

## Overview

- Error in MC
- RNG
- Errors

Condition number

HW3



$$\frac{\text{Inside}}{\text{Total}} \approx \frac{\pi}{4}$$

**Demo:** Computing  $\pi$  using Sampling

**Demo:** Errors in Sampling

## Sampling: Error

The Central Limit Theorem states that with

$$S_N := X_1 + X_2 + \cdots + X_n$$

for the  $(X_i)$  independent and identically distributed according to random variable  $X$  with variance  $\sigma^2$ , we have that

$$\frac{S_N - NE[X]}{\sqrt{\sigma^2 N}} \rightarrow \mathcal{N}(0, 1),$$

i.e. that term approaches the normal distribution. As we increase  $N$ ,  $\sigma^2$  stays fixed, so the asymptotic behavior of the error is

$$\left| \frac{1}{N} S_N - E[X] \right| = O\left(\frac{1}{\sqrt{N}}\right).$$

# Monte Carlo Methods: The Good and the Bad

What are some *advantages* of MC methods?

- easy to code
- does not suffer from "curse of dim." (No <sup>cost</sup> scaling)

What are some *disadvantages* of MC methods?

- Sad error scaling ( $1/\sqrt{N}$ )  
    ↑ expensive
- No immediate error estimate
- not deterministic ← (can work around by seeding)

## Computers and Random Numbers

```
int getRandomNumber()  
{  
    return 4; // chosen by fair dice roll.  
              // guaranteed to be random.  
}
```

[from xkcd]

How can a computer make random numbers?

- complicated formula ←
- hardware randomness
- order "online"

# Random Numbers: What do we want?

What properties can 'random numbers' have?

- ▶ Have a specific distribution  
(e.g. 'uniform'—each value in given interval is equally likely)
- ▶ Real-valued/integer-valued
- ▶ Repeatable (i.e. you may ask to exactly reproduce a sequence)
- ▶ Unpredictable
  - ▶ V1: 'I have no idea what it's going to do next.'
  - ▶ V2: No amount of engineering effort can get me the next number. *← esp. important for cryptography*
- ▶ Uncorrelated with later parts of the sequence  
(Weaker: Doesn't repeat after a short time)
- ▶ Usable on parallel computers



# What's a Pseudorandom Number?

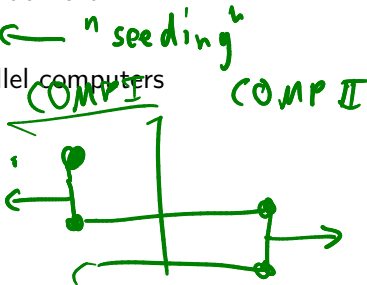
Actual randomness seems like a lot of work. How about 'pseudo-random numbers?'

**Idea:** Maintain some 'state'. Every time someone asks for a number:

$$\text{random\_number, new\_state} = f(\text{state})$$

Satisfy:

- ✓ Distribution
- ✓ 'I have no idea what it's going to do next.'
- ✓ Repeatable (just save the state) ← "seeding"
- ▶ Typically *not* easy to use on parallel computers





## **Demo:** Playing around with Random Number Generators

## Some Pseudorandom Number Generators

Lots of variants of this idea:

- ▶ LC: 'Linear congruential' generators

$$(ax + b) \% c$$

- ▶ MT: 'Mersenne twister'

- ▶ almost all random number generators you're likely to find are based on these—Python's `random` module, `numpy.random`, C's `rand()`, C's `rand48()`.

# Counter-Based Random Number Generation (CBRNG)

What's a CBRNG?

→ (2011)

**Idea:** Cryptography has way stronger requirements than RNGs.  
And the output *must* 'look random'.

(Advanced Encryption Standard) AES algorithm:  
128 encrypted bits = AES (128-bit plaintext, 128 bit key)

We can treat the encrypted bits as random:

128 random bits = AES (128-bit counter, arbitrary 128 bit key)

- ▶ Just use 1, 2, 3, 4, 5, ... as the counter.
- ▶ No quality requirements on counter or key to obtain high-quality random numbers
- ▶ Very easy to use on parallel computers
- ▶ Often accelerated by hardware, faster than the competition

AES

**Demo:** Counter-Based Random Number Generation

# Outline

Python, Numpy, and Matplotlib  
Making Models with Polynomials  
Making Models with Monte Carlo

## → Error, Accuracy and Convergence

Floating Point

Modeling the World with Arrays

The World in a Vector

What can Matrices Do?

Graphs

Sparsity

Norms and Errors

The 'Undo' Button for Linear Operations: LU

LU: Applications

Linear Algebra Applications

Interpolation

Repeating Linear Operations:  
Eigenvalues and Steady States

Eigenvalues: Applications

Approximate Undo: SVD and Least Squares

SVD: Applications

Solving Funny-Shaped Linear Systems

Data Fitting

Norms and Condition

Numbers

Low-Rank Approximation

Iteration and Convergence

Solving One Equation

Solving Many Equations

Finding the Best: Optimization in 1D

Optimization in  $n$  Dimensions

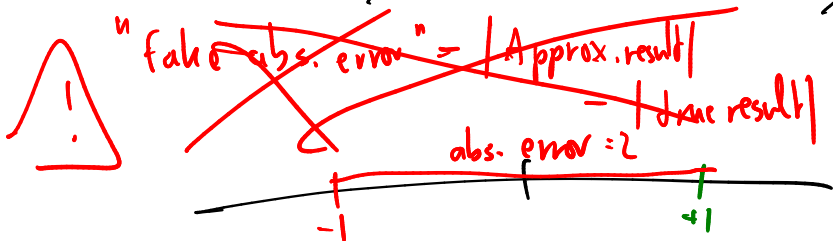
# Error in Numerical Methods

Every result we compute in Numerical Methods is inaccurate. What is our model of that error?

$$\text{Approx result} = \text{true result} + \text{error}$$

Suppose the true answer to a given problem is  $x_0$ , and the computed answer is  $\tilde{x}$ . What is the absolute error?

$$\text{abs. error} = | \text{Approx. result} - \text{true result} |$$



Scen. ①

$$\text{true} = 10,000,000$$

$$\text{abs. err} = 0.2$$

↳ pretty good

$$\text{rel. error} = \frac{0.2}{10,000,000} \approx \text{small}$$

Scen. ②

$$\text{true} = 0.00001$$

$$\text{abs. err} = 0.2$$

↳ pretty bad

$$\text{rel. error} \approx \frac{0.2}{0.0001}$$

$\approx$  big

true value = 10  
abs. error = 0.1

true value = 10,000

# Relative Error

true  
error

$x_0$   
 $\Delta x$

approx  $\tilde{x}$

What is the relative error?

$$\frac{|\Delta x|}{|x_0|} = \frac{|x_0 - \tilde{x}|}{|x_0|} = \frac{\text{Abs. error}}{|\text{true result}|}$$

Why introduce relative error?

to compare result quality

What is meant by 'the result has 5 accurate digits'?

true  $\rightarrow$  12.345000

approx  $\rightarrow$  12.34599957

$$\text{rel error} = 8.099 \cdot 10^{-5} \approx 0.0001 \underline{?}$$




$n$  has  $X$  accurate digits<sup>2</sup>

$\approx$

$$^n \text{ rel. error} \leq 10^{-X/n}$$

## Measuring Error

Why is  $|\tilde{x}| - |x_0|$  a **bad** measure of the error?



If  $\tilde{x}$  and  $x_0$  are vectors, how do we measure the error?

## Sources of Error

What are the main sources of error in numerical computation?

## Digits and Rounding

Establish a relationship between '*accurate digits*' and rounding error.

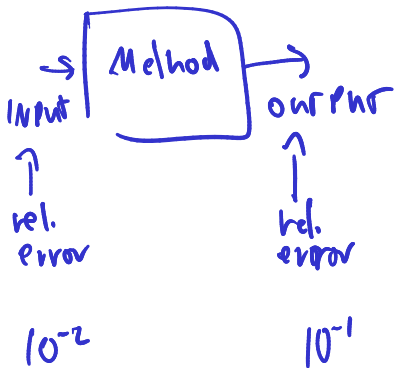
# Condition Numbers

Methods  $f$  take input  $x$  and produce output  $y = f(x)$ .

Input has (relative) error  $|\Delta x| / |x|$ .

Output has (relative) error  $|\Delta y| / |y|$ .

**Q:** Did the method make the relative error bigger? If so, by how much?



(for all inputs)  
upper bound of

$$\text{Condition number} = \frac{\text{rel. error in output}}{\text{rel. error in input}}$$

← cond. number = 10

## $n$ th-Order Accuracy

Often, *truncation error* is controlled by a parameter  $h$ .

Examples:

- ▶ distance from expansion center in Taylor expansions
- ▶ length of the interval in interpolation

A numerical method is called ' *$n$ th-order accurate*' if its truncation error  $E(h)$  obeys

$$E(h) = O(h^n).$$