Enumeration

Probability is defined as the number of times an event occurs as the number of trials goes to infinity. This leads to the previous lecture's method of determining probability: Monte Carlo simulations where you run an experiment a large number of times.

While Monte-Carlo simulations give you an idea of what the probability of an event is, it does not give exact answers when running a finite number of trials. A second way of determining probabilities is to list out all possible outcomes. Assuming each outcome has the same probability, the exact probability of an outcome is

$$p(x) = \left(\frac{\text{number of ways to obtain outcome x}}{\text{total number of possible outcomes}} \right)$$

This technique is called *enumeration*.

Enumeration isn't flashy and has almost no finesse. Instead, it's a brute force way to find probabilities. In some cases, it works very well. In other cases, there are simply too many possible outcomes to list them all.

In this lecture, we'll use enumeration to solve the same problems we solved using Monte-Carlo techniques:

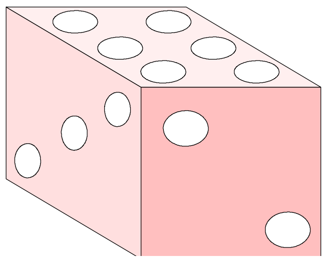
- Probability of rolling a 1 on a 6-sided die
- Probability distribution of the max(d4, d6)
- Max of (d4, d6) vs. d6
- 5-Game Match (tree diagram),
- Rolling 6-Dice
- Drawing a full-house or 3-of-a-kind in poker

Case 1: Rolling a single 6-sided die (d6)

Let's start with the simplest case: *what's the probability of rolling a 1 on a 6-sided die?* In the previous lecture, a Monte-Carlo simulation was run where a 6-sided die was rolled one-million times. The number of times a 1 came up was

- Trial 1: 166,219 times
- Trial 2: 167,090 times
- Trial 3: 166,969 times

From these results, the probability of rolling a 1 is about 0.166, give or take. That's one problem with Monte Carlo simulations: the results give you an idea about the actual probability, but not the actual probability. When we get to a student-t test later in the course, we'll be able to place a bound on what the actual probability is, but even then, it won't tell us the actual probability. With enumeration, you can find the exact probability.



Case 1: Probability of rolling a 1 on a 6-sided die

The basic assumption behind enumeration is that all outcomes have equal probability. The probability of an event is then simply the ratio of

- The number of ways outcome X can happen, and
- The total number of possible outcomes.

For a 6-sided die, the total number of possible outcomes is six:

- N = 6 {1, 2, 3, 5, 6}

There is only one outcome which is equal to one

- M = 1 {1}

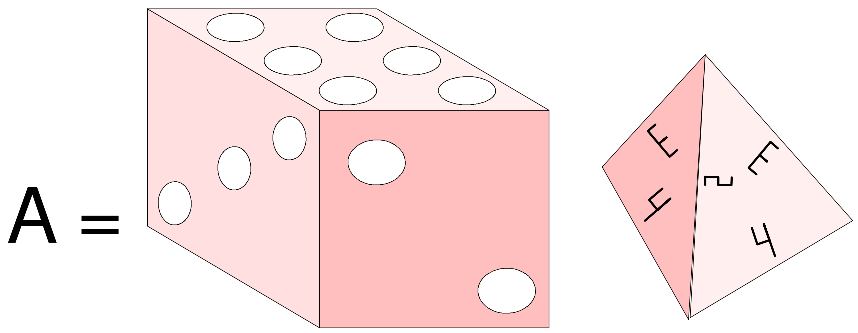
Hence, the probability of rolling a one is

$$p = \frac{M}{N} = 1/6$$

This matches up with Monte Carlo simulations - only more exact.

Case 2: A = max(d4, d6)

As a second case, consider the case where player A rolls a 4-sided die (d4) and a 6-sided die (d6) and takes the larger of the two numbers. What is the probability that A scores 1..6 points - called the *probability density function*? One way to solve this problem is to run a Monte-Carlo simulation. Another approach is to use enumeration.



Case 2: A = max(d4, d6)

With enumeration, you list out all possible outcomes for (d4, d6): there are 24 possibilities (N = 24)

(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1.6)
(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)
(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)
(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)

You then list all possible ways to get each outcome:

- 1: (1,1)
- 2: (2,1), (2,2), (2,3)

3:	(3,1), (3,2), (3,3), (2,3), (1,3)
4:	(1,4), (2,4), (3,4), (4,4), (4,3), (4,2), (4,1)
5:	(1,5), (2,5), (3,5), (4,5)
6:	(1,6), (2,6), (3,6), (4,6)

The odds of each result is then the number of successful outcomes divide by the total number of possible outcomes:

1	p = 1/24
2	p = 3/24
3	p = 5/24
4	p = 7/24
5	p = 4/24
6	p = 4/24

You can also display this as a bar chart (also termed the *probability density function*)

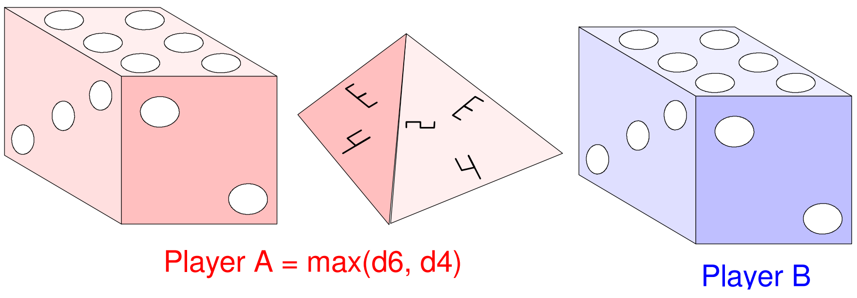
Case 3: max(d4, d6) vs. d6 (conditional probability)

A third example is this:

- Let player A roll a d4 and a d6 and take the larger of the two numbers
- Let player B roll a single 6-sided die

The winner is whoever has the highest score - B wins on ties.

What is the probability that A will win this game?



A rolls two dice and takes the maximum, B rolls a single die

Back from Monte-Carlo analysis, the probability of A winning was about 48.6% (previous lecture). With enumeration, we can find the exact probability.

Enumeration will work for this problem - but there are 144 outcomes to consider. That's a lot. One way to simplify this problem is to use conditional probabilities.

- There are 6 possible outcomes for B (1..6). Each has a probability of 1/6.

Likewise, you can split this problem into six smaller problems

- The probability that A wins given that B rolled a 1, times the probability that B rolled a 1, plus
- The probability that A wins given that B rolled a 2, times the probability that B rolled a 2, etc

Mathematically,

$$p(A) = p(A|B=1)p(B=1) + p(A|B=2)p(B=2) + p(A|B=3)p(B=3) + ...$$

Solving this step by step:

B = 1: A has to score 2 or higher to win. There are 23 ways for A to score 2 or more points, meaning

$$p(A|B=1) = \tfrac{23}{24}$$

$$p(B=1) = \tfrac{1}{6}$$

B = 2: A has to score 3 or higher to win. There are 20 ways for A to score 3 or more points, meaning

$$p(A|B=2) = \tfrac{20}{24}$$

$$p(B=2) = \tfrac{1}{6}$$

B = 3: A has to score 4 or higher to win. There are 15 ways for A to score 4 or more points, meaning

$$p(A|B=3) = \tfrac{15}{24}$$

$$p(B=3) = \tfrac{1}{6}$$

B = 4: A has to score 5 or higher to win. There are 8 ways for A to score 5 or more points, meaning

$$p(A|B=4) = \tfrac{8}{24}$$

$$p(B=4) = \tfrac{1}{6}$$

B = 5: A has to score 6 or higher to win. There are 4 ways for A to score 6 or more points, meaning

$$p(A|B=5) = \tfrac{4}{24}$$

$$p(B=5) = \tfrac{1}{6}$$

B = 6: A loses

$$p(A|B=6) = 0$$

$$p(B=6) = \tfrac{1}{6}$$

Therefore, the probability that A wins is

$$p(A) = \left(\tfrac{23}{24}\right)\left(\tfrac{1}{6}\right) + \left(\tfrac{20}{24}\right)\left(\tfrac{1}{6}\right) + \left(\tfrac{15}{24}\right)\left(\tfrac{1}{6}\right) + \left(\tfrac{8}{24}\right)\left(\tfrac{1}{6}\right) + \left(\tfrac{4}{24}\right)\left(\tfrac{1}{6}\right) + \left(\tfrac{0}{24}\right)\left(\tfrac{1}{6}\right)$$

$$p(A) = \left(\tfrac{70}{144}\right) = 0.486111$$

This matches up with the Monte-Carlo simulations - except that this answer is exact.

Case 4: 5-Game Match (Tree Analysis)

Here's another problem. Suppose you have two players playing a game. Player A has a 60% chance of winning any given game. What's the probability that player A will win a best-of-5 match?

In our last lecture, a Monte-Carlo simulation was run resulting in player A winning in one million matches:

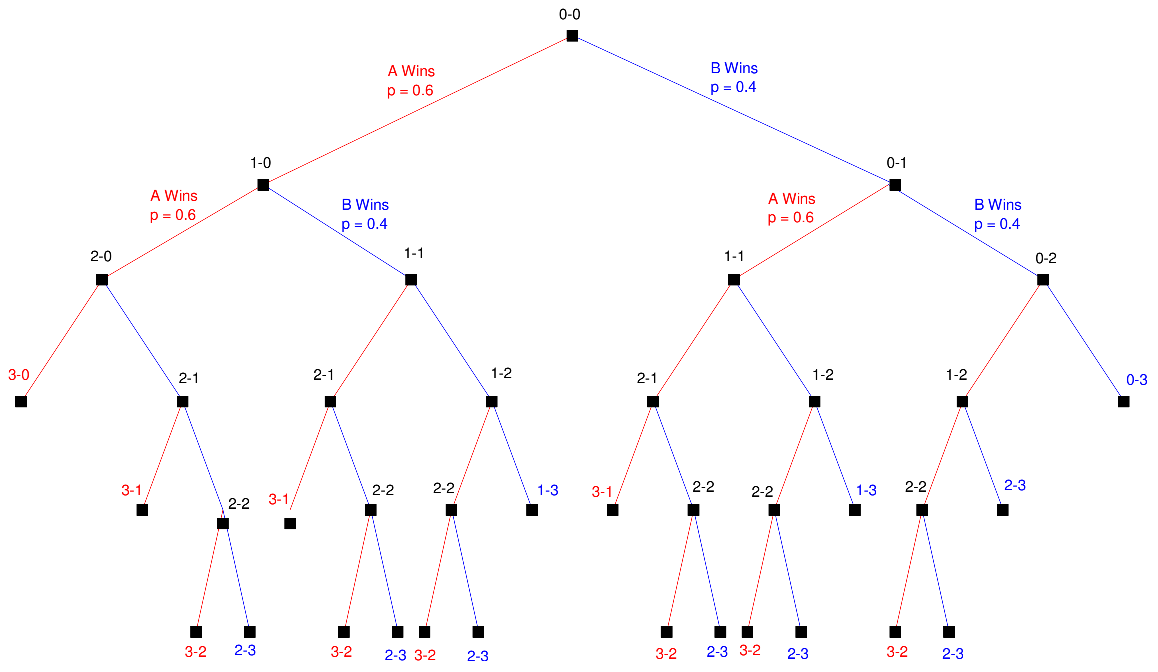
- 682,096 games,
- 683,076 games, and
- 682,342 games

From this, the odds that A will win the match is about 68.2%. Using enumeration, we can compute the exact odds.

One form of enumeration is called a *tree diagram*.

- Each game has two possible outcomes: A wins (p=0.6) or B wins (p=0.4)
- Each following game also has two possible outcomes etc.

For example, for a 5-game series, there are 38 branches that lead to a player winning 3 games:



Tree Diagram: All ways a 5-game series can be played out

Counting the number of ways A wins:

- 1 outcome ends 3-0
- 3 outcomes end in 3-1
- 6 outcomes end in 3-2

The odds of A winning are thus

$$p(A) = 1 \cdot p^3 + 3 \cdot p^3 q + 6 \cdot p^3 q^2$$

$$p(A) = 0.68256$$

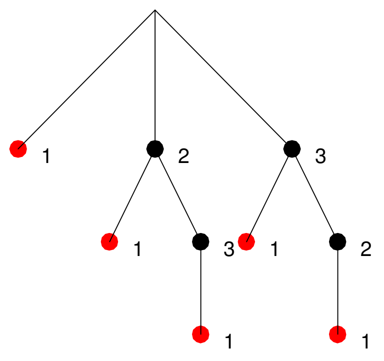
where p = 0.6 (probability of A winning) and q = 0.4 (probability of A losing). Note that this matches up with the Monte-Carlo simulation - except that you get the same answer each time you do the analysis (the answer is exact).

Sidelight: Sampling With and Without Replacement

Tree diagrams work when there the total number of ways a game can play out is finite and small. Sometimes a game can go on indefinitely. For example, consider two different problems:

- A person has three keys for a lock. Each attempt a person tries a key at random. If the lock opens, the correct key was found. If not, he/she tries again
- Case 1: The key that did not work is discarded (sample without replacement)
- Case 2: The key that did not work is kept (sample with replacement)

In the former case, there are a finite number of ways to try three keys until you find the one that works. For example, if key #1 is the correct key, there are a total of nine different paths to take:



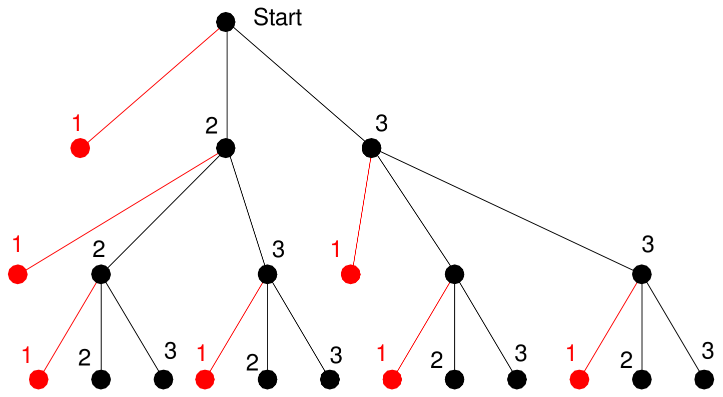
Sample Without Replacement: Max number of attempts is three

Going from the top to the bottom, the number of ways you can try your keys to open the lock are:

```
{1, 21, 231, 31, 321}
```

or there are five possible ways to find the right key.

In the latter case, you always have a 1 in 3 chance of selecting the correct key. If you're unlucky, the number of trials can go to infinity. For the latter case, enumeration doesn't work and we need a different tool. That tool is called a *Markov Chain* and will be covered later in this course.



Sample with replacement results in an infinite series. Each trial has a 1 in 3 chance of success.

Case 5: Dice Games. Enumeration with Matlab

Enumeration is essentially a brute-force solution: you go through every possible outcome and count how many of them were successes. With Matlab, you can write programs to grind out all possibilities using nested for-loops.

For example, consider the case where you roll six 6-sided dice (6d6). Determine the number of ways you can roll

- Two triples (xxx yyy) and
- One triple (xxx aab or xxx abc)

From Monte-Carlo simulations, the number of times you get each of these in one million rolls is

- Two triples: 6337 in 1,000,000 rolls
- One triple: 308,026 in 1,000,000 rolls

This places the odds at approximately 0.63% and 30.8% respectively. What are the exact odds?

First, start with the total number of ways to roll 6d6.

- The first die has six possibilities
- The second die also has six possibilities

and so on resulting in the number of possible die rolls being

$$N = 6^6 = 46,656$$

That seems like a large number, but it's no problem for Matlab.

Start by going through every possible outcome for rolling six dice using nested for-loops

```
for d1 = 1:6
   for d2 = 1:6
      for d3 = 1:6
         for d4 = 1:6
            for d5 = 1:6
               for d6 = 1:6
                  Roll = [d1, d2, d3, d4, d5, d6];
               end
            end
         end
      end
   end
end
```

Once you get your die roll (variable Roll),

- check the frequency of each number and
- sort in decreasing order

```
Roll = [d1, d2, d3, d4, d5, d6];
F = zeros(1,6);
for i=1:6
   F(i) = sum(Roll == i);
end
F = sort(F, 'descend')
```

This results in

- F(1) is the highest frequency of all the dice
- F(2) is the second highest frequency of all the dice
- F(3) is the third highest frequency of all the dice
- etc.

For example, if the roll was {2,5,2,2,5,6} then

- F(1) = 3 (there are three twos in the roll)
- F(2) = 2 (the next highest frequency is two 5's)
- F(3) = 1 (the next highest frequency is one for a single 6)

Once you know F(), you can determine the type of hand

```
if( (F(1) == 3)*(F(2)==3))
    Pair33 = Pair33 + 1;
    end
if( (F(1) == 3)*(F(2)<3))
    Pair3 = Pair3 + 1;
    end
```

By counting, you'll know the total number of hands that result in two and one triples.

The total code is as follows:

```
tic
Pair33 = 0;
Pair3 = 0;
N = 0;

for d1 = 1:6
   for d2 = 1:6
      for d3 = 1:6
         for d4 = 1:6
            for d5 = 1:6
               for d6 = 1:6
                   Dice = [d1,d2,d3,d4,d5,d6];
                   N = N + 1;
                   F = zeros(1,6);
                   for i=1:6
                       F(i) = sum(Dice == i);
                   end
                   F = sort(F, 'descend');
                   if( (F(1) == 3)*(F(2) == 3))
                       Pair33 = Pair33 + 1;
                   end
                   if( (F(1) == 3)*(F(2) < 3))
                       Pair3 = Pair3 + 1;
                   end
               end
            end
         end
      end
   end
end

% probability:
[Pair33]
[Pair3]
[N]
toc
```

The results are

```
Pair33 =         300
Pair3  =       14400
N      =       46656
Elapsed time is 1.972013 seconds.
```

There are

- 300 ways to get two triples,
- 14,000 ways to get one triple, and
- 46,656 total number of ways to roll three dice.

This took a whopping 1.97 seconds on a Windows computer running at 3.4GHz

Case 6: Enumeration with Card Games

Finally, let's use enumeration to determine the odds of getting a full-house or 3-of-a-kind in poker. The first step is to go through every combination of hands. This again consists of nested for-loops. Since you can't repeat cards (you can't have two Ace of spades in your hand), increase the index each loop.

```
for c1 = 1:52
   for c2 = c1+1:52
      for c3 = c2+1:52
         for c4 = c3+1:52
            for c5 = c4+1:52
               Hand = [c1,c2,c3,c4,c5] - 1;
            end
         end
      end
   end
end
```

Matlab code to go through every combination of hands in poker

Once you have your hand, determine the value and suit of each card:

- The value of each card is the card mod 13 plus 1 (1..13 corresponding to Ace ... King)
- The suit is the integer value of the card/13 plus 1 (1..4 corresponding to clubs, diamonds, hearts, and spades)

```
Value = mod(Hand, 13) + 1
Suit = floor(Hand/13) + 1
```

For example

- Card #32 is the 7 of hearts
- Card #33 is the 7 of hearts
- Card #1 is the Ace of clubs
- etc

Once you have your hand, determine what kind of hand this is.

- Determine the frequency of each type of card (Ace through King)
- Sort in descending order, and
- Check if this is a full-house or 3-of-a-kind

```
F = zeros(1,5);
for i=1:13
   F(i) = sum(Value == i);
end
F = sort(F, 'descend');
if( (F(1) == 3)*(F(2) == 2) ) FH = FH + 1; end
if( (F(1) == 3)*(F(2) < 2) ) Pair3 = Pair3 + 1; end
```

The whole program looks like the following:

```
tic
Pair3 = 0;
FH = 0;
N = 0;
for c1=1:52
  for c2 = c1+1:52
    clc
    disp([c1,c2])
    for c3 = c2+1:52
      for c4 = c3+1:52
        for c5 = c4+1:52
          N = N + 1;
          Hand = [c1,c2,c3,c4,c5] - 1;
          Value = mod(Hand,13) + 1;
          Suit = floor(Hand/13) + 1;

          F = zeros(1,13);
          for n=1:13
            F(n) = sum(Value == n);
          end

          F = sort(F, 'descend');
          if ((F(1) == 3)*(F(2) == 2)) FH = FH + 1; end
          if ((F(1) == 3)*(F(2) < 2)) Pair3 = Pair3 + 1; end
        end
      end
    end
  end
end
[N]
[FH]
[Pair3]
toc
```

with the results being

```
N =          2598960
FH =            3744
Pair3 =        54912
Elapsed time is 186.303521 seconds.
```

meaning

- There are 2,598,960 different poker hands
- 3744 of which are full-houses
- 54,912 are three-of-a-kind
- It took 186 seconds to go through every combination on a 3.4GHz computer.

This matches up with the Monte-Carlo results from the last lecture:

In 1,000,000 hands, you expect to get 1440.5 full-houses (Monte-Carlo gave 1369)

$$M = \left(\frac{3744}{2{,}598.960}\right) 1,000,000 = 1440.5$$

You also expect to get 21,128 three-of-a-kinds (Monte-Carlo gave 21,284)

$$M = \left(\frac{54912}{2{,}598.960}\right) 1,000,000 = 21,128.45$$

Summary

While Monte-Carlo simulations give you approximate probabilities, enumeration gives you exact probabilities. Enumeration is a brute-force approach: you list out every possible outcome. Assuming each outcome has equal probability, the odds of an event happening is simply the ratio

$$p = \left(\frac{\text{the number of successful outcomes}}{\text{the total number of outcomes}}\right)$$

For many problems, enumeration works well. For some problems, the number of outcome becomes too large to list. This can happen when

- There are an infinite number of possible outcomes (such as a win-by-3 series), or
- The time it takes to go through all such outcomes is too large

For such cases, we need a different tool.