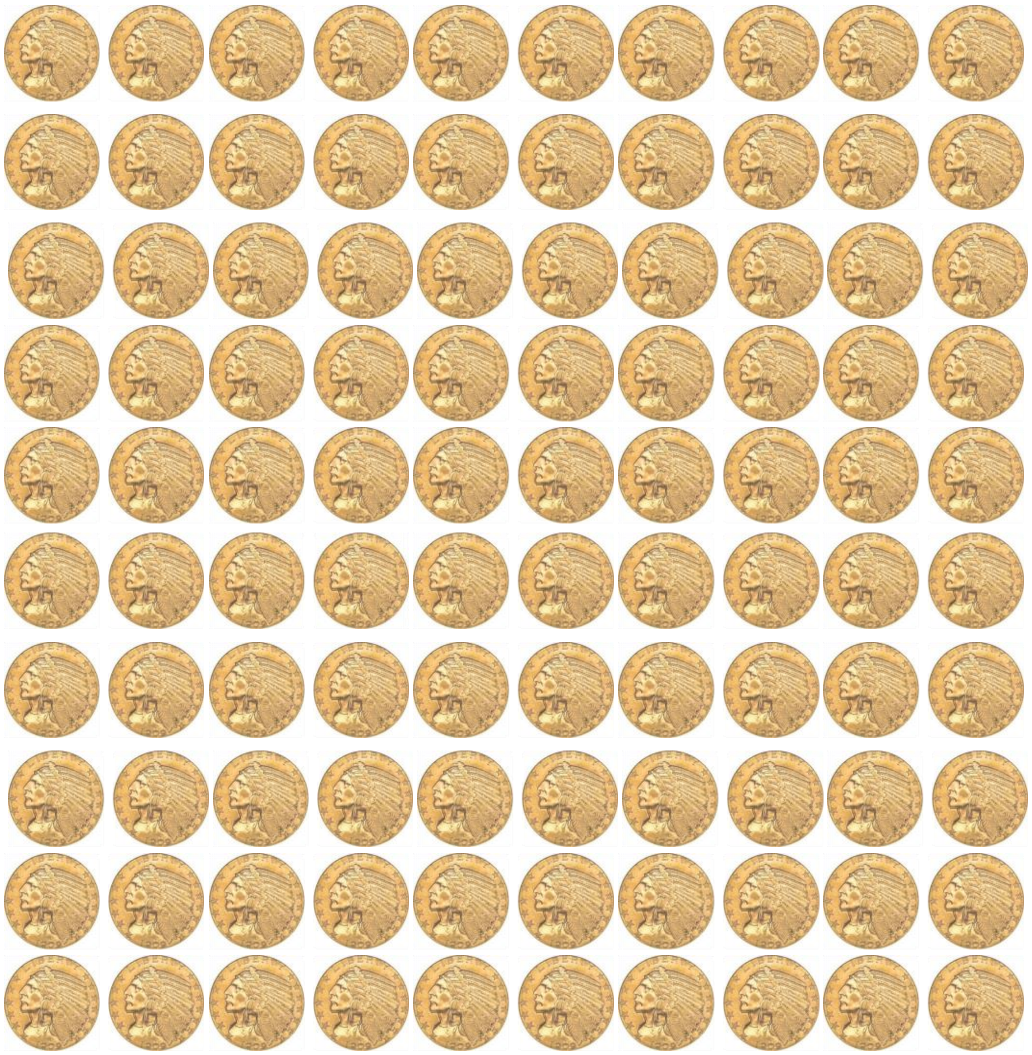
- Given a single coin, estimate fraction of heads you would get if you flipped the coin an infinite number of times
- Consider one flip

How confident would you be about answering 1.0?

Flipping a Coin Twice

Do you think that the next flip will come up heads?

Flipping a Coin 100 Times



Now do you think that the next flip will come up heads?

Flipping a Coin 100 Times

Do you think that the probability of the next flip coming up heads is 52/100?

Given the data, it's your best estimate

But confidence should be low

Why the Difference in Confidence?

- Confidence in our estimate depends upon two things
- Size of sample (e.g., 100 versus 2)
- Variance of sample (e.g., all heads versus 52 heads)
- As the variance grows, we need larger samples to have the same degree of confidence

Roulette

No need to simulate, since answers obvious

Allows us to compare simulation results to actual probabilities

Class Definition

```
class FairRoulette():
    def __init__(self):
        self.pockets = []
        for i in range(1,37):
            self.pockets.append(i)
        self.ball = None
        self.pocketOdds = len(self.pockets) - 1
    def spin(self):
        self.ball = random.choice(self.pockets)
    def betPocket(self, pocket, amt):
        if str(pocket) == str(self.ball):
            return amt*self.pocketOdds
        else: return -amt
    def __str__(self):
        return 'Fair Roulette'
```

Monte Carlo Simulation

```
def playRoulette(game, numSpins, pocket, bet, toPrint):
    totPocket = 0
    for i in range(numSpins):
        game.spin()
        totPocket += game.betPocket(pocket, bet)
    if toPrint:
        print(numSpins, 'spins of', game)
        print('Expected return betting', pocket, '=',\
              str(100*totPocket/numSpins) + '%\n')
    return (totPocket/numSpins)

game = FairRoulette()
for numSpins in (100, 1000000):
    for i in range(3):
        playRoulette(game, numSpins, 2, 1, True)
```

100 and 1M Spins of the Wheel

```
100 spins of Fair Roulette
Expected return betting 2 = 152.0%

100 spins of Fair Roulette
Expected return betting 2 = 116.0%

100 spins of Fair Roulette
Expected return betting 2 = -28.0%

1000000 spins of Fair Roulette
Expected return betting 2 = 0.2312%

1000000 spins of Fair Roulette
Expected return betting 2 = 0.6056%

1000000 spins of Fair Roulette
Expected return betting 2 = -0.4672%
```

Law of Large Numbers

- In repeated independent tests with the same actual probability *p* of a particular outcome in each test, the chance that the fraction of times that outcome occurs differs from *p* converges to zero as the number of trials goes to infinity

Does this imply that if deviations from expected behavior occur, these deviations are likely to be ***evened out*** by opposite deviations in the future?

Gambler's Fallacy

- "On August 18, 1913, at the casino in Monte Carlo, black came up a record twenty-six times in succession [in roulette]. … [There] was a near-panicky rush to bet on red, beginning about the time black had come up a phenomenal fifteen times." -- Huff and Geis, *How to Take a Chance*
- Probability of 26 consecutive reds
- 1/67,108,865
- Probability of 26 consecutive reds when previous 25 rolls were red
- 1/2

Regression to the Mean

TABLE I.

NUMBER OF ADULT CHILDREN OF VARIOUS STATURES BORN OF 205 MID-PARENTS OF VARIOUS STATURES.
(All Female heights have been multiplied by 1·08).

Heights of the Mid-parents in inches.	Heights of the Adult Children.														Total Number of		Medians.
	Below	62·2	63·2	64·2	65·2	66·2	67·2	68·2	69·2	70·2	71·2	72·2	73·2	Above	Adult Children.	Mid-parents.	
Above ..	..	..	..	..	..	..	..	..	..	..	..	1	3	..	4	5	..
72·5	..	..	..	..	..	..	..	1	2	1	2	7	2	4	19	6	72·2
71·5	..	..	..	..	1	3	4	3	5	10	4	9	2	2	43	11	69·9
70·5	1	..	1	..	1	1	3	12	18	14	7	4	3	3	68	22	69·5
69·5	..	..	1	16	4	17	27	20	33	25	20	11	4	5	183	41	68·9
68·5	1	..	7	11	16	25	31	34	48	21	18	4	3	..	219	49	68·2
67·5	..	3	5	14	15	36	38	28	38	19	11	4	..	..	211	33	67·6
66·5	..	3	3	5	2	17	17	14	13	4	..	..	..	..	78	20	67·2
65·5	1	..	9	5	7	11	11	7	7	5	2	1	..	..	66	12	66·7
64·5	1	1	4	4	1	5	5	..	2	..	..	..	..	..	23	5	65·8
Below ..	1	..	2	4	1	2	2	1	1	..	..	..	..	..	14	1	..
Totals ..	5	7	32	59	48	117	138	120	167	99	64	41	17	14	928	205	..
Medians ..	..	..	66·3	67·8	67·9	67·7	67·9	68·3	68·5	69·0	69·0	70·0	..	..	..	..	..

NOTE.—In calculating the Medians, the entries have been taken as referring to the middle of the squares in which they stand. The reason why the headings run 62·2, 63·2, &c., instead of 62·5, 63·5, &c., is that the observations are unequally distributed between 62 and 63, 63 and 64, &c., there being a strong bias in favour of integral inches. After careful consideration, I concluded that the headings, as adopted, best satisfied the conditions. This inequality was not apparent in the case of the Mid-parents.

Francis Galton, 1885

Regression to the Mean

- Following an extreme random event, the next random event is likely to be less extreme
- If you spin a fair roulette wheel 10 times and get 100% reds, that is an extreme event (probability = 1/1024)
- It is likely that in the next 10 spins, you will get fewer than 10 reds
  - But the expected number is not less than 5
- So, if you look at the average of the 20 spins, it will be closer to the expected mean of 50% reds than to the 100% of the first 10 spins

Casinos Not in the Business of Being Fair

Two Subclasses of Roulette

```
class EuRoulette(FairRoulette):
    def __init__(self):
        FairRoulette.__init__(self)
        self.pockets.append('0')
    def __str__(self):
        return 'European Roulette'

class AmRoulette(EuRoulette):
    def __init__(self):
        EuRoulette.__init__(self)
        self.pockets.append('00')
    def __str__(self):
        return 'American Roulette'
```

Comparing the Games

```
Simulate 20 trials of 1000 spins each
Exp. return for Fair Roulette = 6.56%
Exp. return for European Roulette = -2.26%
Exp. return for American Roulette = -8.92%

Simulate 20 trials of 10000 spins each
Exp. return for Fair Roulette = -1.234%
Exp. return for European Roulette = -4.168%
Exp. return for American Roulette = -5.752%

Simulate 20 trials of 100000 spins each
Exp. return for Fair Roulette = 0.8144%
Exp. return for European Roulette = -2.6506%
Exp. return for American Roulette = -5.113%

Simulate 20 trials of 1000000 spins each
Exp. return for Fair Roulette = -0.0723%
Exp. return for European Roulette = -2.7329%
Exp. return for American Roulette = -5.212%
```

Sampling Space of Possible Outcomes

- Never possible to guarantee perfect accuracy through sampling
- Not to say that an estimate is not precisely correct
- Key question:
  - How many samples do we need to look at before we can have justified confidence on our answer?
- Depends upon variability in underlying distribution

Quantifying Variation in Data

$$variance(X) = \frac{\sum_{x \epsilon X}(x-\mu)^2}{|X|}$$

$$\sigma(X) = \sqrt{\frac{1}{|X|}\sum_{x \in X}(x-\mu)^2}$$

- Standard deviation simply the square root of the variance
- Outliers can have a big effect
- Standard deviation should always be considered relative to mean

For Those Who Prefer Code

```
def getMeanAndStd(X):
    mean = sum(X)/len(X)
    tot = 0.0
    for x in X:
        tot += (x - mean)**2
    std = (tot/len(X))**0.5
    return mean, std
```

Confidence Levels and Intervals

- Instead of estimating an unknown parameter by a single value (e.g., the mean of a set of trials), a confidence interval provides a range that is likely to contain the unknown value and a confidence that the unknown value lays within that range
- "The return on betting a pocket 10k times in European roulette is -3.3%. The margin of error is +/- 3.5% with a 95% level of confidence."
- What does this mean?
- If I were to conduct an infinite number of trials of 10k bets each,
  - My expected average return would be -3.3%
  - My return would be between roughly -6.8% and +0.2% 95% of the time

Ripped from the Headlines

- “52% of Utahns approve of President Donald Trump…The latest Tribune-Hinckley Institute poll conducted among 605 registered voters … has a margin of error of plus or minus 3.98 percentage points.”

What is a Utahn?

What does this mean?

Empirical Rule

- Under some assumptions discussed later
  - ~68% of data within one standard deviation of mean
  - ~95% of data within 1.96 standard deviations of mean
  - ~99.7% of data within 3 standard deviations of mean

Applying Empirical Rule

```
resultDict = {}
games = (FairRoulette, EuRoulette, AmRoulette)
for G in games:
    resultDict[G().__str__()] = []
numTrials = 20
for numSpins in (100, 1000, 10000):
    print('\nSimulate betting a pocket for', numTrials,
          'trials of', numSpins, 'spins each')
    for G in games:
        pocketReturns = findPocketReturn(G(), numTrials,
                                         numSpins, False)
        mean, std = getMeanAndStd(pocketReturns)
        resultDict[G().__str__()].append((numSpins,
                                          100*mean,
                                          100*std))
        print('Exp. return for', G(), '=',
              str(round(100*mean, 3))
              + '%,', '+/- ' + str(round(100*1.96*std, 3))
              + '% with 95% confidence')
```

Results

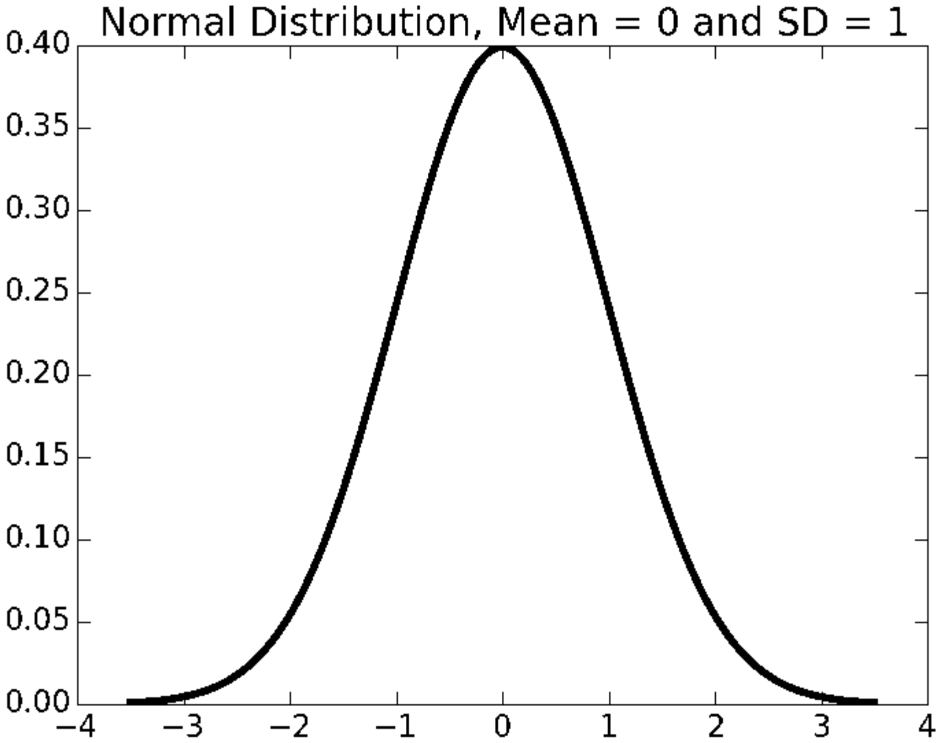
Simulate betting a pocket for 20 trials of 1000 spins each
Exp. return for Fair Roulette = 3.68%, +/- 27.189% with 95% confidence
Exp. return for European Roulette = -5.5%, +/- 35.042% with 95% confidence
Exp. return for American Roulette = -4.24%, +/- 26.494% with 95% confidence

Simulate betting a pocket for 20 trials of 100000 spins each
Exp. return for Fair Roulette = 0.125%, +/- 3.999% with 95% confidence
Exp. return for European Roulette = -3.313%, +/- 3.515% with 95% confidence
Exp. return for American Roulette = -5.594%, +/- 4.287% with 95% confidence

Simulate betting a pocket for 20 trials of 1000000 spins each
Exp. return for Fair Roulette = 0.012%, +/- 0.846% with 95% confidence
Exp. return for European Roulette = -2.679%, +/- 0.948% with 95% confidence
Exp. return for American Roulette = -5.176%, +/- 1.214% with 95% confidence

Assumptions Underlying Empirical Rule

- The mean estimation error is zero
- The distribution of the errors in the estimates is normal

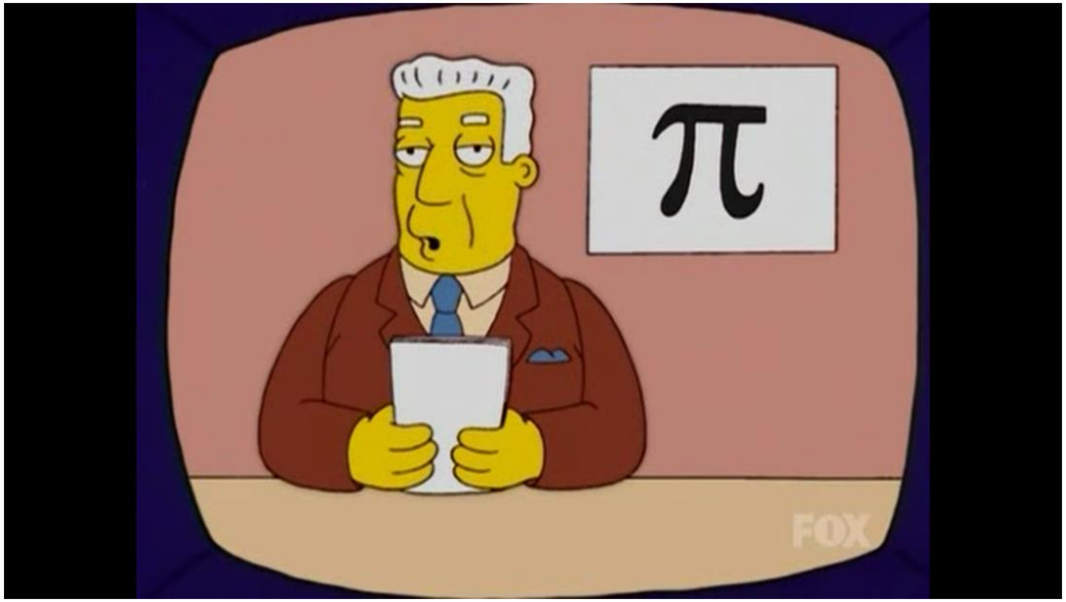


More about distributions next week

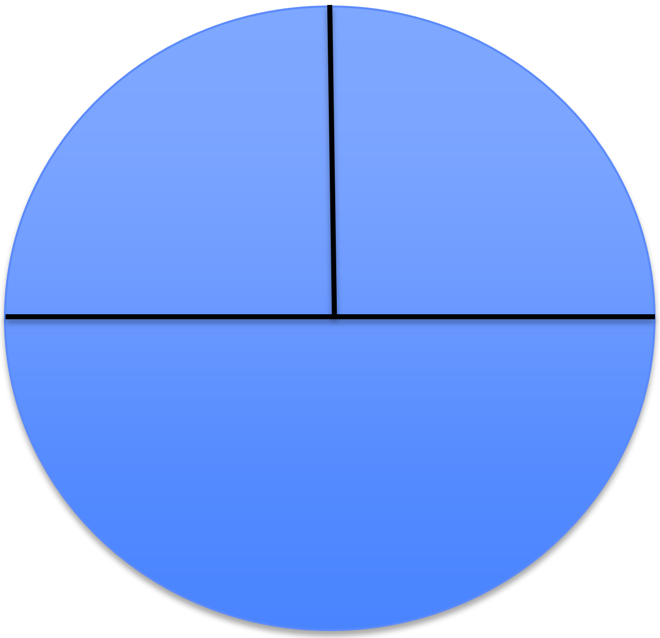
5 Minute Break

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



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

Rhind Papyrus (~1550 BCE)

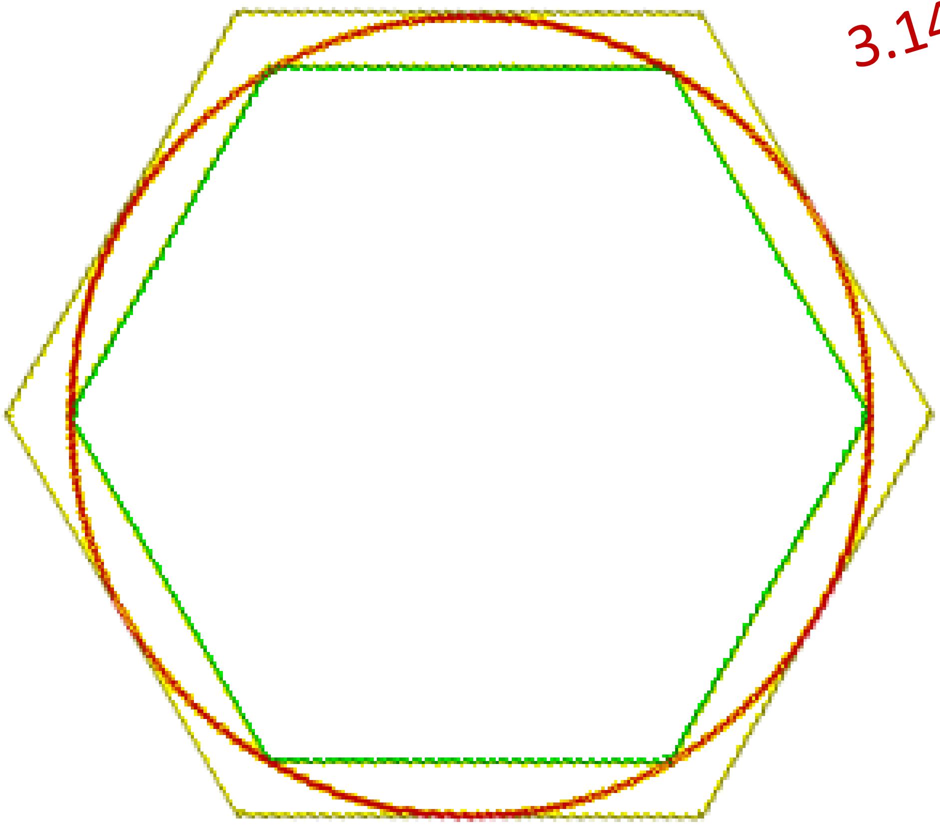
$4*(8/9)^2 = 3.16$

~1100 Years Later

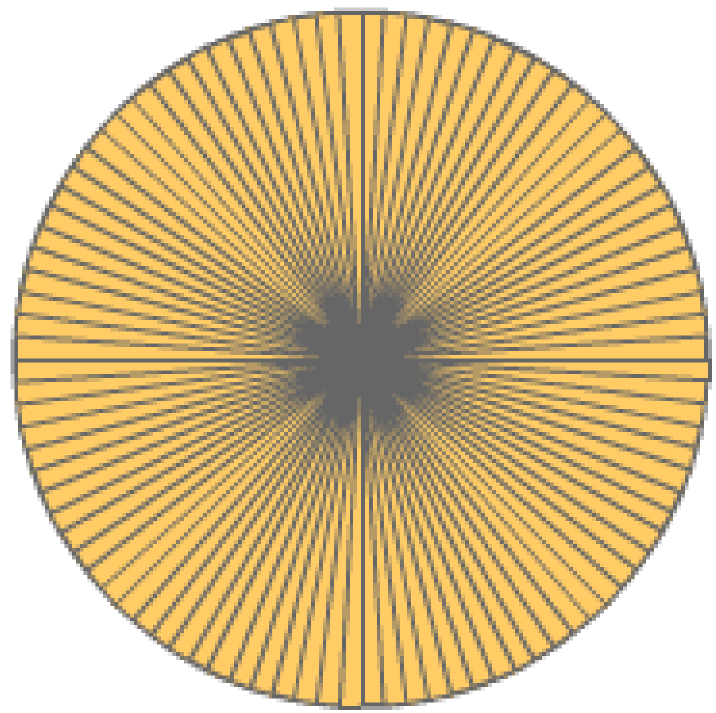
"And he made a molten sea, ten cubits from the one brim to the other: it was round all about, and his height was five cubits: and a line of thirty cubits did compass it round about."

—1 Kings 7.23

~300 Years Later (Archimedes)

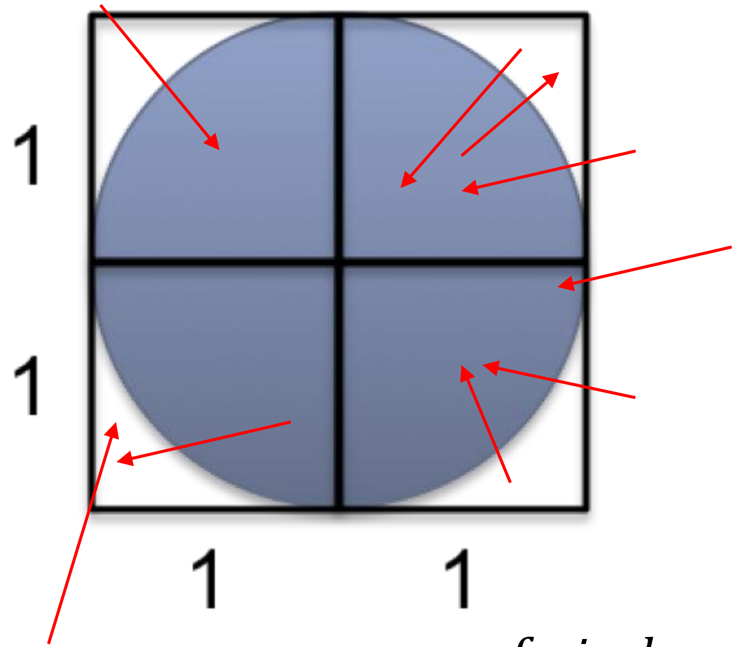


3.14185



223/71 < pi < 22/7

~2000 Years Later (Buffon-Laplace)



$A_s = 2*2 = 4$

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

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

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

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

~200 Years Later

https://www.youtube.com/watch?v=oYM6MIjZ8IY

Simulating Buffon-Laplace Method

```
def throwNeedles(numNeedles):
    inCircle = 0
    for Needles in range(1, numNeedles + 1, 1):
        x = random.random()
        y = random.random()
        if (x*x + y*y)**0.5 <= 1.0:
            inCircle += 1
    return 4*(inCircle/float(numNeedles))
```

Let’s try 10 needles

Simulating Buffon-Laplace Method, cont.

```
def getEst(numNeedles, numTrials):
    estimates = []
    for t in range(numTrials):
        piGuess = throwNeedles(numNeedles)
        estimates.append(piGuess)
    sDev = numpy.std(estimates)
    curEst = sum(estimates)/len(estimates)
    print('Est. = ' + str(curEst) +\
          ', Std. dev. = ' + str(round(sDev, 6))\
          + ', Needles = ' + str(numNeedles))
    return (curEst, sDev)
```

Simulating Buffon-Laplace Method, cont.

```
def estPi(precision, numTrials):
    numNeedles = 1000
    sDev = precision
    while sDev >= precision/1.96:
        curEst, sDev = getEst(numNeedles,
                                           numTrials)
        numNeedles *= 2
    return curEst
```

Output

```
Est. = 3.1484400000000012, Std. dev. = 0.047886, Needles = 1000
Est. = 3.1391799999999987, Std. dev. = 0.035495, Needles = 2000
Est. = 3.1410799999999997, Std. dev. = 0.02713, Needles = 4000
Est. = 3.141435, Std. dev. = 0.016805, Needles = 8000
Est. = 3.141355, Std. dev. = 0.0137, Needles = 16000
Est. = 3.1413137500000006, Std. dev. = 0.008476, Needles = 32000
Est. = 3.141171874999999, Std. dev. = 0.007028, Needles = 64000
Est. = 3.1415896874999993, Std. dev. = 0.004035, Needles = 128000
Est. = 3.141741406249995, Std. dev. = 0.003536, Needles = 256000
Est. = 3.14155671875, Std. dev. = 0.002101, Needles = 512000
```

Is accuracy of estimates monotonically improving?

What is monotonically improving?

Being Right is Not Good Enough

- Not sufficient to produce a good answer
- Need to have reason to believe that it is close to right
- In this case, small standard deviation implies that we are close to the true value of π

Right?

Is it Correct to State

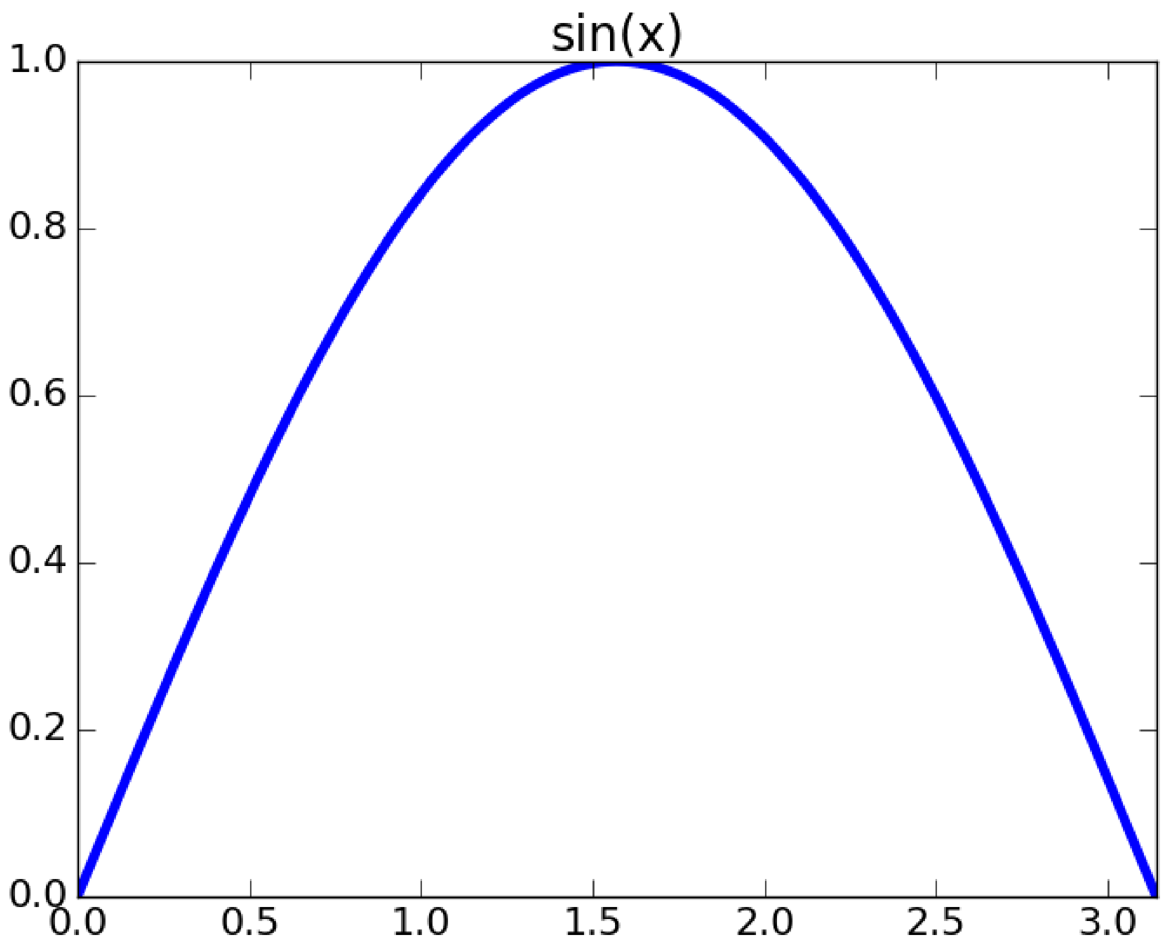
- 95% of the time we run this simulation, we will estimate that the value of pi is between 3.13743875875 and 3.14567467875?
- With a probability of 0.95 the actual value of π is between 3.13743875875 and 3.14567467875?
- Both are factually correct
- But only one of these statements can be inferred from our simulation
- $statiscally\ valid\ \neq true$

Introduce a Bug

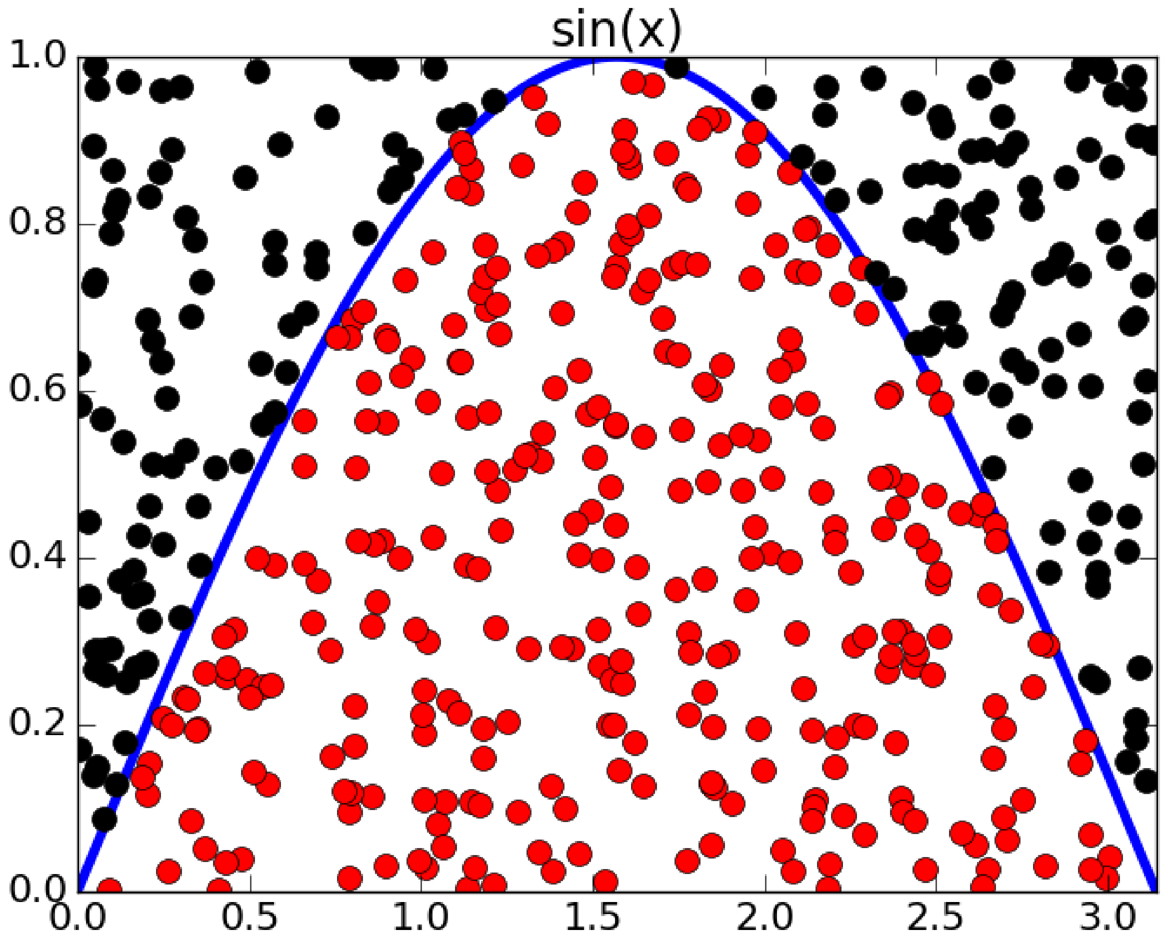
Generally Useful Technique

- To estimate the area of some region, R
  - Pick an enclosing region, E, such that the area of E is easy to calculate and R lies completely within E
  - Pick a set of random points that lie within E
  - Let F be the fraction of the points that fall within R
  - Multiply the area of E by F
- Way to estimate integrals

Sin(x)



Random Points



Plot a Function and Get Min and Max

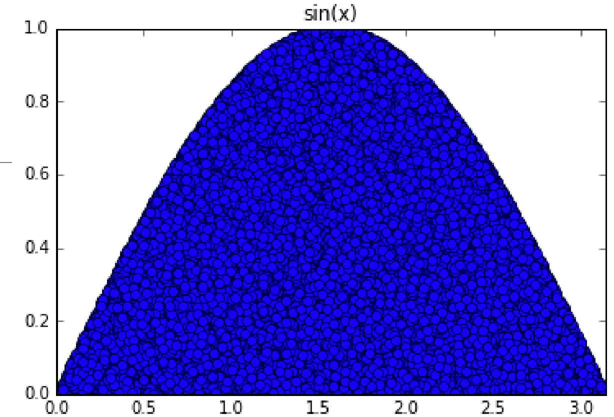
```
import matplotlib.pyplot as plt

def evalFcn(fcn, minX, maxX, toPlot):
    xVals = []
    yVals = []
    incr = 0.001
    curVal = minX
    while curVal < maxX:
        xVals.append(curVal)
        yVals.append(fcn(curVal))
        curVal += incr
    if toPlot:
        plt.plot(xVals, yVals)
        plt.hlines(0, minX, maxX)
        plt.xlim(minX, maxX)
        plt.title(fcn.__name__ + '(x)')
    return min(yVals), max(yVals)
```

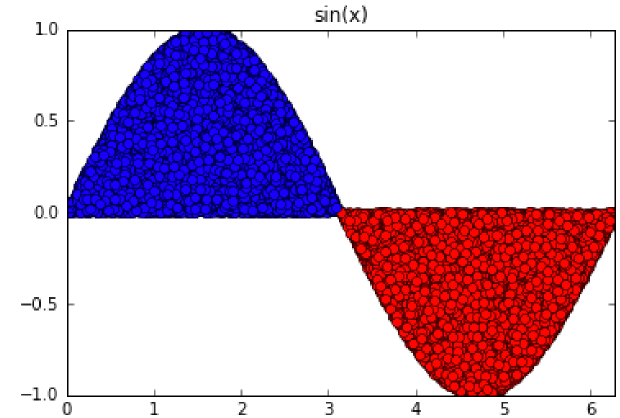
Integrate

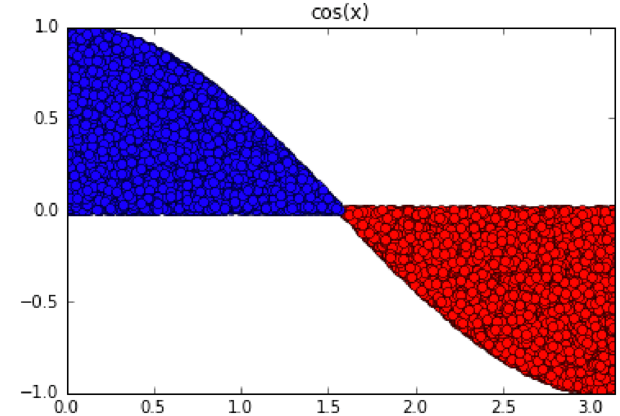
```
def dropNeedles(fcn, minX, maxX, minY, maxY, numNeedles, toPlot):
    underCurve = 0
    for needles in range(1, numNeedles + 1):
        x = random.uniform(minX, maxX)
        y = random.uniform(minY, maxY)
        if y > 0 and y < fcn(x):
            underCurve += 1
            if toPlot and needles%100 == 0:
                plt.plot(x, y, 'bo')
        elif y < 0 and y > fcn(x):
            underCurve -= 1
            if toPlot and needles%100 == 0:
                plt.plot(x, y, 'ro')
    return (underCurve/numNeedles)*(maxX - minX)*(maxY - minY)

def quadrature(fcn, minX, maxX, toPlot = True):
    minY, maxY = evalFcn(fcn, minX, maxX, toPlot)
    print('Integral of', fcn.__name__, 'from', round(minX, 2),
          'to', round(maxX, 2), '=',
          round(dropNeedles(fcn, minX, maxX, minY, maxY,\
                            1000000, toPlot), 2))
```

Test Quadrature

```
quadrature(np.sin, 0, np.pi)
pylab.figure()
quadrature(np.sin, 0, 2*np.pi)
pylab.figure()
quadrature(np.cos, 0, np.pi)
```








```
Integral of sin from 0 to 3.14 = 2.0
Integral of sin from 0 to 6.28 = -0.01
Integral of cos from 0 to 3.14 = 0.0
```