Monte-Carlo Experiments

ECE 341 Random Processes

Lecture #2

Please visit Bison Academy for course syllabus, lecture notes, recorded lectures, homework sets, and solutions

www.BisonAcademy.com

Introduction

The probability of an event is defined as

> *The number of times an event happens as the number of trials goes to infinity.*

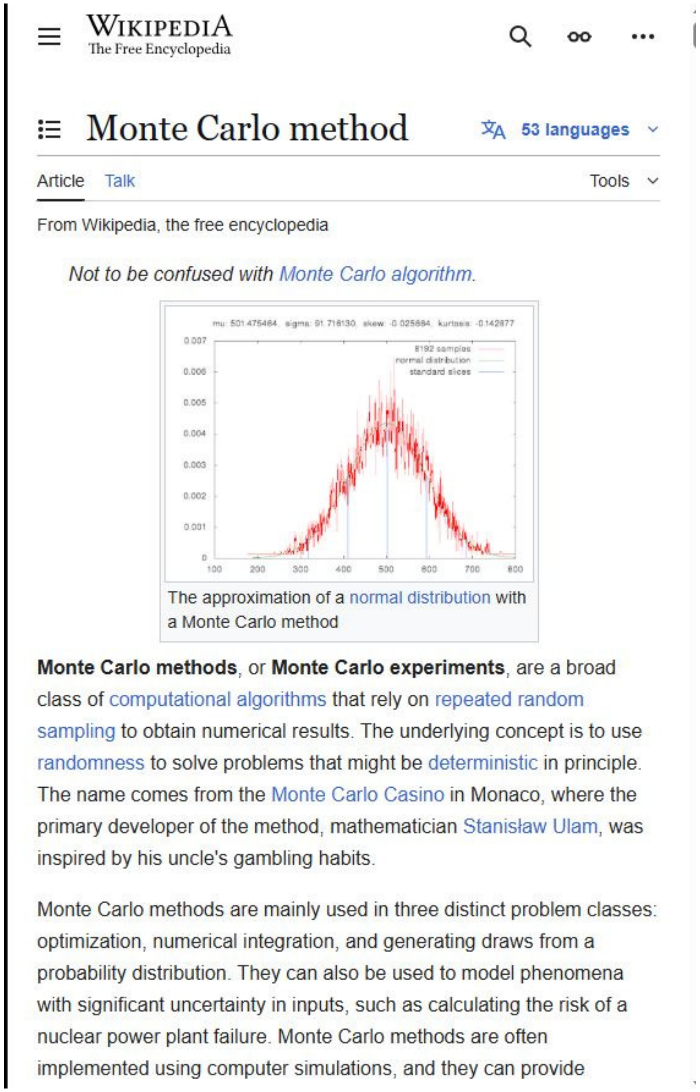
This leads to one way to compute the probability of an event:

- Write a program to play a game one time
- Then play the game one million times
- Count the number of times the event happens

The probability of the event happening is then approximately

- The number of times the event happened,
- Divided by 1 million

This is termed *A Monte Carlo Experiment*



WIKIPEDIA
The Free Encyclopedia

Monte Carlo method — 53 languages

Article Talk — Tools

From Wikipedia, the free encyclopedia

Not to be confused with Monte Carlo algorithm.

The approximation of a normal distribution with a Monte Carlo method

Monte Carlo methods, or **Monte Carlo experiments**, are a broad class of computational algorithms that rely on repeated random sampling to obtain numerical results. The underlying concept is to use randomness to solve problems that might be deterministic in principle. The name comes from the Monte Carlo Casino in Monaco, where the primary developer of the method, mathematician Stanisław Ulam, was inspired by his uncle's gambling habits.

Monte Carlo methods are mainly used in three distinct problem classes: optimization, numerical integration, and generating draws from a probability distribution. They can also be used to model phenomena with significant uncertainty in inputs, such as calculating the risk of a nuclear power plant failure. Monte Carlo methods are often implemented using computer simulations, and they can provide

Matlab

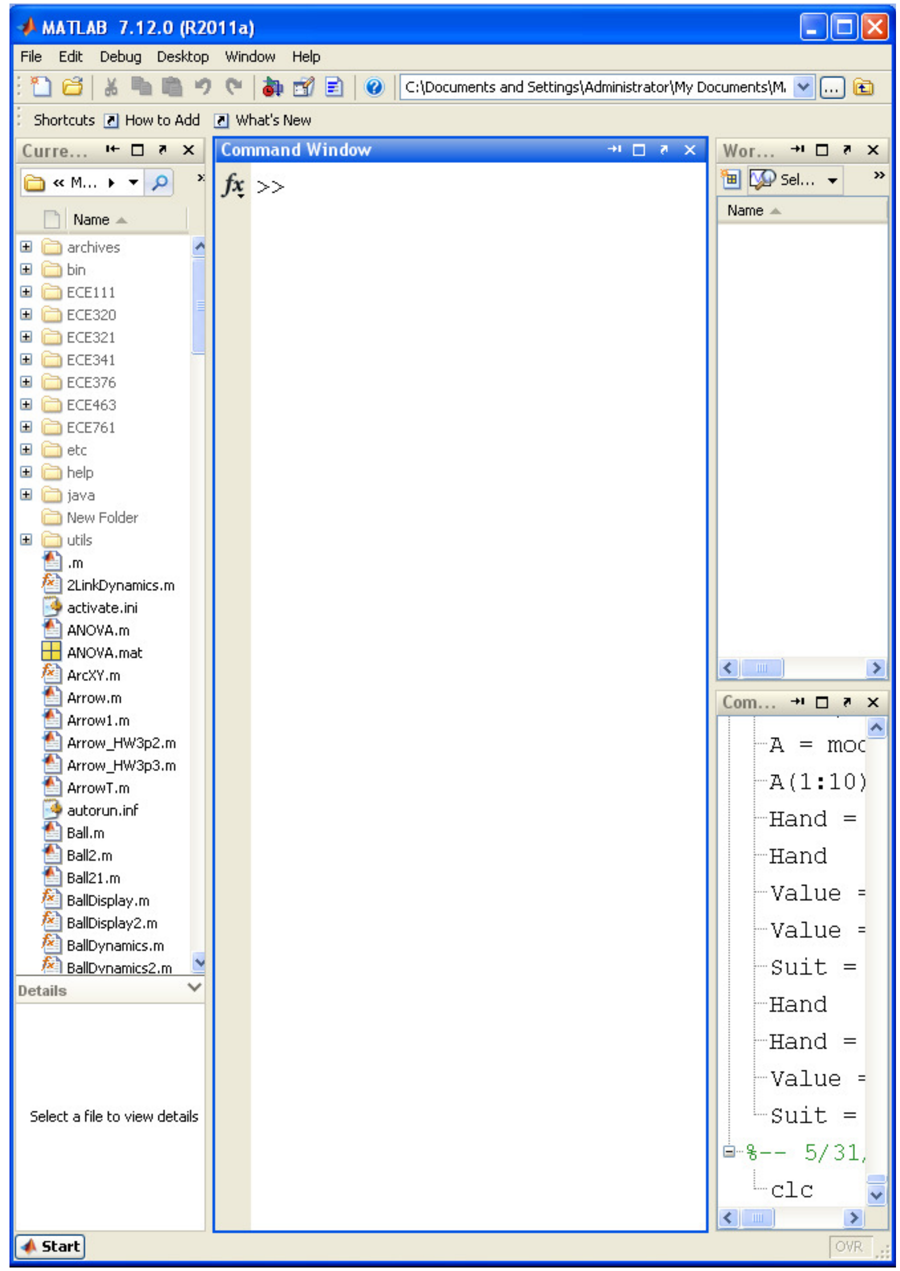
- A little background on Matlab

Matlab is a computer program we'll use throughout this course

- It's essentially a calculator
- It's also a programming language

The default screen includes

- Current folder
  - Current directory - close
- Command Window
  - Keep
- Workspace
  - List of variables - close
- Command History
  - Don't care - close



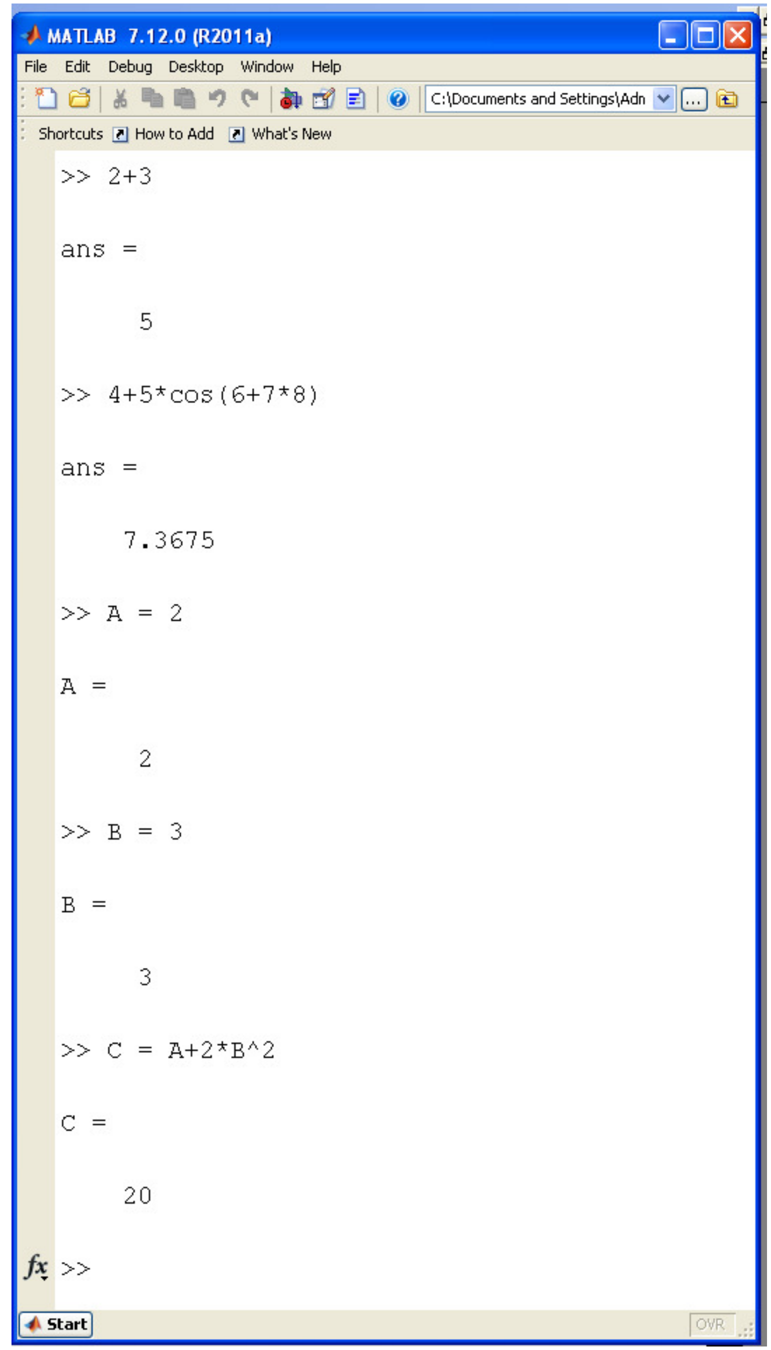
Matlab as a Calculator

The command window acts like a calculator

- You can type in operations
- It then evaluates these operations

You can assign numbers to variables

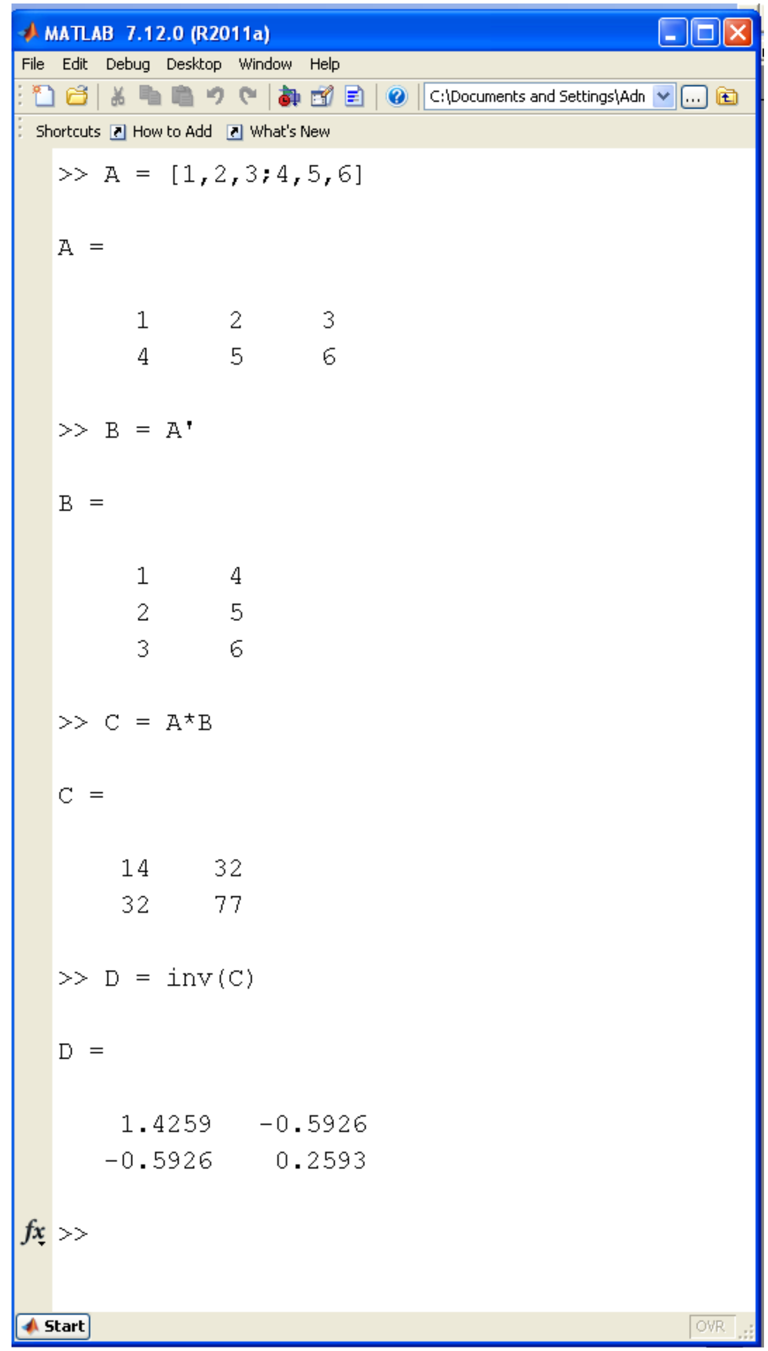
- Case sensitive
- Variables can be functions of variables
- Valid syntax
    - Variable name
    - Equals
    - Operation to evaluate



Matlab is a Matrix Language

Matlab can be treated like a calculator that works with matrices

```
[         start of matrix
]         end of matrix
,         next column
;         next row

+         addition
-         subtraction
*         multiplication
/         division
'         transpose

.*        element-by-element multiply
./        element-by-element division
.^        element raise to a power

inv(A)    matrix inverse
```



Flow Control

Matlab is a programming language with loops

For-Loop

```
for i=1:10
   t = t + dt;
   end
```

While-Loop

```
time = 0;
while(time < 10)
   x = x + dx*dt;
   t = t + dt;
   end
```

If

```
if(time < 10)
   x = 0;
   end
```

If-Else

```
if(x>y)
   points = points + 1;
elseif(x == y)
   points = points + 0.5;
else
   points = points + 0;
end
```

Command Window

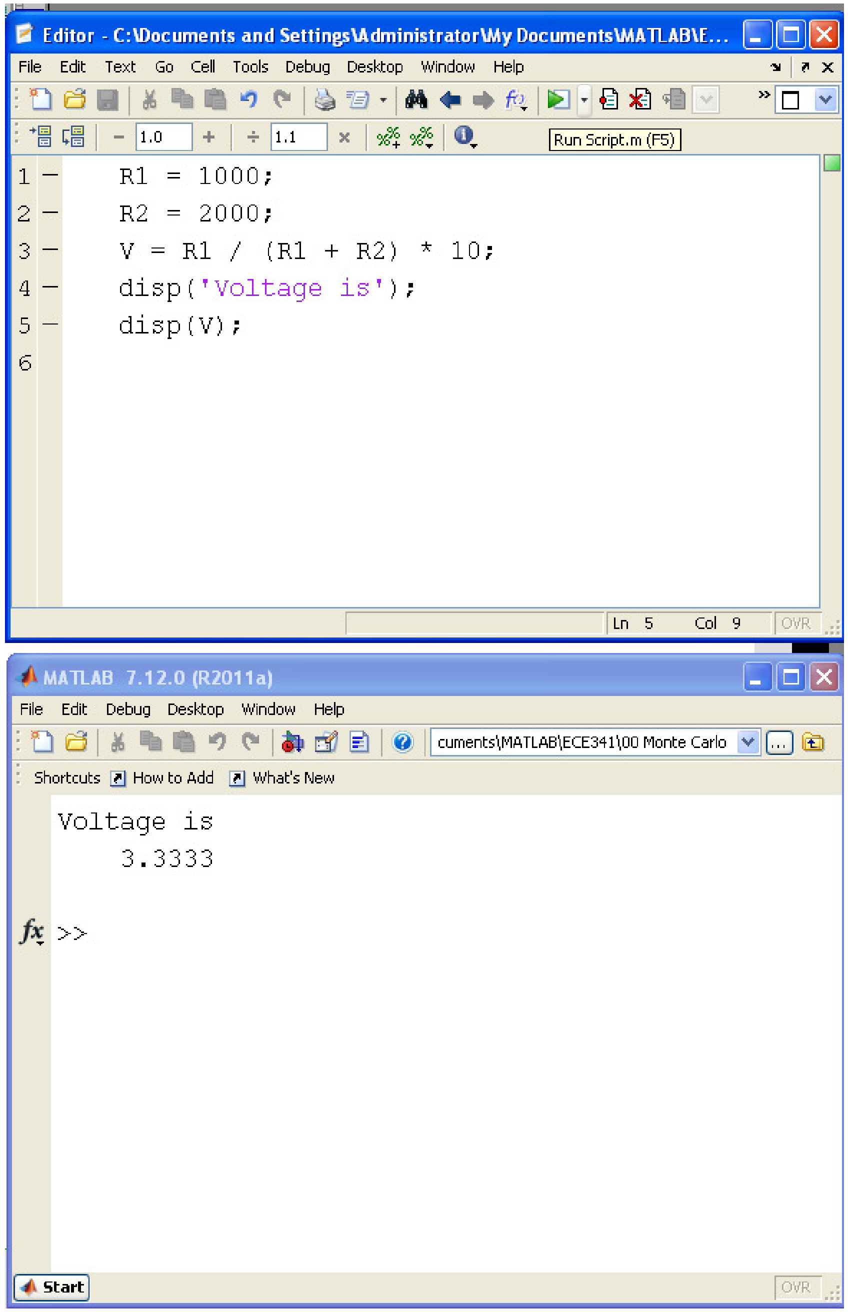
```
>> X = zeros(1,4);
>> for i=1:4
     X(i) = i*i;
     end
>> X

X =

     1     4     9    16

>> x = 10;
>> dx = 0;
>> t = 0;
>> dt = 0.01;
>> while(x>0)
     ddx = -9.8;
     dx = dx + ddx*dt;
     x = x + dx*dt;
     t = t + dt;
     end
>> t

t =

    1.4300
```

Matlab Scripts

- File - New - Script
- control N

If you're going to run the same code over and over, you can place it in a script.

Each time you execute the script, it's like pasting that code into the command window.



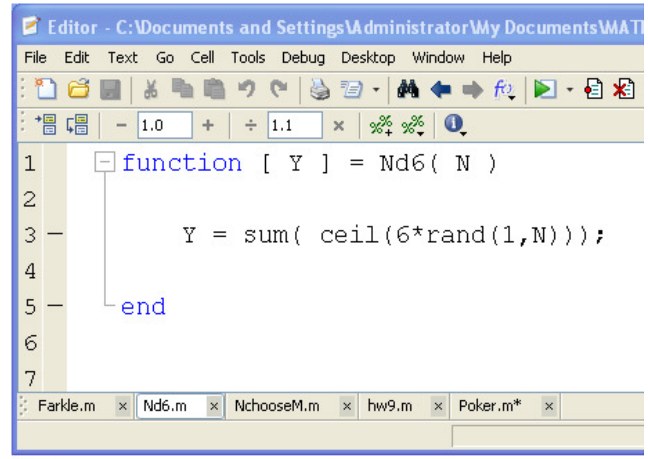
note: This is a convenient way to build a more complex program.

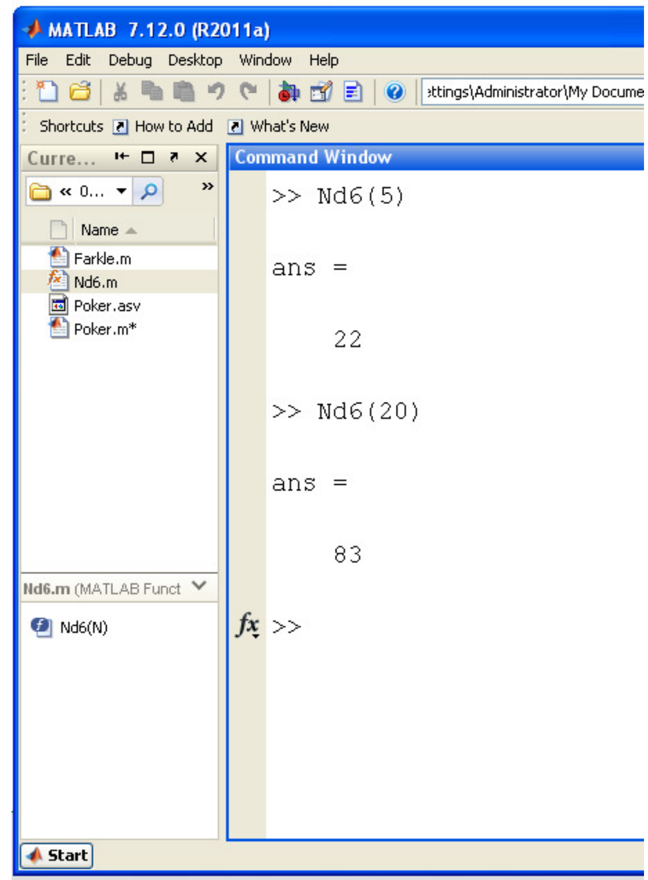
- Display the data when writing the program to see what is happening
- You can fix errors along the way

Matlab Functions

Part of what makes Matlab so powerful is you can create your own functions

- These functions become Matlab commands that other functions can use.
- As companies build up their library of Matlab functions, they get better and better at designing their product.
- The Matlab functions become company proprietary information (design secrets).





Random Numbers in Matlab

- rand(4,1): generate a 4x1 matrix of random numbers in the range of (0,1)
- randn(4,1) generate a 2x3 matrix of normally distributed random numbers
- ceil(6 * rand(4,1)): generate four 6-sided dice (4d6)

Command Window

```
>> rand(4,1)

ans =

    0.1712
    0.7060
    0.0318
    0.2769

>> randn(4,1)

ans =

   -0.8095
   -2.9443
    1.4384
    0.3252

>> ceil(6 * rand(4,1))

ans =

     2
     6
     1
     3

>> 
```

Useful Matlab Commands

command	Description
+ - * /	add, subtract, multiply, divide
^	raise to a power
.^ .* ./	element-by-element operations
ceil(2.3)	round up
floor(2.3)	round down
round(2.3)	round
mod(x,10)	x modulus 10
A'	matrix transpose
inv(A)	matrix inverse
sum(A)	sum of all elements
max(A)	maximum element of A
min(A)	minimum element of A
pause(0.1)	pause 0.1 second

Matlab Command	Description
x = [0:10];	create a row matrix starting from 0 going to 10, step size = 1
x = [0:0.01:10];	go from 0 to 10 step size of 0.01
length(x)	length of matrix x
size(x)	dimensions of matrix x
rand	random number in (0,1)
rand(100,1)	100x1 matrix of random numbers
randn	random with a normal distribution
mean(x)	mean, average, 1st moment
std(x)	standard deviation of x
var(x)	variance of x
disp(A)	display A
tic	start recording time
toc	display time since tic

Monte-Carlo Experiments

- Named after Monte Carlo casino

One way to determine the probabilities

- Write a script to play a game one time
- Next, play the game N times
    - N is large, like a million
- Count how many times an event happens
    - Player A wins

The probability of winning is

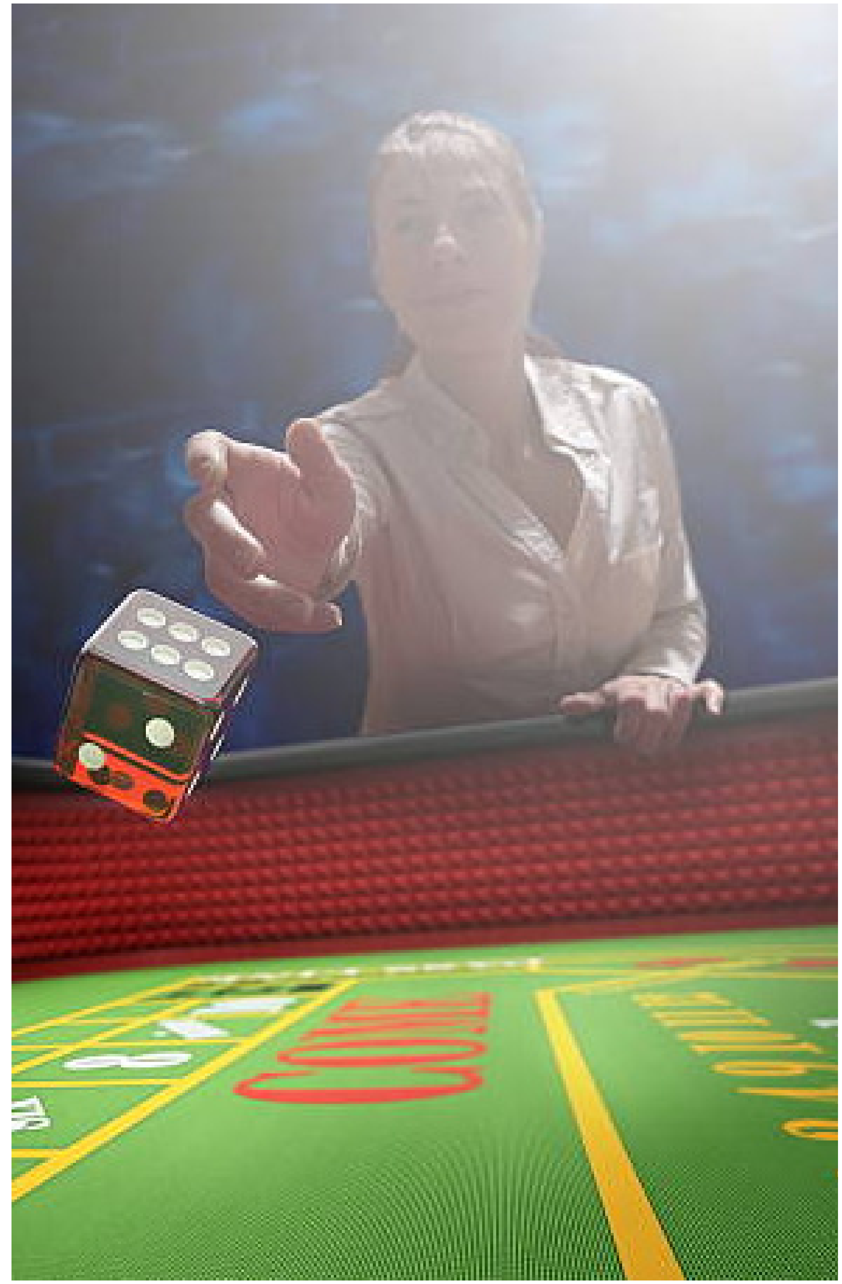
- The number of events
- Divided by the number of games
- (approximately)

Case 1: 6-Sided Die

- What is the probability of rolling a 1 on a 6 sided die?

Start with something simple

- We know what the result should be
- p = 1/6



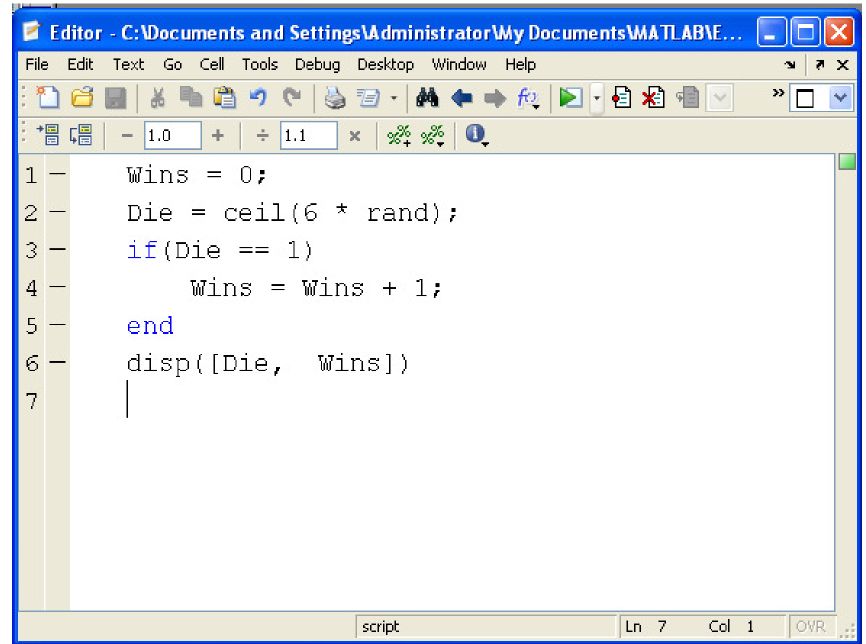
Case 1 (cont'd)

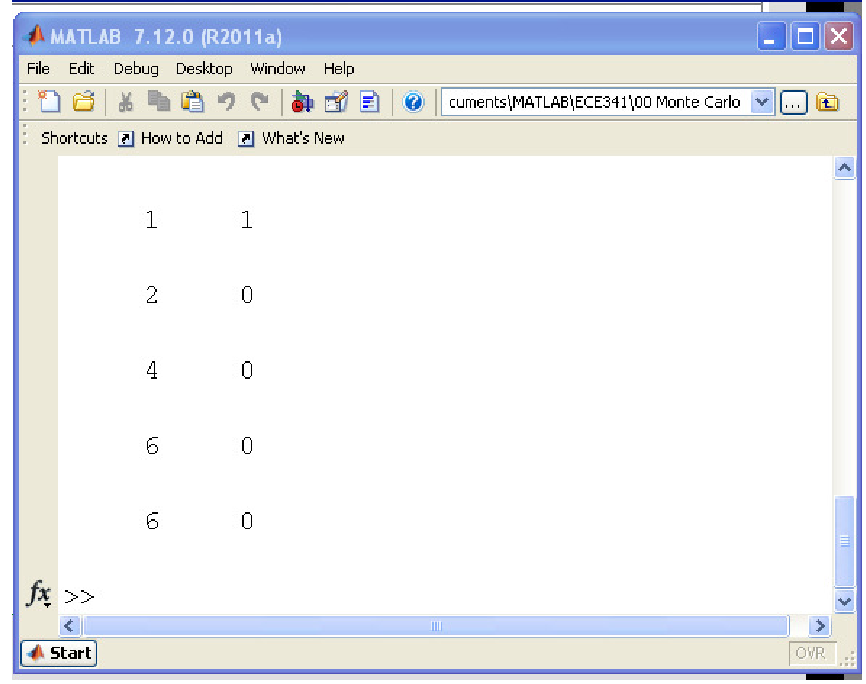
Monte-Carlo Solution:

- Start with playing a game one time
- Script Window (upper image)

Check that the program is working properly

- Command window (lower image)
- Each time you press *run*
  - you get a different result
  - it's random
- When you roll a 1, you win





Case 1 (cont'd)

Once you can play a game one time, play it a million times

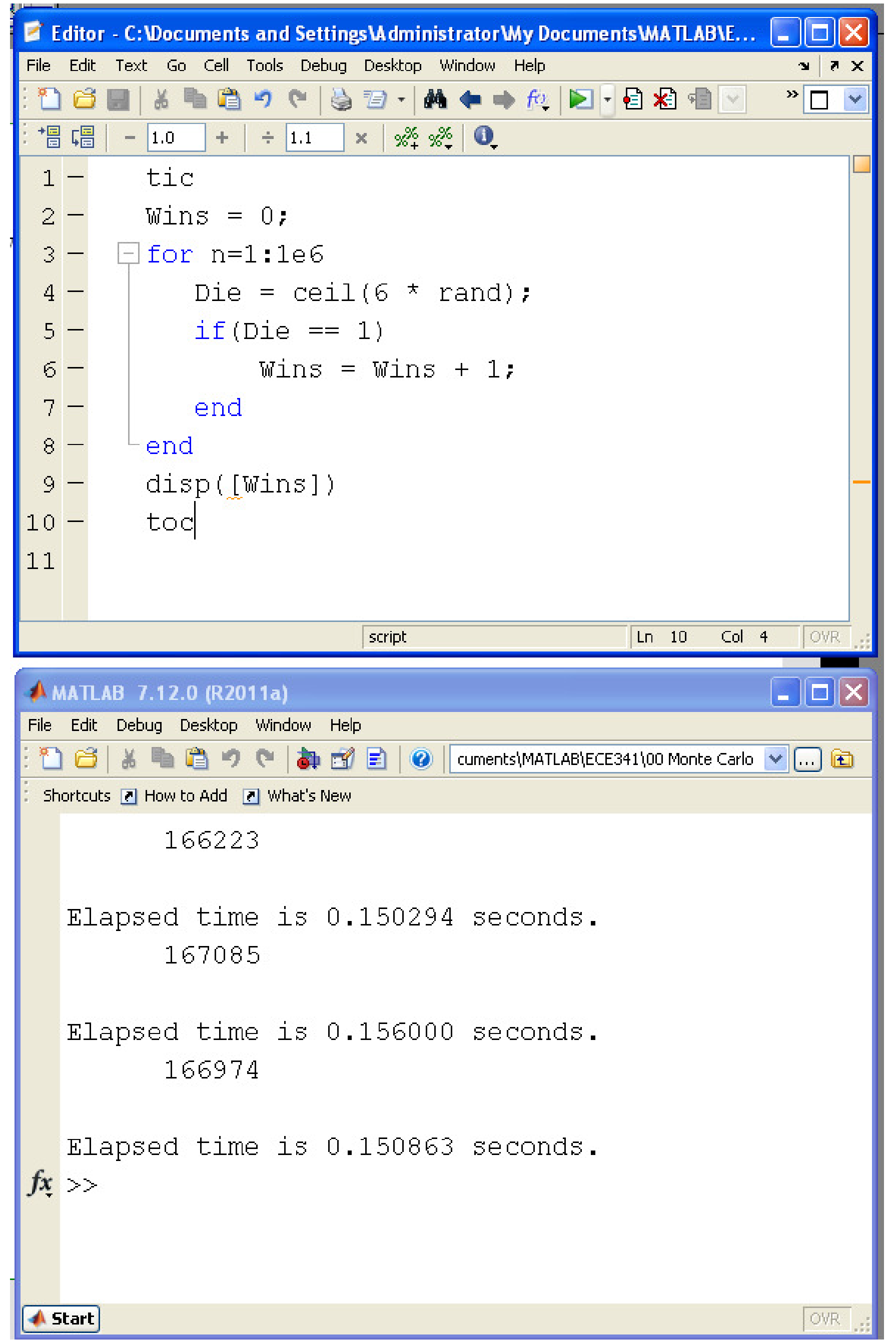
- Place previous code in a for-loop
- Count how many times you win

Result

- Roll the die 1 million times
- Number of games won was
  - 166223, 167085, 166974
- $p \approx 0.166$

Note:

- Results vary each trial (it's random)
- The result is approximate
- The variation tells us something
  - Future topic: t-Test



Case 2: max(d4, d6) vs d6

Problem

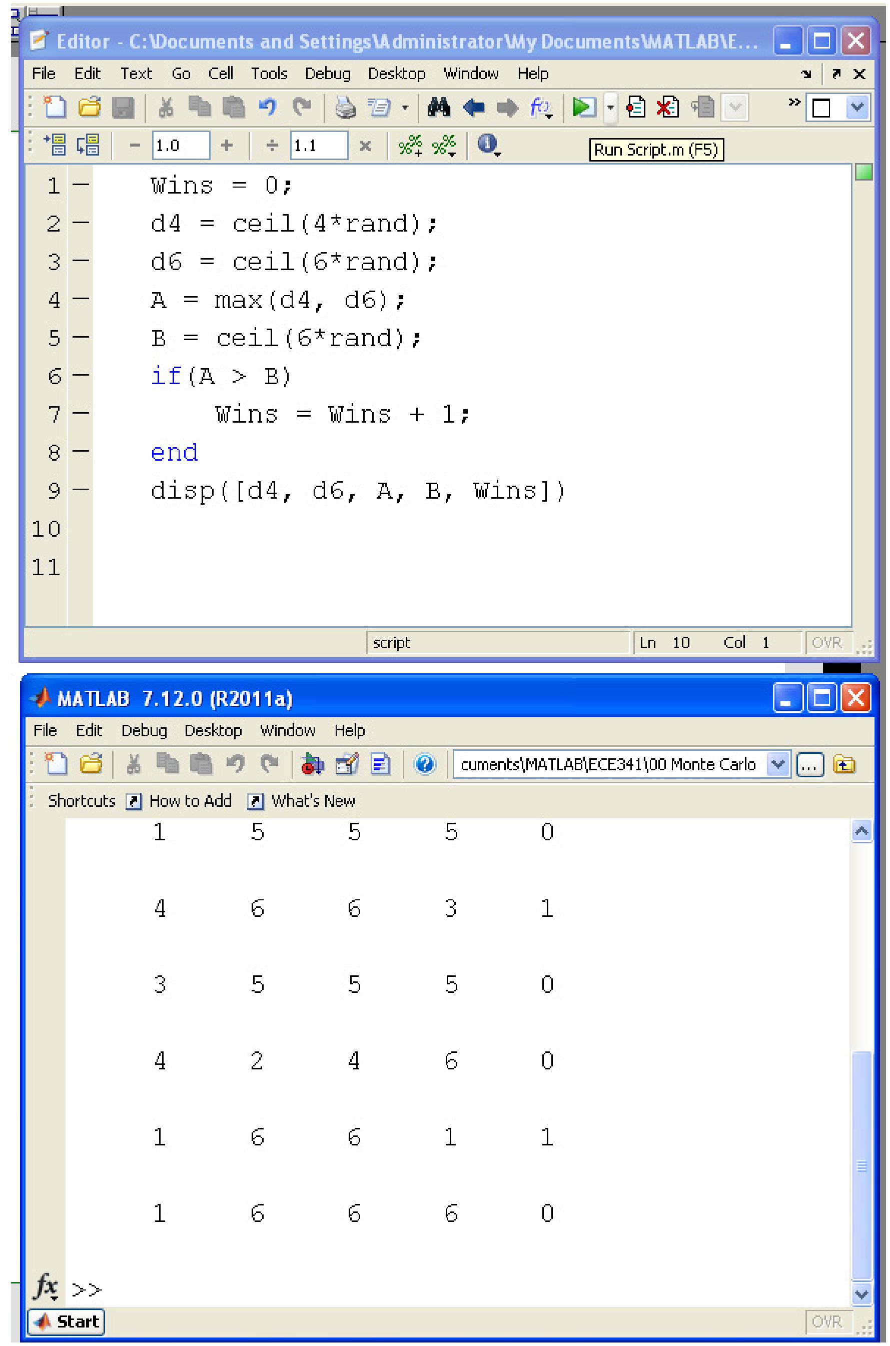
- Player A rolls a d4 and a d6
    - A's score is the maximum of the two
- Player B rolls a single d6
- Highest score wins
- B wins on ties

What is the probability that A will win?

Case 2 (cont'd)

Monte-Carlo Solution

- Start with playing the game one time
- Check that your code works
    - Col#1: d4 is random over [1,4]
    - Col#2: d6 is random over [1,6]
    - Col#3: A is the maximum of (d4, d6)
    - Col#4: B is random over [1,6]
    - Col #5: A wins when A > B



Case 2 (cont'd)

Once that works, repeat 1 million times

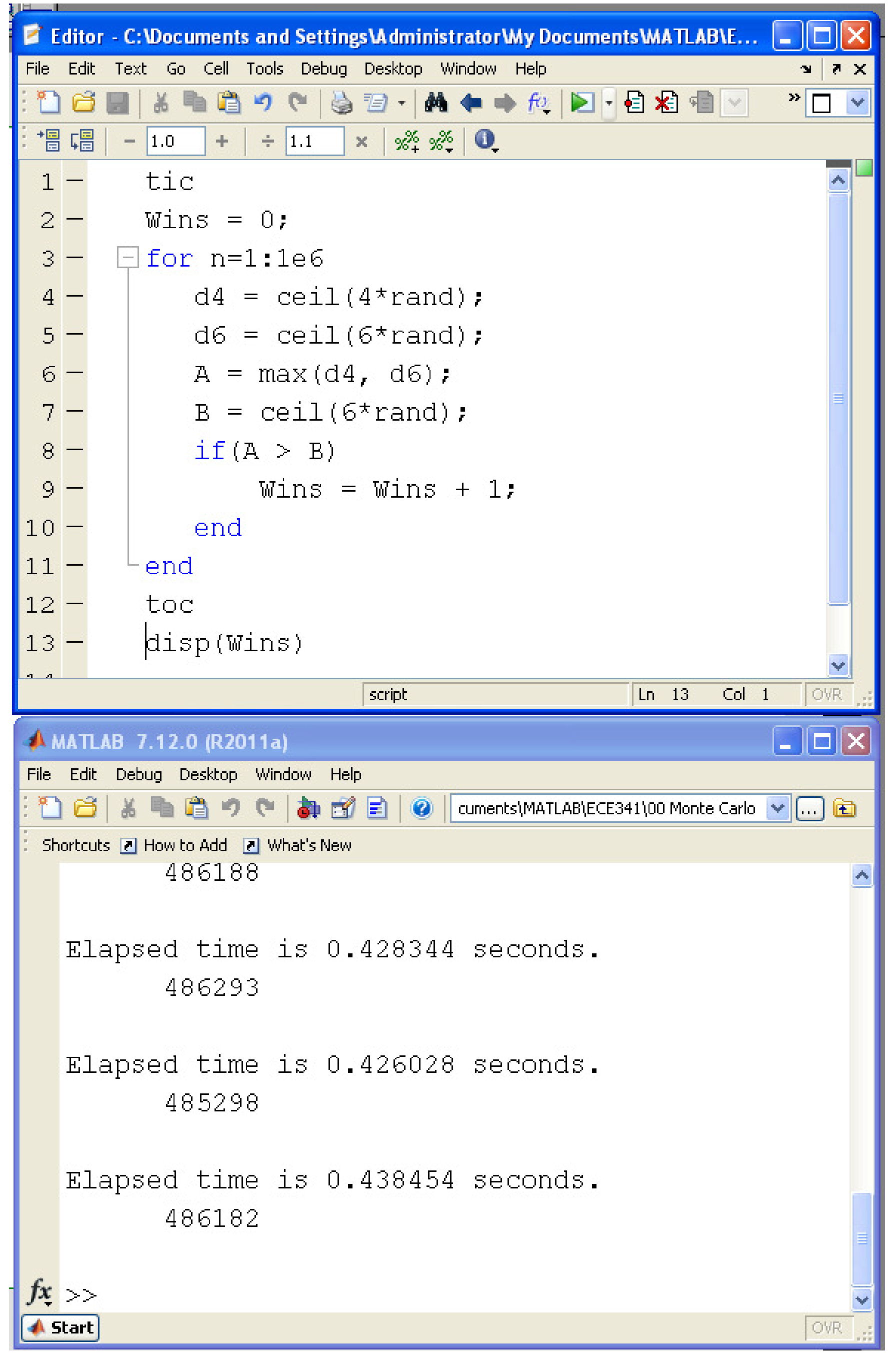
- Place previous code in a for-loop
- Count the number of successes
  - A wins

Result:

- {486188, 486293, 485298, 486182}
- About a 48.6% chance that A wins

Note

- Results are different each run
- Answers are approximate
- The variation tells us something
  - t-Test
  - Future lecture



Case 3: 5-Game Match

Assume A and B are playing a match

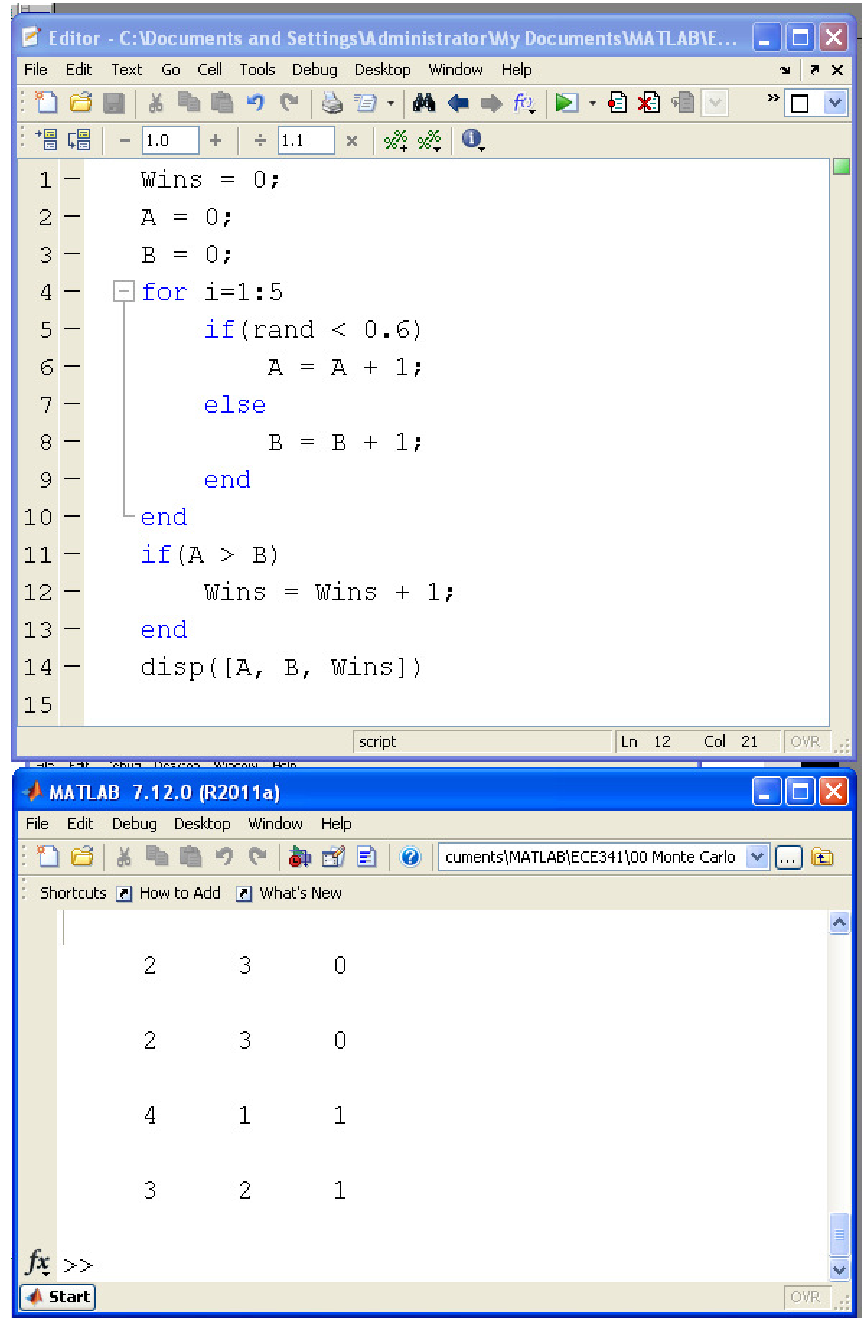
- Each match consists of 5 games
- A has a 60% chance of winning any given game
- Similar to basketball NBA finals

What is the probability that A will win the match?

Case 3 (cont'd)

Monte-Carlo Solution

- Start with playing one game
- Repeat 5 times (a match)
- Record who won the match
- Check that your code is working
  - Wins for A and B vary each match
  - A + B = 5 (5 game match)
  - A wins when A > B



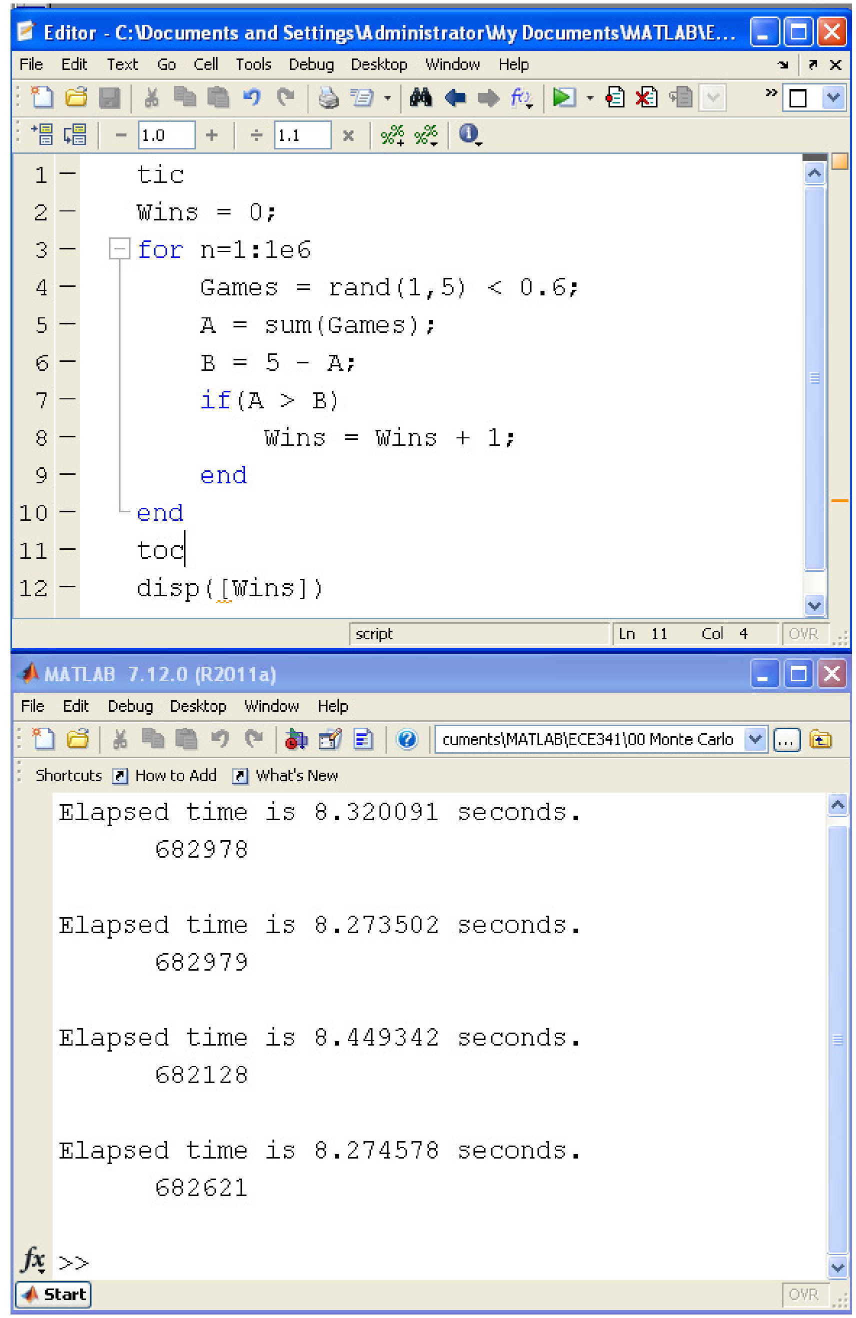
Case 3: (cont'd)

Now repeat 1 million times

- Code modified
- Another way to play a 5-game match

Result

- A wins about 68.2% of the time

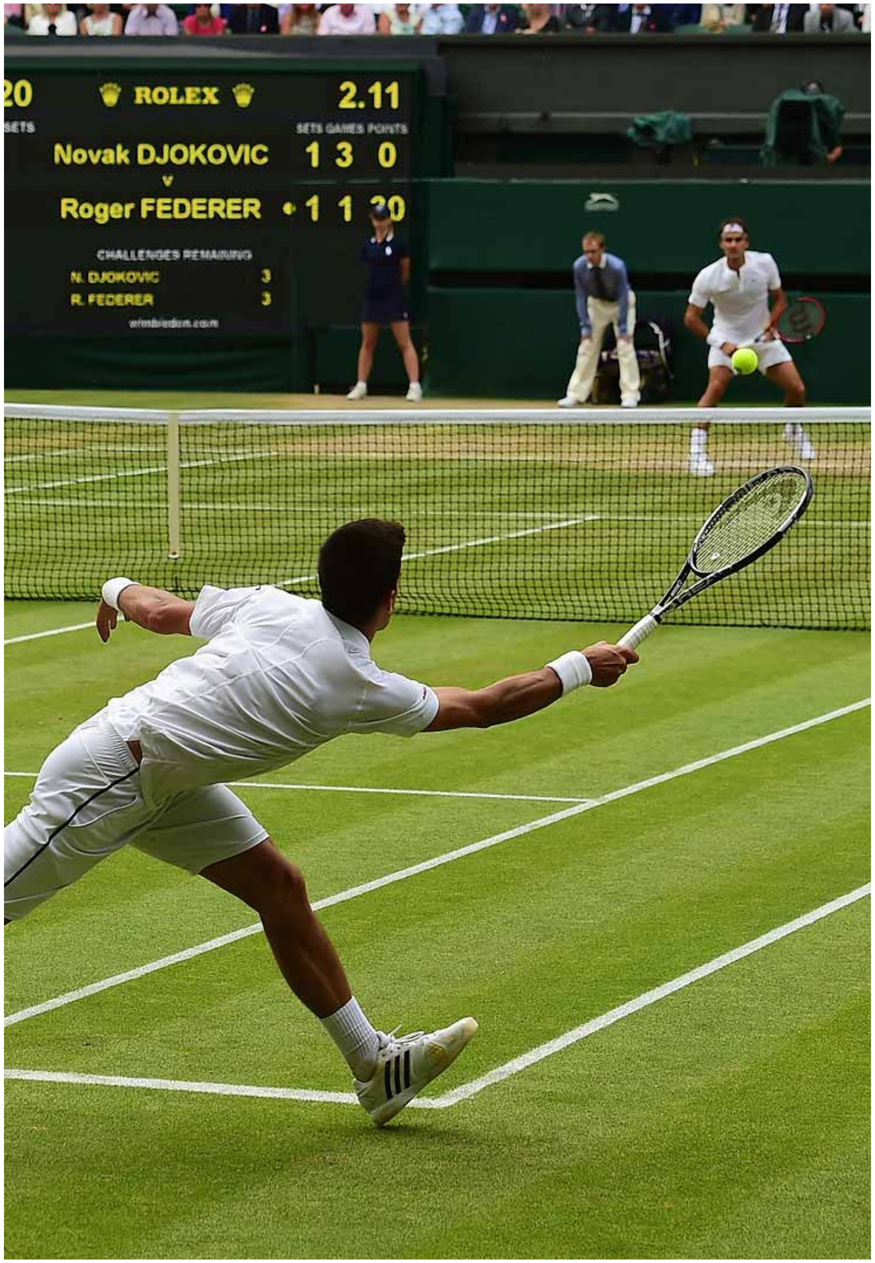


Case 4: Win by 3 Match

Assume A and B are playing a match

- A has a 60% chance of winning any given game
- The match is over when one player is up by 3 games
- Similar to tennis (win by 2)

What is the probability that A will win the match?



Case 4 (cont'd)

Monte-Carlo Solution

- Start by playing a single match
- The for-loop is replaced with a while-loop

```matlab
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])
```

Test your code

- 1st match, A loses 4 games to 7
- 2nd match, A wins 4 games to 1
- 3rd match, A wins 3 games to 0

```
     4     7     0

     4     1     1

     3     0     1

     6     3     1

>>
```

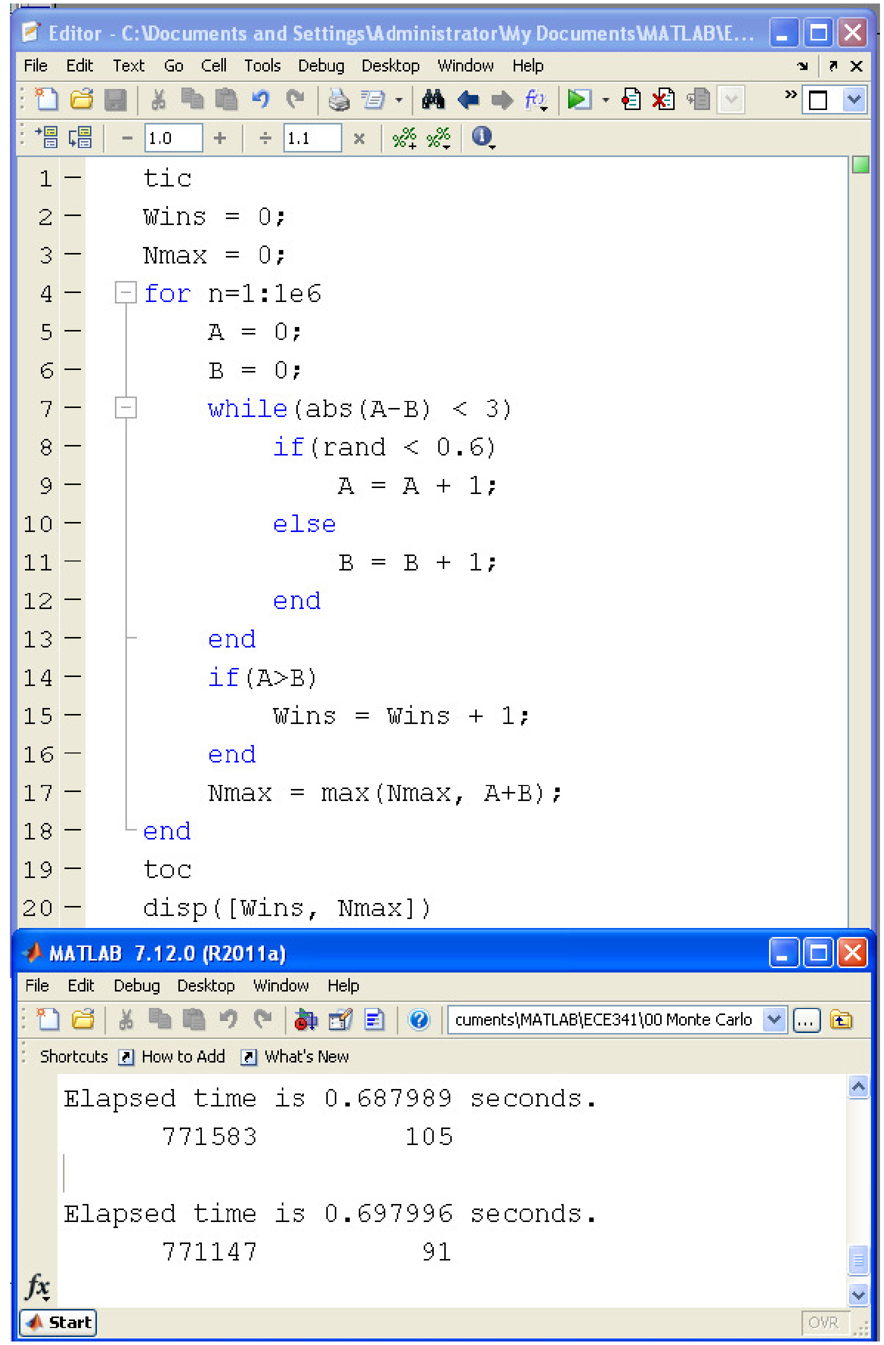
Case 4: (cont'd)

Now play 1 million matches

- Place the previous code inside a for-loop
- Count how many times A wins the match
- Also record the longest match

Result

- A wins 77.1% of the time
- The longest match was 105 games
  - Can be infinite in theory
  - TV hates this format



Case 5: Rolling Dice (Farkle)

Roll six 6-sided dice (6d6)

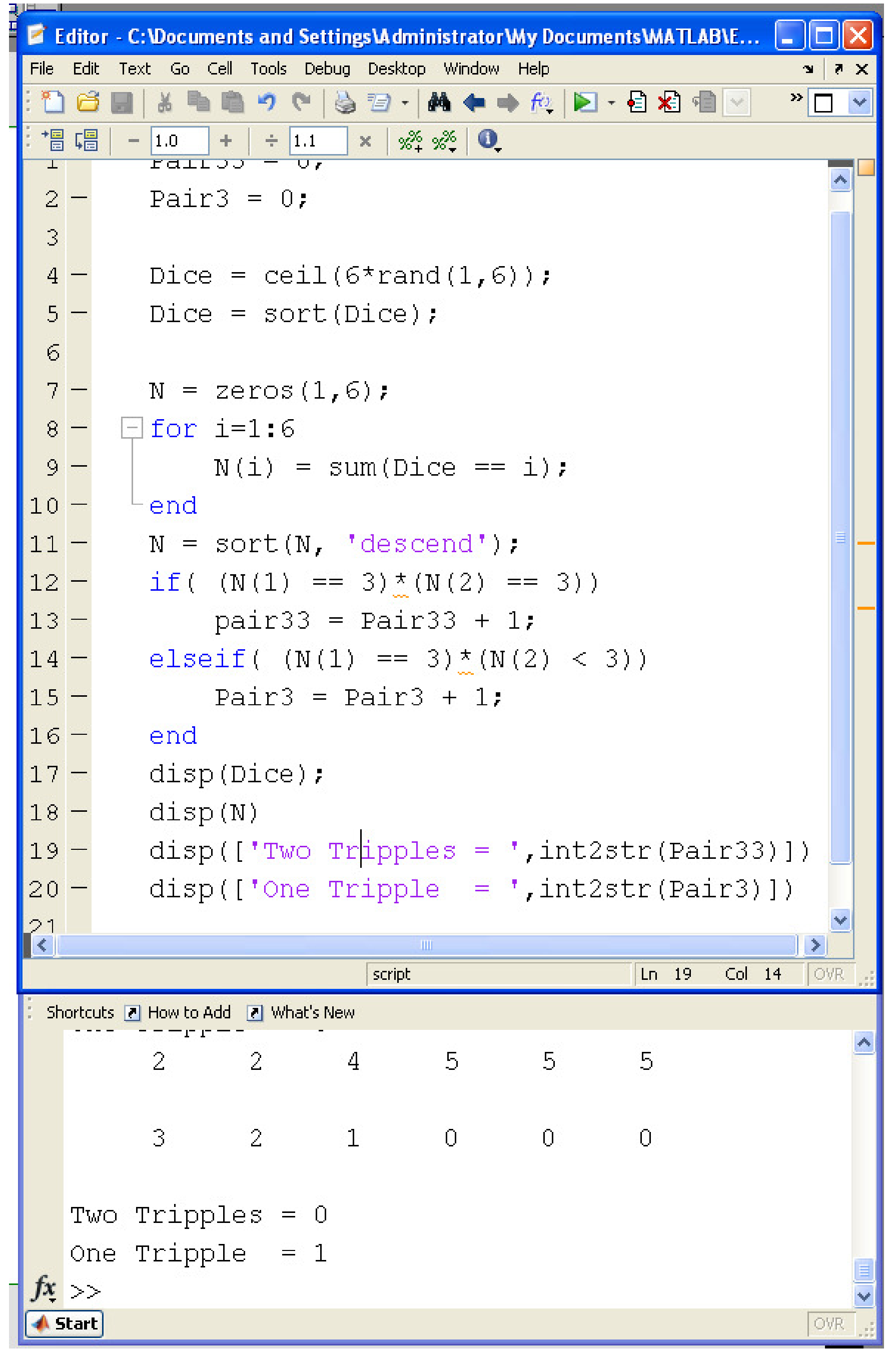
What is the probability of getting

- Two triples: xxx yyy?
- One triple: xxx abc or xxx aab

Case 5 (cont'd)

Monte-Carlo Solution

- Write a program to play the game one time
- Check your code
  - Dice are six 6-sided dice (6d6)
  - N is sorted frequency of numbers
  - Three 5's count as a triple



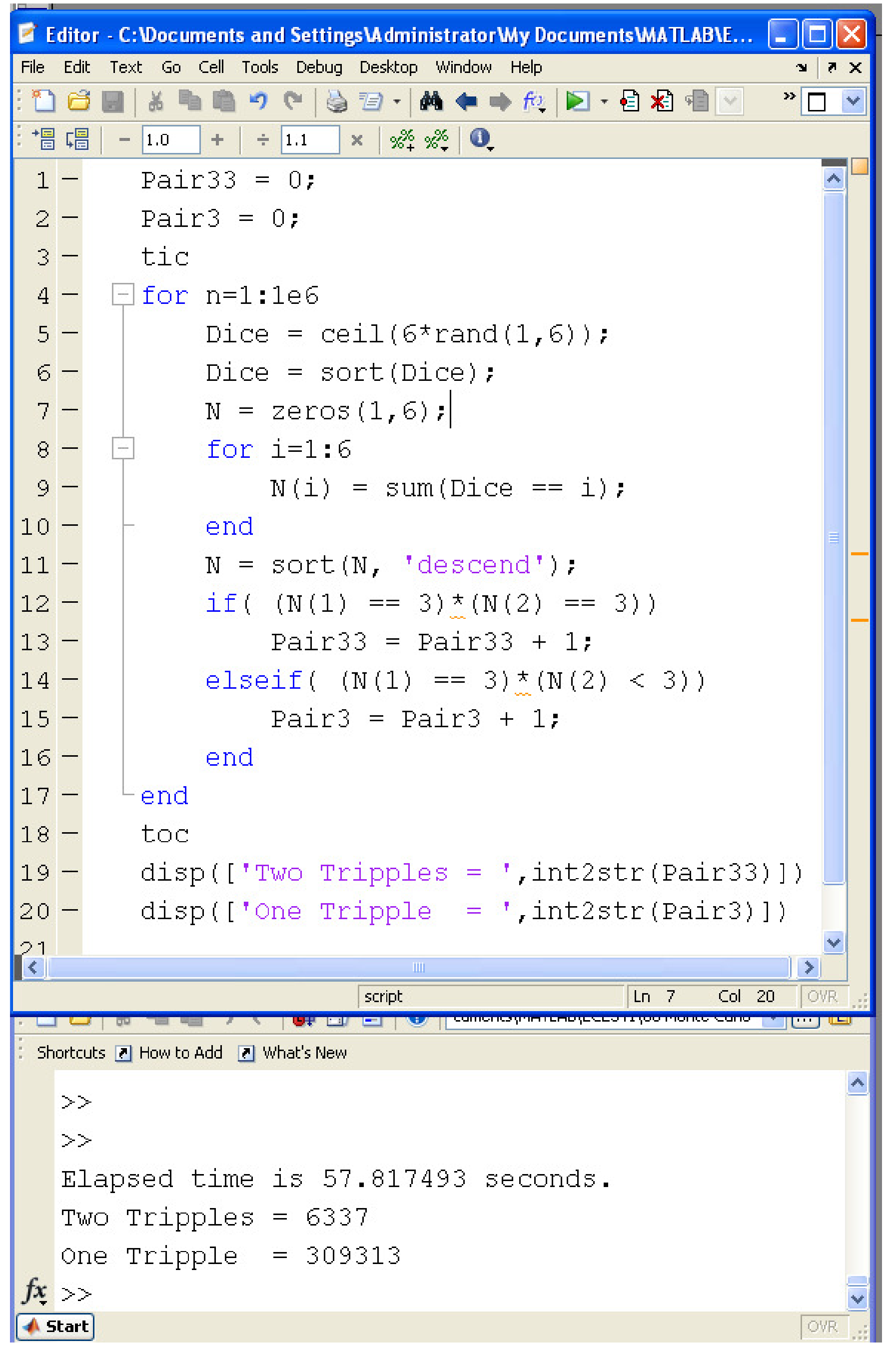
Case 5: (cont'd)

Once that works, roll the dice 1 million times

- Count the number of times you roll
  - Two triples: xxx yyy
  - One triple: xxx aa b or xxx abc

Result

- 6337 times I rolled two triples
  - p = 0.006337
  - approximately
- 309323 times I got one triple
  - p = 0.309323
  - approximately




```
Pair33 = 0;
Pair3 = 0;
tic
for n=1:1e6
    Dice = ceil(6*rand(1,6));
    Dice = sort(Dice);
    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;
    elseif( (N(1) == 3)*(N(2) < 3))
        Pair3 = Pair3 + 1;
    end
end
toc
disp(['Two Tripples = ',int2str(Pair33)])
disp(['One Tripple  = ',int2str(Pair3)])
```



```
>>
>>
Elapsed time is 57.817493 seconds.
Two Tripples = 6337
One Tripple  = 309313
>>
```

Case 6: Poker

In a game of poker

- Start with a deck of 52 cards
- Shuffle the deck
- Deal out 5 cards
    - 5-card stud poker

What is the chance of being dealt

- A full-house?
    - xxx yy
- 3-of-a-kind?
    - xxx ab

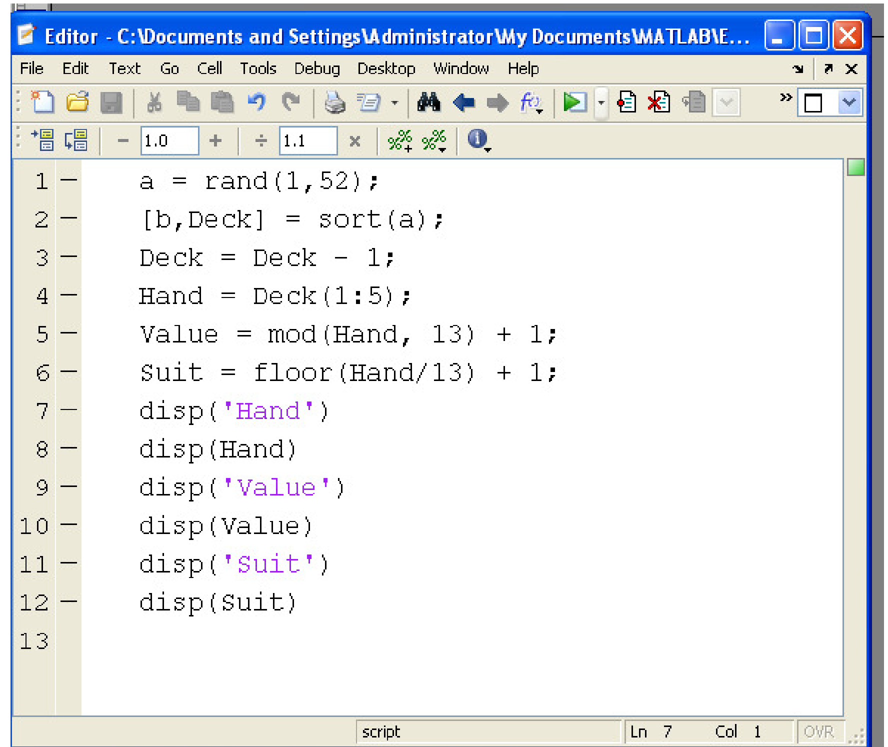
Case 6 (cont'd)

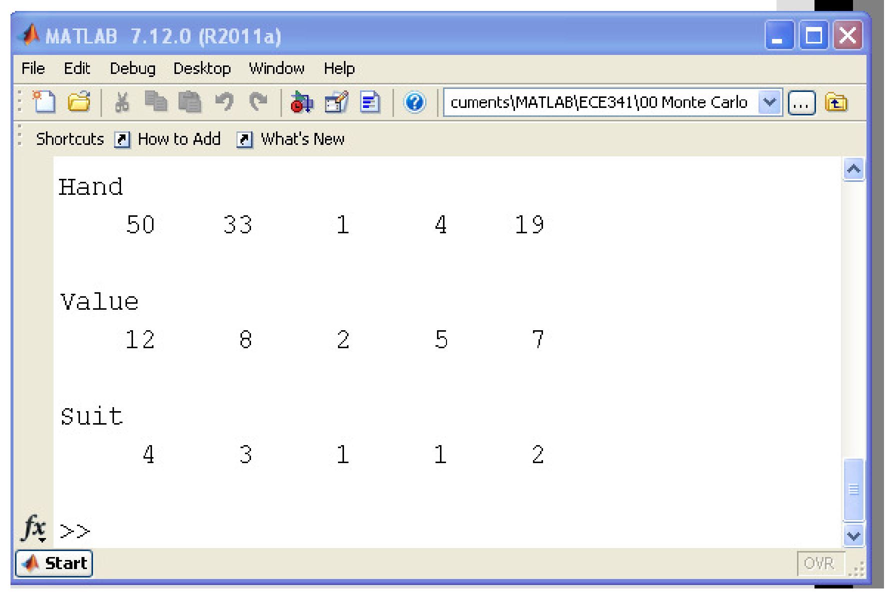
Start by shuffling a deck

- Pick 52 random numbers
- Sort these numbers
- The sort order is the deck
    - Cards 1..52
    - Minus one: 0..51
- Your hand is the first 5 cards of the deck

In this example

- Hand = #50, #33, #2, #4, #19
    - Q Spades
    - 8 Hearts
    - 2 Clubs
    - 5 Clubs
    - 7 Diamonds





Case 6 (cont'd)

Next, check what kind of hand it is

- Count the frequency of each type of card
  - Ace through King
- Sort in descending order
  - N(1) is most frequent card
  - N(2) is second most frequent

In this example

- N(1) = 3
  - There are three 5's
- N(2) = 1
  - There is one seven
- N(3) = 1
  - There is one nine

```
Pair32 = 0;
Pair3 = 0;
a = rand(1,52);
[b,Deck] = sort(a);
Deck = Deck - 1;
Hand = Deck(1:5);
Value = mod(Hand, 13) + 1;
Suit = floor(Hand/13) + 1;
N = zeros(1,13);
for i=1:13
    N(i) = sum(Value == i);
end
N = sort(N, 'descend');
if((N(1)==3)*(N(2)==2))
    Pair32 = Pair32 + 1;
elseif(N(1)==3)
    Pair3 = Pair3 + 1;
end
disp(Value)
disp(N(1:5))
disp([Pair32, Pair3])
```

```
7     9     5     5     5

3     1     1     0     0

0     1

>>
```

Case 6 (cont'd)

Now determine the type of hand you have

- N(1)=3 + N(2)=2: full-house
- N(1)=3 + N(2)=1: 3 of a kind

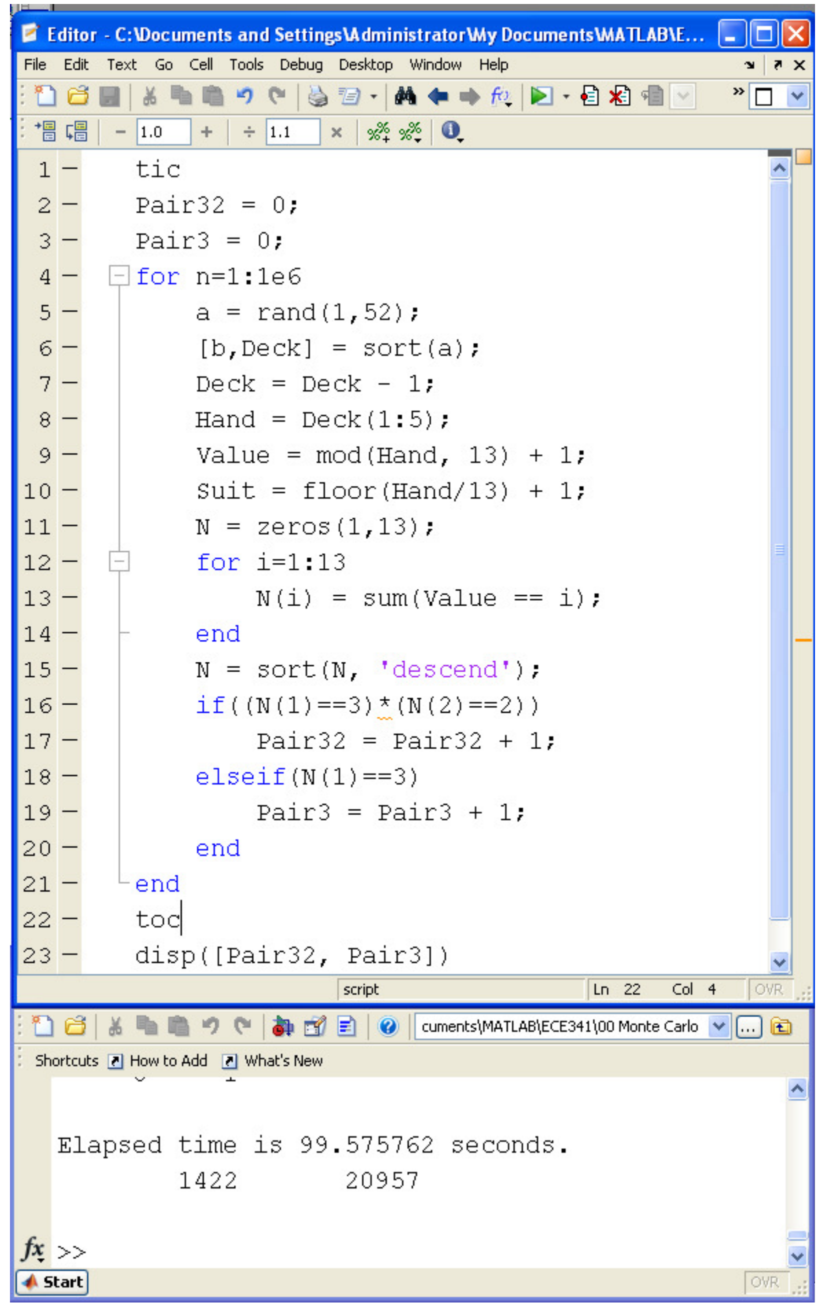
Similar logic for other types of hands

Then repeat 1 million times

- 1422 full-house
  - p = 0.001422
- 20957 3-of-a-kind
  - p = 0.020957

Note:

- Results are different each time
- It's random



Summary

The probability of an event is defined as the frequency that event happens as the number of trials goes to infinity.

This leads to a Monte-Carlo experiment

- Write a program to play a game one time
- Then, repeat a million times
    - (or some large number)
- Count the number of times the event happens
- The probability of the event is then
    - The number of times the event happened
    - Divided by the number of trials
    - (approximately)