Introduction to Matlab

Becoming familiar with MATLAB

- The console
- The editor
- The graphics windows
- The help menu
- Saving your data (diary)

Downloading Matlab

Matlab is free to use for all NDSU students (NDSU pays for a site license). To use the free version, you need to register with Matlab using your NDSU email address (xxxx@ndsu.edu).

- Note: your NDUS email address won't work. NDUS is for all schools in North Dakota (UND, BSC, etc.) The site license only works for NDSU.

If you choose to download Matlab for your personal computer, you'll get options for dozens of toolboxes to include. I personally only use

- Controls Systems toolbox (general)
- Signals and Systems toolbox (general)

If you go onto graduate school, you might want to include some of the more specialized toolboxes. You don't need them for now.

General environment and the console

When you start up Matlab, the screen should look something like this:

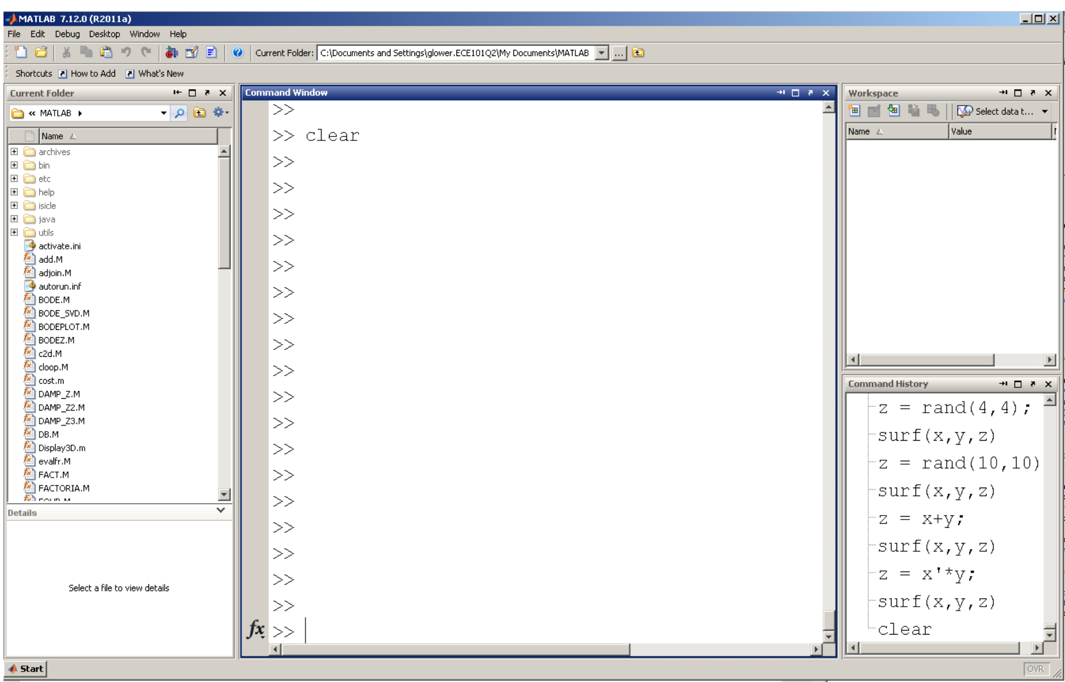


Figure 1: Intial Screen with you Start Matlab

Usually, I close all of the windows except the Command Window. Matlab can be used like a calculator - with some of the command options listed in the appendix. For example, if you want to compute

```
( 2 + 3 ) * 5
```

You type it the way it looks:

```
>> (2 + 3)*5

ans =   25
```

In Matlab, you can save the results and use them later - like the memory functions of your calculator. Valid names for your variables have to start with a letter. Note that Matlab is case sensitive. For example

```
>> a = (2+3)*5

a =  25

>> b = 1.3 * a^3

b =  2.0313e+004
```

Matrices in Matlab

Matlab is also a matrix language. An nxm matrix is an array of numbers arranged in n rows and m columns

$$A_{2x3} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

The syntax to input a matrix are:

```
[            start of a matrix
,            next column (a space also works)
;            next row
]            end of matrix.
```

For example, to input a 2x3 matrix for A:

```
A = [1,2,3 ; 4,5,6]

    1     2     3
    4     5     6
```

Matrix Addition: When adding matrices, each element gets added. You can only add matrices of like dimensions.

```
A = [1,2,3 ; 4,5,6]

    1     2     3
    4     5     6

B = [2,2,2 ; 3,3,3]

    2     2     2
    3     3     3

C = A + B
```

```
   3      4      5
   7      8      9
```

Matrix Multiplication: When multiplying matrices, the inner dimension must match

$$C_{mn} = A_{mx} B_{xn}$$

Element (i,k) of matrix C is computed as:

$$c_{ik} = \Sigma a_{ik} b_{kj}$$

Sample Matlab Code

If you terminate a line with a semi-colon, the result is computed but isn't displayed. If you leave off the semi-colon, the results is displayed. For example:

```
x = 2*pi;            % result is computed and stored in x but isn't displayed
x = 2*pi             % ditto but the result is displayed.

   6.2832
```

If you want a different format for the display, you can use the 'sort', 'long', and 'shorteng' commands:

```
format short
pi

   3.1416

format long
pi

  3.141592653589793


format shorteng
pi^30

  821.2893e+012
```

Matrices

- [start of matrix
-] end of matrix
- , next element
- ; next row

```
A = [1,2,3]

   1      2      3
```

```
B = [1,2,3;4,5,6]

    1     2     3
    4     5     6

C = A'

    1
    2
    3

D = zeros(1,3)

    0     0     0

E = rand(3,2)

   0.5860    0.0835
   0.2467    0.6260
   0.6664    0.6609
```

for loops:

```
x = zeros(1,5);
for i=1:5
  x(i) = i*i;
  end
x

x =

    1     4     9    16    25
```

while

if

if else end

Plotting Functions in Matlab:

Matlab has some pretty good graphics capabilities.

Matlab Plot Command	x axis	y axis	type of function
`plot(x,y)`	linear	linear	$y = ax + b$
`semilogx(x,y)`	log()	linear	$y = a \cdot e^{bx}$
`semilogy(x,y)`	linear	log()	$y = a + b\ln(x)$
`loglog(x,y)`	log()	log()	$y = a \cdot b^{x}$
`subplot(abc)`	Create 'a' rows, 'b' columns of graphs. Starting at #c		

Table 1: Types of Plots in Matlab

For example, plot the function

$$y = 3x + 4$$

```
x = [0:1:10]';
y = 3*x + 4;

subplot(131)
plot(x,y); % connect the points with a line

subplot(132)
plot(x,y,'.');       % plot a dot at each point

subplot(133);
plot(x,y,'.-');      % connect the points and add a dot
```

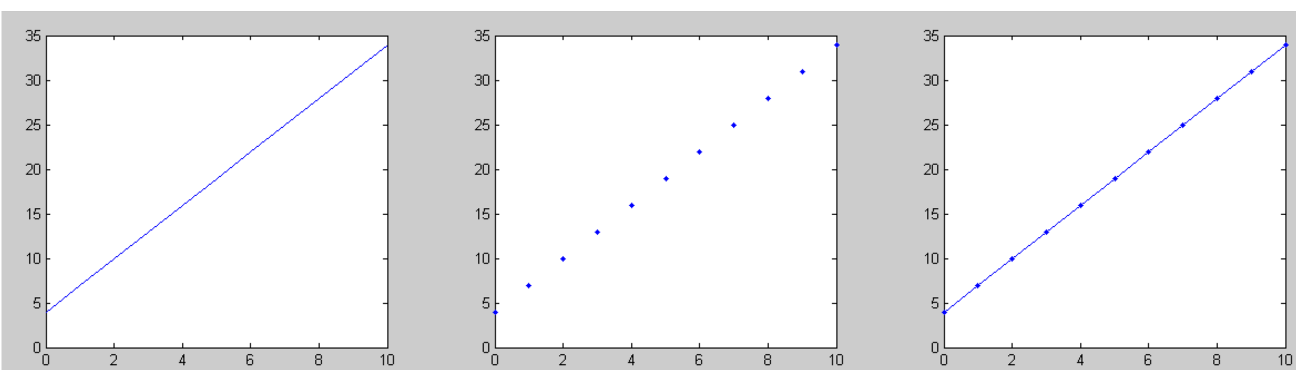


Figure 2: Results of subplots: line, dots, and dotted line

Plot how much money you'd have if you invested it for 10 years at

- 1% interest
- 3% interest
- 6% interest
- 9% interest

Three ways to do this:

```
t = [0:0.01:10]';
y1 = 1000 * exp(0.01*t);
y3 = 1000 * exp(0.03*t);
y6 = 1000 * exp(0.06*t);
y9 = 1000 * exp(0.09*t);

% Method #1
plot(t,y1,t,y3,t,y6,t,y9)

% Method #2
plot(t,[y1,y3,y6,y9])

% Method #3
plot(t,y1)
hold on
plot(t,y3)
plot(t,y6)
plot(t,y9)
hold off
```

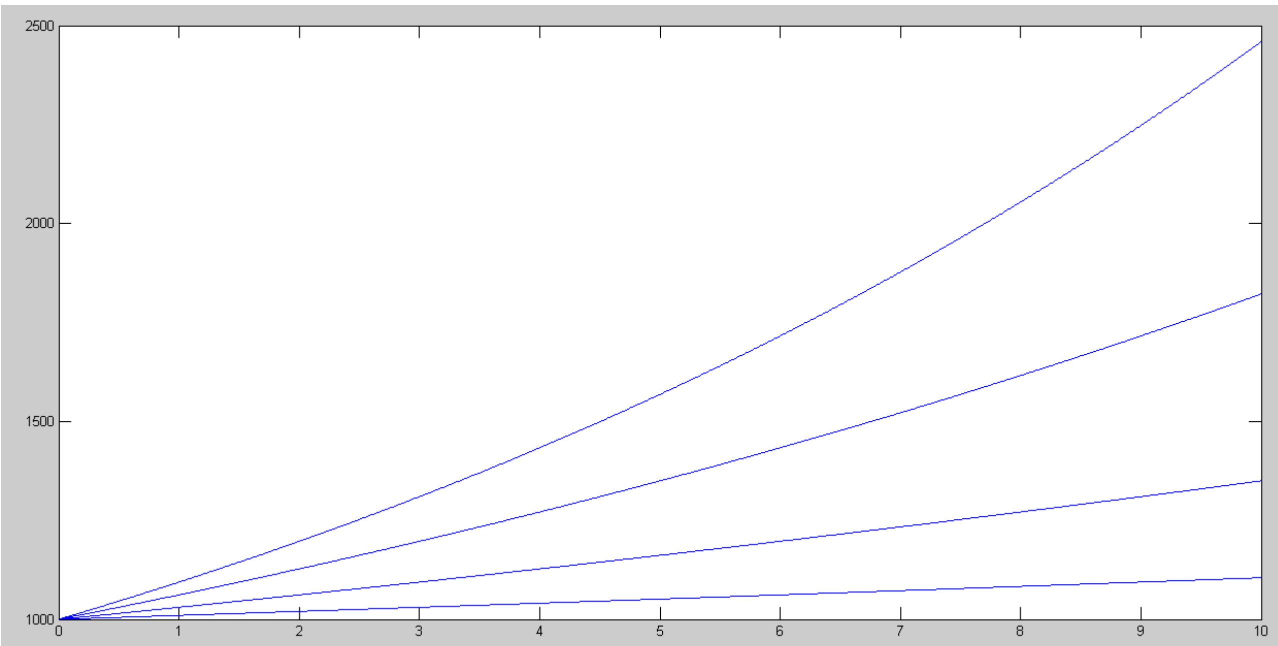


Figure 3: Plotting multiple lines on the same graph

If you invest at 9% interest, you have almost 2.5x more money after 10 years than you'd have at 1% interest.

Polynomials

- poly([a,b,c]) Give a polynomial with roots at (a, b, c)
- roots([a,b,c,d]) Find the roots of the polynomial

$$ax^3 + bx^2 + cx + d = 0$$

Example:

- Determine a polynomial with roots at (1, 2, 3)
- Plot that function,
- Verify that the roots are in fact at (1, 2, 3)

In Matlab:

```
poly([1,2,3])

    1     -6     11     -6
```

meaning

$$y = x^3 - 6x^2 + 11x - 6 = (x-1)(x-2)(x-3)$$

```
roots([1,-6,11,-6])

   3.0000
   2.0000
   1.0000


x = [0:0.01:4]';
y = x.^3 - 6*(x.^2) + 11*x - 6;
plot(x,y);
grid on
```

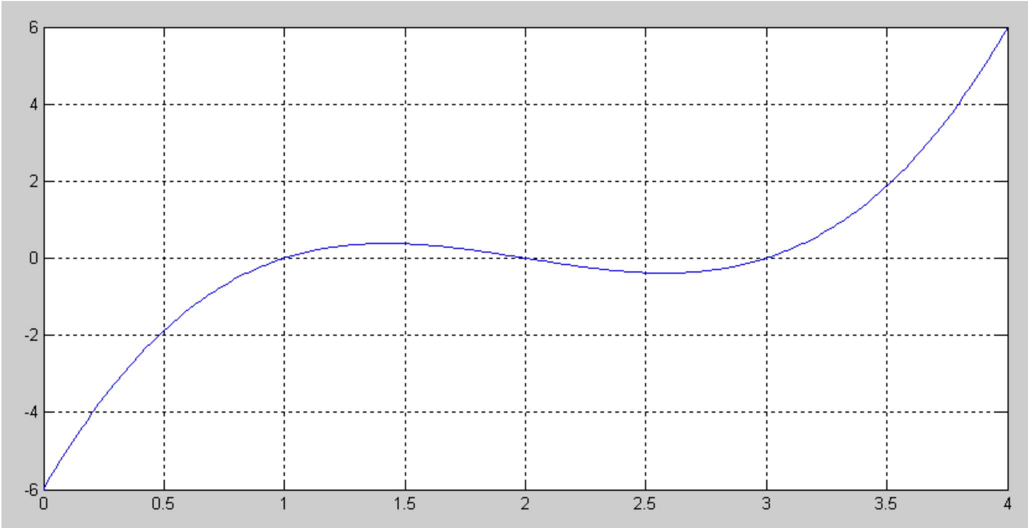


Figure 4: Plotting a polnomial in Matlab. The zero crossings are the roots to the polynomial

3-D Plots

```
mesh(z)
```

Draw a 3-d mesh for the array 'z' with the height being the value in the array.

Example

$$z = x^2 + y^2$$

```
x = [-3:0.1:3]';
y = [-3:0.1:3]';
size(x)

ans =

    61     1

z = zeros(61,61);
for i=1:61
  for j=1:61
     z(i,j) = x(i)^2 + y(j)^2;
     end
  end

mesh(z)
```

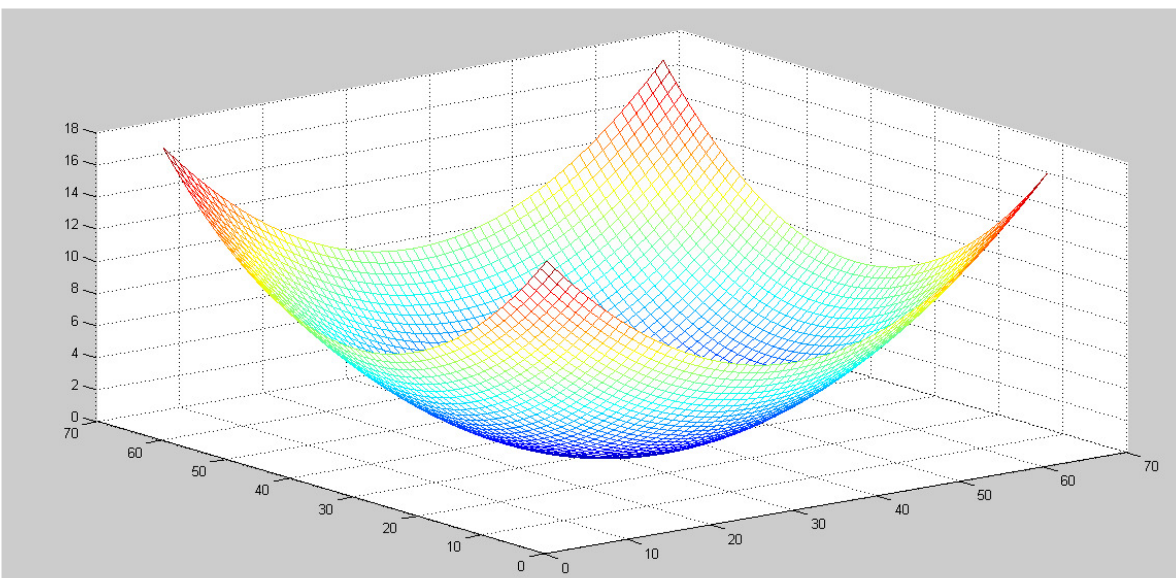


Figure 5: The mesh() funciton draws a 3-d mesh

Another pretty plot:

$$r = \sqrt{x^2 + y^2}$$

$$z(x, y) = \left(\frac{\sin(r)}{r}\right)$$

```
x = [-10:0.1:10]';
y = [-10:0.1:10]';
size(x)

  201     1

z = zeros(201,201);
for i=1:201
  for j=1:201
     r = sqrt(x(i)^2 + y(j)^2);
     z(i,j) = sin(r) / (r + 0.000001);
     end
  end
mesh(z)
```

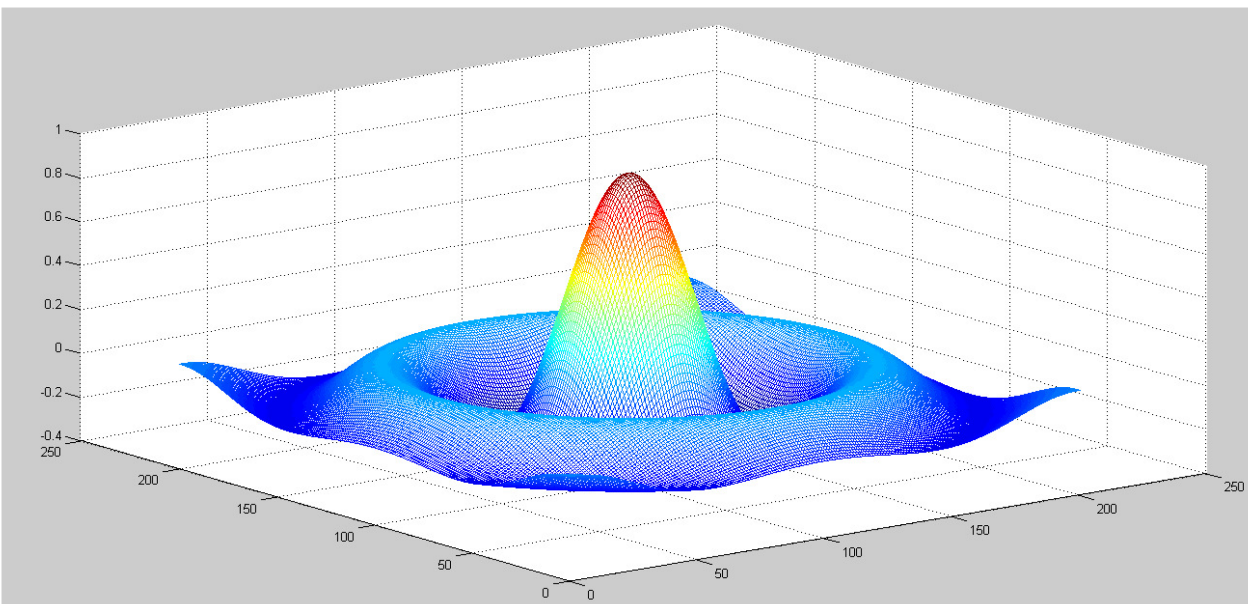


Figure 6: Another example of a mesh() plot in Matlab

Random Numbers: Rolling Dice

```
rand                     random number: (0,1)
randn         standard normal random #
```

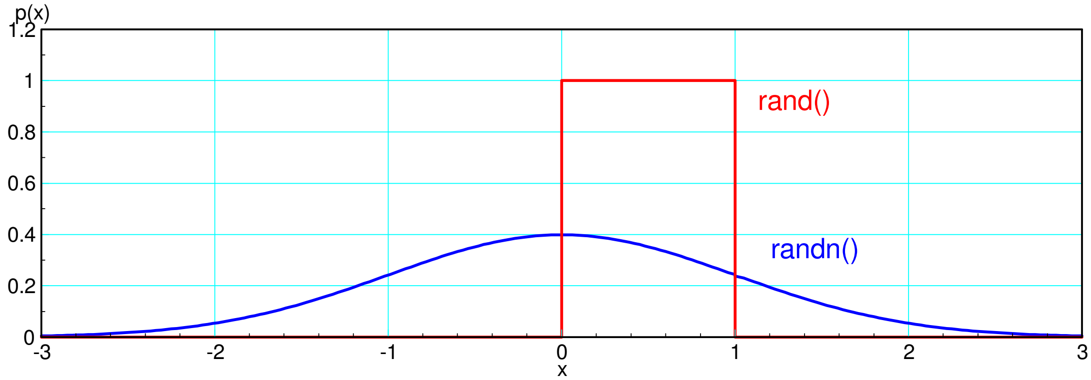


Figure 7: rand() produces a uniform distribution. rnadn() produces a Normal distribution

```
rand(1,5)              1x5 matrix of random #

ceil( 6*rand )         6-sided die

ceil( 8*rand(1,3) )    3d8

sum( 6*rand(5,1) )     sum of 5d6  (level 5 fireball)
```

Example: Generate 1000 random numbers and plot them

```
x = rand(1,1000);
plot(x)
```

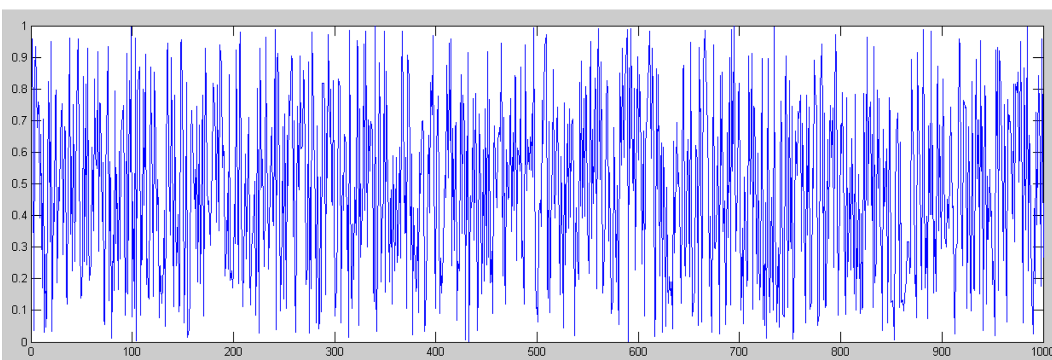


Figure 8: rand() produces a random number in the range of (0,1)

If you sort the numbers, you can see the cumulative distribution function

- approximately
- (rand < 0.6) approximately 60% of the time

```
y = sort(x);
plot(y)
```

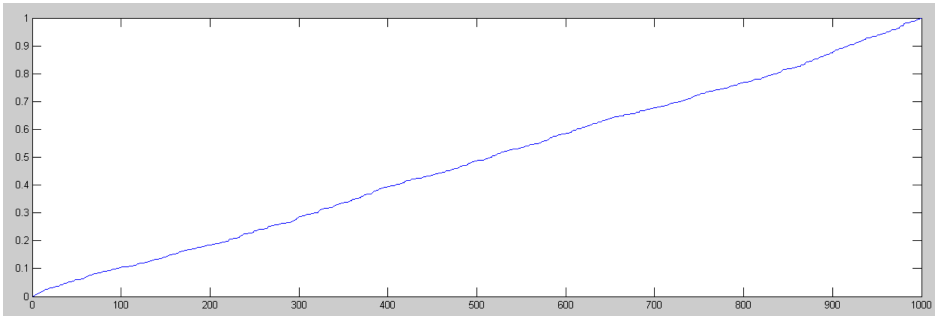


Figure 9: Sorting the results of rand shows that 20% of the time, rand() produces a number less than 0.2

Matlab Scripts

Instead of typing in the command window, you can use a Matlab Script. When executed, (green arrow in the icon bar), scripts act like you input those instructions in the command window.

This is useful

- It lets you build up your code
- It lets you modify existing code
- You can rerun the code over and over

For example,

- The window on the left (Matlab script) computes the damate for summing five 6-sided dice (i.e. a level 5 fireball for D&D fans).
- The window on the right (command window) shows the print-portions when the scipt is run
- Clicking on the run command (green arrow on the top of the script) results in the script executing several times.

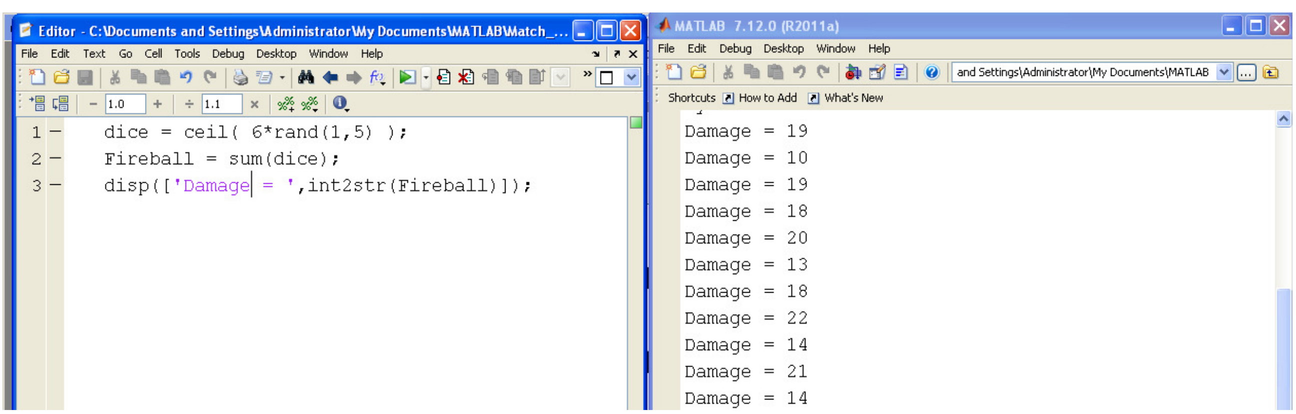


Figure 10: Matlab Script (left) and Command (right).

For-Loops

```
for i=1:100
    statements
    end
```

For-Loops repeat a set of instructions a fixed number of times. For example, assume A and B are playing a game.

- A has a 60% chance of winning any given game.
- How many games will A win in a 100 game match?

This can be done with a for-loop:

- (rand < 0.6) is a boolean operation: it's 1 if true, 0 if false.
- Each game, A has a 60% chance of earning one point (a win)
- 100 games are played in each match (the for-loop).

The resulting Matlab code (in a script window) and results of executing it are as follows:

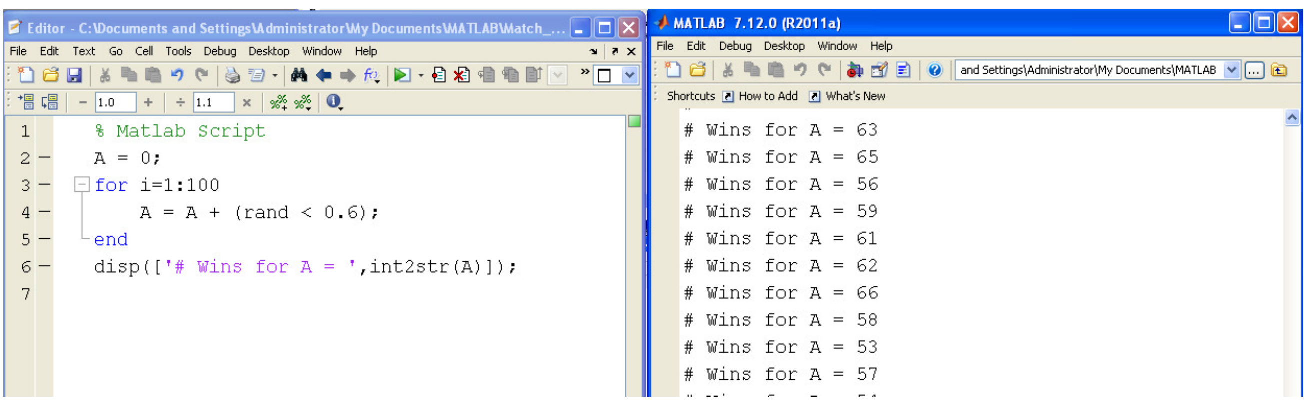


Figure 11: For-Loop: (rand<0.6) is executed 100 times each pass.

Note:

- 0 is false, 1 is true
- On average, A wins 60 games out of 100
- In any given match, A could win 0..100 games

For-Loops & Monte-Carlo Simulations: 20-game match

What is the probability of A winning X games in a 20 game match?

To solve this problem, change the for-loop from 100 to 20. Also keep track of how many times you won X games and then repeat this experiment 100,000 times (termed a Monte-Carlo simulation)

One Matlab program and it's results are as follows:

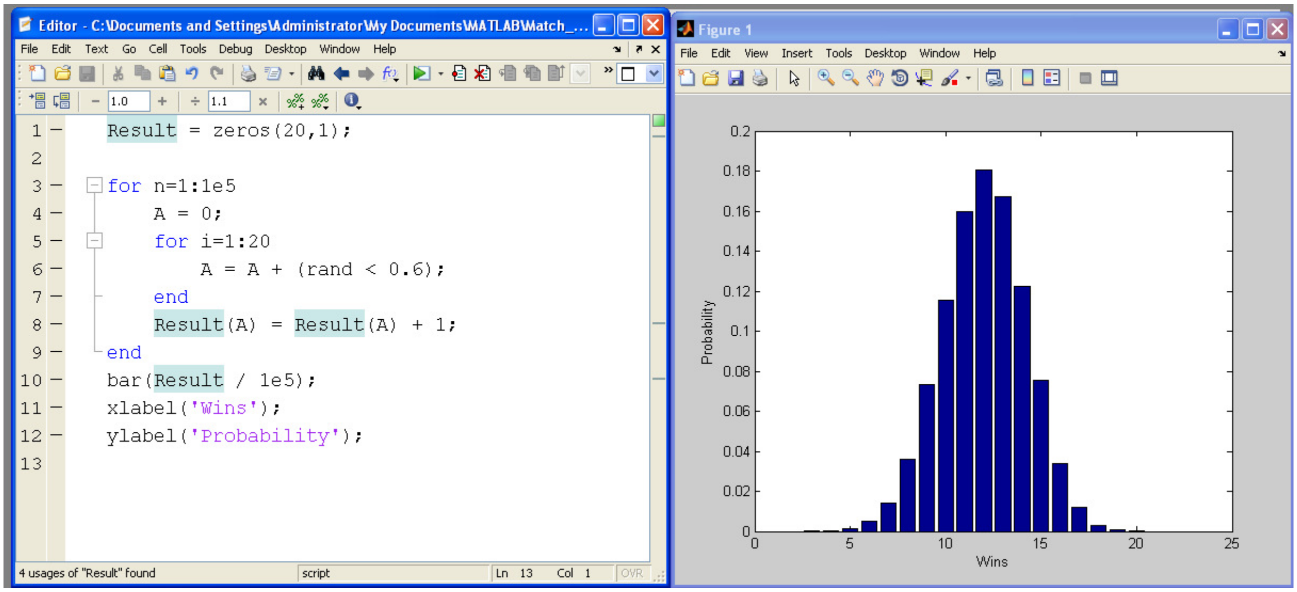


Figure 12: Monte-Carlo simultion: Repeat an experiment 100,000 times

Note: The resulting distribution is a bell-shaped curve. This is called a *Normal Distribution* and is something you'll see over and over and over again.

For-Loops: Level-8 Fireball

Using a similar technique, for-loops can determine the probability of

- Doing 40 damage with a level-8 fireball, or
- Doing 40+ damage with a level-8 fireball.

To solve these problems,

- First, cast a single level-8 fireball
- Determine if the damage was 40 or more than 40
- Then repeat 100,000 times, recording how many times the result was N

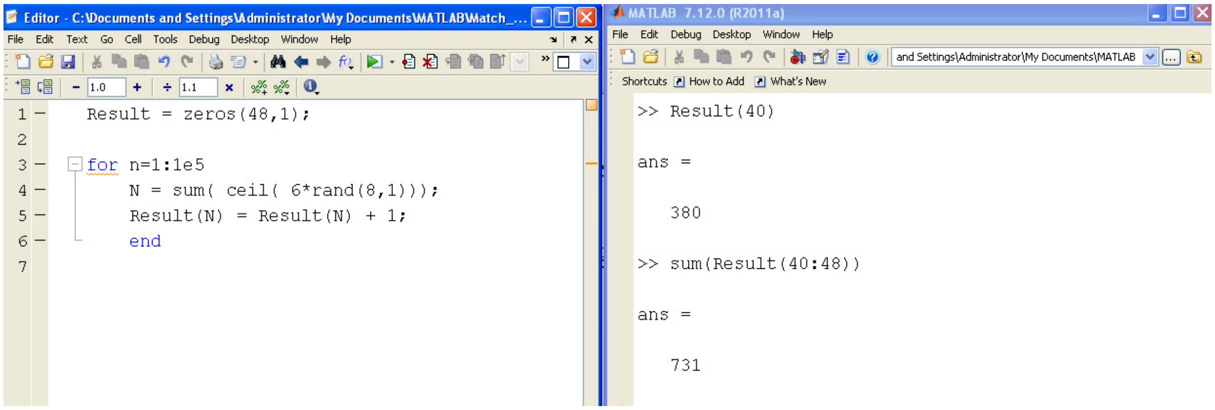


Figure 13: Using for-loops to compute the damage of level-8 fireball

The resulting probability is (approximately)

- p = 380 / 100,000 to do 40 damage exactly
- p = 731 / 100,000 to do 40 or more damage.

If-Statements

```
if(boolean statement)
   statements to execute
   end
```

If-statements allow you to execute a set of instructions based upon a condition. Valid boolean statements are as follows:

```
(N == 3)
(N > 3)
(N >= 3)
(N != 3)
(N >= 3)*(N <= 7)
((N < 3) | (N > 7))
```

If-statements are useful for checking what conditions hold. For example, suppose A and B are playing a match

- Each match consists of 5 games
- A has a 60% chance of winning any given game
- Whoever wins 3+ games wins the match

What is the chance that A wins the match?

Matlab Script

```
Wins = 0;
for n=1:1e5
    A = 0;
    for i=1:5
        if(rand < 0.6)
            A = A + 1;
        end
    end
    if(A >= 3)
        Wins = Wins + 1;
    end
end
disp(['# of wins for A = ',int2str(Wins)]);
```

Matlab Command Window

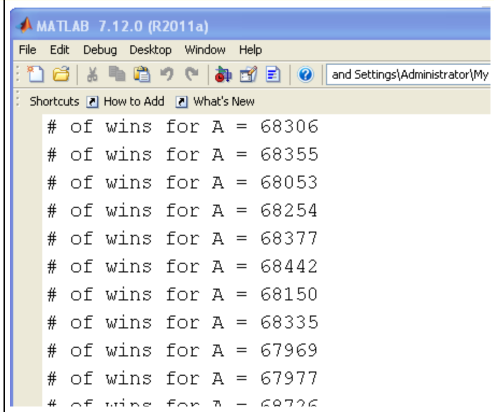


Figure 14: Matlab script and the results in the command windoe

Solution:

- Monte-Carlo simulation of 100,000 matches
- A wins about 68% of the time

If-Statements (cont'd)

Player A casts a level-8 fireball (8d6)

- What is the chance of doing 40 damage?
- What is the chance of doing 40+ damage?

Solution:

- Monte-Carlo Simulation
- Cast 100,000 fireballs
- Count how many times you do 40 damage
- Count how many times you do 40+ damage

Matlab Script

```
eq40 = 0;
gt40 = 0;
for n=1:1e5
    A = sum(ceil(6*rand(8,1)));
    if(A == 40)
        eq40 = eq40 + 1;
    end
    if(A >= 40)
        gt40 = gt40 + 1;
    end
end
disp([eq40/1e5, gt40/1e5])
```

Matlab Command Window

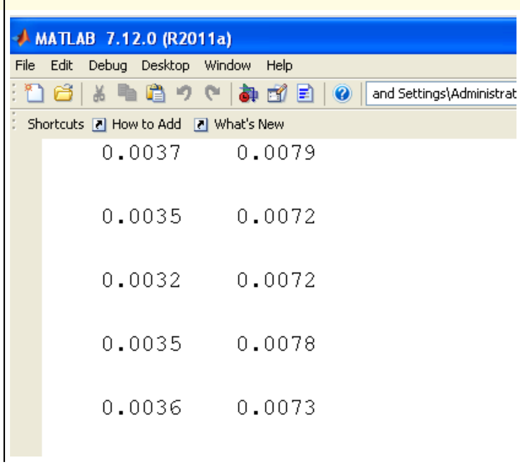


Figure 15: Matlab code for a Monte-Carlo simulation along with the results for different trials

Note:

- There are multiple ways to solve any given problem

If-Elseif-Else Statements

```
if(boolean statement)
   instrucitons #1
elseif(boolean statement)
   instructions #2
else
   instuctions #3
end
```

Execute a set of statements

- If the first condition is true, execute instructions #1 and skip the rest
- If the first condition is false , then check the second boolean statement
- If that is also false, excute instructions #3

Example: A and B play a game

- A has a 30% chance of winning (+1 point)
- A has a 50% chance of a tie (+1/2 point)
- A has a 20% chanec of losing (0 points)

Matlab Script

```
Wins = 0;
X = rand;
if(X < 0.3)
    Wins = Wins + 1;
elseif(X < 0.8)
    Wins = Wins + 0.5;
else
    Wins = Wins + 0;
end
disp(['Wins = ',num2str(Wins)]);
```

Matlab Command Window

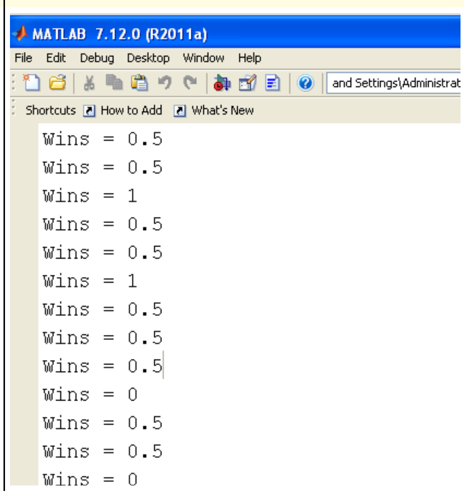


Figure 16: Matlab script for a Monte-Carlo simulation along with the results for different trials

If-Else Statements (cont'd)

A die is loaded:

- 10% of the time, it always rolls a 6
- The rest of the time it's a fair die

What is the chance of doing X damage with a level-5 fireball?

Matlab Script

```
Damage = zeros(30,1);
for n=1:1e5
    X = 0;
    for i=1:5
        if(rand < 0.1)
            d6 = 6;
        else
            d6 = ceil(6*rand);
        end
        X = X + d6;
    end
    Damage(X) = Damage(X) + 1;
end
bar(Damage / 1e5);
xlabel('Damage');
ylabel('probability');
```

Matlab Plot Window

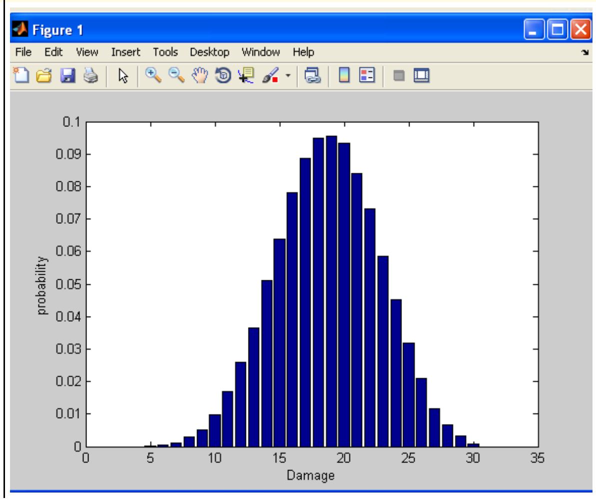


Figure 17: Monte-Carlo simulation for rolling five loaded dice

Note:

- Once again, the result is a bell-shaped curve
- This is *The Central Limit Theorem*
- Almost all distributions converge to this shape

While-Loop

```
while(statement is true)
  do the following
  end
```

While-loops are useful for keeping a program going until a condition is met. This can be

- Keep a simulation until time > 10 seconds,
- Keep playing a game until someone wins,
- Keep itterating until the error is less than 0.001

Example: Count how many times you roll a die until you get a 1

Matlab Script

```
N = 0;
d6 = 0;
while(d6 ~= 1)
    d6 = ceil(6*rand);
    N = N + 1;
end
disp(['Number of Rolls = ',int2str(N)])
```

Matlab Command Window

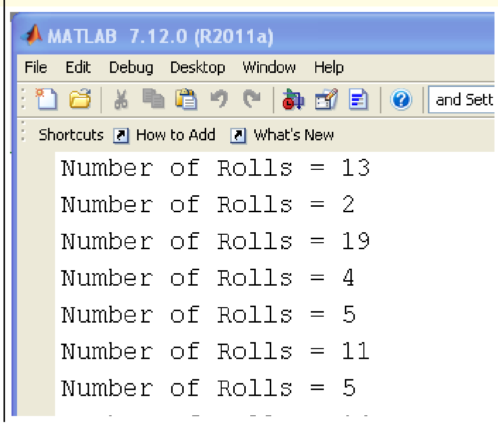


Figure 18: Using a while-loop to determine the number of rolls until you get a 1

Note: in theory, N is unlimited. It could take millions of rolls until y ou get a 1.

While-Loop (cont'd)

What is the chance that it will take 7 or more rolls to get a 1?

Solution:

- Count how many times you roll a die until you get a 1
- Keep track of the number of times this is 7 or more
- Repeat 100,000 times (i.e. run a Monte-Carlo Simulation)

Matlab Script	Matlab Command Window

```
X = 0;
for n=1:1e5
    N = 0;
    d6 = 0;
    while(d6 ~= 1)
        d6 = ceil(6*rand);
        N = N + 1;
    end
    if(N >= 7)
        X = X + 1;
    end
end
disp(['# successes = ',int2str(X)])
```

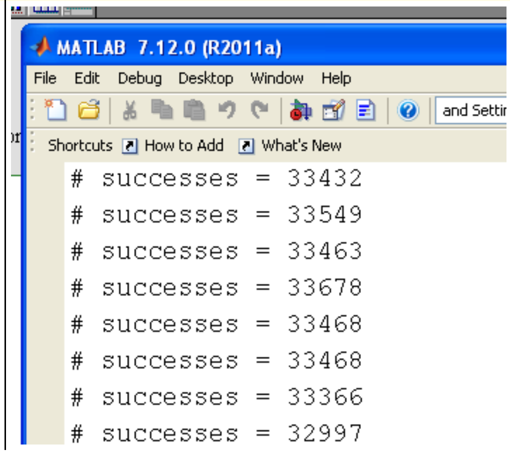


Figure 19: Monte-Carlo simulation to determine the odds of rolling the dice 7 or more times

In 100,000 trials

- It took 7 or more rolls 33,343 times
- There is about a 33.34% chance it will take 7 or more rolls to get a 1

While-Loop (cont'd)

Player A and B are playing a match

- Player A has a 60% chance of winning any given game
- When a player is up 3 games, the match is over

What is the chance player A wins the match?

Solution: Play a single match

- If A wins, A gains 1 point.
- If A loses, A loses 1 point.
- Keep playing until A is up 3 or down 3

Matlab Script

```
A = 0;
while(abs(A) < 3)
    if(rand < 0.6)
        A = A + 1;
    else
        A = A - 1;
    end
end
if(A == 3)
    Wins = 1;
else
    Wins = 0;
end
disp(['Wins = ',int2str(Wins)])
```

Matlab Command Window

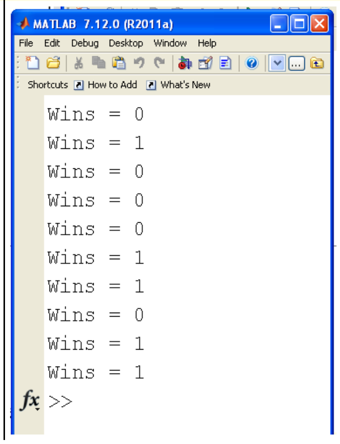


Figure 20: Matlab code for playing a gain one time

Now play 100,000 matches

- A wins about 77% of the time with this format
- TV hates this format since a match can take a very long time

Matlab Script

```
Wins = 0;
for n=1:1e5
    A = 0;
    while(abs(A) < 3)
        if(rand < 0.6)
            A = A + 1;
        else
            A = A - 1;
        end
    end
    if(A == 3)
        Wins = Wins + 1;
    end
end
disp(['Wins = ',int2str(Wins)])
```

Matlab Command Window

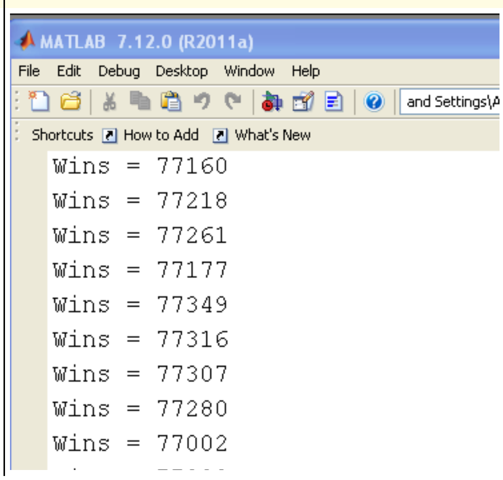


Figure 21: Matlab code for playing 100,000 matches

Sort Command

Sort data in either

- Ascending order (default), or
- Descending order

Return

- The sorted data (one returned parameter)
- The sort order (two returned parameters)

Matlab Command Window

```
X = rand(1,5)
X =     0.1744    0.7055    0.4435    0.1654    0.0592

a = sort(X)
a =     0.0592    0.1654    0.1744    0.4435    0.7055

a = sort(X, 'descend')
a =     0.7055    0.4435    0.1744    0.1654    0.0592

[a,b] = sort(X)
a =     0.0592    0.1654    0.1744    0.4435    0.7055
b =        5         4         1         3         2
```

Figure 22: Example of using the sort() funciton in Matlab

Sort Command & Poker

Shuffle a deck of 52 playing cards

Deal out 5 cards (poker hand)

- Hand is the card number (1..52)
- Value is Ace..King (1..13)
- Suit is Club / Diamond / Heart / Spade (1...4)

Result:

- {4s, Jh, 3s, 7h, 6s}
- {4h, Jh, Ac, Jc, 5d}

Matlab Script

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

Matlab Command Window

```
Hand =     42    36    41    32    44
Value =     4    11     3     7     6
Suit =      4     3     4     3     4


Hand =     29    36    13    10    17
Value =     4    11     1    11     5
Suit =      3     3     1     1     2
```

Figure 23: Using the sort() command to shuffle a deck of cards

Check for a Full-House

Problem: Determine if your hand is a full-house.

Solution:

- Count how many Aces, twos, etc. are in your hand (variable P)
- Sort P to determine the frequency of pairs in descending order

Matlab Script	Matlab Command Window

`Deck = rand(1,52);`
`[a,b] = sort(Deck);`
`Hand = b(1:5);`
`Value = mod(Hand, 13) + 1;`
`Suit = ceil(Hand / 13);`
 	
`P = zeros(1,13);`
`for i=1:13`
`    P(i) = sum(Value == i);`
`end`
`P = sort(P, 'descend');`
 	
`disp('--------')`
`disp(Value);`
`disp(P(1:5));`
	`--------`
	`V 2 2 10 7 4`
	`P 2 1 1 1 0`
	
	`--------`
	`V 11 5 5 12 4`
	`P 2 1 1 1 0`
	
	`--------`
	`V 4 3 1 12 8`
	`P 1 1 1 1 1`
	
	`--------`
	`V 4 4 1 4 2`
	`P 3 1 1 0 0`

Figure 24: Matlab code for dealing out five cards for a poker hand and checking for pairs

Note:

- Hand #1 has a pair of 2's
- Hand #2 has a pair of 5's
- Hand #3 has no pairs
- Hand #4 has three 4's

Now checkI

- if P(1) = 3 (three of a kind) and P(2) = 2 (pair) then you have a full-house

Matlab Script	Matlab Command Window

<pre>FH = 0;
 	
for n=1:1e5
    Deck = rand(1,52);
    [a,b] = sort(Deck);
    Hand = b(1:5);
    Value = mod(Hand, 13) + 1;
    Suit = ceil(Hand / 13);
 	
    P = zeros(1,13);
    for i=1:13
        P(i) = sum(Value == i);
    end
    P = sort(P, 'descend');
 	
    if((P(1)==3)*(P(2)==2))
        FH = FH + 1;
    end
end
disp(['Full House = ',int2str(FH)])</pre>
	<pre>Full House = 141
	Full House = 133
	Full House = 162
	Full House = 124
	Full House = 119
	Full House = 154</pre>

Figure 25: Monte-Carlo simulation for finding the probability of being dealt a full-house

Note:

- The number of full-houses you get in 100,000 hands of poker changes each time you run the Monte Carlo simulation (it's a random process)

With this, you can answer questions, such as

- What is the range in numbers I'll get?
- What is the actual probability of getting a full house?

This is something covered in ECE 341 Random Processes and week #12 in ECE 111.

Summary

Matlab is a fairly friendly computer language

You can use the command window as a calculator

- Adds, subtracts, multiplies, divides

Scripts allow you to try & modify code as you write it

For-loops let you run code multiple times

- Monte-Carlo simulations...

If-statements allow you to check for conditions

- If the sum is 25 or more...

While-loops let you run code until an event happens

- repeat until you roll a 1

Matlab Commands

Display

- format short display results to 4 decimal places
- format long display results to 13 decimal places
- format short e display using scientific notation
- format long e display using scientific notation

Polynomials

- poly(x)
- roots(x)
- conv(x,y)

Analysis

- sqrt(x) square root of x
- log(x) log base e
- log10(x) log base 10
- exp(x) e^x
- exp10(x) 10^x
- abs(x) |x|
- round(x) round to the nearest integer
- floor(x) round down (integer value of x)
- ceil(x) round up to the next integer
- real(x) real part of a complex number
- imag(x) imaginary part of a complex number
- abs(x) absolute value of x, magnitude of a complex number
- angle(x) angle of a complex number (answer in radians)
- unwrap(x) remove the discontinuity at pi (180 degrees) for a vector of angles
- sum(x) sum the columns of x
- prod(x) multiply the columns of x

Trig Functions

- sin(x) sin(x) where x is in radians
- cos(x) cos()
- tan(x) tan()
- asin(x) arcsin(x)
- acos(x) arccos(x)
- atan(x) arctan(x)
- atan2(y,x) angle to a point (x,y)

Probability and Statistics

- factorial(x) x!
- gamma(x) x!
- rand(n,m) create an nxm matrix of random numbers between 0 and 1
- randn(n,m) create an nxm matrix of random numbers with a normal distribution
- length(x) return the dimensions of x
- mean(x) mean (average) of the columns of x
- std() standard deviation of the columns of x

Display Functions

- plot(x) plot x vs sample number
- plot(x,y) plot x vs. y

Rolling Dice:

```
rand         generate a random number in the range of (0, 1)
rand(1,5)    generate a 1x5 matrix of random numbers in the range of (0, 1)

randn        generate a random number with a normal distribution with mean=0, std=1
```

Example: Generate a random number between 0 and 1:

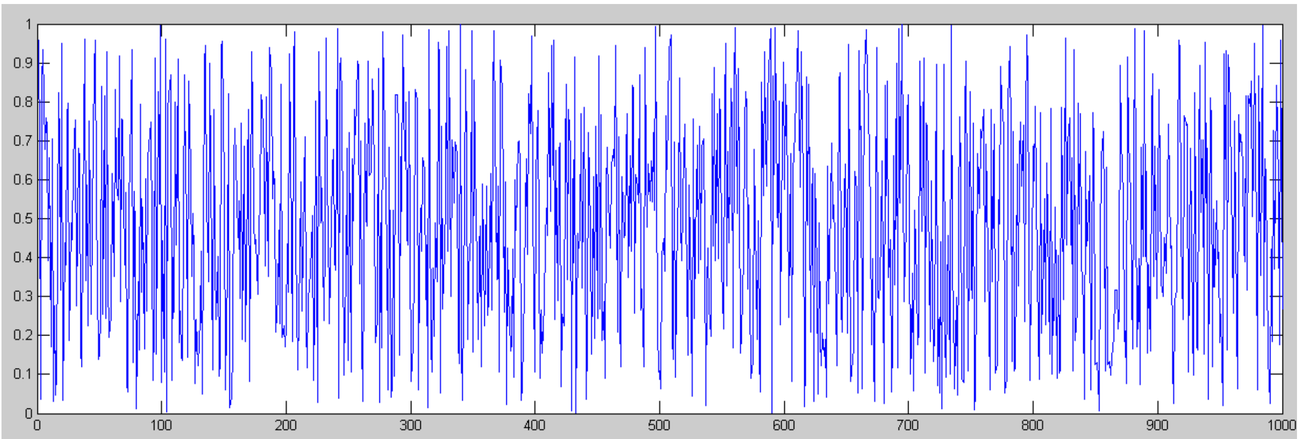
```
rand

   0.8147

rand

   0.9058
```

Generate 5 random numbers between 0 and 1

```
rand(1,5)

   0.8003    0.1419    0.4218    0.9157    0.7922
```

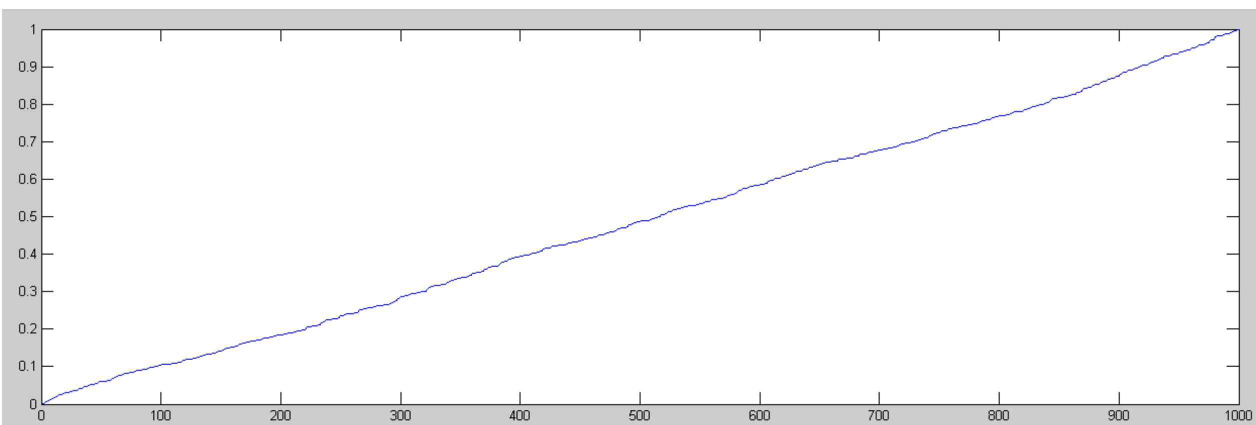
Generate 1000 random numbers and plot them

```
x = rand(1,1000);
plot(x)
```



If you sort the numbers, you can see the probability distribution function (approximately)

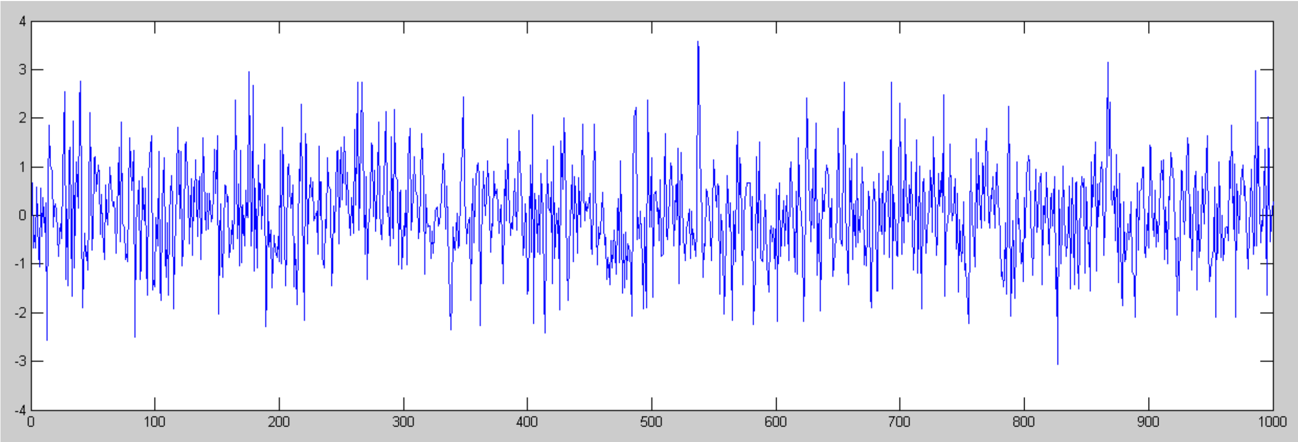
```
y = sort(x);
plot(y)
```



Each number from (0, 1) should have the same likelihood of coming up, meaning when you sort the numbers, the result should be a straight line

Compare this to a Normal distribution

```
x = randn(1,1000);
plot(x)
```



This is more what you'd see for

- Stock prices
- Temperatures
- Height of people,
- etc.

You have to watch which random number generator you're using. *rand()* is different from *randn()*

With this you can have fun with rolling dice:

Problem: Find the sum or rolling five six-sided dice. (5d6)

Solution: Generate five random numbers in the range of (0,1)

```
rand(1,5)

    0.7298    0.8908    0.9823    0.7690    0.5814
```

Looks right. Now generate five random integers in the range of (1,6)

```
dice = ceil( 6*rand(1,5) )

     3     6     2     4     4

dice = ceil( 6*rand(1,5) )

     1     4     3     1     3
```

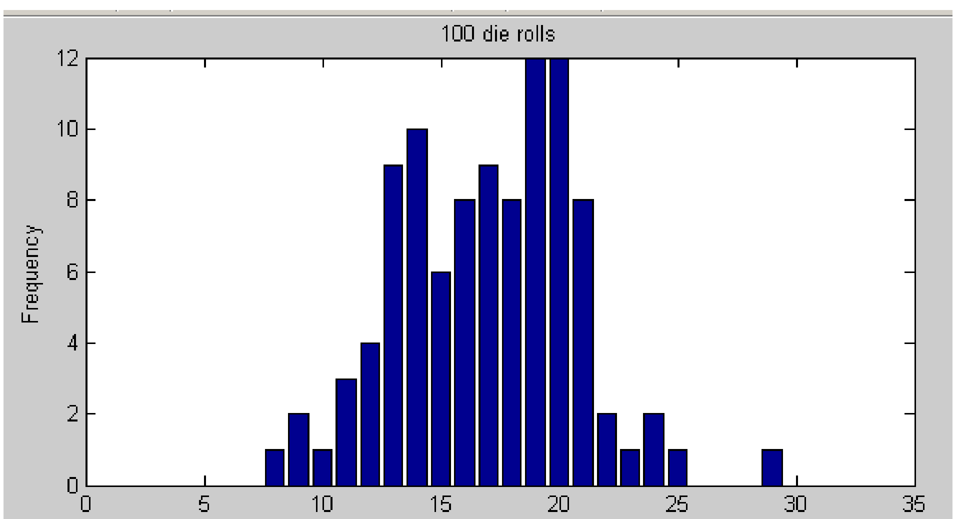
Looks right. Now, find the total:

```
sum( ceil( 6*rand(1,5) ) )

    8

sum( ceil( 6*rand(1,5) ) )

   16
```

Each time you roll the dice, you get a different answer. It's random.

Problem: Roll 5d6 one-hundred times and record how many rolls you get for each total:

```
X = zeros(30,1);
for i=1:100
  D = sum( ceil( 6*rand(1,5) ) );
  X(D) = X(D) + 1;
  end
bar(X)
xlabel('Die Total');
ylabel('Frequency');
title('100 die rolls')
```

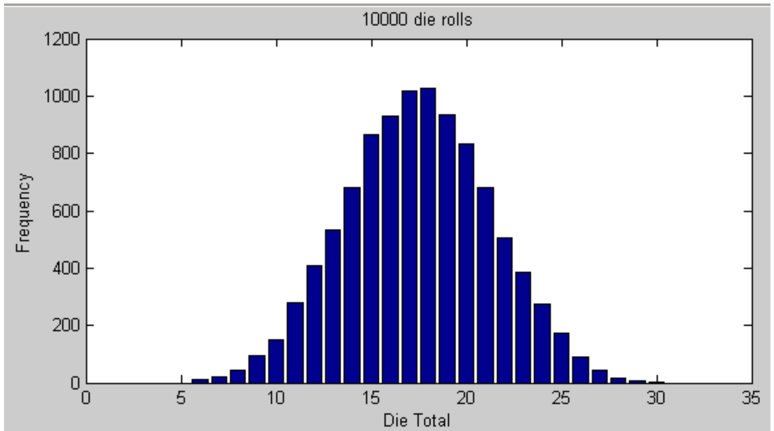


This looks pretty erratic. If you roll the dice 10,000 times, it becomes much better behaved:

Roll 5d6 10,000 times and record the frequency of each outcome:

```
X = zeros(30,1);
for i=1:10000
  D = sum( ceil( 6*rand(1,5) ) );
  X(D) = X(D) + 1;
  end
bar(X)
```

```
xlabel('Die Total');
ylabel('Frequency');
title('10000 die rolls')
```



This is the *Central Limit Theorem*. As the sample size goes to infinity, you get a Normal distribution.

Problem: What's the probability of rolling a 25 or higher with 5d6?

Solution: Toss the dice 1,000,000 times and count the number of times you got 25 or higher. The probability is then approximately this result divided by 1,000,000

```
X = 0;
for i=1:1000000
  D = sum( ceil( 6*rand(1,5) ) );
  if (D >= 25)
     X = X + 1;
     end
  end

X

      32348

X / 1e6

   0.0323
```

There is approximately a 3.23% chance of rolling a 25 or higher with 5d6

Problem: Instead of adding the total of five 6-sided dice (5d6), add d6 + d8 + d10 + d12 + d20. What's the distribution look like?

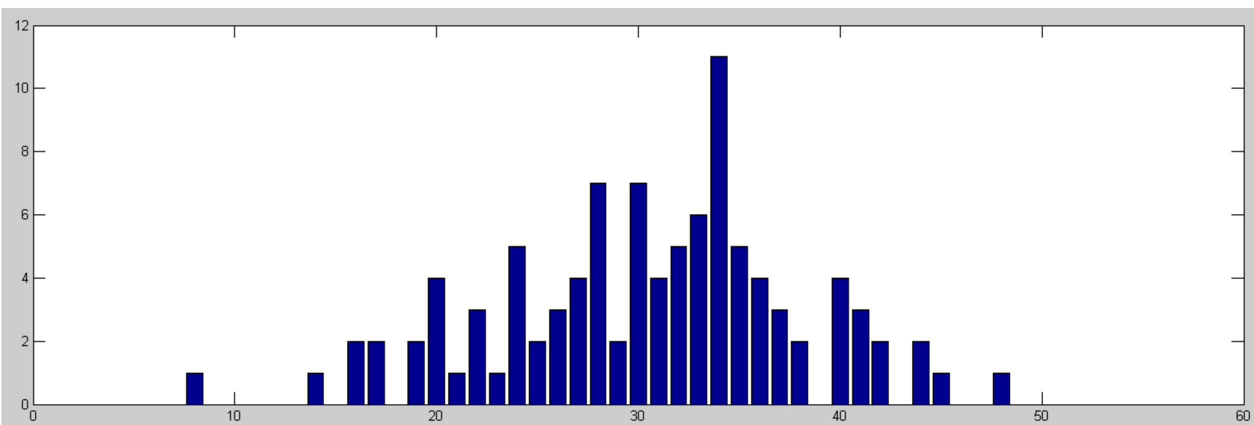
Solution: Roll the dice once

```
d6 = ceil(6*rand);
d8 = ceil(8*rand);
```

```
 d10 = ceil(10*rand);
 d12 = ceil(12*rand);
 d20 = ceil(20*rand);

 [d6,d8,d10,d12,d20]

     1     5     4     1     2

 ROLL = sum([d6,d8,d10,d12,d20])

    13
```

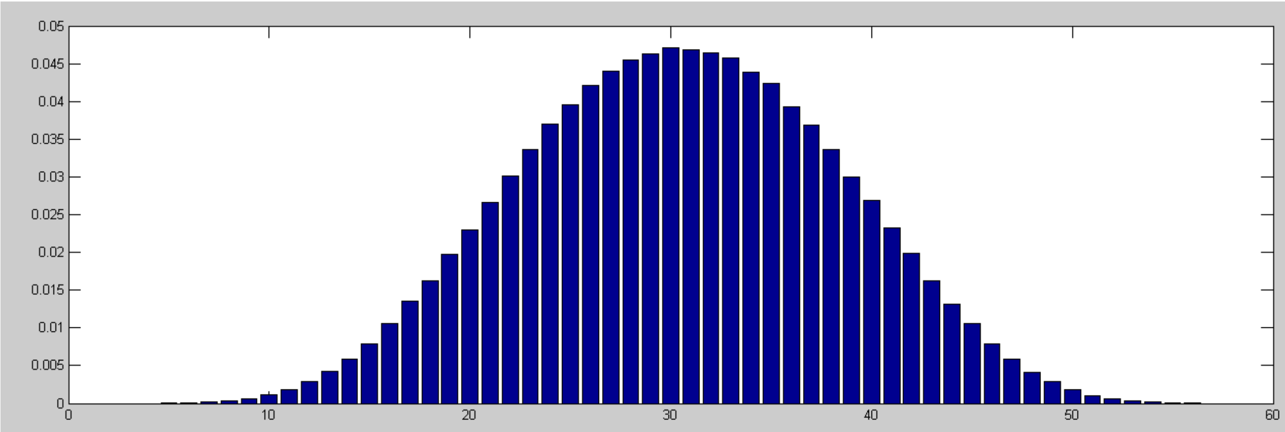
Now do it 100 times

```
RESULT = zeros(56,1);
for i=1:100
   d6 = ceil(6*rand);
   d8 = ceil(8*rand);
   d10 = ceil(10*rand);
   d12 = ceil(12*rand);
   d20 = ceil(20*rand);
   ROLL = sum([d6,d8,d10,d12,d20]);
   RESULT(ROLL) = RESULT(ROLL) + 1;
   end
bar(RESULT)
```



This is pretty erratic. To get a smoother plot, use the Central Limit theorem and roll the dice a lot mode (1,000,000 times):

```
RESULT = zeros(56,1);
for i=1:1000000
   d6 = ceil(6*rand);
   d8 = ceil(8*rand);
   d10 = ceil(10*rand);
   d12 = ceil(12*rand);
   d20 = ceil(20*rand);
   ROLL = sum([d6,d8,d10,d12,d20]);
   RESULT(ROLL) = RESULT(ROLL) + 1;
   end

bar(RESULT / 1000000)
```



What's the probability of rolling a 17?

```
RESULT(17) / 1e6

    0.0135
```

There's a 1.35% chance of rolling a 17.

What's the probability of rolling a 50 or higher?

```
sum(RESULT(50:56)) / 1e6

    0.0040
```

There's a 0.40% chance of rolling a 50 or higher.

Matlab Commands

Display

- format short display results to 4 decimal places
- format long display results to 13 decimal places
- format short e display using scientific notation
- format long e display using scientific notation

Analysis

- sqrt(x) square root of x
- log(x) log base e
- log10(x) log base 10
- exp(x) e^x
- exp10(x) 10^x
- abs(x) |x|
- round(x) round to the nearest integer
- floor(x) round down (integer value of x)
- ceil(x) round up to the next integer
- real(x) real part of a complex number
- imag(x) imaginary part of a complex number
- abs(x) absolute value of x, magnitude of a complex number
- angle(x) angle of a complex number (answer in radians)
- unwrap(x) remove the discontinuity at pi (180 degrees) for a vector of angles

Polynomials

- poly(x)
- roots(x)
- conv(x,y)

Trig Functions

- sin(x) sin(x) where x is in radians
- cos(x) cos()
- tan(x) tan()
- asin(x) arcsin(x)
- acos(x) arccos(x)
- atan(x) arctan(x)
- atan2(y,x) angle to a point (x,y)

Probability and Statistics

- factorial(x) (x-1)!
- gamma(x) x!
- rand(n,m) create an nxm matrix of random numbers between 0 and 1
- randn(n,m) create an nxm matrix of random numbers with a normal distribution
- sum(x) sum the columns of x
- prod(x) multiply the columns of x
- sort(x) sort the columns of x from smallest to largest

- length(x) return the dimensions of x
- mean(x) mean (average) of the columns of x
- std() standard deviation of the columns of x

Display Functions

- plot(x) plot x vs sample number
- plot(x,y) plot x vs. y
- semilogx(x,y) log(x) vs y
- semilogy(x,y) x vs log(y)
- loglog(x,y) log(x) vs log(y)
- mesh(x) 3d plot where the height is the value at x(a,b)
- contour(x) contour plot
- bar(x,y) draw a bar graph
- xlabel('time') label the x axis with the word 'time'
- ylabel() label the y axis
- title() put a title on the plot
- grid() draw the grid lines

Useful Commands

- hold on don't erase the current graph
- hold off do erase the current graph
- diary create a text file to save whatever goes to the screen
- linepace(a, b, n) create a 1xn array starting at a, increment by b
- logspace(a,b,n) create a 1xn array starting at 10^a going to 10^b, spaced logarithmically
- subplot() create several plots on the same screen
- disp('hello') display the message *hello*

Utilities

- format set the display format
- zeros(n,m) create an nxm matrix of zeros
- eye(n,m) create an nxm matrix with ones on the diagonal
- ones(n,m) create an nxm matrix of ones
- help help using different functions
- pause(x) pause x seconds (can be a fraction). Show the graph as well
- clock the present time
- etime the difference between to times
- tic start a stopwatch
- toc the number of seconds since tic