Monte Carlo Techniques

Professor Stephen Sekula

Guest Lecture – PHY 4321/7305

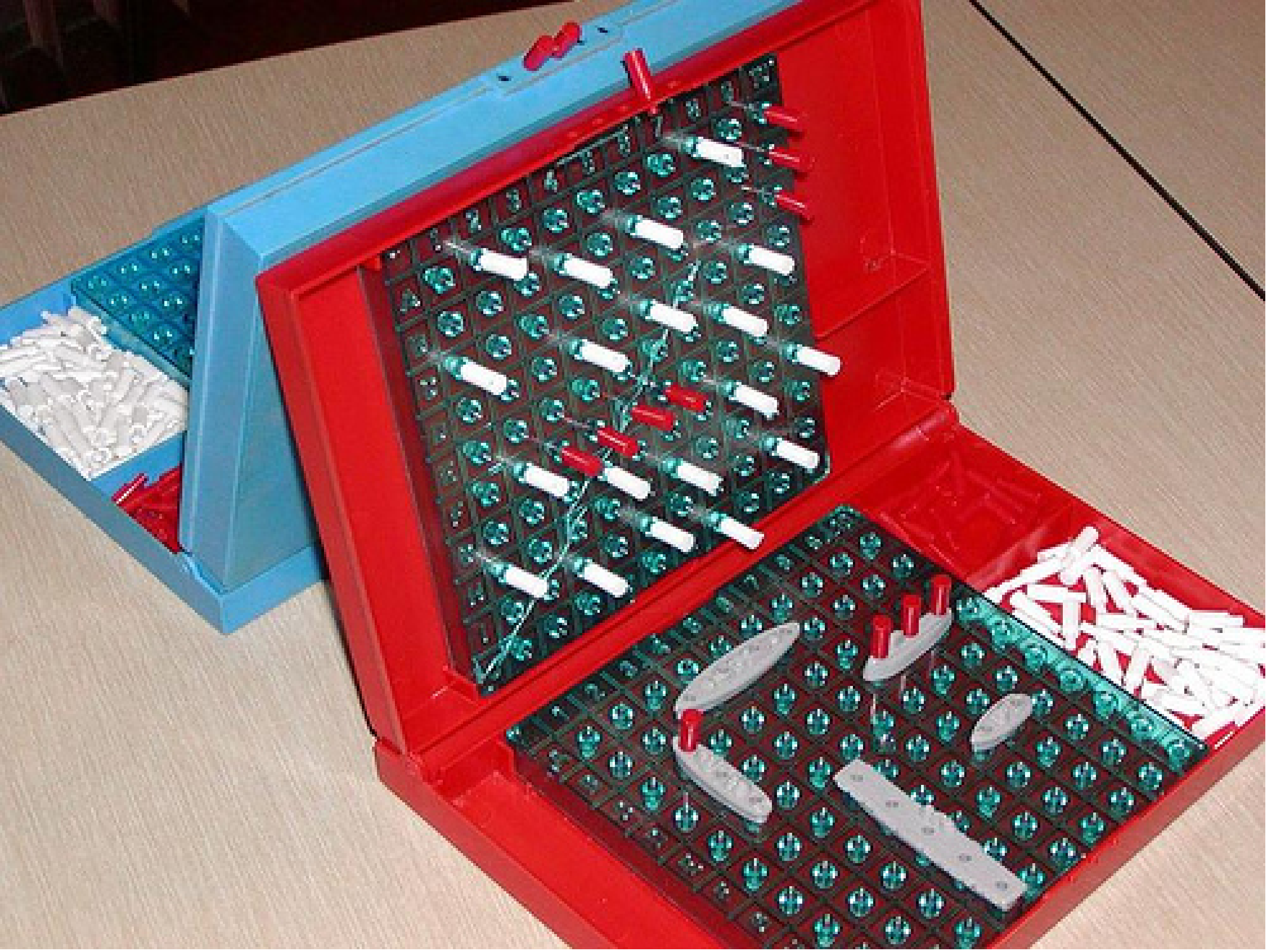
SMU | DEDMAN COLLEGE OF HUMANITIES & SCIENCES

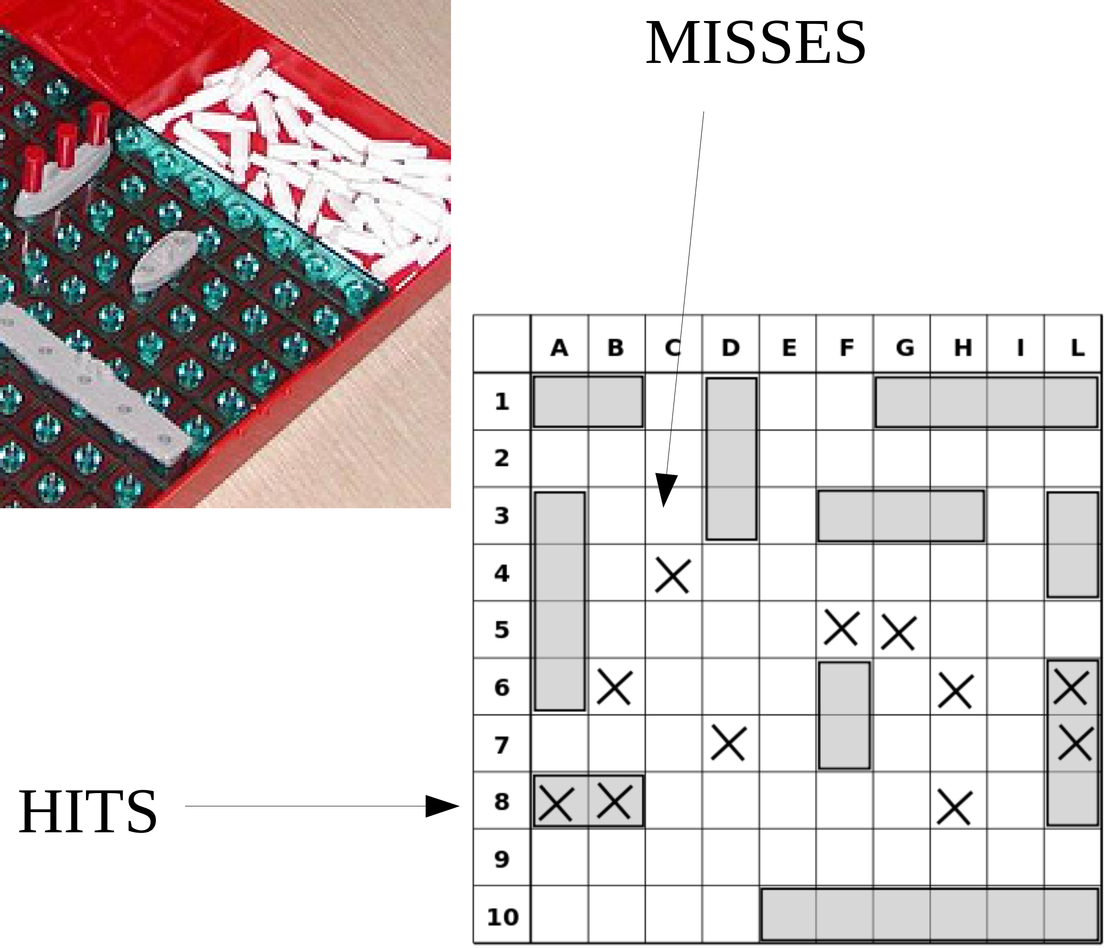
What are “Monte Carlo Techniques”?

- 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 underlying probabilities that lead to the outcomes
- 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

HAVE YOU EVER (KNOWINGLY) USED “MONTE CARLO TECHNIQUES”?

EVER PLAYED "BATTLESHIP"?



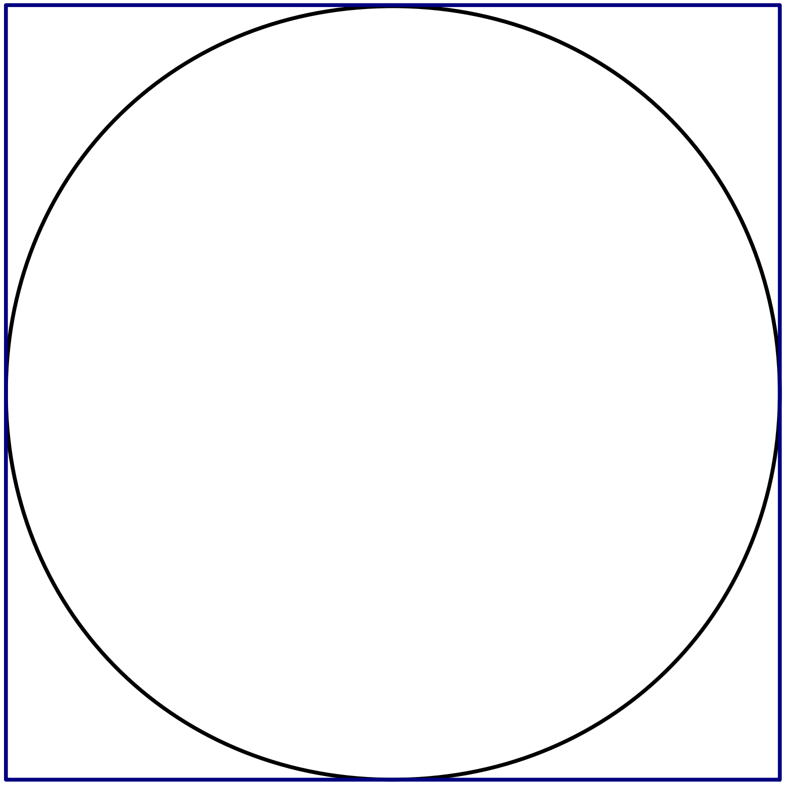


IF SO, YOU HAVE APPLIED MONTE CARLO TECHNIQUES.

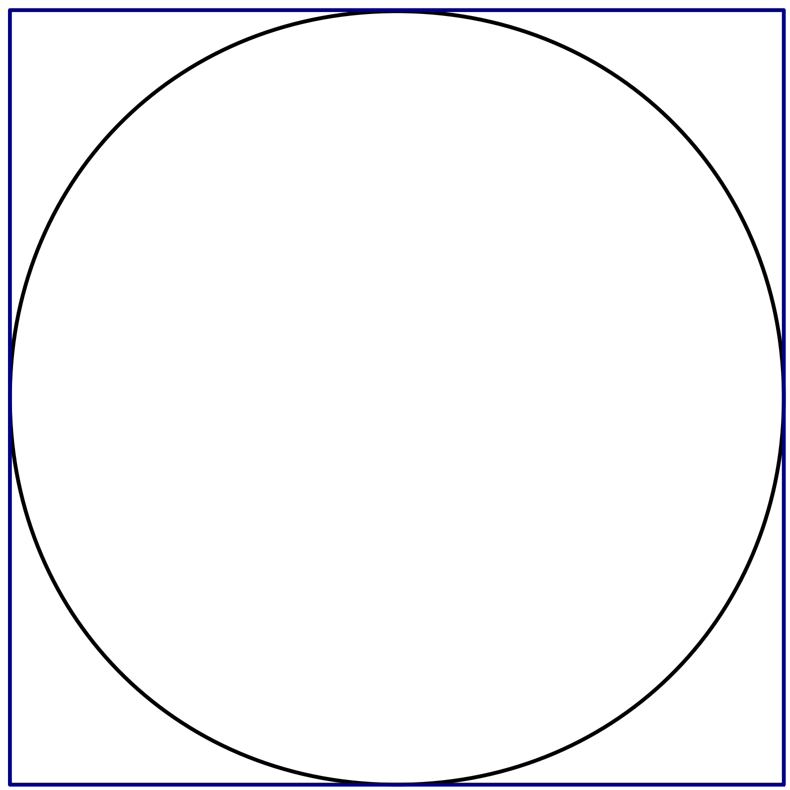
A Simple Physical Example

- Let's illustrate this class of techniques with a simple physical example: numerical computation of π
- π: the ratio of the circumference of a circle to its diameter.
- 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.

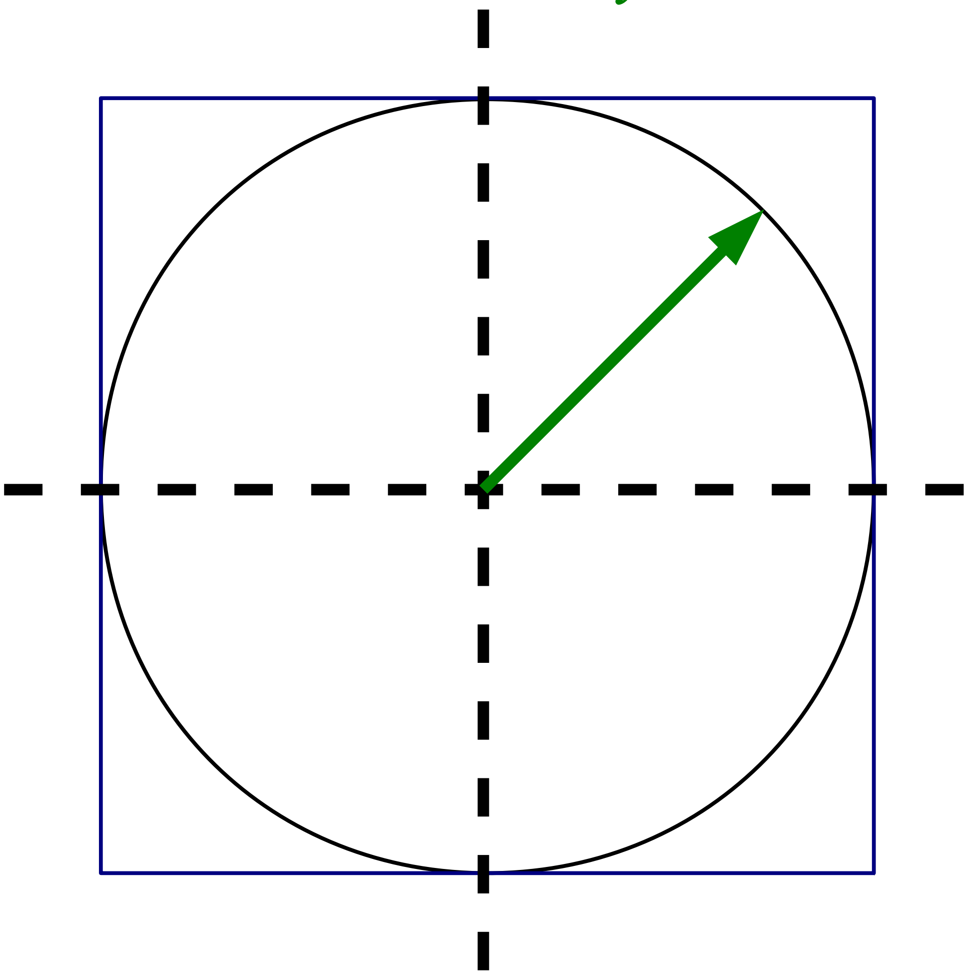


What do we know?

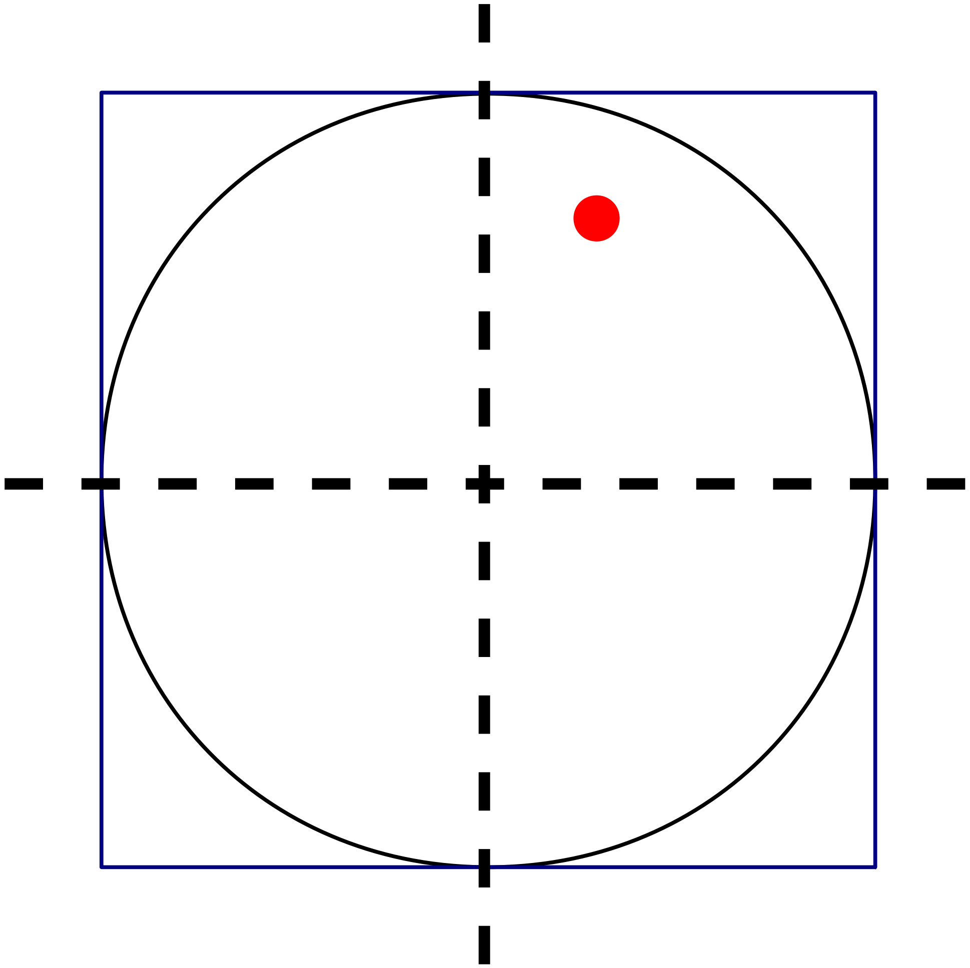


We know the relationship between the radius of a circle and the x and y coordinate of a point on the radius:

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



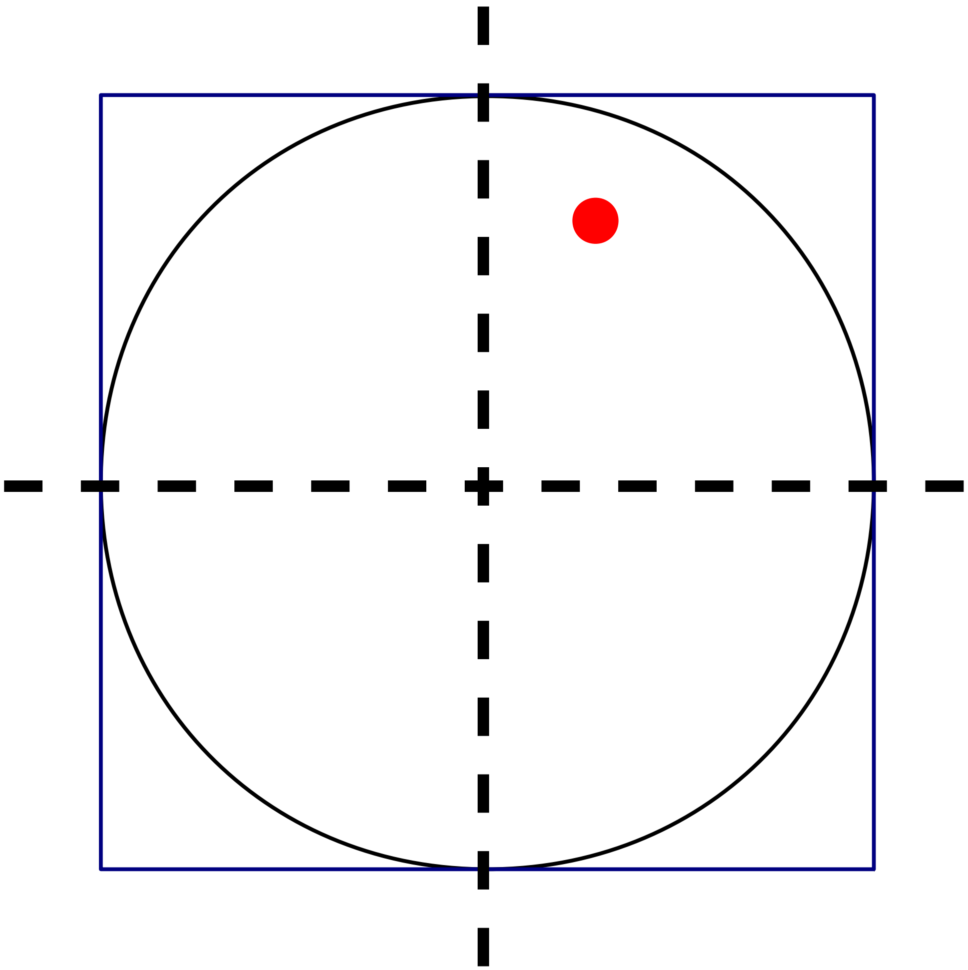
Let us imagine that we have a way of randomly throwing a dot into the square (imagine a game of darts being played, with the square as the board...)



Knowns:

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

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?

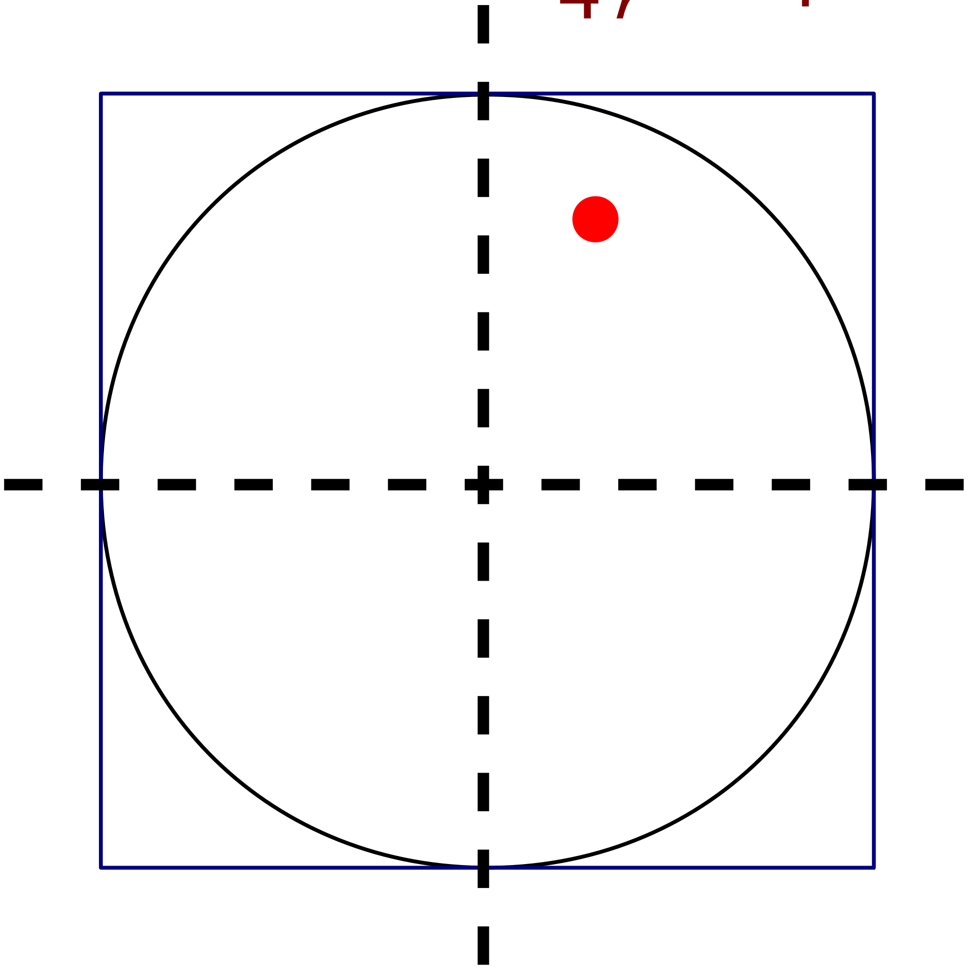


Knowns:

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

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

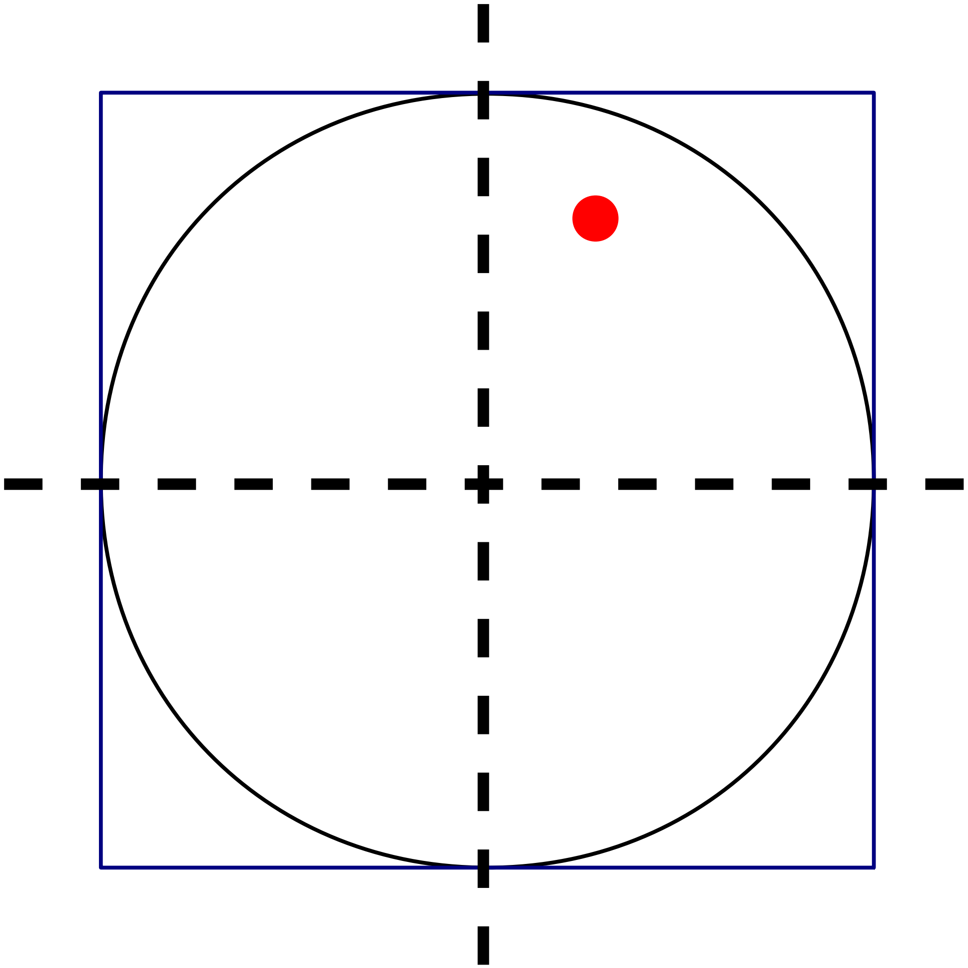
$$P(\text{in}|\text{dot}) = \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$$



Knowns:

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

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



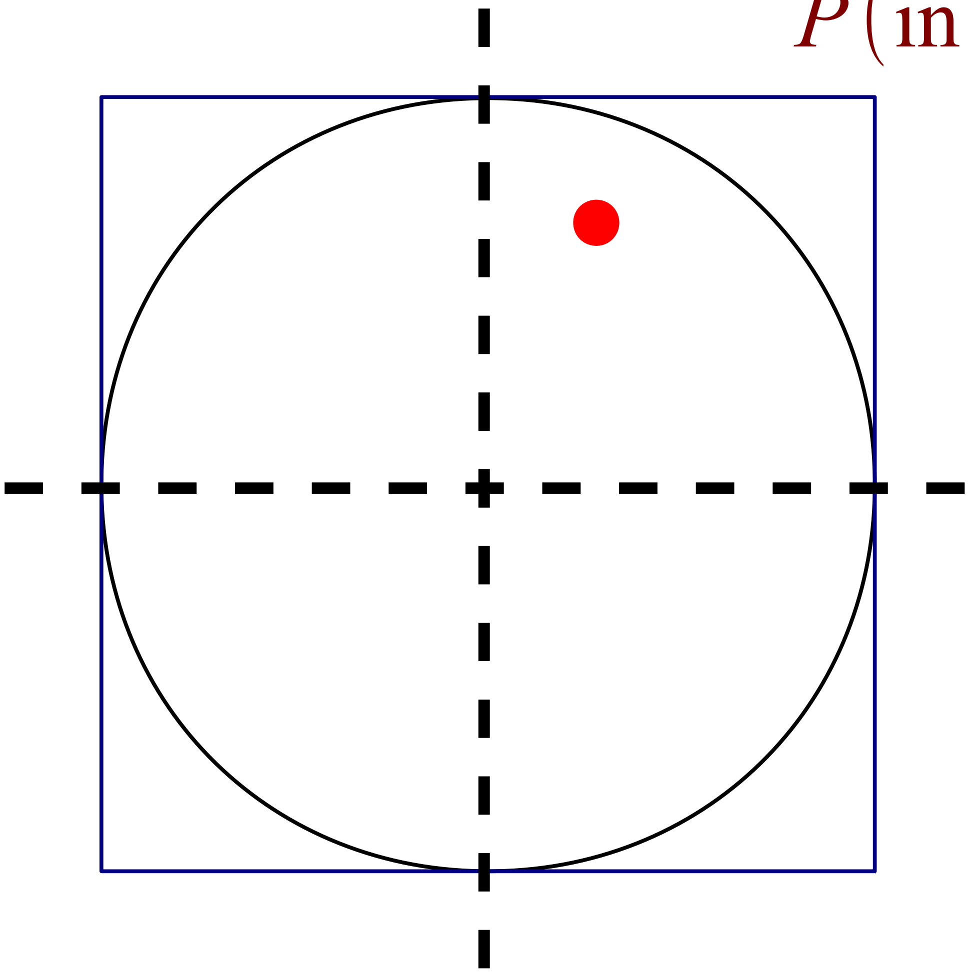
Knowns:

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

$$P(\mathrm{in}|\mathrm{dot})=\frac{\pi}{4}$$

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:

$$P(\text{in}|\text{dot}) = \frac{N_{\text{in}}}{N_{\text{total}}}$$

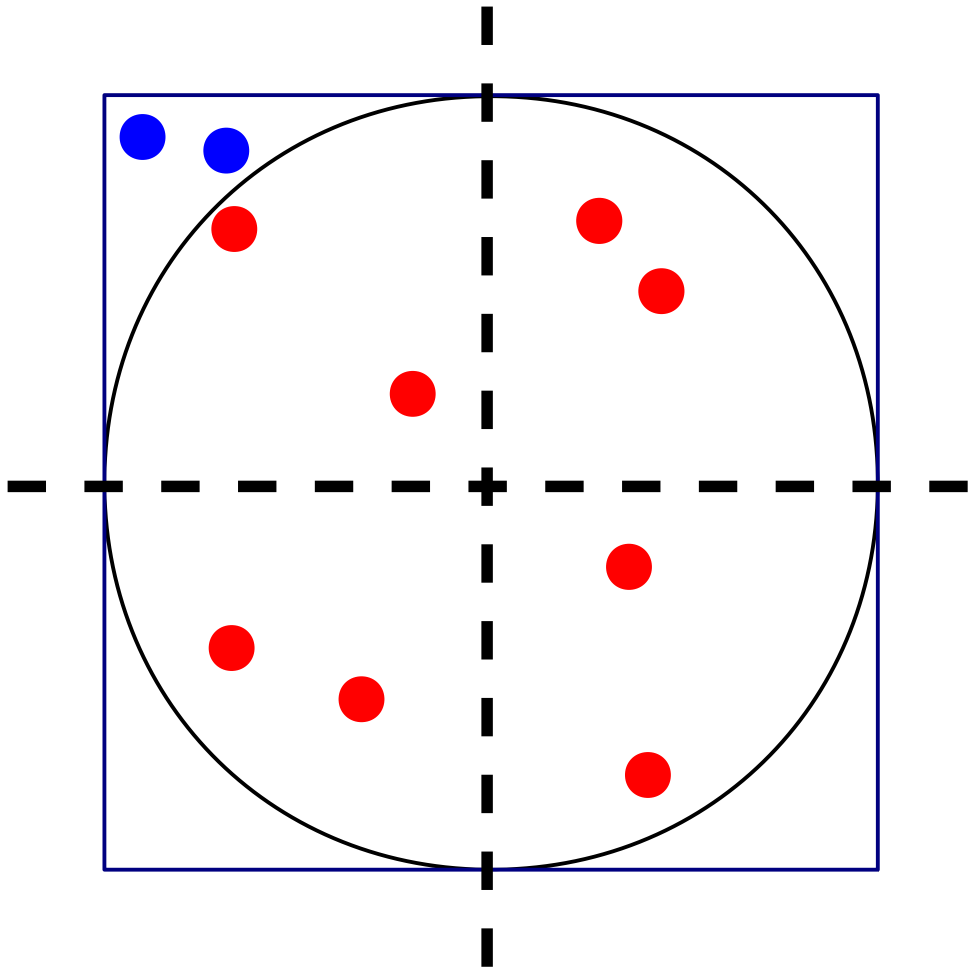


Knowns:

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

$$P(\text{in}|\text{dot}) = \frac{\pi}{4}$$

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:



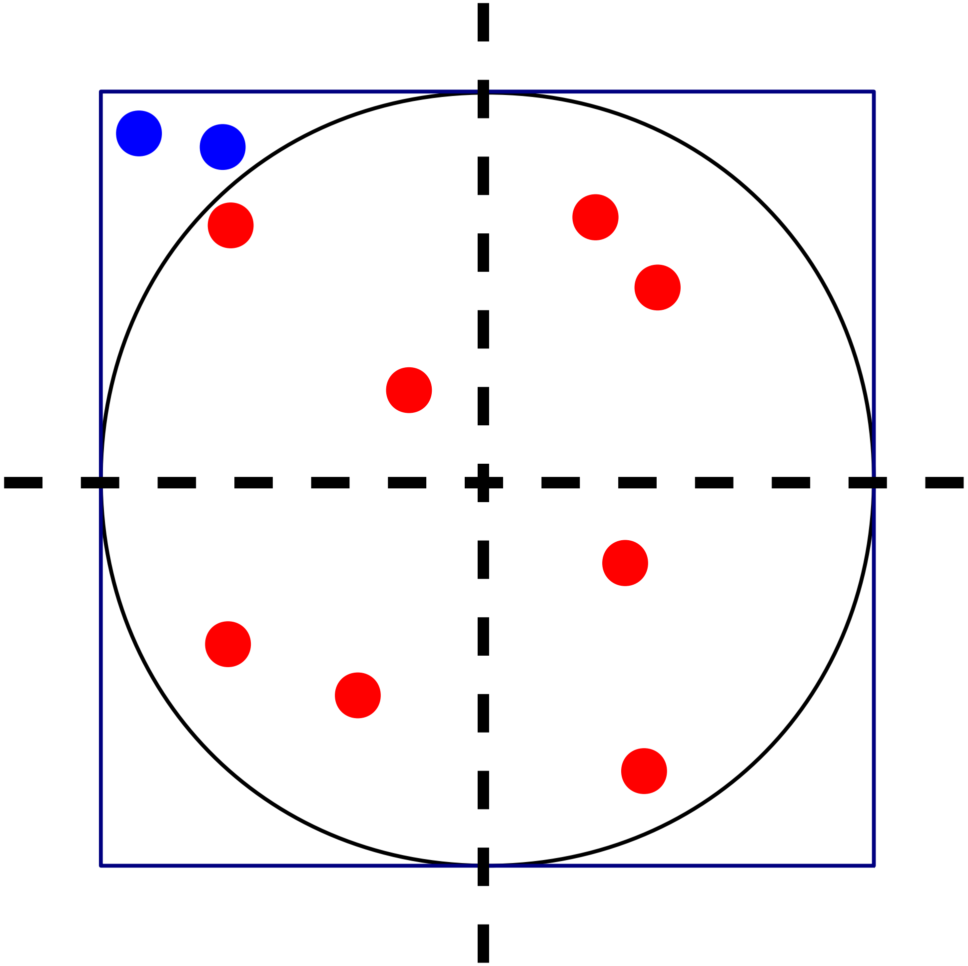
Knowns:

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

$$\frac{N_{in}}{N_{total}} = \frac{\pi}{4}$$

π is then simply determined numerically via:

$$\pi = 4\frac{N_{in}}{N_{total}}$$



Knowns:

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

$$\frac{N_{in}}{N_{total}} = \frac{\pi}{4}$$

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 FOSC 60 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.

sekula : python

File Edit View Bookmarks Settings Help

```
sekula@alchemist:~$ python
Python 2.7.10 (default, Oct 14 2015, 16:09:02)
[GCC 5.2.1 20151010] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> Ntotal = 100
>>> print Ntotal
100
>>>
```

You type it, it does it.

sekula : bash | SMU-Lecture-MonteCarlo : emacs | trunk : bash | EShield2 : bash | sekula : python sekula : python

File Edit View Bookmarks Settings Help

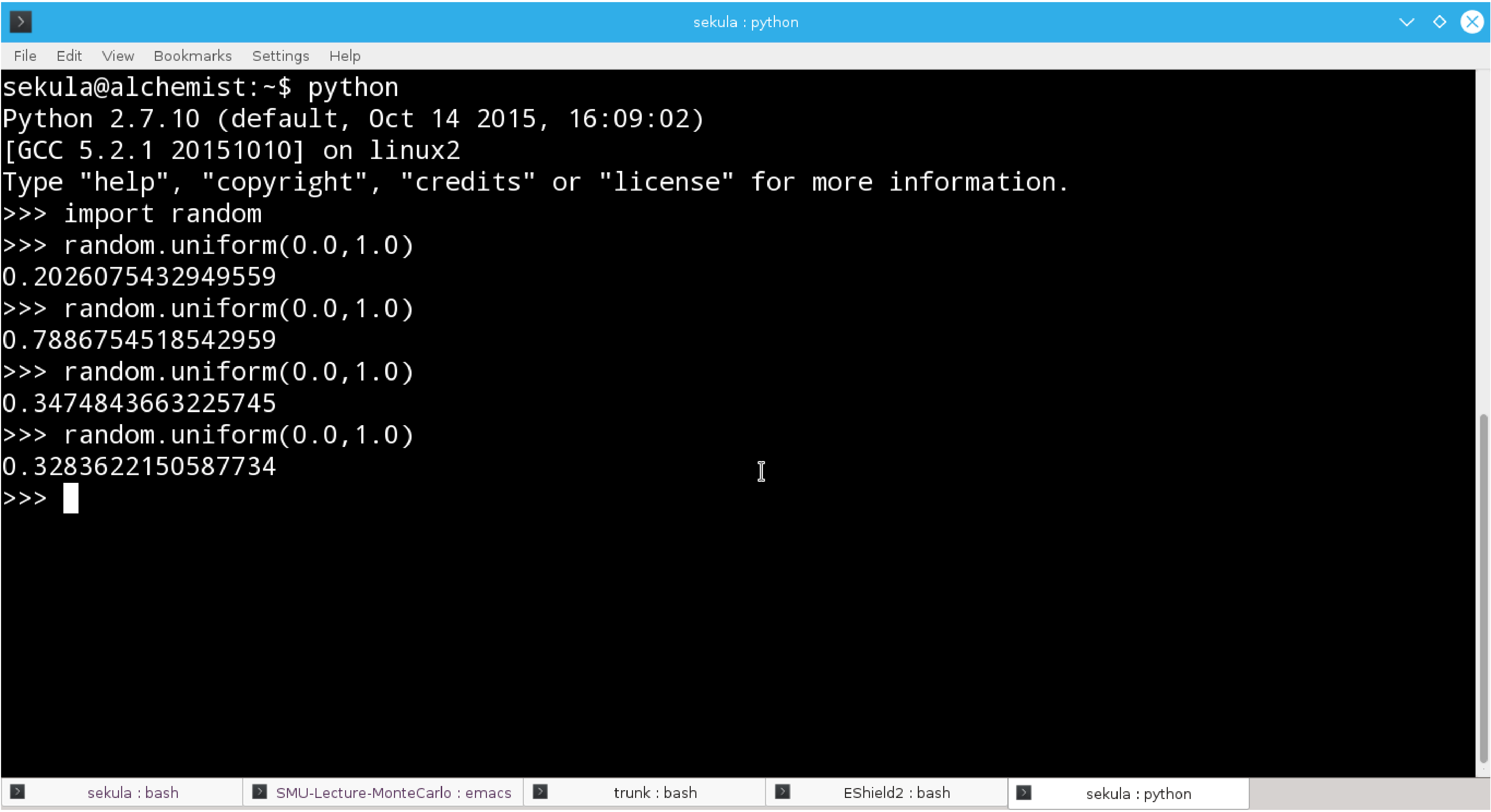
```
sekula@alchemist:~$ python
Python 2.7.10 (default, Oct 14 2015, 16:09:02)
[GCC 5.2.1 20151010] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> Ntotal=100.0
>>> Nin=55.0
>>> print Ntotal/Nin
1.81818181818
>>> 
```

sekula : bash | SMU-Lecture-MonteCarlo : emacs | trunk : bash | EShield2 : bash | sekula : python

Uniform Random Numbers

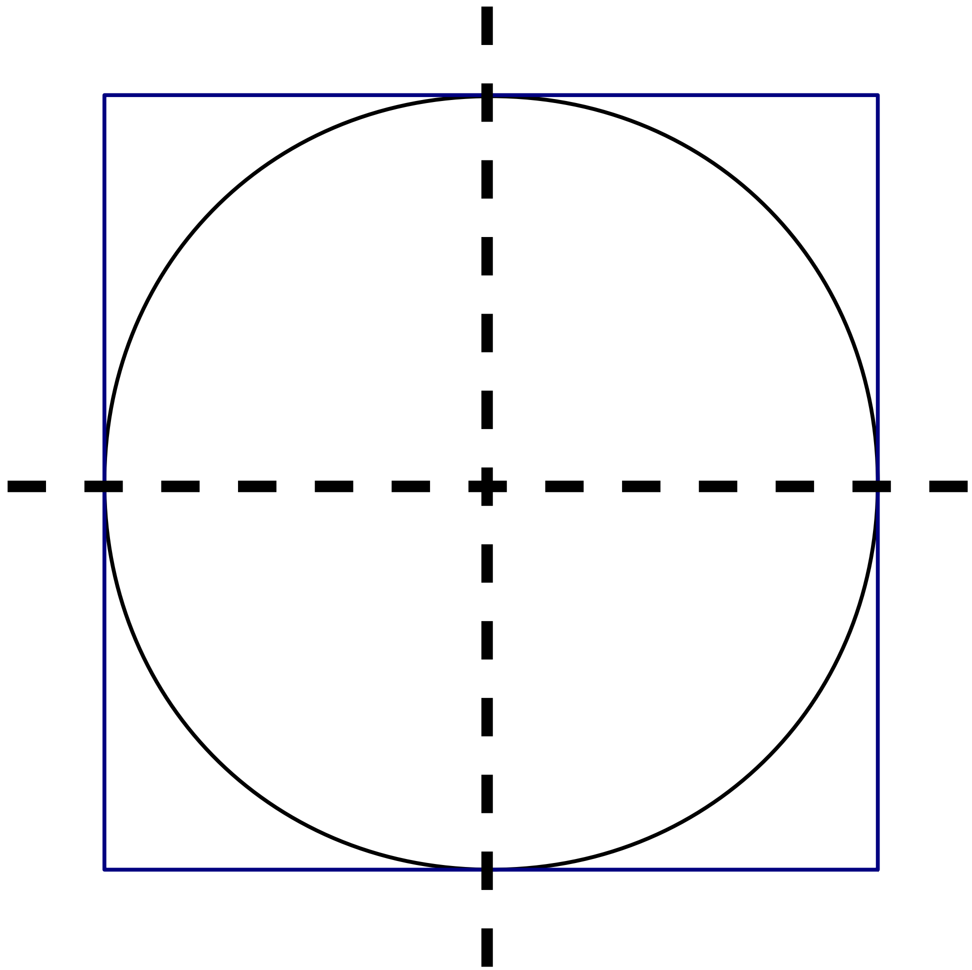
- Computers can generate (pseudo)random numbers using various algorithms
  - this is a whole lecture in and of itself – if you're interested in pseudo-random numbers, etc. go do some independent reading
- We will utilize the “rand” function in OCTAVE to obtain our uniform random numbers

https://docs.python.org/2/library/random.html

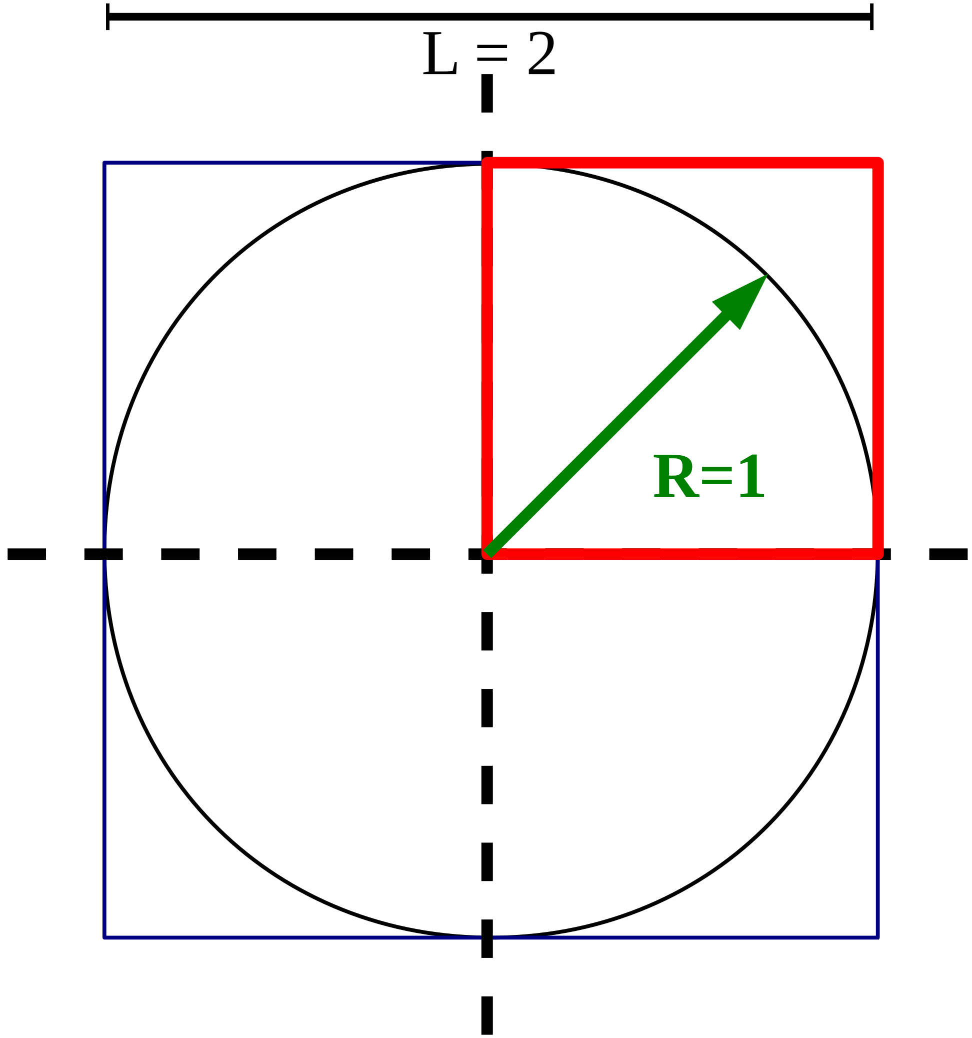


"random.uniform(0.0,1.0)" generates a uniform random floating-point decimal number between 0 and 1 (inclusive)

Designing our “game board”



Designing our “game board”



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

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

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

https://docs.python.org/2/library/math.html

```
sekula@alchemist:~$ python
Python 2.7.10 (default, Oct 14 2015, 16:09:02)
[GCC 5.2.1 20151010] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import random
>>> import math
>>> x = random.uniform(0.0,1.0)
>>> y = random.uniform(0.0,1.0)
>>> r = math.sqrt(x**2 + y**2)
>>> print x
0.432817827706
>>> print y
0.209008763831
>>> print r
0.480641171081
>>>
```

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)
```

```
sekula@alchemist:~$ python
Python 2.7.10 (default, Oct 14 2015, 16:09:02)
[GCC 5.2.1 20151010] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import random
>>> import math
>>> Ntotal = 100
>>> Nin = 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)
...     print "x=%f, y=%f, r=%f" % (x,y,r)
...
x=0.745232, y=0.473733, r=0.883059
x=0.531765, y=0.718660, r=0.894006
x=0.654799, y=0.077412, r=0.659359
x=0.981373, y=0.091605, r=0.985639
x=0.251635, y=0.769428, r=0.809530
x=0.132645, y=0.603831, r=0.618228
x=0.794403, y=0.603306, r=0.997524
x=0.806367, y=0.183981, r=0.827090
```

“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 $r = \sqrt{(x^2+y^2)} < R$, it's in the circle; otherwise, it is out of the circle!

```
sekula@alchemist:~$ python
Python 2.7.10 (default, Oct 14 2015, 16:09:02)
[GCC 5.2.1 20151010] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import random
>>> import math
>>> 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 < R:
...         Nin = Nin + 1
...
>>> my_pi = 4.0*float(Nin)/float(Ntotal)
>>>
>>> print my_pi
3.12
>>>
```

A working program.

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

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 or out of the circle of radius, R. Thus the uncertainty on N_{in} can be treated as a binomial error:

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

- Propagating this to π:

$$\sigma_\pi = 4 \cdot \sigma_{N_{in}}/N_{total} = 4\sqrt{\frac{N_{in}}{N_{total}^2}\left(1-\frac{N_{in}}{N_{total}}\right)}$$

Precision (continued)

- Relative error:

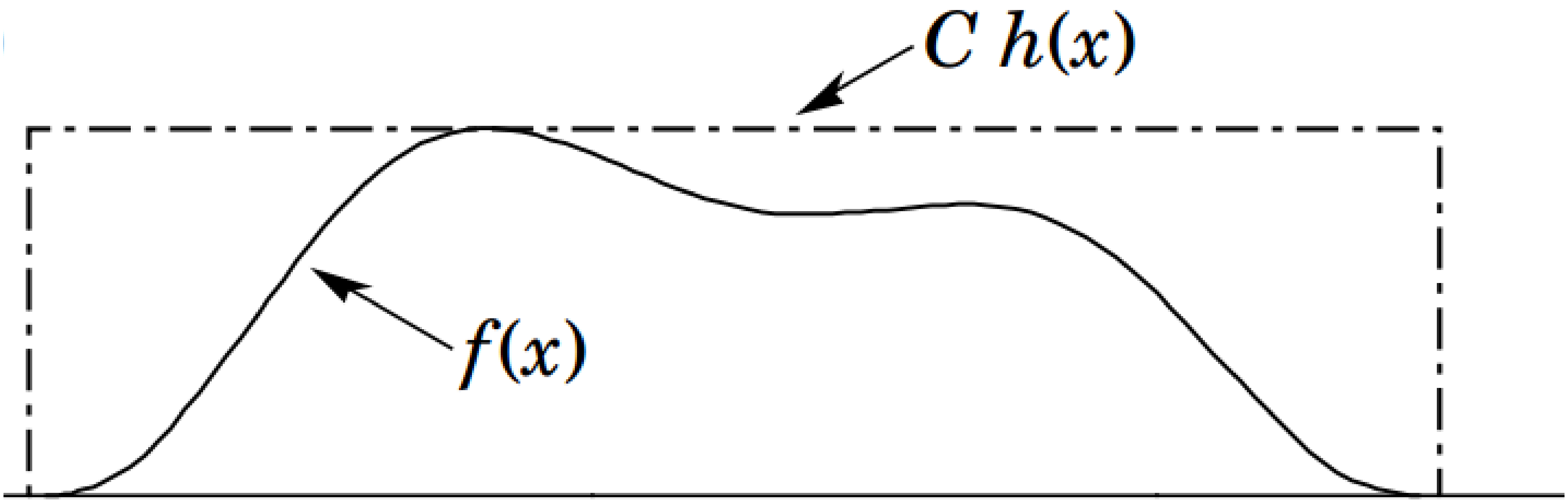
$$\frac{\sigma_\pi}{\pi} = \sqrt{\frac{1}{N_{\text{in}}} - \frac{1}{N_{\text{total}}}}$$

- For 100 trials, $\sigma_\pi/\pi = 5.0\%$ (e.g. 3.20 ± 0.16)
- For 1000 trials, $\sigma_\pi/\pi = 1.7\%$ (e.g. 3.14 ± 0.05)
- For 10,000 trials: $\sigma_\pi/\pi = 0.5\%$ (e.g. 3.134 ± 0.016)

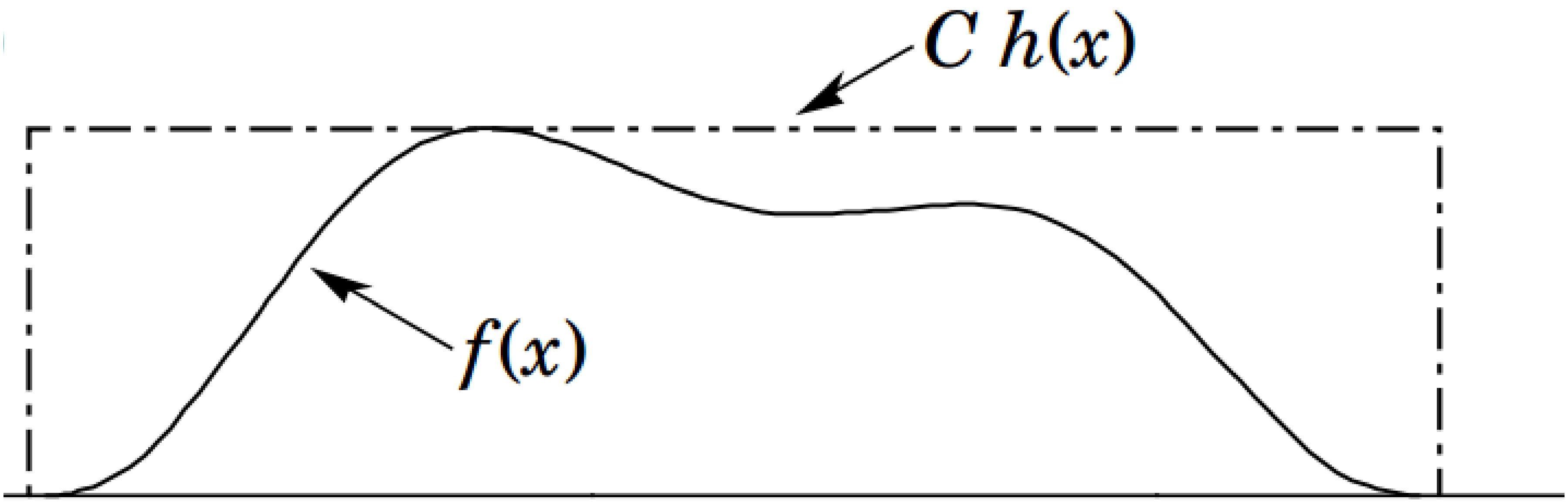
Note that uncertainty scales only as $1/\sqrt{N_{\text{total}}}$

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!



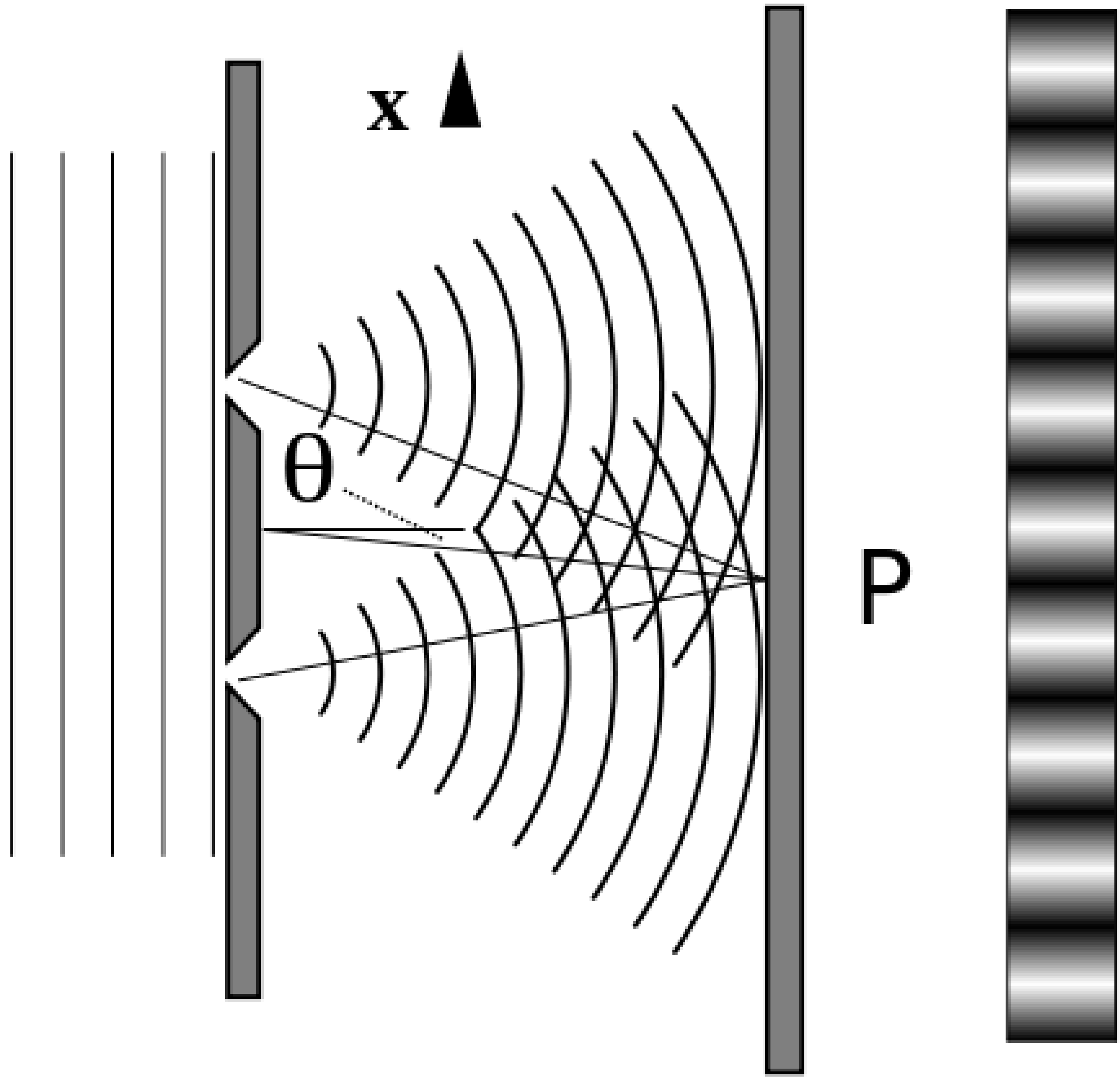
$$\frac{N_{\text{hits}}}{N_{\text{total}}} = \frac{I\left(f\left(x\right)\right)}{A\left(h\left(x\right)\right)}$$

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


x
θ
P

Young's Double-Slit Experiment Simulation

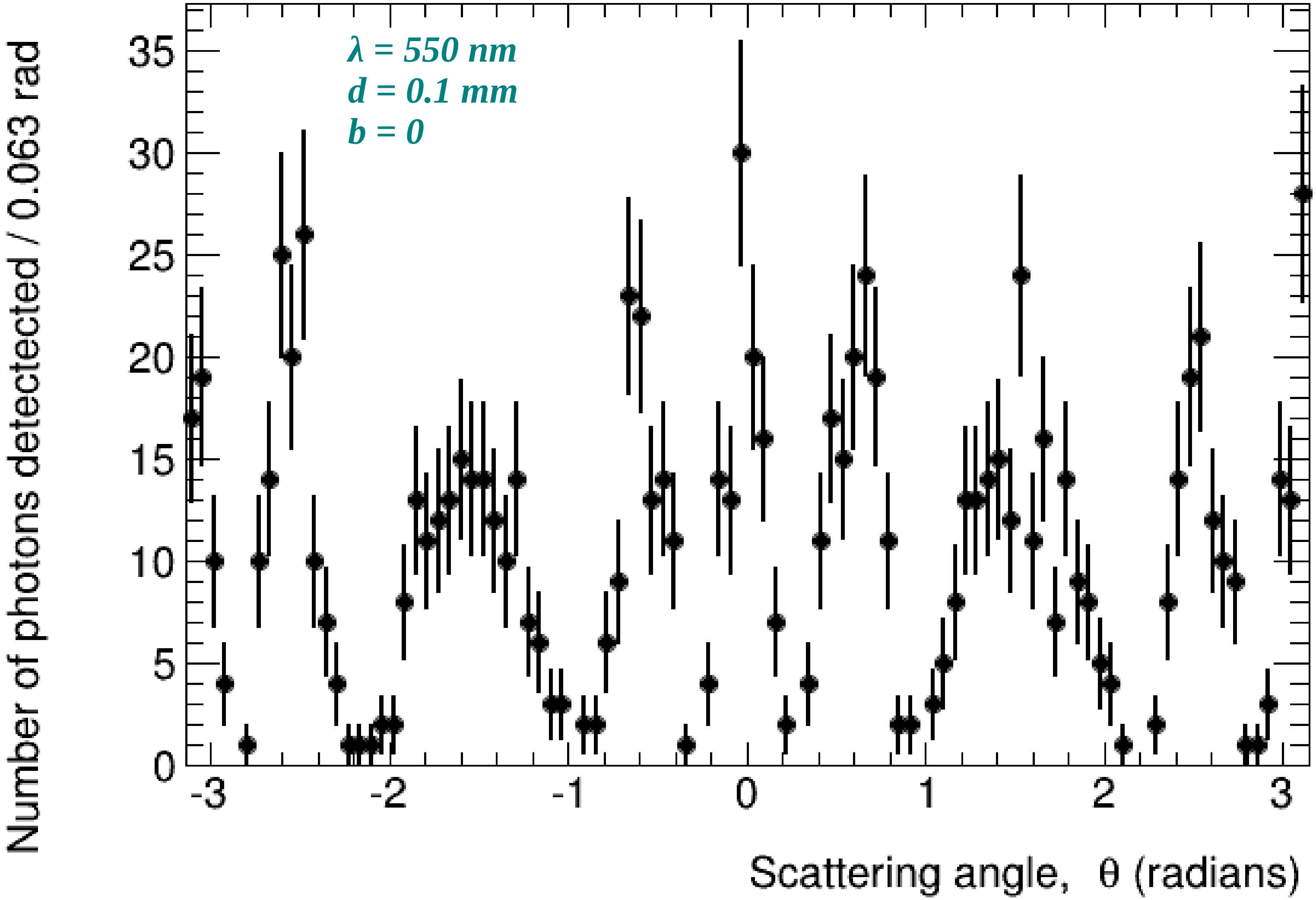
- Consider slits of width, b, separated by a distance, d.
- Given the function that describes the probability of finding a photon at a given angle:

$$I(\theta) \propto \cos^2\left[\frac{\pi d \sin(\theta)}{\lambda}\right] \mathrm{sinc}^2\left[\frac{\pi b \sin(\theta)}{\lambda}\right]$$

$$\mathrm{sinc(x)} = \begin{cases} \sin(x)/x \quad (x \neq 0) \\ 1 \quad (x = 0) \end{cases}$$

Next Steps

- Need the max. value of I(θ)
  - occurs at θ = 0
- Use that to compute the height of the box; the width of the box is 2π (ranging from -π to +π)
- "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/
- Octave: (open-source, free)
  http://www.gnu.org/software/octave/
- Mathematica: (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