#### Monte Carlo Techniques

# Professor Stephen Sekula Guest Lecture – PHY 4321/7305



22

# What are "Monte Carlo Techniques"?

- Computational algorithms that rely on repeated random sampling in order to obtain numerical results
- Basically, you run a simulation over and over again to calculate the possible outcomes, either based on probability or to determine probability
- Like playing a casino game over and over again and recording all the game outcomes to determine the underlying rules of the game
- Monte Carlo is a city famous for its gambling hence the name of this class of techniques



Rock Flow Dynamics

http://rfdyn.com/

Monte Carlo modeling of geological hydrodynamics for hydrocarbon extraction



MATLAB is used to Monte Carlo simulate financial evolution





Design and use of "phantoms" to model radiation dose in human tissue – necessary for prepping a patient for radiation-based cancer therapy.

Stephen J. Sekula - SMU

#### HAVE YOU EVER (KNOWINGLY) USED "MONTE CARLO TECHNIQUES"?

Stephen J. Sekula - SMU



#### **EVER PLAYED "BATTLESHIP"?**

MISSES



IF SO, YOU HAVE APPLIED MONTE CARLO TECHNIQUES.

HITS

# A Simple Physical Example

- Let's illustrate this class of techniques with a simple physical example: numerical computation of  $\pi$
- π: the ratio of the circumference of a circle to its diameter: 3.14159265359...
- It's difficult to whip out a measuring tape or ruler and accurately measure the circumference of an arbitrary circle.
- The Monte Carlo method avoids this problem entirely

Begin by drawing a square, inscribed into which is a circle. The properties of the square are much easier to measure.



#### What do we know?



We know the relationship between the radius of a circle and the x and y coordinate of a point on the circle boundary:

$$r = \sqrt{x^2 + y^2}$$

Let us imagine that we have a way of randomly throwing a dot into the square (imagine a game of darts being played, with the square as the board...)



There is a probability that a uniformly, randomly thrown dot will land in the circle, and a probability that it will land out of the circle. What are those probabilities?



Probability of landing in the circle is merely given by the ratio of the areas of the two objects:



That's nice – but we're missing a piece . . . just what is that probability on the left side? How can we determine it?



ANSWER: "numerically" - by throwing dots uniformly in the square and counting the number that land inside the circle, divided by the number that we have thrown in total:



ANSWER: "numerically" - by throwing dots uniformly in the square and counting the number that land inside the circle, divided by the number that we have thrown in total:



Stephen J. Sekula - SMU

 $\pi$  is then simply determined numerically via:



# The Pieces

- Random numbers
  - needed to "throw dots" at the board
- Uniformity of coverage
  - we want to pepper the board using uniform random numbers, to avoid creating artificial pileups that create new underlying probabilities
- Code/Programming
  - You can do this manually with a square, an inscribed circle, coordinate axes, and a many-sided die.
  - But that limits your time and precision computers are faster for such repetitive tasks

# Computational Examples

- I will demonstrate the underlying computation framework principles using PYTHON, a free and open-source programming language and computation framework
  - Why? Because every PC in the lab has PYTHON installed and ready to go!
- At the end of this, you will have a program you can take with you and adapt into ANY language.
- If you've never seriously written code before, today is your "lucky" day

## **Basics of Coding**

- Numbers all programming languages can minimally handle numbers: integers, decimals
- Variables placeholders for numbers, whose values can be set at any time by the programmer
- Functions any time you have to repeatedly perform an action, write a function. A "function" is just like in math it represents a complicated set of actions on variables
- Code an assembly of variables and functions whose goal is determined by the programmer. "Task-oriented mathematics"
- Coding is the poetry of mathematics it takes the basic rules of mathematics and does something awesome with them.

# Python and Jupyter

- Python is a programming language
- Jupyter is a framework for developing web-based interactive python software
- We will use Python and Jupyter together today. I think it makes programming more fun and also more share-able.
  - Jupyter notebooks can be shared with other people, who can improve them and share them again.

# Starting Jupyter

Start a "Terminal" - a window containing a prompt where you can enter commands. This is still a primary way in which computer scientists, physicists, and any other discipline that must command a computer interacts with the machine: you talk to it by typing commands.

At the terminal prompt, type this and then press the <ENTER> key (a key stroke will be enclosed in angle-brackets in this talk).

#### jupyter notebook

You will be told to either copy-and-paste or click a link to open a browser; or, the browser might open on its own.

Stephen J. Sekula - SMU

💭 Jupyter	Logout
Files Running Clusters	
Select items to perform actions on them.	Upload New - 2
	Text File
Fractals.ipynb	Folder Terminal
pythonbasics_honors.ipynb	Natabaska
Untitled.ipynb	Python 3
named_colors.png	R.

After the get the above page in your browser, select from the upperright "New" menu to make a Python (2,3) notebook. This will produce a blank Python canvas (in a new tab or window) upon which you can paint your mathematical masterpiece.

JUPYTER Untitled Last Checkpoint: 2 minutes ago (unsaved changes)	Cogout
File Edit View Insert Cell Kernel Help	Python 3 O
►     ∞     ▲     ▲     ►     ►     C     Code     CellToolbar	
In []:	

This is the place where you can begin to experiment with Python programming. It's simple. You type a Python program command, then press <SHIFT+ENTER>, and the "kernel" (the running Python executable in the background) will process your code and return any results that are obtained.

If you wish to alter or re-run a command, click on it and press <SHIFT+ENTER> again.



A simple example of defining a variable ("Ntotal"), setting its value ("=" is the "assignment operator" in Python, and using 100 without a decimal point after the number indicates this is an integer), and then printing its value. To obtain the line-break, press <ENTER>. To execute the whole block of code you just wrote, press <SHIFT+ENTER>.



Try defining a new variable in the next block of code ("Nin") and set it to 55.0 (decimal points indicate that this number is a "floating point number" - a number with decimal precision). Print the mathematical operation of dividing Ntotal by Nin ("/" is the "divide by" operator in Python).

Note that although Ntotal was an integer and Nin was a floating point number, the math operation return a floating point number.

## Uniform Random Numbers

- Computers can generate (pseudo)random numbers using various algorithms
  - this is a whole lecture in and of itself if you're interested in pseudo-random numbers, etc. go do some independent reading
- We will utilize the PYTHON "random" library and its ".uniform()" function to obtain our uniform random numbers

https://docs.python.org/3/library/random.html

In	[6]:	<pre>import random</pre>
		random.uniform(0.0,1.0)

Out[6]: 0.9249294856462658

- In [7]: random.uniform(0.0,1.0)
- Out[7]: 0.9162778141642234
- In [8]: random.uniform(0.0,1.0)
- Out[8]: 0.9856041081761423
- In [9]: random.uniform(0.0,1.0)
- Out[9]: 0.2455499563864707

"random.uniform(0.0,1.0)" generates a uniform random floating-point decimal number between 0 and 1 (inclusive) Stephen J. Sekula - SMU 29

### Designing our "game board"



# Designing our "game board"



We don't need the whole game board – we can just use onequarter of it. This keeps the program simple!

Alternative: you can rescale the output of "rand" to generate random numbers between -1 and 1

 $\frac{(1/4)\pi r^2}{(1/4)4r^2} = \frac{\pi}{4} = P(\text{in}|\text{dot})$ 

https://docs.python.org/3/library/math.html

```
In [3]: import math
        import random
        Ntotal = 100
        Nin = 0
        x = random.uniform(0.0,1.0)
        y = random.uniform(0.0,1.0)
         r = math.sqrt(x**2 + y**2)
        print(x)
        print(y)
        print(r)
        0.0903585398665
        0.316618141258
        0.329259340187
```

# Repetition

- You don't want to manually type 100 (or more) computations of your dot throwing
- You need a loop!
- A "loop" is a small structure that automatically repeats your computation an arbitrary number of times
- In PYTHON:

```
Ntotal = 100
Nin = 0
for i in range(1,Ntotal):
    x = random.uniform(0.0,1.0)
    y = random.uniform(0.0,1.0)
```

[11]:	<pre>import math Ntotal = 10 Nin = 0 for i in rat x = ran y = ran r = mat print(";</pre>	0 dom.uniform() dom.uniform() h.sqrt(x**2 x=%f, y=%f, j	): 0.0,1.0) 0.0,1.0) + y**2) r=%f" % (x,y,r))
	x=0.476185,	y=0.221475,	r=0.525170
	x=0.675048,	y=0.422341,	r=0.796280
	x=0.009795,	y=0.332586,	r=0.332730
	x=0.579938,	y=0.966590,	r=1.127220
	x=0.106765,	y=0.803517,	r=0.810579
	x=0.903053,	y=0.829679,	r=1.226325
	x=0.853741,	y=0.029755,	r=0.854259
	x=0.811355,	y=0.187595,	r=0.832759
	x=0.130509,	y=0.604732,	r=0.618654
	x=0.016680,	y=0.388991,	r=0.389348
	x=0.436060,	y=0.385543,	r=0.582058
	x=0.258471,	y=0.011611,	r=0.258732
	x=0.644286,	y=0.443687,	r=0.782281
	x=0.848350,	y=0.130553,	r=0.858337
	x=0.964408,	y=0.829467,	r=1.272045
	x=0.757361,	y=0.874556,	r=1.156911
	x=0.902675,	y=0.656903,	r=1.116397
	x=0.999765,	y=0.092378,	r=1.004024
	x=0.385890,	y=0.897866,	r=0.977279
	x=0.279021,	y=0.097120,	r=0.295441
	x=0.603913,	y=0.038930,	r=0.605167
	x=0.983921,	y=0.212304,	r=1.006565

In

"Loops" are powerful – they are a major workhorse of any repetitive task coded up in a programming language.

# Final Piece

- So we have generated a dot by generating its x and y coordinates throwing uniform random numbers...
- How do we determine if it's "in" or "out" of the circle?
- ANSWER:
  - if  $r = \sqrt{(x^2+y^2)} \le R$ , it's in or on the circle; otherwise, it is out of the circle!

```
In [14]: Ntotal = 100
         Nin = 0
         R = 1.0
         for i in range(1,Ntotal):
             x = random.uniform(0.0,1.0)
             y = random.uniform(0.0,1.0)
              r = math.sqrt(x**2 + y**2)
             if r \leq R:
                  Nin = Nin + 1
                  # alternatively, Nin += 1 (auto-increment by 1)
         my pi = 4.0*float(Nin)/float(Ntotal)
         print("pi = %f" % (my pi))
         pi = 3.320000
```

#### A working program.

#### You can increase $N_{total}$ to get increased precision!

Stephen J. Sekula - SMU

### A Comment on Precision

• Given finite statistics, each set of trials carries an uncertainty ( $\pi \pm \sigma_{\pi}$ ). A point, (x,y), can either be in/on or out of the circle of radius, R. Thus the uncertainty on N<sub>in</sub> can be treated as a <u>binomial error</u>:

$$\sigma_{N_{\text{in}}} = \sqrt{N_{\text{total}}} \cdot p(1-p)$$
 where  $p = N_{\text{in}}/N_{\text{total}}$ 

• Propagating this to  $\pi$ :  $\sigma_{\pi} = 4 \cdot \sigma_{N_{\text{in}}} / N_{\text{total}} = 4 \sqrt{\frac{N_{\text{in}}}{N_{\text{total}}^2} \left(1 - \frac{N_{\text{in}}}{N_{\text{total}}}\right)}$ 

Stephen J. Sekula - SMU

# Precision (continued)

• Relative error:

$$\frac{\sigma_{\pi}}{\pi} = \sqrt{\frac{1}{N_{\rm in}} - \frac{1}{N_{\rm total}}}$$

- For 100 trials,  $\sigma_{\pi}/\pi = 15.2\%$  (e.g.  $3.04 \pm 0.46$ )
- For 1000 trials,  $\sigma_{\pi}/\pi = 4.8\%$  (e.g.  $3.15 \pm 0.15$ )
- For 10,000 trials:  $\sigma_{\pi}/\pi = 1.5\%$  (e.g.  $3.141 \pm 0.047$ )
- For 100,000 trials:  $\sigma_{\pi}/\pi = 0.48\%$  (e.g. 3.128 ± 0.015)

#### Note that uncertainty scales only as $1/\sqrt{N}_{total}$

Stephen J. Sekula - SMU

improves by  $\sqrt{10}=3.1623$ 

improves by  $\sqrt{10=3.1623}$ 

```
In [27]: Ntotal = 100000
Nin = 0
R = 1.0
for i in range(1,Ntotal):
    x = random.uniform(0.0,1.0)
    y = random.uniform(0.0,1.0)
    r = math.sqrt(x**2 + y**2)
    if r <= R:
        Nin = Nin + 1
        # alternatively, Nin += 1 (auto-increment by 1)

my_pi = 4.0*float(Nin)/float(Ntotal)
my_pi_uncertainty = my_pi * math.sqrt(1.0/float(Nin) + 1.0/float(Ntotal))
print("pi = %.6f +/- %.6f (percent error= %.2f)" % (my_pi, my_pi_uncertainty, 100.0*my_pi_uncertainty/my_pi))
pi = 3.128360 +/- 0.014933 (percent error= 0.48)</pre>
```

Play around with computing all kinds of things, like the percent error, the absolute error, and printing extended and nicely formatted numbers. Python is fun! Imagine if you had learned this earlier – how much better a homework result could you have prepared by attacking problems both analytically AND numerically?

# Why is this powerful?

- You've just learned how to compute an integral NUMERICALLY.
- You can apply this technique to any function whose integral (area) you wish to determine
- For instance, consider the next slide.



- Given an arbitrary function, f(x), you can determine its integral numerically using the "Accept/Reject Method"
- **First**, find the maximum value of the function (e.g. either analytically, if you like, or by calculating the value of f(x) over steps in x to find the max. value, which I denote F(x))
- **Second**, enclose the function in a box, h(x), whose height is F(x) and whose length encloses as much of f(x) as is possible.
- **Third**, compute the area of the box (easy!)
- **Fourth**, throw points in the box using uniform random numbers. Throw a value for x, denotes x'. Throw a value for y, denoted y'. If y' < f(x'), it's a hit! If not, it's a miss!



$$\frac{N_{\text{hits}}}{N_{\text{total}}} = \frac{I(f(x))}{A(h(x))}$$

This, in the real world, is how physicists, engineers, statisticians, mathematicians, etc. compute integrals of arbitrary functions. Learn it. Love it. It will save you.

# **Generating Simulations**

- The Monte Carlo technique, given a function that represents the probability of an outcome, can be used to generate "simulated data"
- Simulated data is useful in designing an experiment, or even "running" an experiment over and over to see all possible outcomes



### Young's Double-Slit Experiment Simulation

- Consider slits of width, b, separated by a distance, d.
- Given the function that describes the probability of finding a photon at a given angle:

$$I(\theta) \propto \cos^{2} \left[ \frac{\pi d \sin(\theta)}{\lambda} \right] \operatorname{sinc}^{2} \left[ \frac{\pi b \sin(\theta)}{\lambda} \right]$$
$$\operatorname{sinc}(x) = \begin{cases} \sin(x)/x & (x \neq 0) \\ 1 & (x = 0) \end{cases}$$

Stephen J. Sekula - SMU

# Next Steps

- Need the max. value of  $I(\theta)$ 
  - occurs at  $\theta = 0$
- Use that to compute the height of the box; the width of the box is  $2\pi$  (ranging from  $-\pi$  to  $+\pi$ )
- "Throw" random points in the box until you get 1000 "accepts"
- Now you have a "simulated data" sample of 1000 photons scattered in the two-slit experiment.



1000 simulated photons scattered through a double-slit experiment. This was done in C++ using the free ROOT High-Energy Physics data analysis framework, so I could easily generate a *histogram* – a binned data sample.

### Resources

- Python: (open-source, free) http://www.python.org/
- Project Jupyter: (open-source, free) http://jupyter.org/
- Mathematica: (closed-source, non-free) http://www.wolfram.com/mathematica/
- Maxima (open-source, free "Mathematica") http://maxima.sourceforge.net
- Monte Carlo Techniques: http://en.wikipedia.org/wiki/Monte\_Carlo\_method