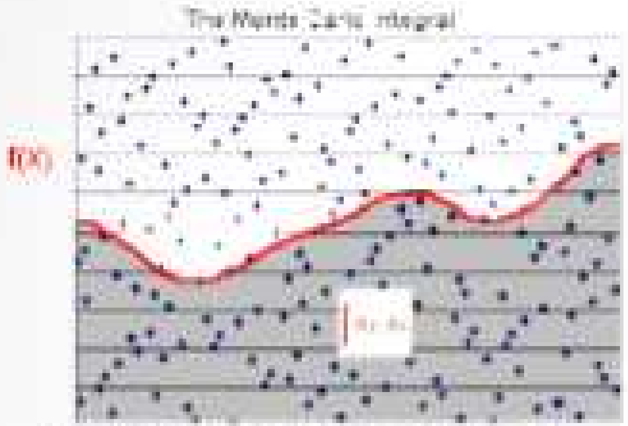
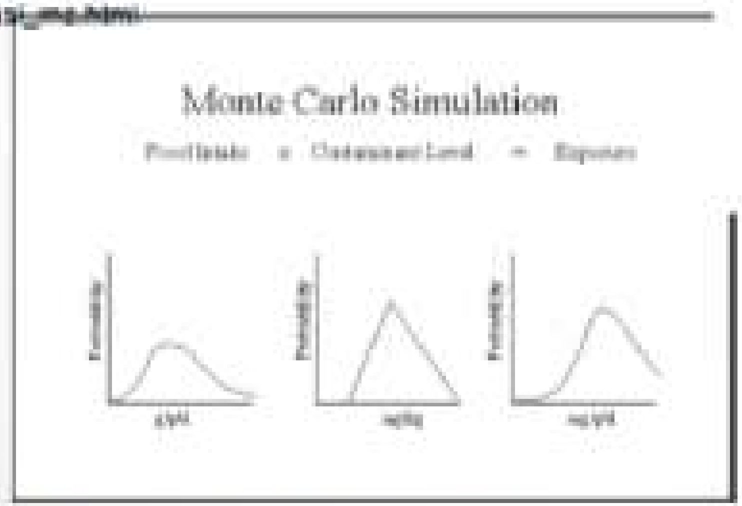
Simulations in Python

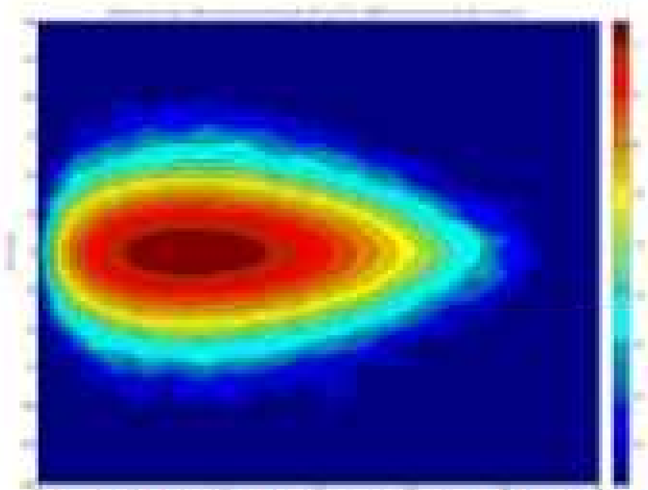
Some Applications



http://marcoagd.usuarios.rdc.puc-rio.br/quasi_mc.html



US Food and Drug Administration



Dr.-Ing. Mathias Westhäuser. Statistical Analysis of Fiber Optical Systems using Multicanonical Monte Carlo Methods (http://www.hft.e-technik.tu-dortmund.de/forschung/projekt.php?id=18&lang=en)

What is a Monte Carlo method?

- An algorithm that uses a source of (pseudo) random numbers
- Repeats an "experiment" many times and calculates a statistic, often an average
- Estimates a value (often a probability)
- ... usually a value that is hard or *impossible* to calculate analytically

Simple example: dice statistics

- We can **analyze** throwing a pair of dice and get the following probabilities for the sum of the two dice:

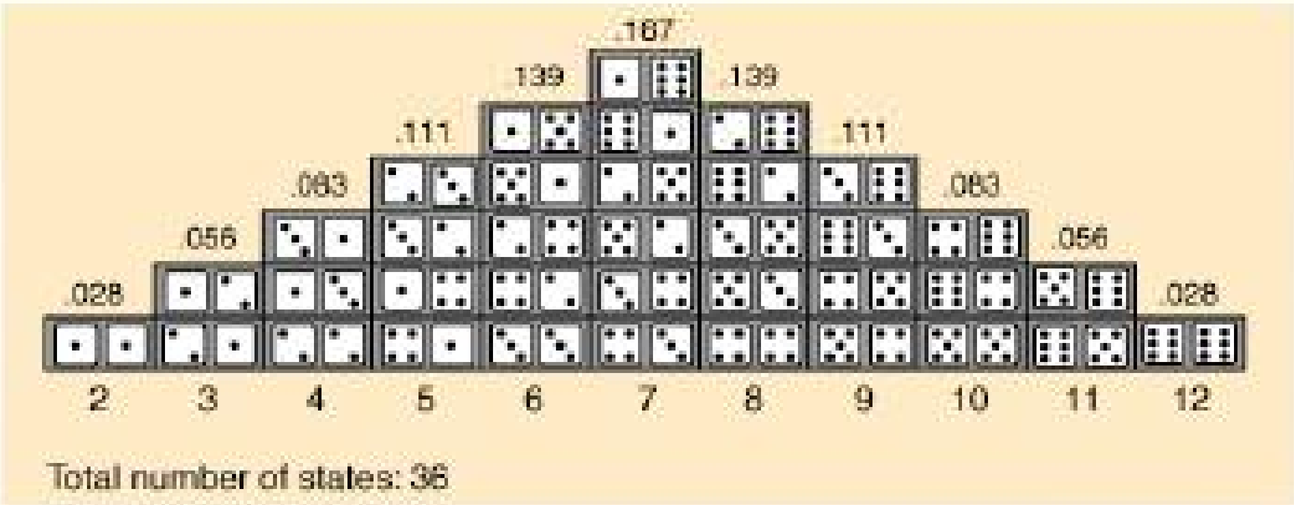
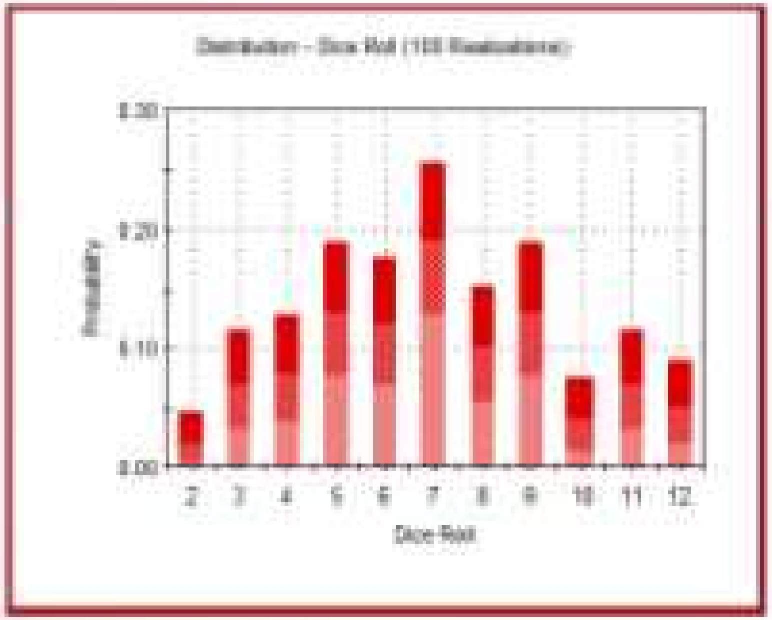


Image source:
[illegible] via
http://www.goldsim.com/Web/Introduction/Probabilistic/MonteCarlo/

Simple example: dice statistics

- … **or** we can throw a pair of dice 100 times and record what happens, or 10000 times for a more accurate estimate.



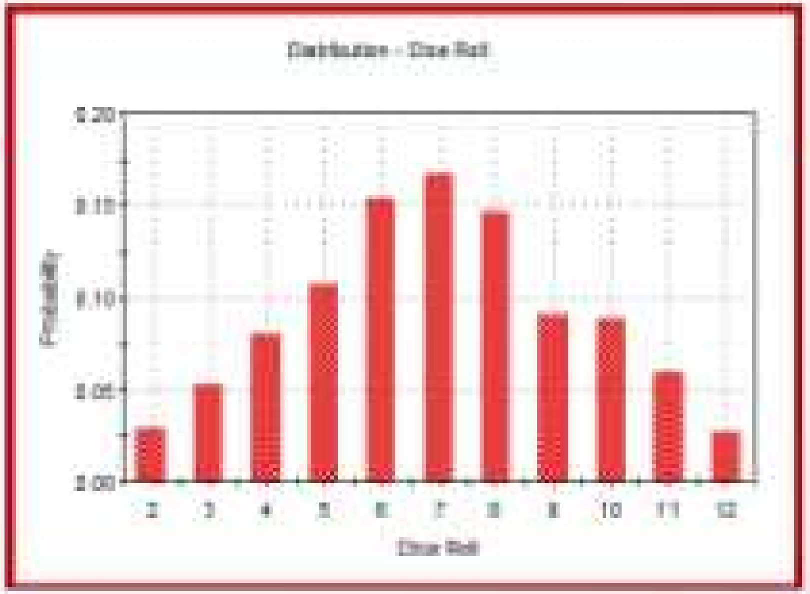


Image source:
[illegible] via
http://www.goldsim.com/Web/Introduction/Probabilistic/MonteCarlo/

A game of dice

```
def dice_game() :
    strikes = 0
    winnings = 0
    while strikes < 3 : # 3 strikes and you're out
        die1 = roll() # a random number 1…6
        die2 = roll()
        if die1 == die2 :
            strikes = strikes + 1
        else :
            winnings = winnings + die1 + die2
    return winnings   # in cents
```

The Hungry Dice Player

- In our simple game of dice:
  Can I expect to make enough money playing it to buy lunch?
- That is, what is the expected (average) value won in the game?
- We could figure it out by applying laws of probability
- …or use a Monte Carlo method

Monte Carlo method for the hungry dice player

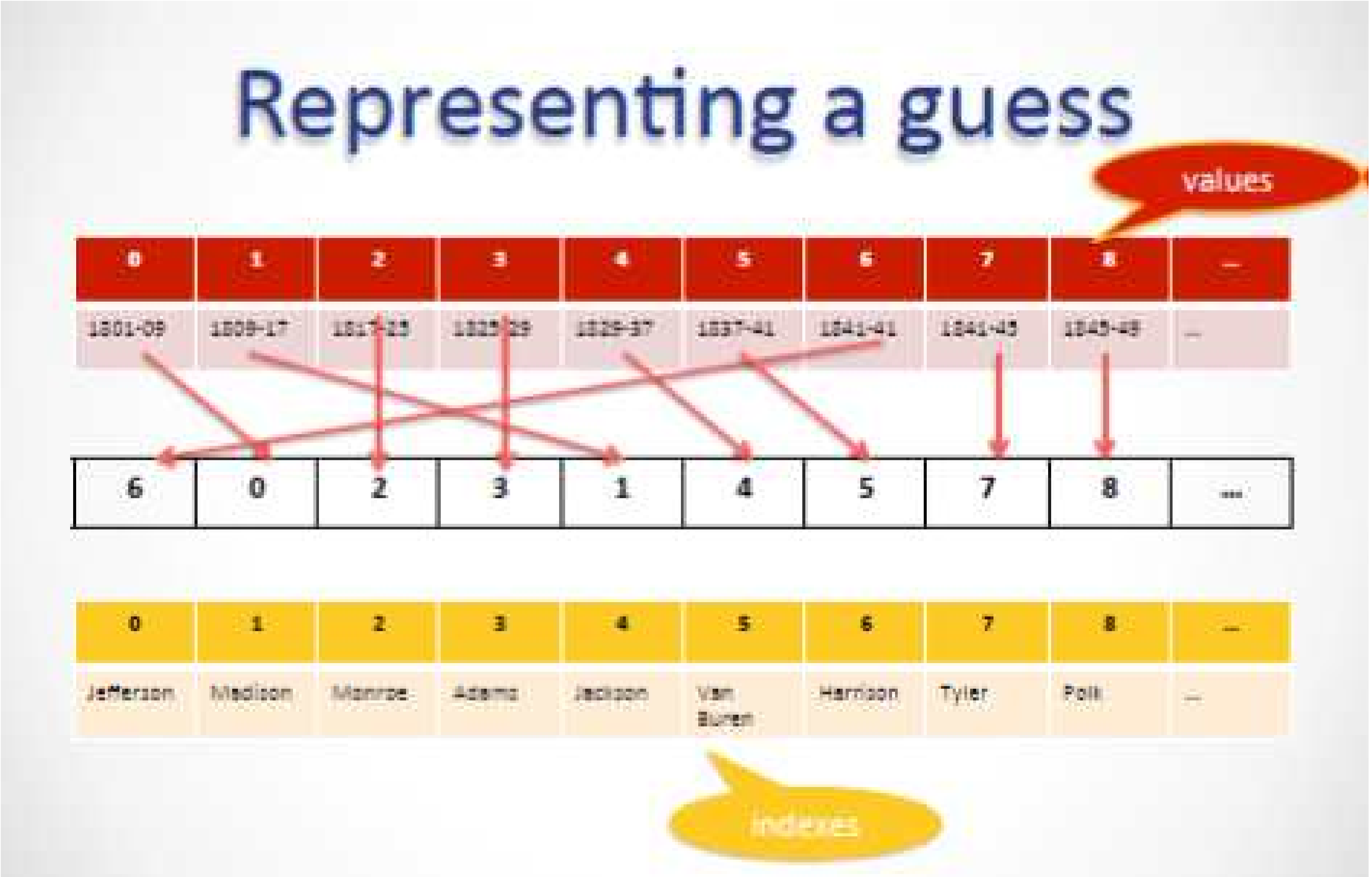
```
def average_winnings(runs) :
    # runs is the number of experiments to run
    total = 0
    for n in range(runs) :
        total = total + dice_game()
    return total/runs
>>> [round(average_winnings(10),2) for i in range(5)]
[85.8, 94.8, 120.7, 123.3, 90.0]
>>> [round(average_winnings(100),2) for i in range(5)]
[105.97, 102.95, 107.74, 134.4, 114.54]
>>> [round(average_winnings(1000),2) for i in range(5)]
[106.84, 107.11, 105.59, 104.28, 106.41]
>>> [round(average_winnings(10000),2) for i in range(5)]
[104.94, 105.71, 105.81, 105.74, 104.62]
```

The Clueless Student

A clueless student faced a pop quiz: a list of the 24 Presidents of the 19th century and another list of their terms in office, but scrambled. The object was to match the President with the term. If the student guesses a random one-to-one matching, how many matches will be right out of the 24, on average?

The quiz

1. Monroe	a. 1801-1809
2. Jackson	b. 1869-1877
3. Arthur	c. 1885-1889
4. Madison	d. 1850-1853
5. Cleveland	e. 1889-1893
6. Jefferson	f. 1845-1849
7. Lincoln	g. 1837-1841
8. Van Buren	h. 1853-1857
9. Adams	i. 1809-1817
etc.	etc.


Representing a guess
values
0
1
2
3
4
5
6
7
8
…
1801-09
1809-17
1817-25
1825-29
1829-37
1837-41
1841-41
1841-45
1845-49
…
6
0
2
3
1
4
5
7
8
…
0
1
2
3
4
5
6
7
8
…
Jefferson
Madison
Monroe
Adams
Jackson
Van Buren
Harrison
Tyler
Polk
…
indexes

Representing a guess

- Representing a guess – examples:
  [0, 1, 2, 3, 4, 5, ..., 23] represents a completely correct guess
  [1, 0, 2, 3, 4, 5, ..., 23] represents a guess that is correct except that it gets the first two presidents wrong.
  - A guess is just a permutation (shuffling) of the numbers 0 ... 23.
- Let's define a *match* in a guess to be any number *k* that occurs in position *k*. (*E.g.,* 0 in position 0, 10 in position 10)

- With this representation, our question becomes: *if I pick a random shuffling of the numbers 0...23, how many (on average) matches occur?*

Randomly permuting a list

To get a random shuffling of the numbers 0 to 23 we use the `shuffle` function from module `random`:

```
>>> nums = list(range(10))
>>> nums
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> shuffle(nums)
>>> nums
[4, 5, 3, 2, 0, 9, 6, 1, 8, 7]
>>> shuffle(nums)
>>> nums
[3, 6, 1, 4, 5, 8, 2, 9, 0, 7]
```

We will solve a more general problem

Algorithm

- Input: *pairs* (number of things to be matched), *samples* (number of samples to test)
- Output: average number of correct matches per sample
- Method

1. Set *num_correct* = 0
2. Do the following *samples* times:
   a. Set *matching* to a random permutation of the numbers 0...*pairs*-1
   b. For *i* in 0...*pairs*, if *matching*[*i*] = *i* add one to *num_correct*
3. The result is *num_correct* / *samples*

Code for the clueless student

```
from random import shuffle
# pairs is the number of pairs to be guessed
# samples is the number of samples to take
def student(pairs, samples) :
    num_correct = 0
    matching = list(range(pairs))
    for i in range(samples) :
        shuffle(matching)         # generate a guess
        for j in range(pairs) :
            if matching[j] == j :
                num_correct = num_correct + 1
    return num_correct / samples
```

Running the code

- The mathematical analysis says the expected value is exactly 1 (**no matter how many matches are to be guessed**).

```
>>> student(24, 10000)
0.9924
>>> student(24, 10000)
1.0071
>>> student(10, 10000)
1.0224
>>> student(10, 10000)
0.9999
>>> student(5, 10000)
1.0039
>>> student(5, 10000)
0.9826
```

More samples – smaller error

```
>>> 1 - student(5, 1000)
0.03600000000000003
>>> 1 - student(5, 10000)
0.005900000000000016
>>> 1 - student(5, 100000)
0.0014100000000000223
>>> 1 - student(5, 1000000)
-0.0006679999999998909
```

The Umbrella Quandary

- Mr. X walks between home and work every day
- He likes to keep an umbrella at each location
- But he always forgets to carry one if it's not raining
- If the probability of rain is p, how many trips can he expect to make before he gets caught in the rain? (Assuming that if it's not raining when he starts a trip, it doesn't rain during the trip.)

The trivial cases

- What if it always rains?
- What if it never rains (ok, that was too easy)
- So we only need to think about a probability of rain greater than zero and less than one

Solving the umbrella quandary

- Analysis of the problem can be done with Markov chains
- But we're just humble programmers, we'll simulate and measure

Simulating an event with a given probability

- In contrast to the clueless student problem we're given a probability of an event
- We want to simulate that the event happens, with the given probability *p* (where *p* is a number between 0 and 1)
- Technique: get a random float between 0 and 1; if it's less than *p* simulate that the event happened

```
if random() < p :
    raining = True
```

Representing home, work, and umbrellas

- Use 0 for home, 1 for work, and a two-element list for the number of umbrellas at each location
- How should we initialize?
- ```
  location = 0
  umbrellas = [1, 1]
  ```

Figuring out when to stop

- We want to count the number of trips before Mr. X gets wet, so we want to keep simulating trips until he does.
- To keep track:
- ```
  wet = False
  trips = 0
  while (not wet) :
    …
  ```

Changing locations

- Mr. X walks between home (0) and work (1)
  - To keep track of where he is:
    ```
    location = 0 # start at home
    ```
  - To move to the other location:
    ```
    location = 1 - location
    ```
  - To find how many umbrellas at current location:
    ```
    umbrellas[location]
    ```

Putting it together

```
from random import random

def umbrella(p) :              # p is the probability of rain
    wet = False
    trips = 0
    location = 0
    umbrellas = [1, 1]   # index 0 stands for home, 1 stands for work
    while (not wet) :
        if random() < p :   # it's raining
            if umbrellas[location] == 0 :  # no umbrella
                wet = True
            else :
                trips = trips + 1
                umbrellas[location] -= 1           # take an umbrella
                location = 1 - location              # switch locations
                umbrellas[location] += 1           # put umbrella
        else :    # it's not raining, leave umbrellas where they are
            trips = trips + 1
            location = 1 - location
    return trips
```

Running simulations

```
>>> umbrella(.5)
22
>>> umbrella(.5)
4
>>> umbrella(.5)
13
>>> umbrella(.5)
2
>>> umbrella(.5)
2
```

Great, but we want averages

- One experiment doesn't tell us much—we want to know, **on average**, if the probability of rain is *p*, how many trips can Mr. X make without getting wet?
- We add code to run `umbrella(p)` 10,000 times for different probabilities of rain, from $p = .01$ to .99 in increments of .01
- We accumulate the results in a list that will show us how the average number of trips is related to the probability of rain.

Running the experiments

```
# 10,000 experiments for each probability from .01
to .99
# Accumulate averages in a list
def test() :
    results = [None]*99
    p = .01
    for i in range(99) :
        trips = 0
        for k in range(10000) :
            trips = trips + umbrellas(p)
        results[i] = trips/10000
        p = p + .01
    return results
```

Crude plot of results