**Administrative**

Quiz 1: Wed, Nov 12th 7:30-9:00
Quiz 1 review session: Thur, Nov 6th 7:30
Homework 2 is due Wed, Nov 5th

**Probability (10 min.)**

- Events can be independent (e.g. dice rolls) or dependent (e.g. picking socks 1 at a time from a drawer)
- \( P(A \text{ and } B) = P(A) \times P(B) \) if A and B are independent
- \( P(A) + P(\text{not } A) = 1 \)
- **Problems**
  - What is the probability of rolling a 6 five times in a row? \( 1/6^5 \)
  - What is the probability of getting 6 at least once in 10 rolls?
    - \( P(\text{at least one 6}) = P(6 \text{ and } 9 \text{ 1's}) + P(6 \text{ and } 8 \text{ 1's and } 1 \text{ 2}) + \ldots + P(6 \text{ and } 9 \text{ 2's}) \ldots \text{ too much work!!} \)
    - \( P(\text{at least one 6}) = 1 - P(\text{no 6's}) = 1 - (5/6)^{10} \)
  - What is the probability of getting 6 at least twice in 10 rolls?
    - \( P(\text{at least two 6's}) = 1 - P(\text{no 6's}) - P(\text{one 6}) = 1 - (5/6)^{10} - 10 \times (1/6) \times (5/6)^9 \)
- **Law of large numbers.** doesn’t necessarily talk about the distribution of the outcomes, only the average
  - **Definition:** as the number of trials approaches infinity, the average value from the trials approaches the expected value.
- Bernoulli trials (trials with only a “success” or “failure.” we talk a lot about when trials are independent)

**Intro to Inferential Statistics (20 min.)**

As you have seen in lecture and undoubtedly in other parts of the world, one of the most useful applications of computational technology is running simulations. Simulations are useful for estimating values that are too tedious or too hard to solve for analytically.

The use of simulations for estimating an unknown value is based on inferential statistics. In short, inferential statistics is guided by the principle that a random sample from a population tends to exhibit the same properties as the entire true population. For example, if you want to survey MIT students about their favorite course (even though you known the answer is 6.0002), you would give the survey to a random sample of students, and you *may* be able to assume that those results are representative of ALL MIT students. If you want to find out the average value of a dice roll, you can roll a die some number of times and record the results. We’ll talk about how many samples we need later.

**Terminology review (3 min.)**
• mean
  ○ true mean $\mu$
    ■ In a lot of real-world problems, we have no way of knowing the true mean. We can only try to estimate its value using random sampling.
  ○ sample mean $\bar{x}$
    ■ The mean calculated from our sample of the population (e.g. the mean of 100 dice rolls)

• standard deviation (true and sample)
  ○ How close the values in the population are to the mean
    ■ Low std implies that there are a lot of values that are close to the mean
    ■ High std implies that there are a lot of values that are far from the mean

Distributions (5 min.)
We can often describe the distribution of values in a set of data or a population.
• Uniform distribution
  ○ every possible value is equally probable
  ○ e.g. value of rolling a single die (⅙ probability for each value)
  ○ e.g. coin toss
• Normal distribution
  ○ Commonly known as a bell curve or a Gaussian distribution
  ○ Any normal distribution can be completely specified by 2 parameters: mean and standard deviation; knowing these two values is equivalent to knowing the entire distribution
  ○ Peak at mean, falls off symmetrically around mean and asymptotically approaches 0
  ○ Why do we care so much about normal distributions?
    ■ Many naturally occurring distributions are close to normal
    ■ They can be used to produce confidence intervals
    ■ Sample means often fall in a normal distribution, under certain conditions

Example 1: Suppose we run 10000 trials of 25 dice rolls each. In each trial, we record the mean dice roll value (over all 25 dice). These 10000 sample means will be distributed normally.

    import pylab
    import random

    def mean_dice_rolls(num_trials, num_rolls):
        mean_per_trial = pylab.zeros(num_trials)
        dice_values = [1,2,3,4,5,6]

        for i in range(num_trials):
            total_so_far = 0
            for j in range(num_rolls):
total_so_far += random.choice(dice_values)
mean_per_trial[i] = total_so_far/float(num_rolls)

print "Gaussian mean:", pylab.mean(mean_per_trial), ", Gaussian std:", pylab.std(mean_per_trial)
pylab.figure(1)
pylab.hist(mean_per_trial)
pylab.show()

mean_dice_rolls(10000, 25)

Your output should look like this:

Confidence intervals (5 min.)

- When we try to estimate an unknown value using inferential statistics, we often like to describe our confidence in the estimate that we come up with.
- Say we take a sample of 20 6.0002 students and look at their mean midterm score (Q: what is the proper statistical term for this value?). We can say that have a 95% confidence level that the true midterm average falls within +/-10% of the sample mean.
- Almost always, increasing the confidence level will widen the confidence interval
• If your confidence level is 95%, you say that there is a 95% probability that the calculated confidence interval from a given sample will encompass the true value of a parameter.
• Think of this as taking 100 samples from some true population. 95 of the confidence intervals calculated from these samples encompass the true mean. This does not mean that the true value of the parameter falls within the interval with a 95% probability - the true value actually falls into any given interval with a probability of 0 or 1.
• Calculating confidence intervals requires the distribution of sample means to be normal. If this is true, there is a very easy way of estimating confidence levels given the mean and std:
  ▪ 68% of the data will fall within 1 standard deviation (i.e. +/- σ) of the mean
  ▪ 95% of the data will fall within 2 stds of the mean
  ▪ 99.7% of the data will fall within 3 stds of the mean
• For example, suppose we find that the mean 6.0002 midterm score is 50% with a standard deviation of 5%. If we got more random samples of 20 students each, we can predict that 95% of the time, the sample mean will be 50% +/- 10%.

Monte Carlo Simulations (30 min.)
• A technique that simulates repeated random sampling to obtain a numerical solution to a problem.

Example 1: Finding socks in a dark room
Suppose it’s 6am and you’re trying to find a pair of socks in a dark room. If you own 5 differently colored pairs of socks, what is the probability that you’re going to get a match by drawing 1 sock at a time?

```python
import numpy as np
import random

def socksSim(num_trials, num_colors):
    num_matches = 0

    for trial in range(num_trials):
        # Create a new drawer of socks
        socks = []
        for sock_color in range(num_colors):
            socks.extend([[sock_color, sock_color]])

        # Draw sock 1
```
sock1 = socks[random.randint(0, len(socks)-1)]
socks.remove(sock1)

# Draw sock 2
sock2 = socks[random.randint(0, len(socks)-1)]

if sock1 == sock2:
    num_matches += 1

return num_matches/float(num_trials)

print "Percentage of matching pairs:", socksSim(10000, 5)

Actually, this problem is easy enough to solve analytically. Is our estimated solution right?
P(any sock) * P(matching sock) = 1 * 1/9 = 0.111111...

Example 2: Pill Simulation
100 people become stranded on an island and end up getting sick with Guttag-itis, but there’s no doctor among them, so nobody knows how sick each of them are, or how many pills they need to survive. However, one of them remembers reading that the disease will require at most 5 pills to be completely cured in the worst case of the disease. Less severe cases could require anywhere between 1 and 4 pills to be cured, but need diagnosis by a doctor.

The ship they crashed from only has 320 pills, so if everyone is severely sick, not all of them can live. However, if everyone is only a little sick, they can all survive. It turns out they’re all selfish, though, so everyone wants to take all the pills they can. They agree on the following scheme:

- Everyone gets 5 tickets, since nobody will need more than 5 pills
- The total 500 tickets will be put into a hat, and 320 tickets will be pulled at random

Because they’re all selfish, they’ll immediately take any pills they get. On average, how many of the people will survive the outbreak of Guttag-itis?

Example 3: Constructing a fair Dartboard
You’re trying to build a new dartboard, and you’ve already assigned each section a set number of points. You’re also pretty happy with the overall size of the dartboard, but you’re not sure exactly how large to make each section. You want the average score to be between 48 and 52 (ideally 50), and so you decide to run simulations to see what the radii of your sections should be.
Dartboard Pseudocode:

While True:
1. Randomly generate section sizes
2. Run num_trials trials:
   2.1. Throw num_darts darts at the dartboard
3. If average score over all trials is not between 48 and 52, go to step 1.
4. Otherwise, return section sizes

import math
import random

class Dartboard:
    def __init__(self, radii, values):
        self.radii = radii
        self.values = values
        self.score = 0

    def put_dart(self, x, y):
        radius = math.sqrt(x*x+y*y)
        section_index = len(self.radii) - 1

        while radius < self.radii[section_index] and section_index != 0:
            section_index -= 1
            self.score += self.values[section_index]

    def get_score(self):
        return self.score

    def throw_darts_and_calculate_score(board, num_darts):
        for i in xrange(num_darts):
            # simulate throwing the dart at a random spot on the board
            x = random.random() * 10
            y = random.random() * 10
            board.put_dart(x, y)
        return board.get_score()

def find_best_board(num_trials=500, num_darts=5):
    # all dartboards have a radius of 10 and have 6 sections
    section_values = [40, 20, 10, 4, 2, 1] # vary these and have some fun
while True:
    # randomly sample from the set of all possible dartboards by randomly picking 5 radii with sizes between 0-10

    radii = []

    for j in xrange(5):
        # pick a random radius between 0 and 10
        radius = round(random.random() * 10, 2)
        radii.append(radius)
    radii.append(10)
    radii.sort()

    trials = []

    # for num_trials number of trials, simulate dart-throwing and record the score
    for i in xrange(num_trials):
        board = Dartboard(radii, section_values)
        score = throw_darts_and_calculate_score(board, num_darts)
        trials.append(score)
        trials.append(score)
        if avg_score >= 48 and avg_score <= 52:
            return radii

    print "Best dart board radii: ", find_best_board()}