
ECE 341: Random Processes

www.BisonAcademy.com

Instructor:	Jacob Glower
Office Location:	ECE 201A
Office Phone:	none at present
Class Hours:	Noon - 2:30pm Monday - Friday Also live streamed on Zoom
Office Hours	M-F, 7-8pm on Zoom
Text:	Bison Academy
References	Get a used text book on Statistics. They sell for \$6 on Amazon StatTrek.com is also a good reference (and free)

Catalog Description: Principles of probability. Application of probability and statistics to electrical and computer engineering problems. 3 lectures. Prereq: MATH 266. F, S

Topics Covered: Combinatorics, trees, apriori, aposteriori, conditional probabilities, pdf, cdf, expectations, single and multi random variables, discrete and continuous functions of random variables, analysis via statistics, MATLAB, and Monte Carlo approaches.

Course Objectives: By the end of the semester, students should be able to

- Explain what a random process is and give examples,
- Determine the mean, variance, and expected value of a random process given its cumulative distribution function or its probability density function via calculus as well as Monte-Carlo (MATLAB) simulations.
- Determine what type of distribution applies to a random process including: binomial, Poisson, geometric, hyper-geometric, and normal distributions.
- Determine if two random processes have similar means using a t-test.
- Determine if a random process has a given distribution using a chi-squared test.

Grading

- Midterms: 1 unit each
- Homework 1 unit
- Total: Average of all above

Final Percentage:

- 100% - 90% A
- 89% - 80% B
- 79% - 70% C
- 69% - 60% D
- < 59% E

Policies: A student may take a makeup exam if he/she misses an exam due to an emergency, illness, or plant trip and notifies me in advance of the exam. Late homework will not be accepted once the solutions are posted online.

Homework: Homework is due by 8am the following day. (I'll start grading the homework at 8am and hopefully have them finished by 11am at the start of office hours).

Chat GPT and AI: Chat GPT and AI is notorious for submitting nonsense answers and/or copies of other people's work. Use of Chat GPT and AI is discouraged (and not allowed on tests). **Along these lines, students are responsible for whatever they submit.** If your solution makes no sense, I will assume that's your work. If your solution is a copy of someone else's work, I will also assume that is also your intent.

Testing: All tests will be open-book, open-notes, calculators, matlab, the internet permitted. Individual effort though (i.e. no working together, using ChatGPT, Chegg, AI, or similar tools). Midterms serve to identify who put in the time solving the homework problems. My goal in writing tests is add new twists you haven't seen yet so that

- If you did the homework and are comfortable with the concepts and tools, you'll have a shot at the midterms.
- If you copied someone else's homework, you'll be lost.

The best way to study for the midterm is to make up your own midterm. There's only so many ways to ask a question.

The rules for tests are:

- Tests will be posted at 6am, due at 7am the next day.
- 2 hour time limit from when you start
- Each test will be different (each student generates random numbers for the test)
- All tests are open-book, open notes, calculators, Matlab all permitted
- Providing or receiving help from others prohibited
- Posting test on-line and/or using an on-line solution prohibited

Special Needs - Any students with disabilities or other special needs, who need special accommodations in this course are invited to share these concerns or requests with the instructor as soon as possible.

Academic Honesty - All work in this course must be completed in a manner consistent with NDSU University Senate Policy, Section 335: Code of Academic Responsibility and Conduct. Violation of this policy will result in receipt of a failing grade.

ECE Honor Code: On my honor I will not give nor receive unauthorized assistance in completing assignments and work submitted for review or assessment. Furthermore, I understand the requirements in the College of Engineering and Architecture Honor System and accept the responsibility I have to complete all my work with complete integrity.

Matlab & Monte-Carlo Simulations

Introduction & Random Processes:

ECE 341 Random Processes is really a course on statistics for engineers. A random process is

- A repeatable event,
- Whose outcome is different each time it happens.

For example, rolling a 6-sided die is a random process

- You can roll the die over and over again, and
- Each time you roll the die, the result changes (although repeats are possible)

The high temperature each day is a random process

- Each day's high temperature changes each day, and
- Each result is different

Collecting data in a lab is a random process

- A given lab experiment can be run over and over again, and
- Each time you run the experiment, you'll get slightly different outcomes.

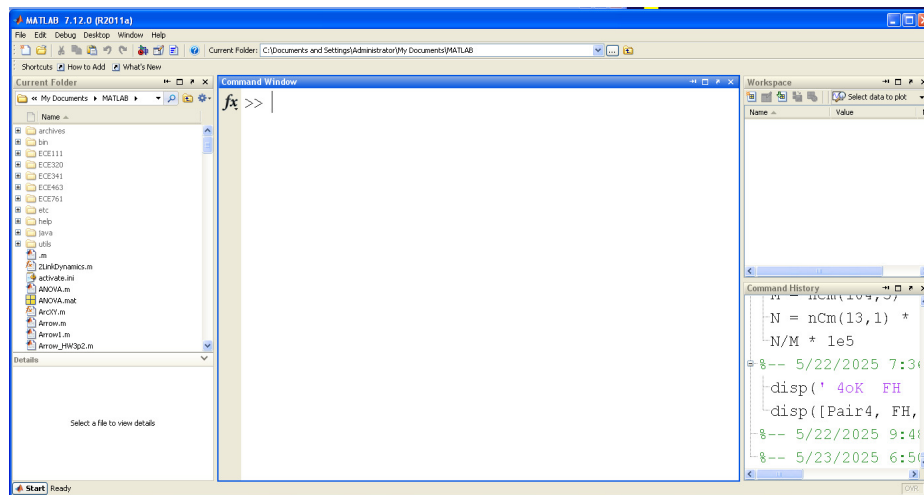
This course looks at

- How to compute the odds of random processes, such as drawing 5 cards from a 52-card deck),
- Modeling discrete random processes,
- Modeling continuous random processes, and
- How to analyze data from lab experiments.

The main tool we'll use in this course is Matlab. Matlab is essentially a programmable calculator that's become a standard tool throughout engineering.

Matlab Overview

Matlab is a computer program which is essentially a calculator. When you start Matlab, the screen typically looks like this:



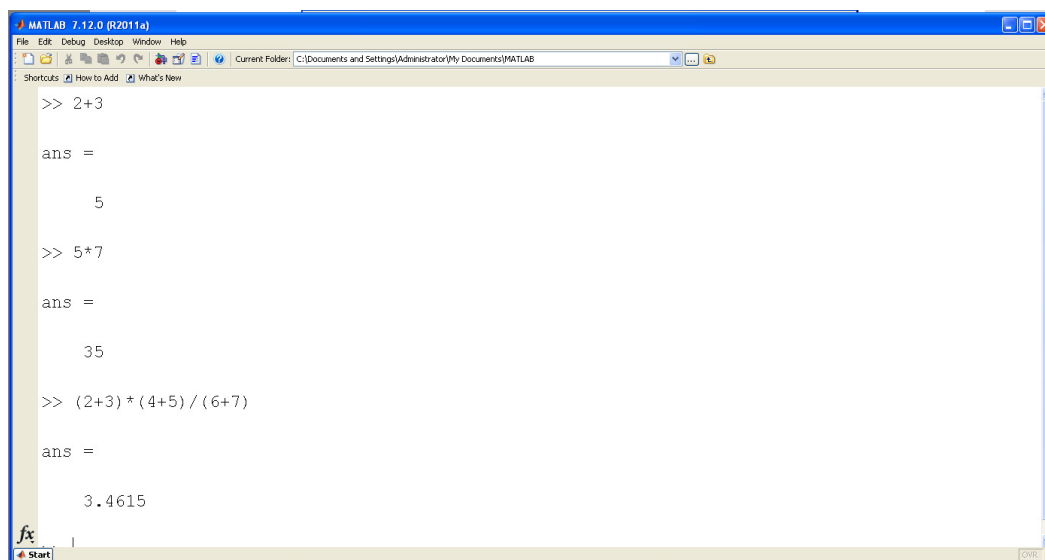
Matlab Default Display

This shows several windows

- The left side shows the current directory you're using. I usually don't care and close this window.
- The upper right side shows a list of your variables. Again, I don't care and close this window.
- The lower right side shows a list of previously used commands. Again, I don't care and close this window.

What's left is the command window. That's the main window I use.

The Command Window is very similar to the Shell window in Python - for all you Python programmers out there. This window lets you use Matlab like a calculator. For example, you can do calculations almost the same way you'd write them on paper.



Matlab being used as a calculator

Valid commands in Matlab are as follows

Symbol	Example	Name
+	2 + 3	addition
-	2 - 3	subtraction
*	2 * 3	multiplication
/	2 / 3	division
^	2 ^ 3	raise to the power
%	% 2 ^ 3	comment

In addition, Matlab has other functions you have to spell out. A short list is:

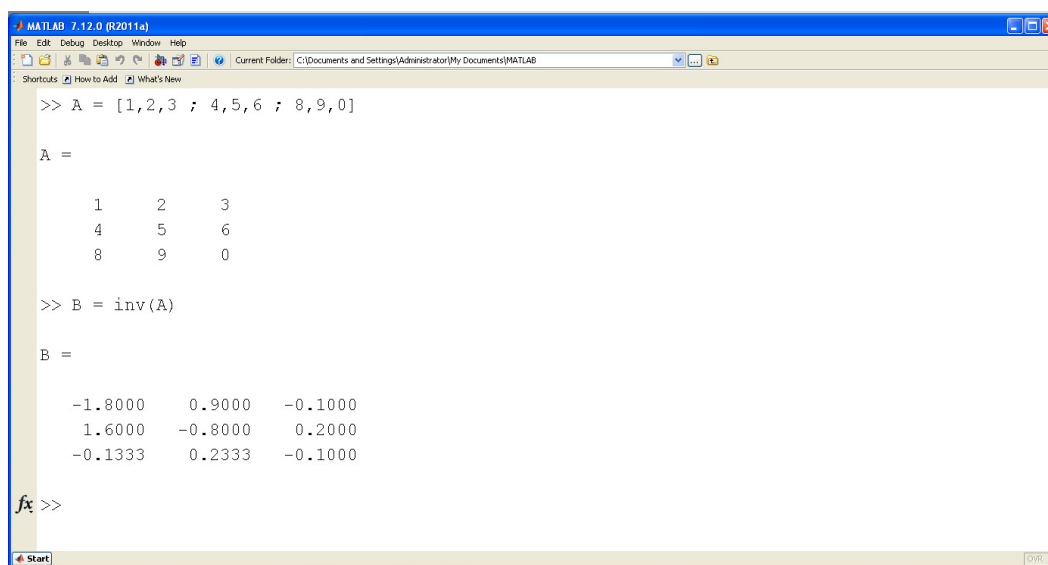
Symbol	Example	Name
log	log(3)	natural log
log10	log10(3)	log base 10
exp	exp(3)	e raised to the power of 3
sin	sin(3)	sin(3 radians)
cos	cos(3)	cos(3 radians)
tan	tan(3)	tan(3 radians)
floor	floor(2.3)	round down
ceil	ceil(2.3)	round up
round	round(2.3)	round to nearest integer
mod	mod(x,7)	modulus. return x mod 7

Matlab has several random-functions

- rand return a random number in the range of (0,1)
- randn return a random number with a normal distribution with mean=0, var=1

Matlab is a matrix language.

- [start of a matrix
-] end of a matrix
- , next column (space also works)
- ; next row (carriage return also works)
- ' transpose
- inv(A) inverse of matrix A
- zeros(m,n) create an nx3 matrix of zeros
- eye(n,n) create an nxn identity matrix
- A * B multiply matrix A by matrix B (matrix multiplication)
- A ^ 2 square a matrix (equal to A*A)
- A .* B element-by-element multiplication
- A.^2 element-by-element squaring
- sum(A) returns the sum of the terms in A



The screenshot shows the MATLAB 7.12.0 (R2011a) command window. The user has entered two commands: `A = [1,2,3 ; 4,5,6 ; 8,9,0]` and `B = inv(A)`. The output for `A` is a 3x3 matrix with rows [1, 2, 3], [4, 5, 6], and [8, 9, 0]. The output for `B` is the inverse of `A`, a 3x3 matrix with rows [-1.8000, 0.9000, -0.1000], [1.6000, -0.8000, 0.2000], and [-0.1333, 0.2333, -0.1000].

```
>> A = [1,2,3 ; 4,5,6 ; 8,9,0]

A =

     1     2     3
     4     5     6
     8     9     0

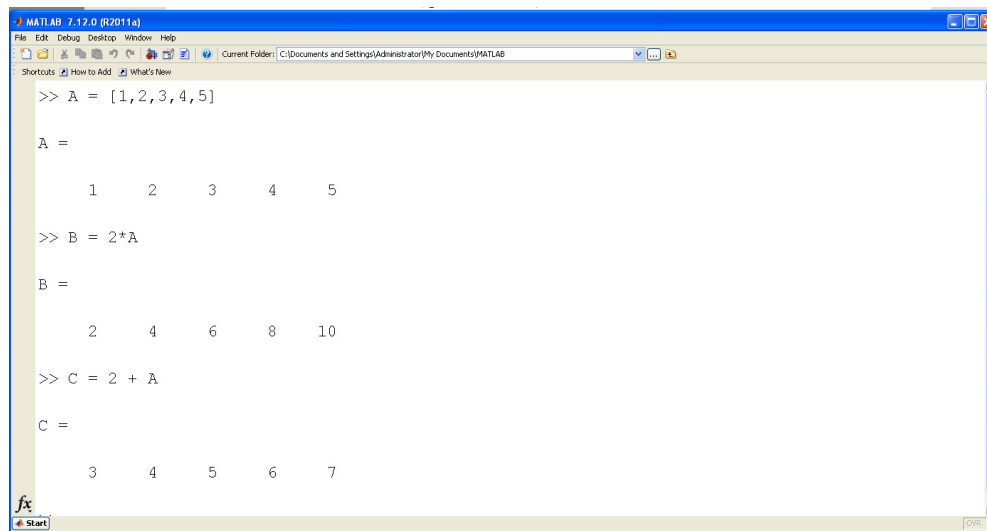
>> B = inv(A)

B =

    -1.8000    0.9000   -0.1000
     1.6000   -0.8000    0.2000
    -0.1333    0.2333   -0.1000
```

Matrix operations in Matlab

Addition, subtraction, and multiplication all valid operations in Matlab. Note that when adding or multiplying a matrix with a scalar, all entries are changed. This is different from how Python handles matrices.



The screenshot shows the MATLAB 7.12.0 (R2011a) command window. The user has entered three commands: `A = [1,2,3,4,5]`, `B = 2*A`, and `C = 2 + A`. The output for `A` is a 1x5 row vector [1, 2, 3, 4, 5]. The output for `B` is a 1x5 row vector [2, 4, 6, 8, 10]. The output for `C` is a 1x5 row vector [3, 4, 5, 6, 7].

```
>> A = [1,2,3,4,5]

A =

     1     2     3     4     5

>> B = 2*A

B =

     2     4     6     8    10

>> C = 2 + A

C =

     3     4     5     6     7
```

In Matlab, adding or multiplying a matrix with a scalar affects all entries

if, for-loops, while-loops

Matlab supports if-statements, for-loops, and while-loops. The syntax for if-statements is as follows. Note that

- Indentation is decorative in Matlab. It does help the reader understand the code though.

- for-statements must be terminated with an end statement

```
A = 3;
if(A == 2)
    disp('A equals 2')
elseif(A==3)
    disp('A equals 3')
else
    disp('A is not 2 or 3')
end
A equals 3
```

if-statements in Matlab

for-statements allow you to execute a set of commands over and over. The basic part of a for-loop is

```
for i=1:5
    % code goes here
end
```

for-loop in Matlab

In Matlab

- The loop starts with i=1
- When you reach the end-statement, the loop repeats with i incremented by one
- The loop stops when i is greater than 5 (slightly different than Python)

For example, to print the squares of the numbers 1..5, use the following code:

```
for i=1:5
    x = i*i;
    disp([i, x])
end

1      1
2      4
3      9
4     16
5     25
```

for-loops. Repeat the code five times

While-loops repeat as long as the condition is true. For example, the following code repeats until t>10

```
t = 0;
dt = 0.01;
x = 10;
while(t < 10)
    x = x + 0.1;
    t = t + dt;
end
```

while-loop in Matlab

Matlab Graphics

Matlab has pretty good graphing capabilities. The main ones we use in this class are:

- `plot(x,y)`
- `plot(x1,y1,x2,y2)`
- `bar(x)`
- `x = [-5:0.01:5];`

The last command creates a row matrix

- starting at -5
- stepping by 0.01
- stopping at +5 inclusive)

The `plot()` command plots in Cartesian coordinates $x()$ vs. $y()$. For example, if you want to find the solutions to

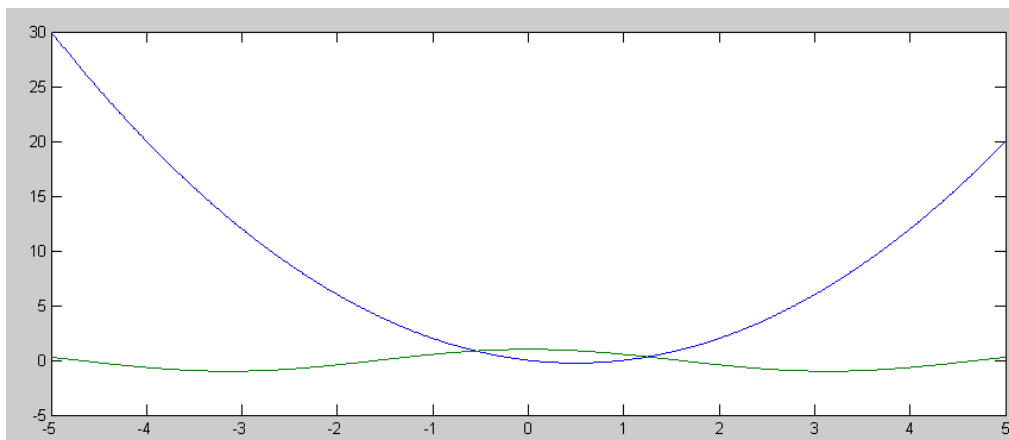
$$y = x^2 - 1$$

$$y = \cos(x)$$

you could graph these functions and see where they intersect

```
x = [-5:0.01:5];  
y1 = x.^2 - x;  
y2 = cos(x);  
plot(x,y1,x,y2);
```

Matlab code to plot two functions



Result of a `plot()` command.

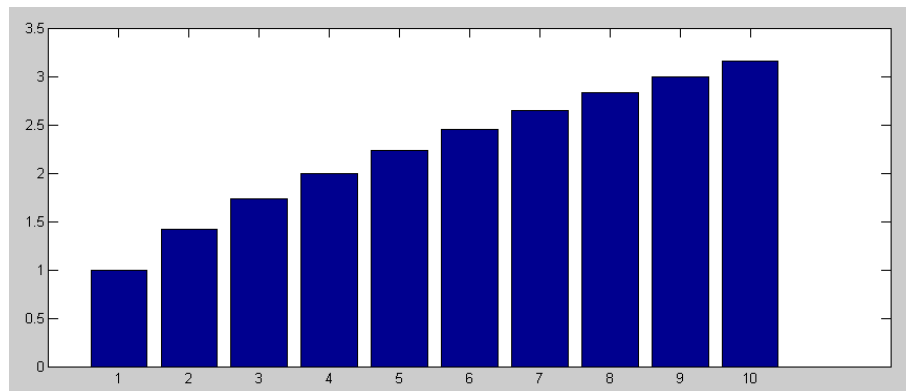
`bar()` draws a bar-chart with

- the x-axis is the number of the data entry
- the y-axis is the value of the contents

For example, plotting a bar-chart showing the square root of the numbers 1..10 would be:


```
x = [1:10];  
y = sqrt(x);  
bar(x,y);
```

creating a bar-chart in Matlab

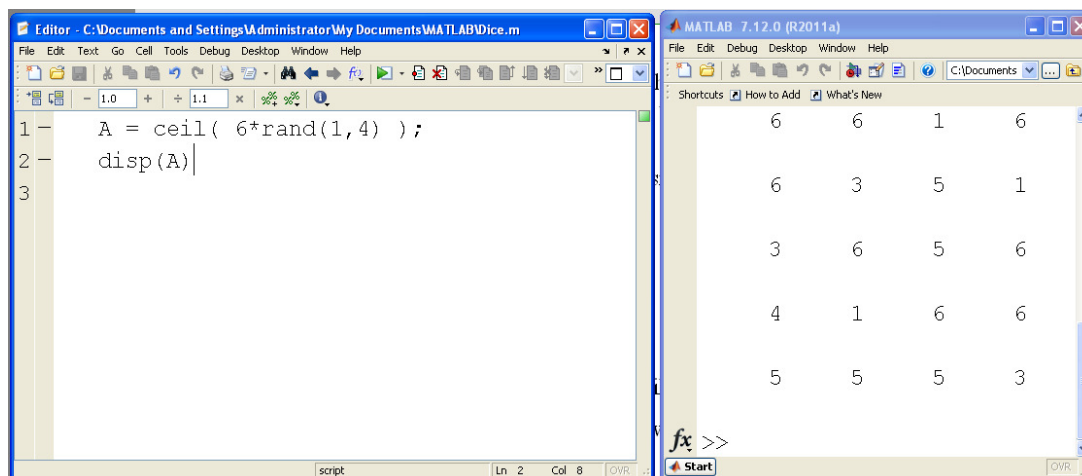


resulting bar-chart

Matlab Scripts (Matlab Programs)

Matlab is also a programming language. Rather than typing code in the command window, you can create a Matlab script (file-new-script). This opens a window where you can type in commands then execute them by clicking the green arrow.

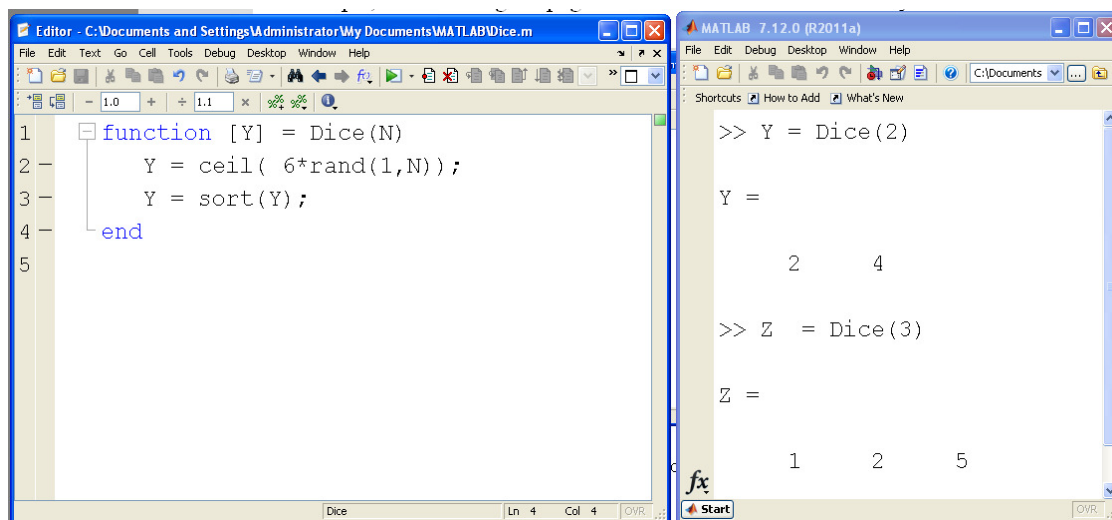
For example, the following script generates six random die rolls every time it's called



Script (left) & Command (right) windows. The script executes each time you click the green arrow (run)

Matlab Functions

What really makes Matlab powerful is you can create your own functions (subroutines in c-speak). For example, if I'm going to be rolling a lot of dice, it would be convenient to have a command Dice(N) which returns the result or rolling N 6-sided dice. A Matlab function which does this is:



Function example

The first line tells Matlab

- This is a function (a subroutine) - as opposed to a script
- The function returns a variable *Y*
- The name of the function is *Dice*
- One parameter is passed to the function (*N*)

The logic inside the function tells you how to compute *Y* in terms of *N*

The end-statement tells Matlab when to end the function.

We'll be using functions and script a lot in this course.

Monte-Carlo Simulations

One reason Matlab is so powerful is it is very good at running Monte Carlo simulations. A Monte-Carlo simulations is one way to find the probability of an event:

- First, code is written to simulate a single event, such as rolling a 6-sided die.
- Once that code has been tested, it is run a bunch of times with the outcome of each simulation recorded

The odds of an event happening is then the ration

$$p(x) \approx \left(\frac{\text{the number of times } x \text{ occurs}}{\text{the number of simulations run}} \right)$$

What makes Monte-Carlo simulations so powerful is that you can determine the probabilities of any function you can program. Even if you can't compute the odds by hand. In this lecture, we'll look use Monte-Carlo techniques to compute:

- The odds of rolling a 1 on a 6-sided die
- The odds that the highest roll of two dice (d4 & d6) beats a 6-sided die
- The odds of player A winning a 5-game match (basketball playoffs)
- The odds of player A winning a win-by-3 match,
- The odds of rolling 4-of-a-kind when rolling six dice, and
- The odds of being dealt three-of-a-kind in poker.

Case 1: 6-Sided Die

Let's start out with the simplest case: determine the odds of rolling a 1 on a 6-sided die. Obviously, the answer is $1/6$. This can be confirmed with a Monte-Carlo simulation.

First, write a Matlab script that rolls a die one time.

```
Win = 0;
Die = ceil( 6*rand );
if(Die == 1)
    Win = Win + 1;
end
disp([Die, Win])
```

Matlab Script (run over and over)

Running this script over and over verifies

- The die roll is correct (it's a random number in the range of $[1,6]$, and
- The win condition is correct ($\text{Win} = 1$ when the die roll is a one)

Die	Win
1	1
5	0
6	0
1	1
3	0
2	0

Testing the Matlab script

Now that you can play the game one time, play it a million times by adding a for-loop

```
Win = 0;
tic
for n=1:1e6
    Die = ceil( 6*rand );
    if(Die == 1)
        Win = Win + 1;
    end
end
toc
disp([Win])
```

Script Window

```
Elapsed time is 0.154412 seconds.
166219
Elapsed time is 0.154566 seconds.
167090
Elapsed time is 0.154347 seconds.
166969
```

Command Window

The *tic toc* commands are just for fun: they show you how long the program took to execute (154ms)

The number in the command window is the number of successes (166219) in 1 million tries. Note that

- You get a different answer each time you run the program (this is a random process)
- The different results allow us to place a bound on the probability of rolling a 1 (something we'll cover when we get to student-t distributions), and
- The odds of rolling a one is about 0.166 (ish).

Case 2: max(2d6) vs. d8

A harder problem is as follows:

- Let player A roll a 4-sided die and a 6-sided die and take the maximum.
- Let player B roll a single 6-sided die
- Whoever has the highest score wins (B wins on ties)

What is the probability that A will win?

Here, the answer isn't so obvious. With Matlab and a Monte-Carlo simulation, we can find the answer (approximately at least).

Step1 is again to play a single game.

```
Win = 0;
d4 = ceil(4*rand);
d6 = ceil(6*rand);
A = max(d4, d6);
B = ceil(6*rand);
if(A>B)
    Win = Win + 1;
end
disp([d4, d6, A, B, Win])
```

Script Window

d4	d6	A	B	Win
2	4	4	4	0
2	2	2	4	0
1	2	2	2	0
1	2	2	1	1
2	2	2	4	0

Command Window

From the command window, the code looks good:

- The d4 and d6 are random numbers in the range of [1,4] and [1,6]
- A is the maximum of (d4, d6)
- B is a random number in the range of [1,6]
- A wins only if $A > B$ (B wins on ties)

Now that this code works, repeat 1 million times

```
Win = 0;
tic
for n=1:1e6
    d4 = ceil(4*rand);
    d6 = ceil(6*rand);
    A = max(d4, d6);
    B = ceil(6*rand);
    if(A>B)
        Win = Win + 1;
    end
end
toc
disp([Win])
```

Script Window

```
Elapsed time is 0.439249 seconds.
485961
Elapsed time is 0.439099 seconds.
486520
Elapsed time is 0.442024 seconds.
486615
```

Command Window

It looks like A has a about a 48.6% chance of winning any given game

Case 3: 5-Game Match

Next, consider the problem where

- A and B are playing a given match
- A has a 60% chance of winning any given game
- The match is over after 5 games (best-of-5 match)

What is the probability that A will win the match?

We'll solve this problem several times throughout this course. For now, let's solve using a Monte-Carlo simulation.

First step is to play a single match:

```
Wins = 0;
A = 0;
B = 0;
for i=1:5
    if(rand < 0.6) A = A + 1;
    else B = B + 1;
    end
end
if(A > B)
    Wins = Wins + 1;
end
disp([A,B,Wins])
```

Script Window

A	B	Wins
4	1	1
4	1	1
2	3	0
4	1	1
3	2	1

Command Window

So far, the code looks good:

- The number of games A and B wins varies
- A usually wins more games
- A wins the match whenever A wins 3 or more games

Once a single game works, play a series of 1 million games

```
Wins = 0;
tic
for n=1:1e6
    A = 0;
    B = 0;
    for i=1:5
        if(rand < 0.6) A = A + 1;
        else B = B + 1;
        end
    end
    if(A > B)
        Wins = Wins + 1;
    end
end
toc
disp(Wins)
Wins = 0;
```

script window

```
Elapsed time is 0.902470 seconds.  
682096  
Elapsed time is 0.901853 seconds.  
683076  
Elapsed time is 0.900268 seconds.  
682342
```

command window

A has about a 68.2% chance of winning the match.

Case 4: Win-By-3 Match

Suppose the format changes to win-by-three? What's the probability that A wins the match?

Again, start by writing the code for playing a single match. Here, the for-loop is replaced by a while-loop (keep playing until someone is up 3 games).

```
Wins = 0;  
A = 0;  
B = 0;  
while(abs(A-B) < 3)  
    if(rand < 0.6) A = A + 1;  
    else B = B + 1;  
    end  
end  
if(A > B)  
    Wins = Wins + 1;  
end  
disp([A, B, Wins])
```

script window

A	B	Wins
3	0	1
6	3	1
6	3	1
3	6	0

command window

So far, so good:

- A usually wins
- The match ends when one player is up 3 games
- The duration of the match varies

Once you can play a single match, play a million matches:

```

Wins = 0;
MaxGames = 0;
tic
for n=1:1e6
    A = 0;
    B = 0;
    while(abs(A-B) < 3)
        if(rand < 0.6) A = A + 1;
        else B = B + 1;
        end
    end
    if(A > B)
        Wins = Wins + 1;
    end
    MaxGames = max(MaxGames, A+B);
end
toc
disp([Wins, MaxGames])

```

script window

```

Elapsed time is 0.696762 seconds.
771633      87
Elapsed time is 0.693483 seconds.
772376      97
Elapsed time is 0.695759 seconds.
772023      81

```

command window

With this format

- A has about a 77.2% chance of winning the match, and
- The longest match was 97 games (!)

TV really doesn't like this format.

Case 5: Rolling Dice

Next, let's look at a dice game. Suppose you roll six 6-sided dice. What's the chance of rolling

- Two triples (die values are xxx yyy), and
- One triple (die values are xxx aab or xxx abc)?

To run a Monte-Carlo simulation, start by rolling six dice.

```

Dice = ceil(6*rand(1,6));
[Dice]

```

script window

```

Dice =      6      2      5      5      5      6
Dice =      2      1      4      3      6      4
Dice =      4      2      1      6      4      5

```

command window

Looks good:

- Each roll consists of six numbers (six dice)
- Each die is in the range of 1..6

Next, count the frequency of each number and sort in descending order

```
Dice = ceil(6*rand(1,6));
N = zeros(1,6);
for i=1:6
    N(i) = sum(Dice == i);
end
N = sort(N, 'descend')
[Dice]
[N]
```

script window

```
Dice =      1      6      4      1      3      2
N      =      2      1      1      1      1      0
```

command window

Looks good:

- The highest frequency is two (two 1's)
- The next highest frequency is one

Now check for the number of two-triples and one-triple and repeat one million times

```
Pair33 = 0;
Pair3 = 0;
tic
for n=1:1e6
    Dice = ceil(6*rand(1,6));
    N = zeros(1,6);
    for i=1:6
        N(i) = sum(Dice == i);
    end
    N = sort(N, 'descend');
    if((N(1)==3)*(N(2)==3)) Pair33 = Pair33 + 1; end
    if((N(1)==3)*(N(2)<3)) Pair3 = Pair3 + 1; end
end
toc
[Pair33, Pair3]
```

script window

```
Elapsed time is 55.056778 seconds.
ans =      6337      308026
```

command window

Meaning, the odds of rolling:

- Two triples is about 6337 in one million, and
- One triple is about 308,026 in one million

Poker (full-house and 3-of-a-kind)

Finally, let's look at a trickier problem. Suppose you shuffle a 52-card deck of cards and deal out five cards (i.e. a poker hand). What's the chance of being dealt 3-of-a-kind?

This program is a little trickier. First, shuffle a deck of 52 cards and deal out the top 5 cards

```
X = rand(1,52);
[a,Deck] = sort(X);
Deck = Deck - 1;
Hand = Deck(1:5);
Value = mod(Hand,13) + 1;
Suit = floor(Hand/13) + 1;
```

```
disp(Hand)
disp(Value)
disp(Suit)
```

script window

Hand	34	37	29	13	44
Value	9	12	4	1	6
Suit	3	3	3	2	4

command window

The first three lines of code generate the numbers [0,51] in random order (the shuffled deck of cards)

Line #4 takes the top five cards for you hand

- cards number 34, 37, 29, 13, and 44

Line #5 determines the value of each card using the modulus command

- 9, Q (card #12), 4, Ace, 6

Line #6 determines the suit

- heart, heart, heart, diamond, spade

Once you have your hand, determine the frequency of each card and sort the frequency in descending order

```
N = zeros(1,13);
for n=1:13
    N(n) = sum(Value == n);
end
disp(N)
```

script window

A	2	3	4	5	6	7	8	9	10	J	Q	K
1	0	0	1	0	1	0	0	1	0	0	1	0

command window

Sort based upon frequency then check for a 3-of-a-kind

- The highest frequency is 3
- The second highest frequency is 1

```

FH = 0;
Pair3 = 0;
N = zeros(1,13);
for n=1:13
    N(n) = sum(Value == n);
end
[N,a] = sort(N, 'descend');
if ((N(1) == 3)*(N(2) == 2)) FH = FH + 1; end
if ((N(1) == 3)*(N(2) < 2)) Pair3 = Pair3 + 1; end

```

script window

Finally, once you can deal out one hand, deal out a million:

```

tic
Pair3 = 0;
FH = 0;

for n = 1:1e6
    X = rand(1,52);
    [a,Deck] = sort(X);
    Deck = Deck - 1;
    Hand = Deck(1:5);
    Value = mod(Hand,13) + 1;
    Suit = floor(Hand/13) + 1;

    N = zeros(1,13);
    for n=1:13
        N(n) = sum(Value == n);
    end
    [N,a] = sort(N, 'descend');

    if ((N(1) == 3)*(N(2) == 2)) FH = FH + 1; end
    if ((N(1) == 3)*(N(2) < 2)) Pair3 = Pair3 + 1; end
end
[FH]
[Pair3]
toc

```

script window

```

FH      =      1369
Pair3    =      21284
Elapsed time is 110.689897 seconds.

```

command window

There is about a 0.13% chance of being dealt a full-house

$$p \approx \left(\frac{1369}{1,000,000} \right) = 0.001369$$

There is about a 2.1% chance of being dealt a 3-of-a-kind.

$$p \approx \left(\frac{21,284}{1,000,000} \right) = 0.021284$$

This took almost two minutes to compute, however.

Summary

The definition of probability is the percent of time an event occurs as the number of trials goes to infinity. This sets up one way to compute probabilities: Monte Carlo simulations. With a Monte Carlo simulation,

- You first run an experiment one time.
- Once that works, repeat the experiment a large number of times (one million in this lecture).

The probability of the event happening is then approximately the number of successes divided by the number of trials. Matlab is a tool which makes such calculations fairly easy - although it may take some programming.

•