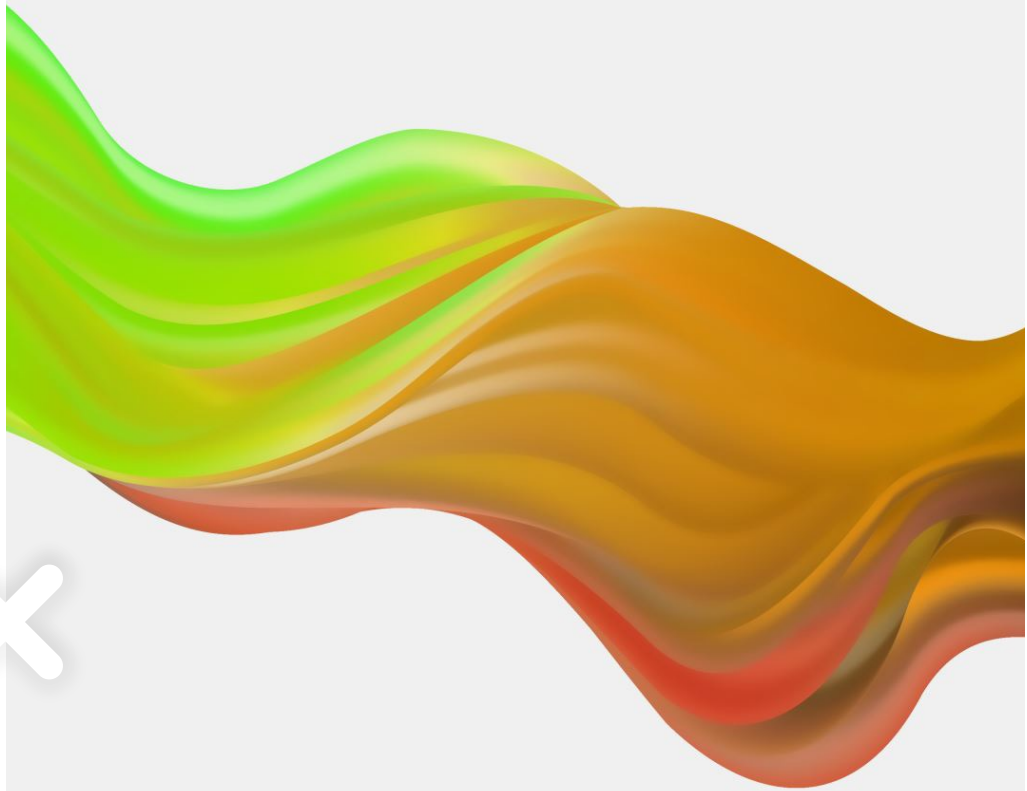
Computer Programming

MENG2020

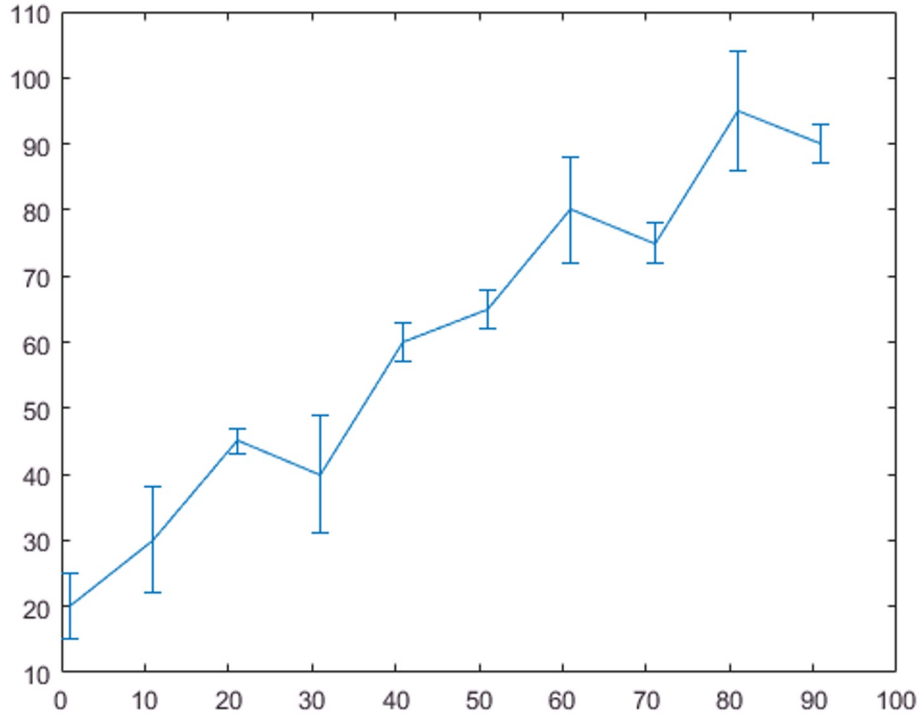


REVIEW

Plots in 2D and 3D

Plots with Error Bars : Example 2

```
x = 1:10:100;
y = [20 30 45 40 60 65 80 75 95 90];
err = [5 8 2 9 3 3 8 3 9 3];
errorbar(x,y,err)
```



Specialized Plots: Bar Plots

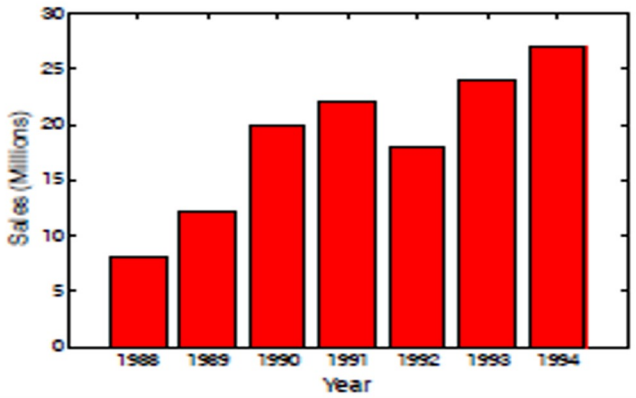
- MATLAB has lots of special types of plots like bar, stairs, stem, pie…

bar

Vertical Bar Plot

Function format:

```
bar(x,y)
```




```
yr=[1988:1994];
sle=[8 12 20 22 18 24 27];
bar(yr,sle,'r')
xlabel('Year')
ylabel('Sales (Millions)')
```

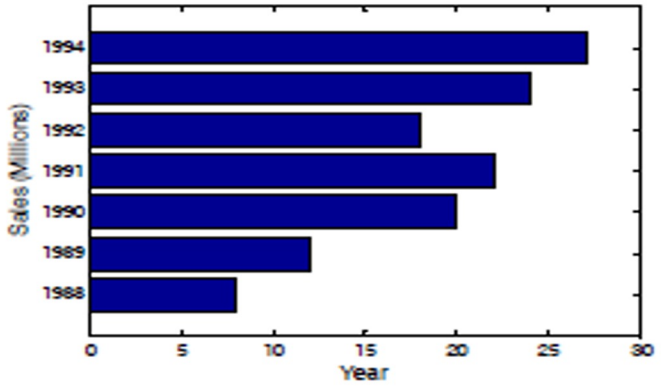
The bars are in red.

barh

Horizontal Bar Plot

Function format:

```
barh(x,y)
```

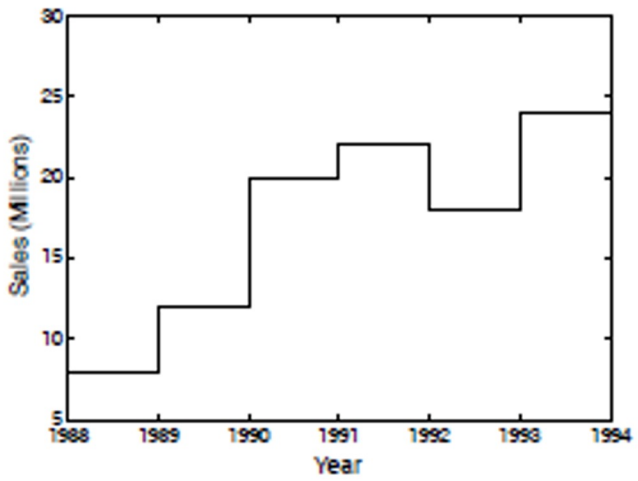


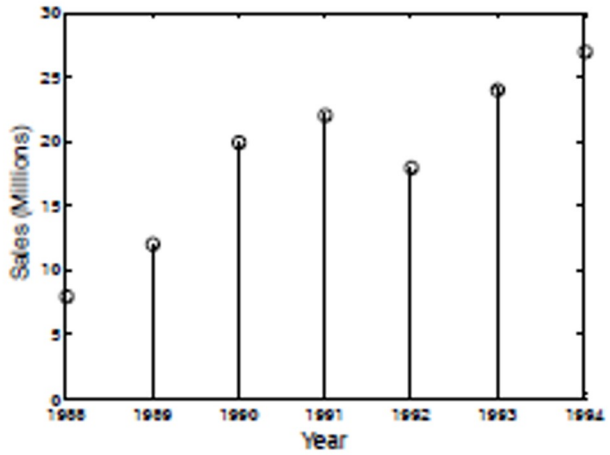

```
yr=[1988:1994];
sle=[8 12 20 22 18 24 27];
barh(yr,sle)
xlabel('Sales (Millions)')
ylabel('Year')
```

Specialized Plots: Stairs and Stem Plots

stairs

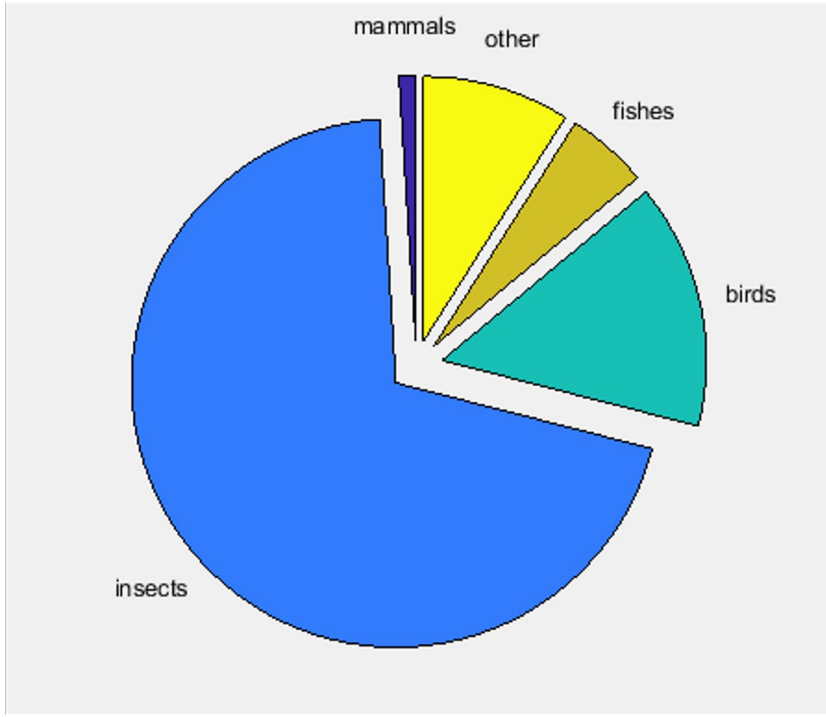
stem

Stairs Plot
 	
Function format:
 	
`stairs(x,y)`
		`yr=[1988:1994];`
		`sle=[8 12 20 22 18 24 27];`
		`stairs(yr,sle)`

Stem Plot
 	
Function Format
 	
`stem(x,y)`
		`yr=[1988:1994];`
		`sle=[8 12 20 22 18 24 27];`
		`stem(yr,sle)`

Pie Chart Plot : An Example

```
%percentage of animals in a small park
%mammals, insects, birds, fishes, other
pc = [0.01, 0.70, 0.15, 0.05, 0.09];
labels = {'mammals', 'insects', 'birds', 'fishes', 'other'};
explode = [1 1 1 1 1];
pie (pc, explode, labels);
```



Histograms : A Simple Example

Let's have an example. Let's roll a die 1000 times and place the rolls into a vector.

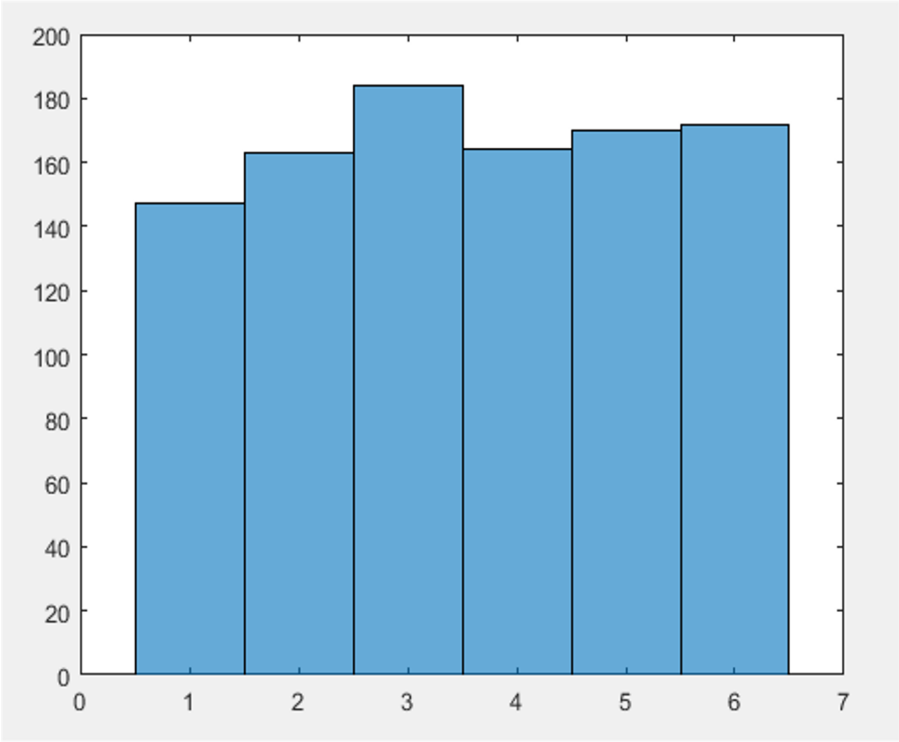
```
rolls = randi (6, 1, 1000);
```

Array 1 x 1000 (row vector size 1000) of numbers between 1 and 6

Now let's make a simple histogram of these rolls.

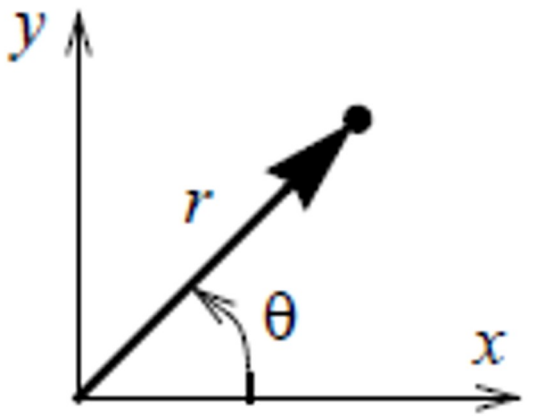
```
histogram (rolls)
```

Note: The number of bins is automatically determined by MATLAB. To fix the number of bins yourself, look at the next slide.



Polar Plots

- In *polar coordinates*, points in a plane are specified by (r,θ).
- r is the distance from the origin.
- θ is the angle from the positive, horizontal axis. θ is positive in the counterclockwise direction.



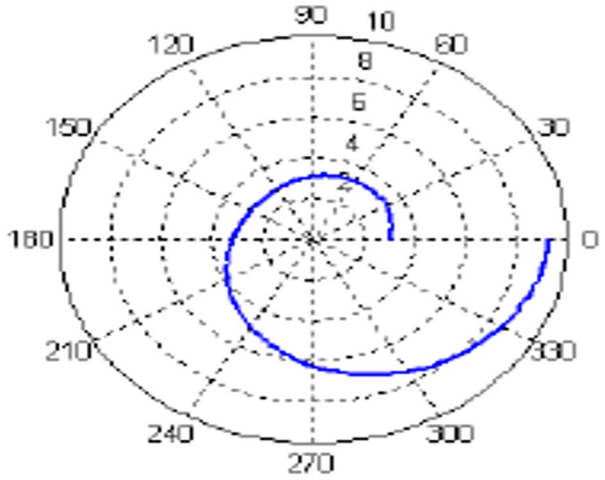
Polar Plots: The polar Command

To make polar a plot in MATLAB, use

```
polar(theta,radius,'line specifiers')
```

- *theta* is expressed in radians.
- Both *theta* and *radius* must be vectors, and **of the same size**.
- The line specifiers are the same as in *plot*.

```
t=linspace(0,2*pi,200);
r=3*cos(0.5*t).^2+t;
polar(t,r)
```



3D Line Plots

A three-dimensional line plot is a plot obtained by connecting points in 3D space. The MATLAB command for a 3D line plot is **plot3**.

```
plot3(x,y,z,'line specifiers','PropertyName',property value)
```

x, y, and z are vectors of the coordinates of the points.

(Optional) Specifiers that define the type and color of the line and markers.

(Optional) Properties with values that can be used to specify the line width, and marker's size and edge and fill colors.

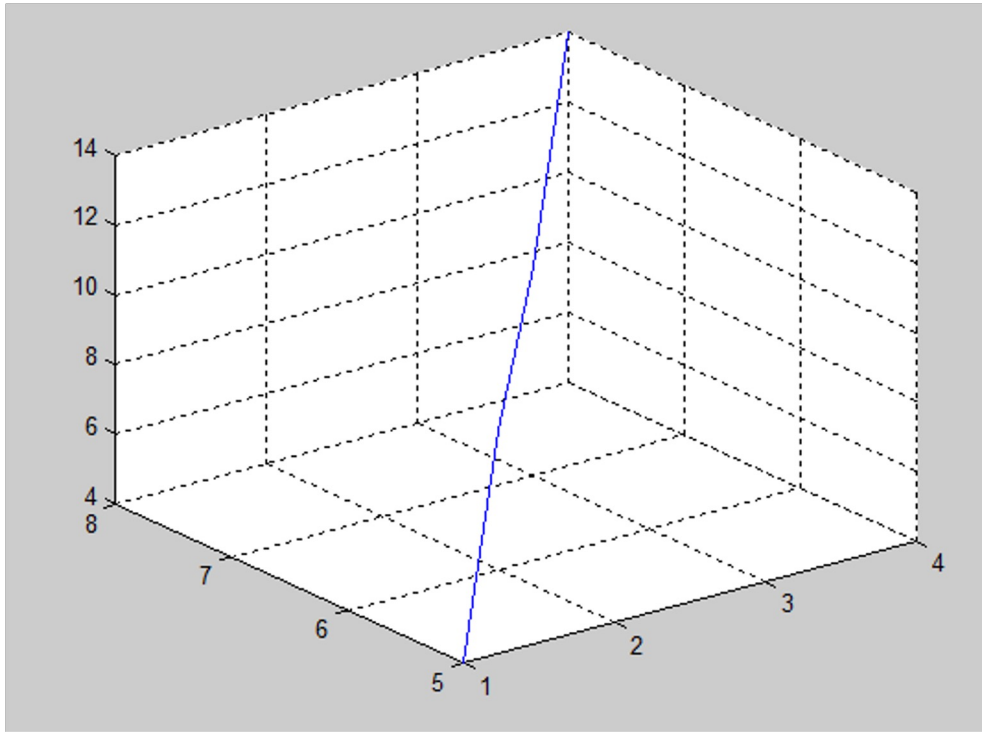
- *x,y, and z must be vectors of the same size.*

- *The line specifiers and the property and values pairs are the same as in 2-D plots.*

3D Line Plots : Example 1

Here is a first example of the **plot3** command. Simple, linear, but rather uninteresting.

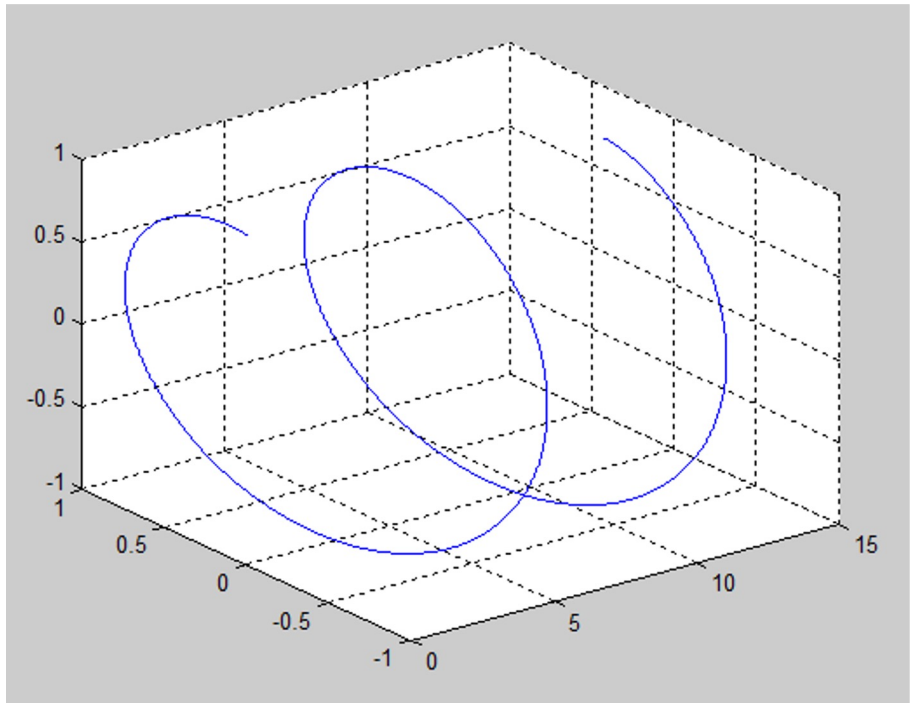
```
x = [1 2 3 4];
y = [5 6 7 8];
z = [4 8 10 14];
plot3 (x,y,z)
grid on
```



3D Line Plots : Example 2

Let's look at a more beautiful plot now.

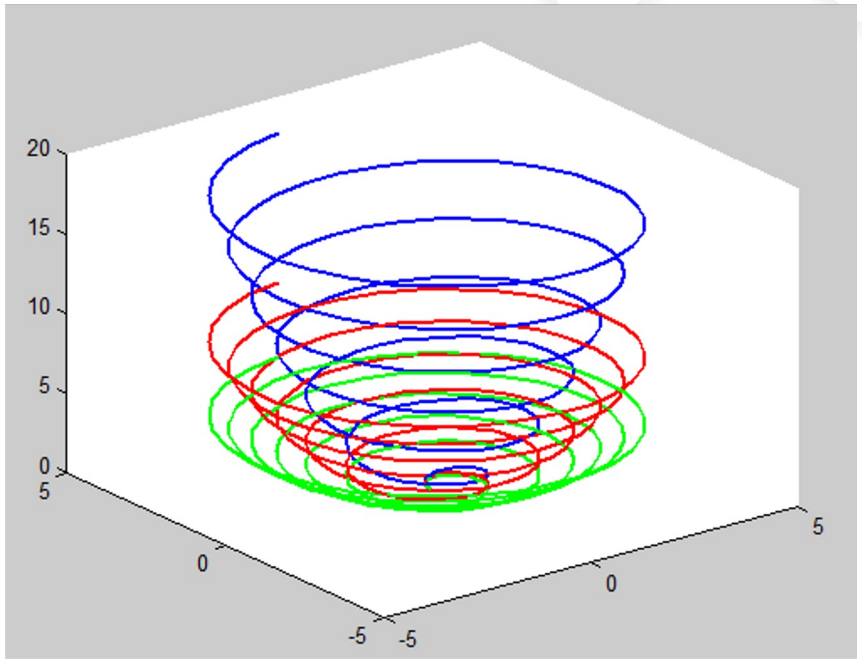
```
a = [0:0.001:4*pi];
b = sin (a);
c = cos (a);
plot3 (a,b,c)
grid on
```



3D Line Plots : Example 3

Now let's look at this much prettier plot!

```
t = 0:0.1:6*pi;
x = sqrt(t) .* sin (2*t);
y = sqrt(t) .* cos (2*t);
z1 = 0.5 * t;
z2 = 0.5 * t;
z1 = t;
z3 = 0.25*t;
plot3(x,y,z1,'b',x,y,z2,'r',x,y,z3,'g','linewidth',2)
grid off
```

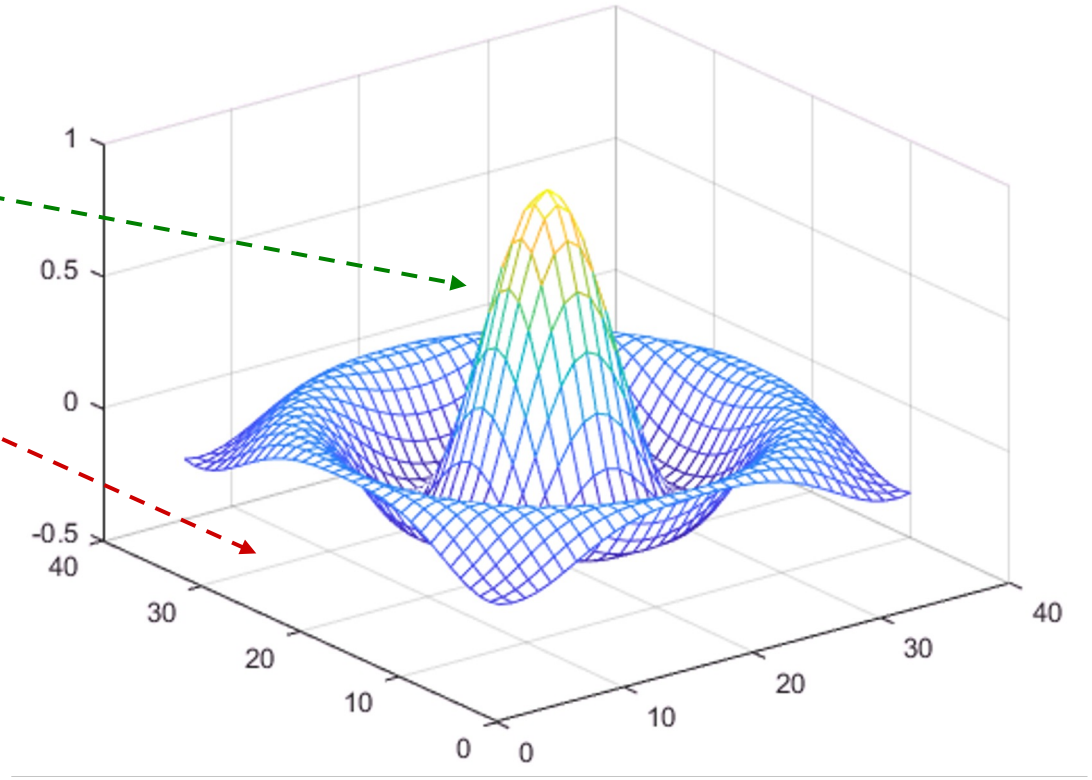


3D Mesh and Surface Plots

- You can think of 3D mesh and surface plots like floating graphs hovering above a base plane that is like a two-dimensional grid.

The dependent variable (z) defines the floating graph itself

The two independent (x,y) variables define the two-dimensional base plane



Making 3D Mesh and Surface Plots

1c. We now have two vectors x and y. Let's use now the **meshgrid** command to create a 2-D grid based on the coordinates contained in vectors x and y. X is a matrix where each row is a copy of x, and Y is a matrix where each column is a copy of y. See the result on the next slide.

```
[X,Y] = meshgrid(x,y)
```

X is the matrix of the x coordinates of the grid points.
Y is the matrix of the y coordinates of the grid points.

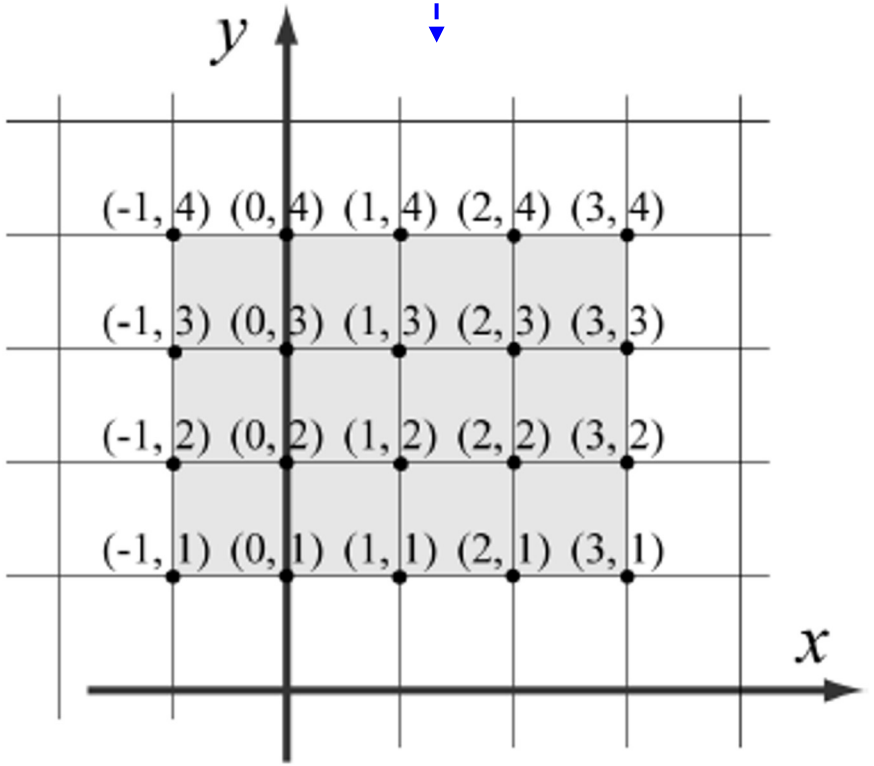
x is a vector that divides the domain of x.
y is a vector that divides the domain of y.

Making 3D Mesh and Surface Plots

Step 1 in MATLAB.
The base plane grid is ready.

```
>> x=-1:3;
>> y=1:4;
>> [X,Y]=meshgrid(x,y)
X =
    -1     0     1     2     3
    -1     0     1     2     3
    -1     0     1     2     3
    -1     0     1     2     3
Y =
     1     1     1     1     1
     2     2     2     2     2
     3     3     3     3     3
     4     4     4     4     4
```

This is what the plane looks like

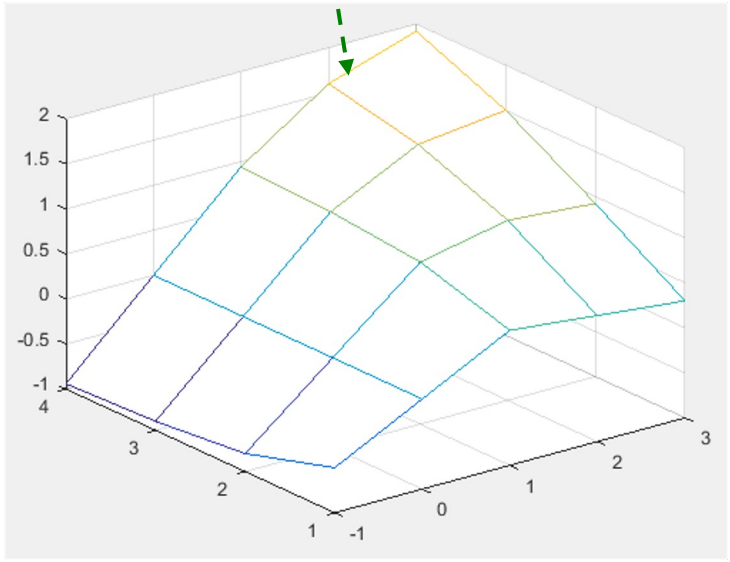


Making 3D Mesh and Surface Plots

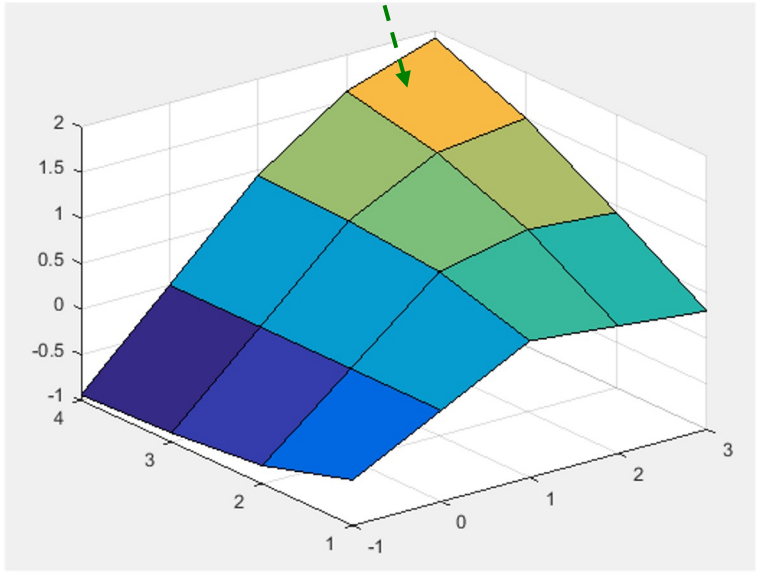
Step 3: Create the 3D plot

This step is very easy. You can create here a mesh plot or a surface plot using the three calculated matrices X, Y, and Z. We use the MATLAB commands **mesh** and **surf**.

mesh (X,Y,Z)



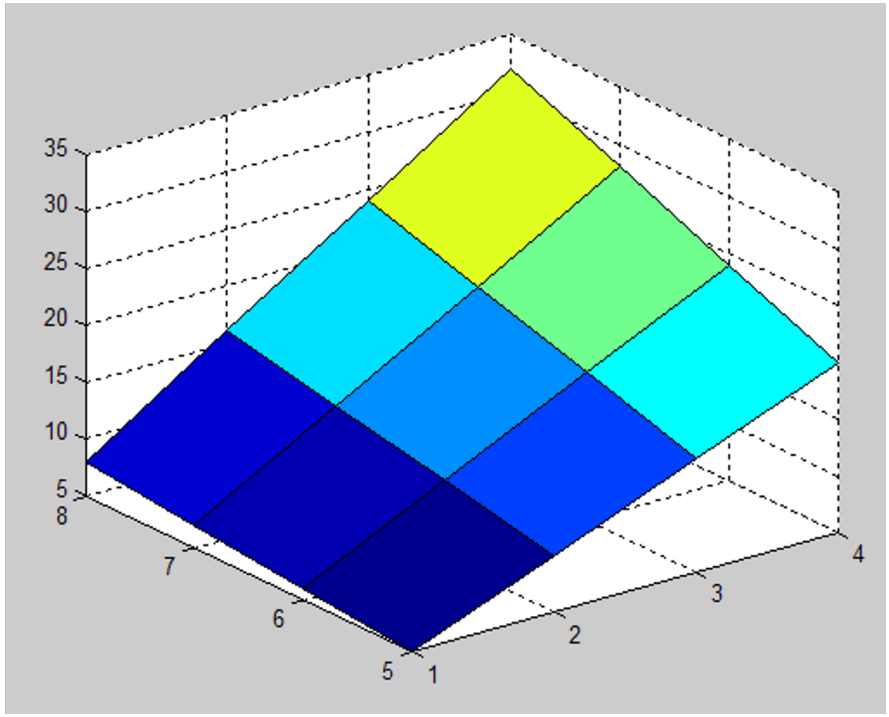
surf (X,Y,Z)



3D Surface Plot : Another Example

- This is the same plot as the previous slide but as a surface plot.

```
x = [1 2 3 4];
y = [5 6 7 8];
[X,Y] = meshgrid (x,y);
Z = X.*Y;
surf (X,Y,Z)
```



3D Mesh and Surface Plots

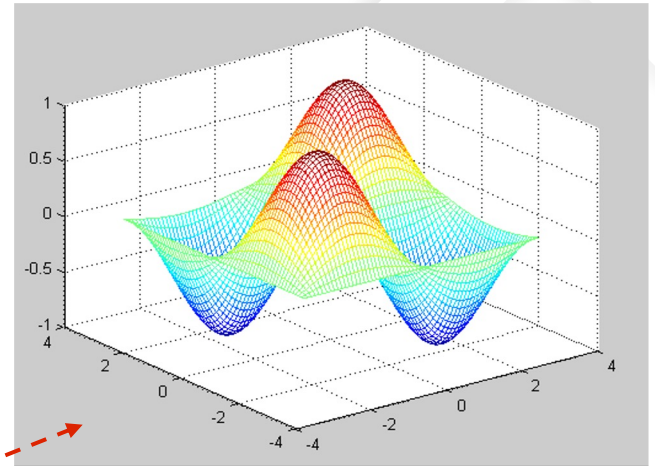
. Let's see more beautiful plots.

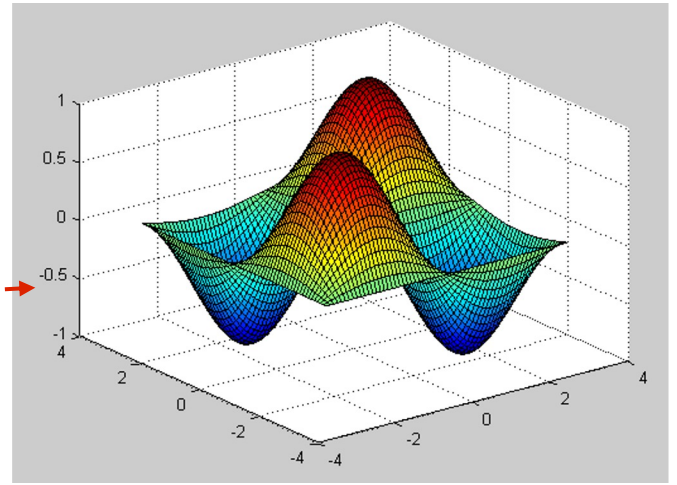
```
c = [-pi:0.1:pi];
d = [-pi:0.1:pi];
[C,D] = meshgrid (c,d);

E = sin (C).* sin (D);

mesh (C,D,E)
surf (C,D,E)
```

Step 1

Step 2

Step 3

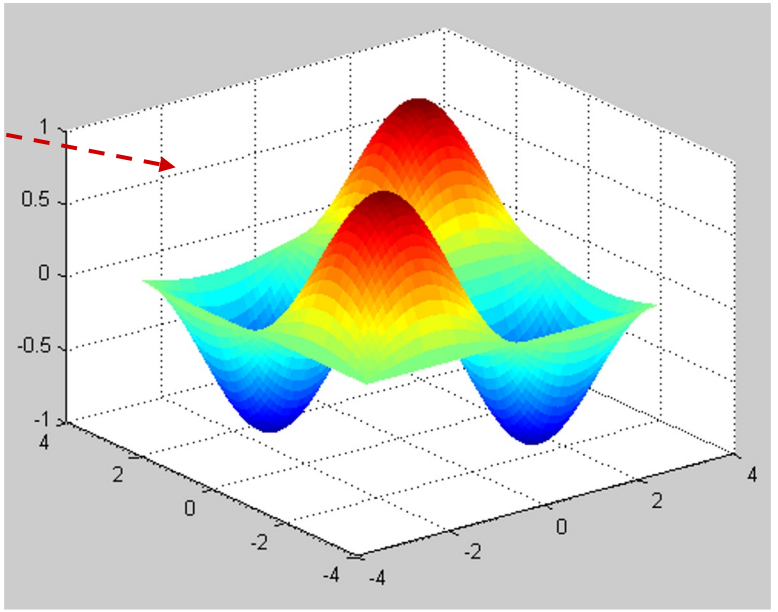


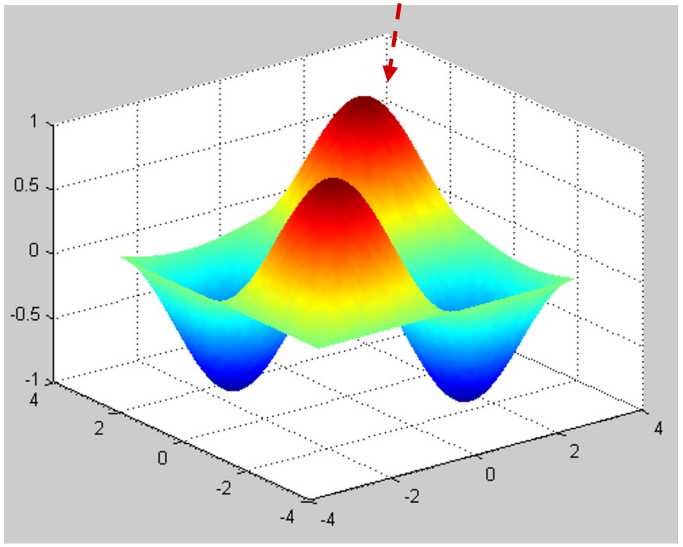


3D Mesh and Surface Plots

You can change the surface shading with the **shading** command (*shading faceted* is the default seen on the previous slide).

```
shading flat
shading interp
```

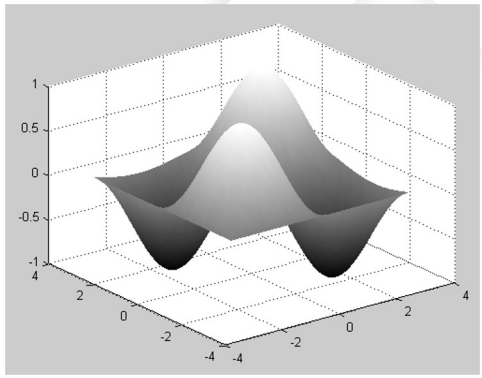


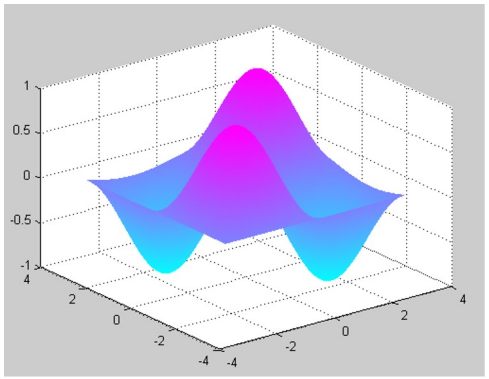


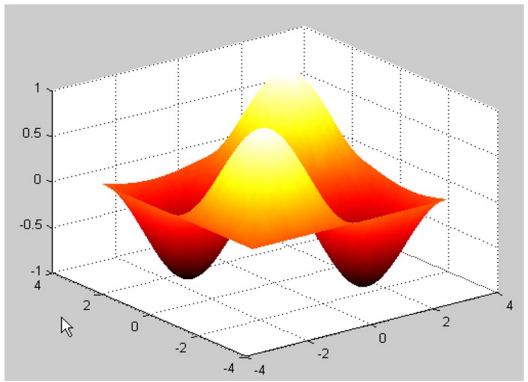
3D Mesh and Surface Plots

You can also change the coloring of the graph with **colormap**.

```
colormap (gray)
colormap (cool)
colormap (hot)
```







Other values for colormap include parula, spring, summer, autumn, winter, bone, copper, lines, colorcube, prism, pink, flag, jet, and hsv. Try them!

3D Polar Plots

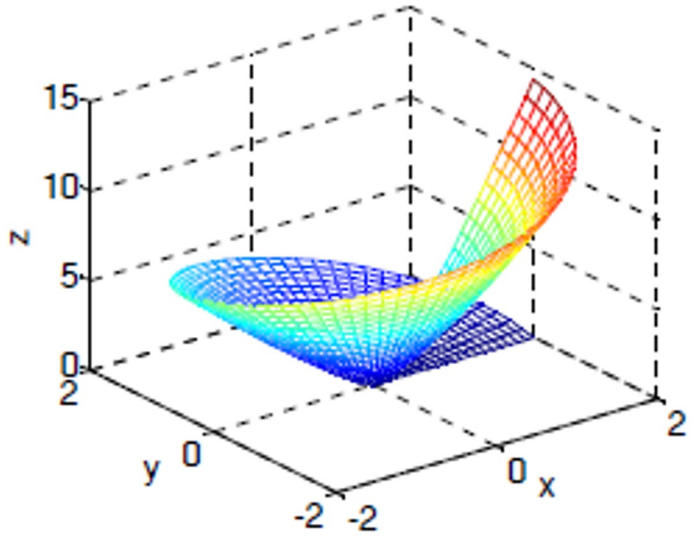
For example the following script creates a plot of the function $z = r\theta$ over the domain $0 \le \theta \le 360°$ and $0 \le r \le 2$.

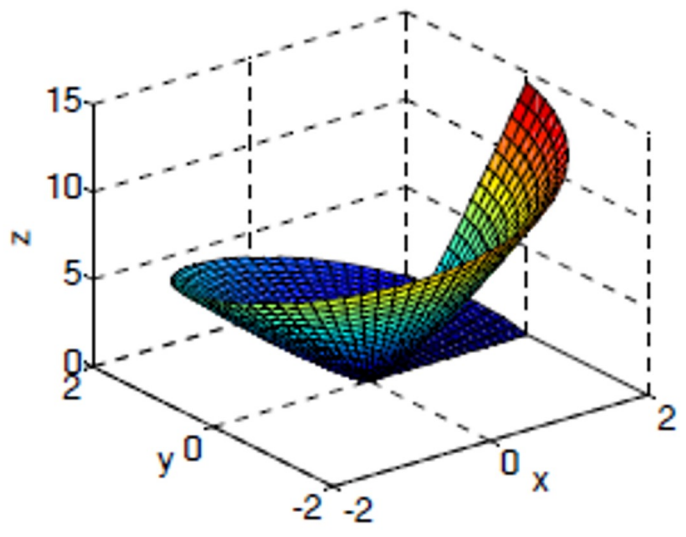
```
[th,r]=meshgrid((0:5:360)*pi/180,0:.1:2);
Z=r.*th;
[X,Y] = pol2cart(th,r);
mesh(X,Y,Z)
```

1. Make base grid (polar).
2. Calculate Z at each point.
3. Convert base grid into cartesian.
4. Plot.

Type `surf(X,Y,Z)` for surface plot.

The figures created by the program are:

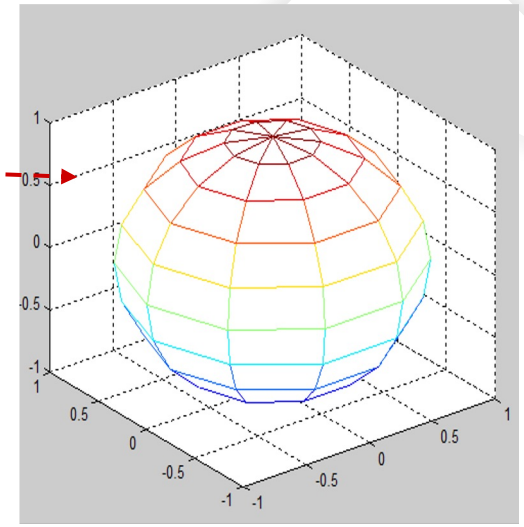




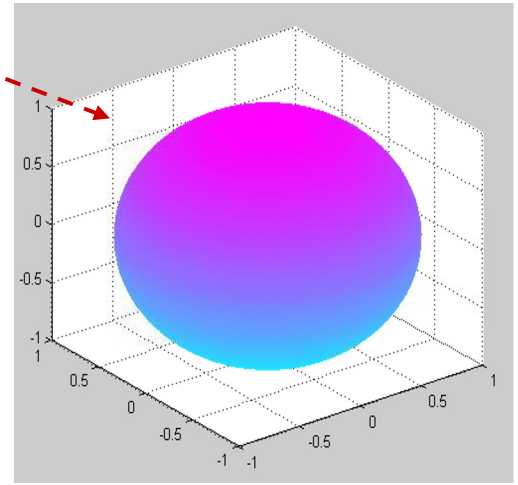
3D Plots for Special Graphs : Sphere

It is extremely easy to plot a sphere. The sphere function returns the coordinates of a sphere with a specified number of faces (20 if the number is not specified).

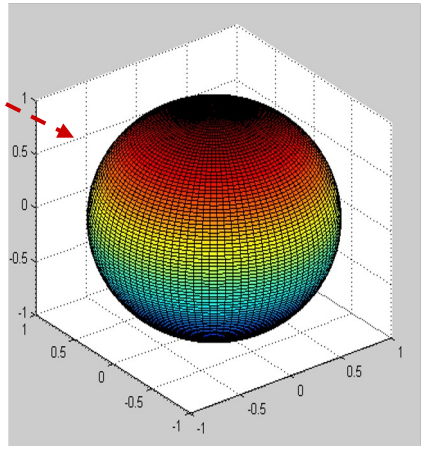
```
[X,Y,Z] = sphere (10);
mesh (X,Y,Z)
```



```
[X,Y,Z] = sphere (50);
surf (X,Y,Z)
shading interp
colormap cool
```



```
[X,Y,Z] = sphere (100);
surf (X,Y,Z)
```



The view Command

The **view** command controls the direction from which you view your plot. The command is **view(az,el)** or **view([az el])**

- **az** (***azimuth***): the angle (in degrees) in the *x y* plane measured from the negative *y* axis and positive is in the counterclockwise direction.
- **el** (***elevation***): the angle of elevation (in degrees) from the *x y* plane. Positive is the direction of the positive *z* axis.

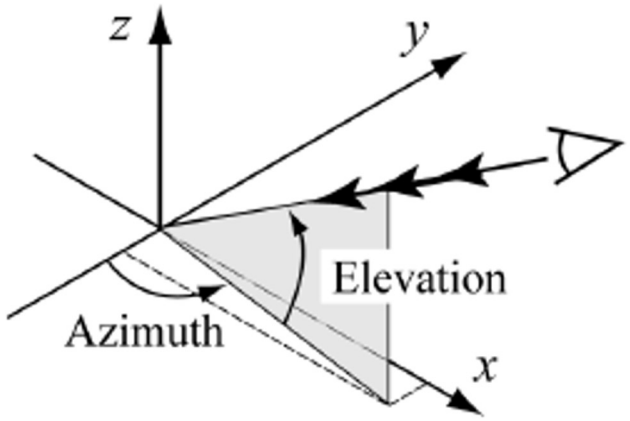
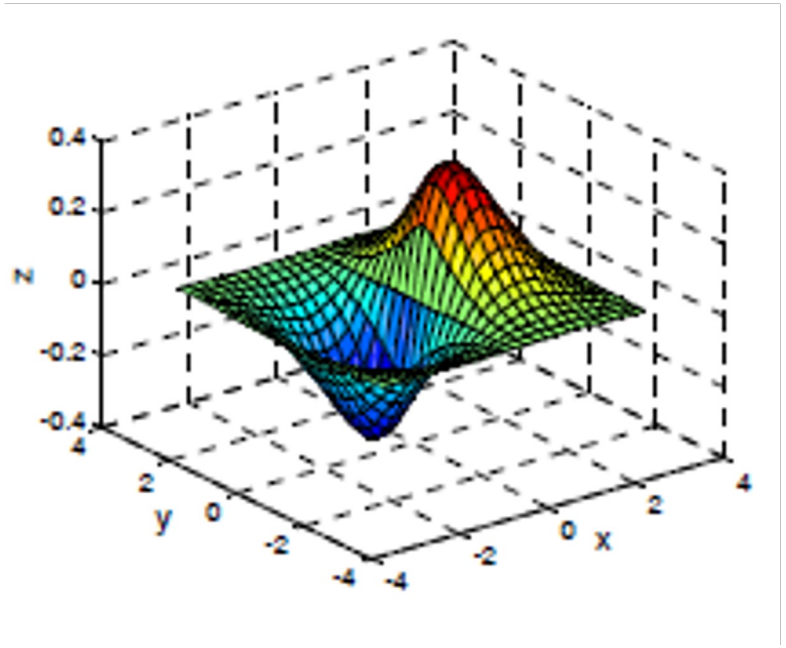


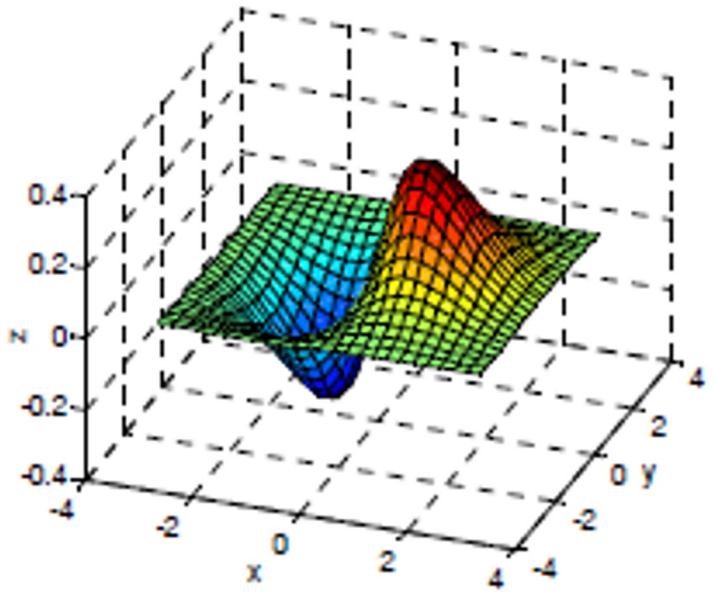
Figure 10-3: Azimuth and Elevation angles.

The view Command : Examples

Default view angles are *az* = -37.5° and *el* = 30°



az = -37.5° and *el* = 30°



az = 20° and *el* = 35°

The view Command : Projection

You can project a 3D curve onto a 2D plane by specific settings, as you would expect, of azimuth and elevation (a value of 0 in one direction projects or "flattens" the 3D plot onto the plane at 0 for that direction. Here a few typical values:

Projection plane	*az* value	*el* value
x-*y* (top view)	0	90
x-*z* (side view)	0	0
y-*z* (side view)	90	0

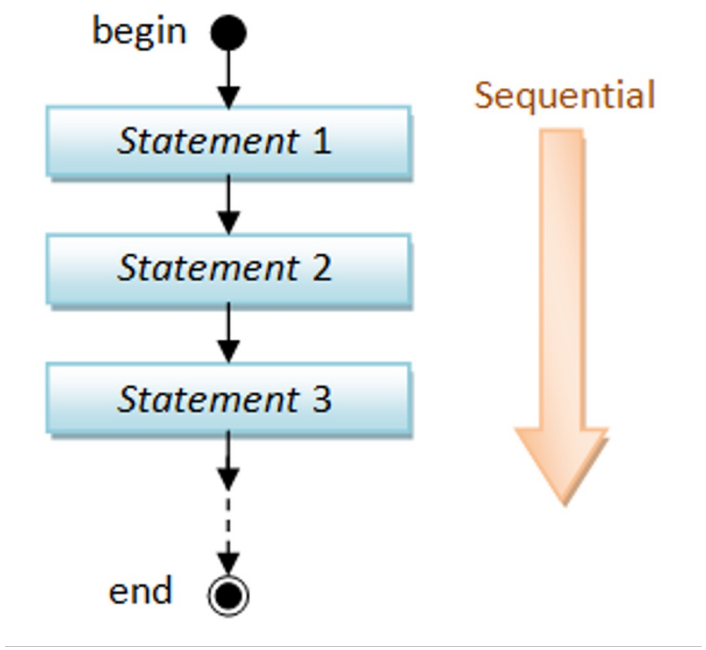
About our lecture

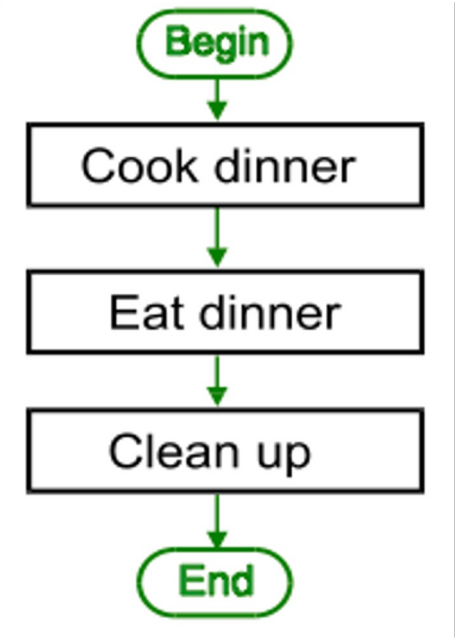
Control Structures

- Programming, as we know already, consists in writing commands (instructions or statements) one after the other. Known as scripts or programs, those sets of instructions tell the computer what to do by way of a programming language (in our case: MATLAB).
- Control structures combine individual instructions into a single logical unit with one entry point at the top and one exit point at the bottom. There are three kinds of control structures:
  - *Sequence (default control structure)*
  - *Selection (branches)*
  - *Repetition (loops)*

Control Structures : The Sequence

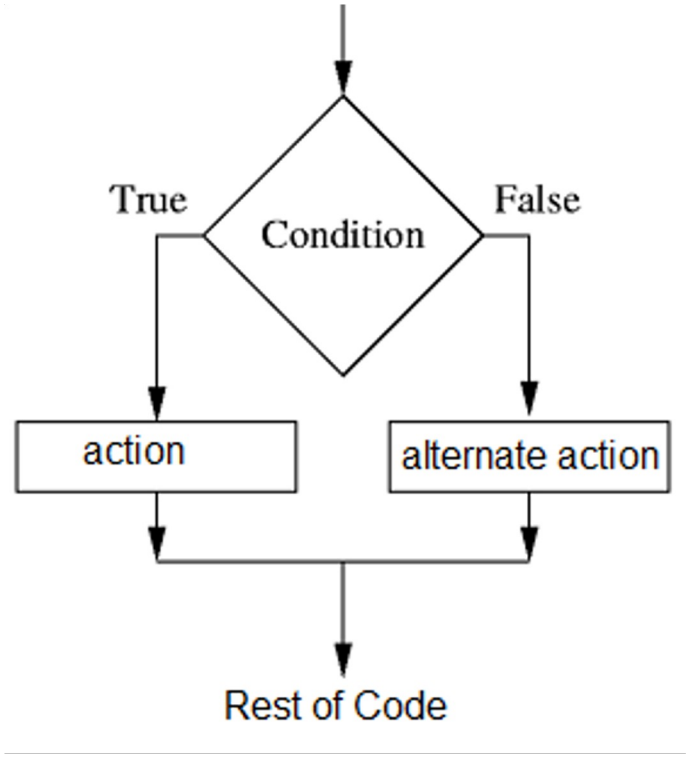
1. Sequence: The control structure we know already. The instructions are executed one after the other (remember the recipe to bake bread or the directions to Port Dover?). All the scripts we have made so far use that control structure. It is also the default control structure, meaning that all programs have at least one sequence structure in them.





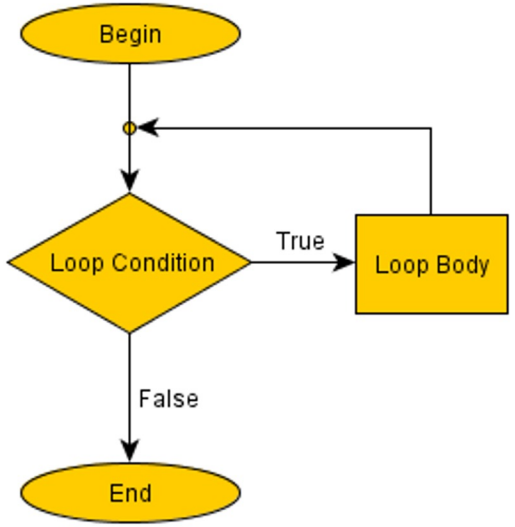
Control Structures : The Selection

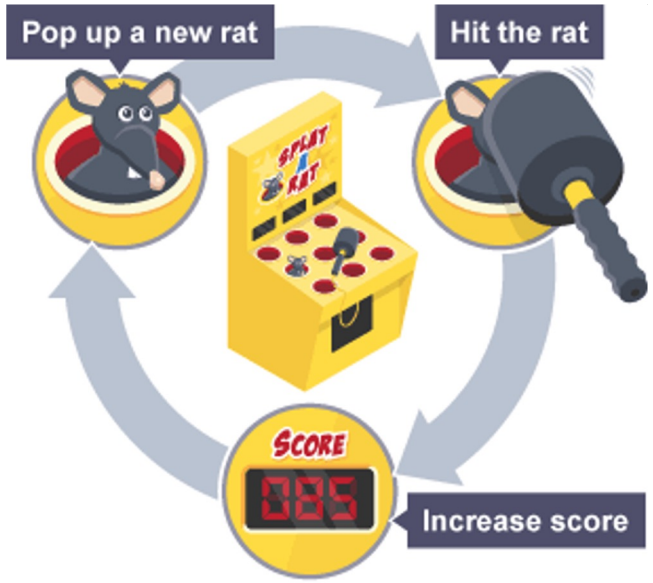
2. Selection: Selection or branching will provide different alternatives based on certain conditions (remember the sorting the mail algorithm?).



Control Structures : The Repetition

3. Repetition: Repetition or iteration or loops will provide the possibility of repeating certain operations based on certain conditions (remember the washing the dishes algorithm?).





- *Selection and repetition rely on **conditions**. So we will now see how to program conditions with relational and logical operators.*

Conditions

- A condition is an expression that is either true or false (A Boolean).

Example: temperature = 28;
So ***temperature < 0*** will be **false**.
But ***temperature > 20*** will be **true**.

=> In Matlab, false has a numerical value of 0 and true a numerical value of 1.

The comparison (relational) operators are:

Relational Operator	Description
<	Less than.
>	Greater than.
<=	Less than or equal to.
>=	Greater than or equal to.
==	Equal to.
~=	Not Equal to.

Caution! The equality relational operator is == not =, the latter being the assignment operator as we already know.

Comparing Arrays

- Conditions can be used to compare two arrays. They must have the exact same dimensions as MATLAB does the comparison element-by-element. The result is an array that has same dimensions as other two but only contains 1's and 0's.

```
>> v1 = [7 8 3 9 1 7 3 2 0 6];
>> v2 = [6 9 2 6 9 7 6 3 8 8];

% is v1 greater than v2?

% is v1 the same as v2? Keep the answer
in another vector v3.
```

Comparing Arrays

- Conditions can be used to compare two arrays. They must have the exact same dimensions as MATLAB does the comparison element-by-element. The result is an array that has same dimensions as other two but only contains 1's and 0's.

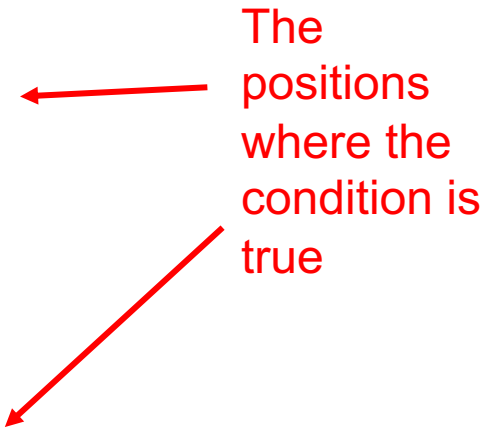
```
>> v1 = [7 8 3 9 1 7 3 2 0 6];
>> v2 = [6 9 2 6 9 7 6 3 8 8];
>> v1 > v2
    ans = 1 0 1 1 0 0 0 0 0 0

%the answer in a vector
>> v3 = v1 == v2
     v3 = 0 0 0 0 0 1 0 0 0 0
```

The positions where the condition is true

Comparing Scalars and Arrays

- When comparing an array to a scalar, MATLAB compares the scalar to every element of the array. The result is an array that has same dimensions as array but only contains 1's and 0's.

```
>> v1 = [7 8 3 9 1 7 3 2 0 6];
>> v1 > 5
   ans = 1 1 0 1 0 1 0 0 0 1

%the answer in a vector
>> v4 = v1 == 3
    v4 = 0 0 1 0 0 0 1 0 0 0
```



Q:
How you can count the number of elements that satisfy the condition?

EXAMPLE:

```
>> v1 = [7 8 3 9 1 7 3 2 0 6];
>> v2 = [6 9 2 6 9 7 6 3 8 8];

%how many 3s in v1?

%how many are the same in v1 and v2?

%how many above 5 in v1?

%how many prime numbers between 1 and 20?
```

Counting the 1s in Logical Arrays

- You can easily count the number of elements that satisfy the condition by doing the sum of all the 1s in the resulting vector. Here are a few examples:

```
>> n3 = sum (v1 == 3) %how many 3s in v1?
n3 = 2

%how many are the same in v1 and v2?
>> v1_eq_v2 = sum (v1 == v2)
v1_eq_v2 = 1

>> v1_over_5 = sum (v1 > 5) %how many above 5 in v1?
v1_over_5 = 5

%how many prime numbers between 1 and 20?
>> nprimes = sum (isprime (1:20))
nprimes = 8
```

Notes on Logical Arrays

- A *logical vector* or a *logical array* is a vector or array that has only **logical** 1's and 0's.
- 1's and 0's obtained from mathematical operations do not count (they really are 0s and 1s).
- 1's and 0's from relational comparisons do work however (these are really *true* or *false* values).
- The first time a logical vector or array is used in an arithmetic operation, MATLAB changes it to a numerical vector or array.

Example of Logical Indexing

- What are the numbers between 1 and 10 that are multiples of 3?

Example of Logical Indexing

- What are the numbers between 1 and 10 that are multiples of 3?

```
%vector of numbers between 1 and 10
>> numbers = 1:10
numbers    = 1   2   3   4   5   6   7   8   9 10

%the 1s point to the numbers that are multiples of 3
>> multiples = rem(numbers, 3) == 0
multiples = 0   0   1   0   0   1   0   0   1   0
```

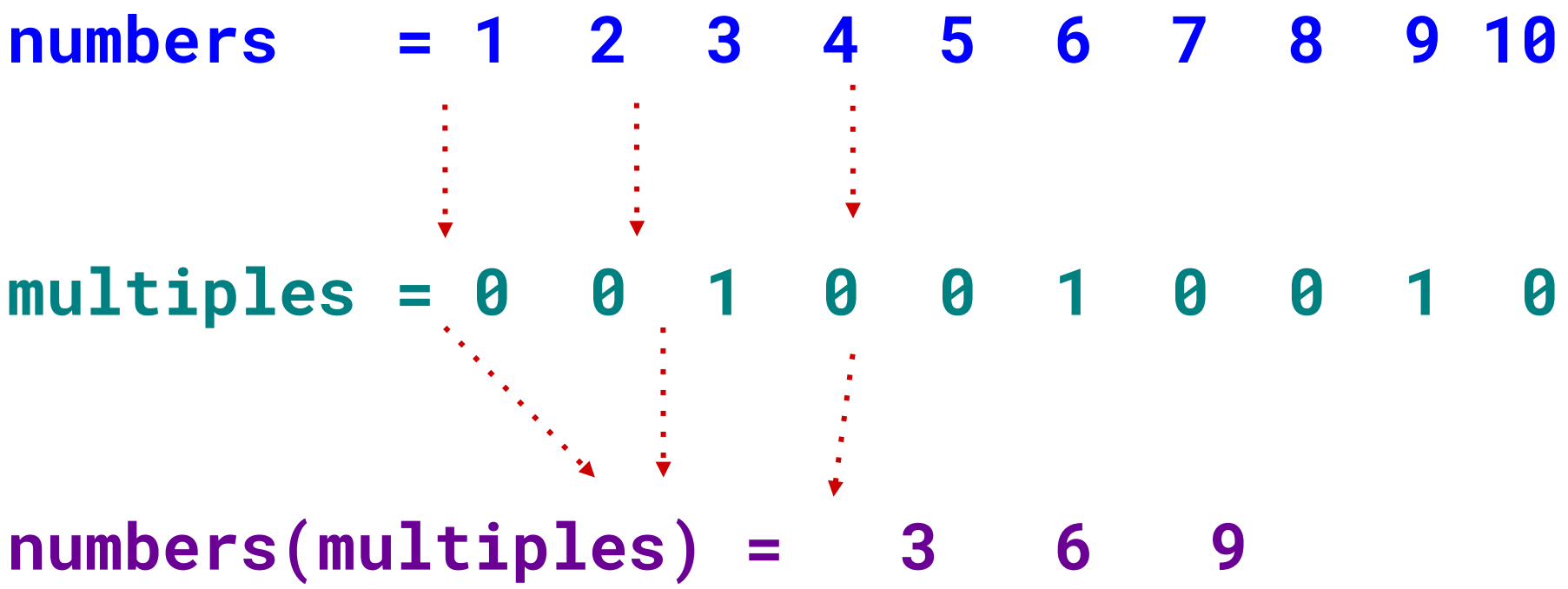
Example of Logical Indexing

- What are the numbers between 1 and 10 that are multiples of 3?

```
%vector of numbers between 1 and 10
>> numbers = 1:10
numbers    = 1   2   3   4   5   6   7   8   9 10

%the 1s point to the numbers that are multiples of 3
>> multiples = rem(numbers, 3) == 0
multiples = 0   0   1   0   0   1   0   0   1   0

%the numbers that are multiples of 3
>> multiples_of_3 = numbers(multiples)
multiples_of_3 = 3   6   9
```

Example of Logical Indexing

- Think of **numbers(multiples)** as pulling out of the **numbers** vector all the elements that have a 1 in the corresponding element of the **multiples** vector.



Logical and Arithmetic Arrays

Have a look at this vector containing only 1s and 0s.

```
>> v5 = [ 0 0 1 0 0 1 0 0 1 0 ]
v5 = 0  0  1  0  0  1  0  0  1  0
```

v5 is the same as *multiples* from the previous slide isn't it? But is it really? We know that *numbers (multiples)* gives us 3 6 9. So numbers (v5) should give us the same but instead we get an error.

```
>> numbers (v5)

???
```

Logical and Arithmetic Arrays

Have a look at this vector containing only 1s and 0s.

```
>> v5 = [ 0 0 1 0 0 1 0 0 1 0 ]
v5 = 0  0  1  0  0  1  0  0  1  0
```

v5 is the same as *multiples* from the previous slide isn't it? But is it really? We know that *numbers (multiples)* gives us 3 6 9. So numbers (v5) should give us the same but instead we get an error.

```
>> numbers (v5)
```

★ ***Subscript indices must either be real positive integers or logicals.***

- *Explanation:* ***multiples*** *is the result from a comparison.* ***v5*** *is just a set of integers and is of a totally different type. They look the same but they are not the same! By the way MATLAB clearly indicates a logical value with the word Logical in blue.*

Logical Operators

A logical expression contains one or more comparison and/or logical operators. They are:

&: The AND operator: True only when all the operands are true.

|: The OR operator: False only when all the operands are false.

,: The XOR operator: True only when either operand is true but not both operands.

~: The NOT operator; False when the operand is true, and true when the operand is false.

Logical Operators Truth Table

- *A truth table* gives the output of a logical operation for every possible combination of inputs. Remember that logical values (true or false) are represented by integer constant and variables. False is always 0. True is all the non-zero values. 1 always means true of course, but 2, 4.5, and -10000 are also values interpreted as true.

- Here is the MATLAB truth table:

INPUT		OUTPUT				
A	B	AND A&B	OR A|B	XOR (A,B)	NOT ~A	NOT ~B
false	false	false	false	false	true	true
false	true	false	true	true	true	false
true	false	false	true	true	false	true
true	true	true	true	false	false	false

Updated Operators Precedence Rule

Precedence	Operation
1 (highest)	Parentheses (If nested parentheses exist, inner have precedence).
2	Exponentiation.
3	Logical NOT (~).
4	Multiplication, division.
5	Addition, subtraction.
6	Relational operators (>, <, >=, <=, ==, ~=).
7	Logical AND (&).
8 (lowest)	Logical OR (|).

Note: If there is an assignment operation like y = (2+x/3) < 7, it is done last. What is the result?

Updated Operators Precedence Rule

Precedence	Operation
1 (highest)	Parentheses (If nested parentheses exist, inner have precedence).
2	Exponentiation.
3	Logical NOT (~).
4	Multiplication, division.
5	Addition, subtraction.
6	Relational operators (>, <, >=, <=, ==, ~=).
7	Logical AND (&).
8 (lowest)	Logical OR (|).

Note: If there is an assignment operation like y = (2+x/3) < 7, it is done last. The answer for y is 1, a logical 1.

Logical Operator Examples

How to determine if a person is a teenager (aged between 13 and 19 inclusive):

```
>> age=[45 47 15 13 11]
age =  45 47 15 13 11
```

Logical Operator Examples

How to determine if a person is a teenager (aged between 13 and 19 inclusive):

```
>> age=[45 47 15 13 11]
age =  45 47 15 13 11

>> teenager = 13 <= age & age <= 19
teenager = 0 0 1 1 0
```

Another example:

```
>> x=3; y=4; z=10;
>> w = x == 3 | y < 10
```

What do you think? Is w true or false?

Important!
You must use **two** separate comparisons!
We'll explain why in a few slides.

Fast Evaluation of Logical Expressions

Look at this example:

```
>> x=3; y=4; z=10;
>> ww = x/3-z < x & z >= x/y | x < y;
```

Now is *ww* true or false?

Fast Evaluation of Logical Expressions

Look at this example:

```
>> x=3; y=4; z=10;
>> ww = x/3-z < x & z >= x/y | x < y;
```

Now is *ww* true or false?

- The trick is to divide the expression in sub-expressions between the | and evaluate the easiest first. As soon as you find a true sub-expression, the whole expression is true.

Another example:

```
>> a=1; b=2; c=3;
>> d = a<5-3 & b==5-c & b>4
```

Now is *d* true or false?

- When there is no | operator, you divide the expression in parts between the & operators. If one part is false, the whole expression is false.

Building Logical Expressions

Are x and y both greater than 10?
Is either x equal to 1 or 3?
Is x an even number?
Is x between y and z?

Building Logical Expressions

Are x and y both greater than 10?

```
x > 10 & y > 10
```

Is either x equal to 1 or 3?

```
x == 1 | x == 3
```

Is x an even number?

```
mod (x,2) == 0
```

Is x between y and z?

```
x >= y & x <= z
```

```
%never y <= x <= z
```

May not always fail, but could at the worst possible time, depending on the value in x - you never know, so don't ever do this! For example if x=7; then 3 <= x <= 5 would be true!!

Comparing Characters

True or false?

```
'9' >= '0'
'a' < 'e'
'B' <= 'A'
'Z' == 'z'
```

Character values are based on the ASCII code. Three things to remember:

1. Digits are in order (0 the smallest, 9 the largest).
2. Letters are in order (A the smallest, Z the largest).
3. Lowercase and uppercase letters are different.

Is a letter lowercase?

```
letter >= 'a' & letter <= 'z'
```

Does the variable *ch* contain a letter?

```
(ch >= 'a' & ch <= 'z') | (ch >= 'A' & ch <= 'Z')
```

Conditions Complements

The opposite (or complement) of (x == 0) is ~(x==0) or (x ~= 0).

The opposite of (x > 3) is ~(x > 3) or (x <=3).

The opposite of (a == 10 & b > 5) is ~(a == 10 & b > 5) or (a ~= 10 | b <=5)

De Morgan's Theorem:

The complement of exp1&exp2 is comp_exp1|comp_exp2.

The complement of exp1|exp2 is comp_exp1&comp_exp2.

Built-In Logical Functions

- MATLAB has some built-in functions or commands for doing logical operations and related calculations. Three are equivalent to the logical operators
  - **and(A,B)** : is the same as A&B
  - **or(A,B)** : is the same as A|B
  - **not(A)** : is the same as ~A

Built-In Logical Functions

MATLAB also has other Boolean functions you can use, but these don't have simple short forms as on the previous slide.

Note: Addresses are indices (location within the array).

Function	Description	Example

`xor(a,b)`	Exclusive or. Returns true (1) if one operand is true and the other is false.	`>> xor(7,0)`
		`ans =`
		`1`
		`>> xor(7,-5)`
		`ans =`
		`0`

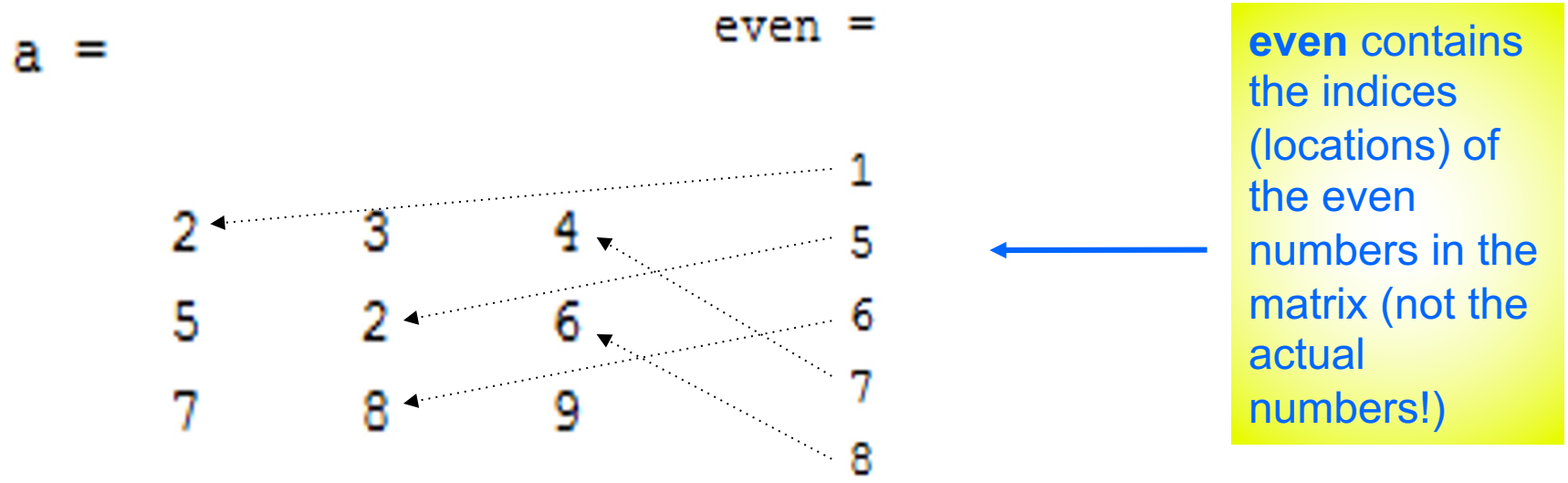
`all(A)`	Returns 1 (true) if all elements in a vector A are true (nonzero). Returns 0 (false) if one or more elements are false (zero).
	If A is a matrix, treats columns of A as vectors, returns a vector with 1's and 0's.
		`>> A=[6 2 15 9 7 11];`
		`>> all(A)`
		`ans =`
		`1`
		`>> B=[6 2 15 9 0 11];`
		`>> all(B)`
		`ans =`
		`0`

`any(A)`	Returns 1 (true) if any element in a vector A is true (nonzero). Returns 0 (false) if all elements are false (zero).
	If A is a matrix, treats columns of A as vectors, returns a vector with 1's and 0's.
		`>> A=[6 0 15 0 0 11];`
		`>> any(A)`
		`ans =`
		`1`
		`>> B = [0 0 0 0 0 0];`
		`>> any(B)`
		`ans =`
		`0`

`find(A)`
 	
`find(A>d)`
	If A is a vector, returns the indices of the nonzero elements.
	If A is a vector, returns the address of the elements that are larger than d (any relational operator can be used).
		`>> A=[0 9 4 3 7 0 0 1 8];`
		`>> find(A)`
		`ans =`
		`2 3 4 5 8 9`
		`>> find(A>4)`
		`ans =`
		`2 5 9`

Built-In Logical Functions : An Example

```
a = [2 3 4; 5 2 6; 7 8 9];
even = find(mod(a,2) == 0);
```



even contains the indices (locations) of the even numbers in the matrix (not the actual numbers!)

- *Remember that matrix indices are column major, going down then right from top left corner.*

The Conditional (if) Statement

- A *conditional statement* is a command that allows MATLAB to decide whether or not to execute some code that follows the statement.

- If statements are very common in scripts or functions. They come in three general forms:

A simple if with one alternative:
if-end

A simple if with two alternatives:
if-else-end

A nested if with three or more alternatives:
if-elseif-else-end etc...

elseif can be repeated (nested) for more alternatives

The if-end Structure

Syntax of a simple if statement:

```
if condition
    a statement if condition is true;
      another statement if condition is
true;
end

remaining lines of code...
```

You can add many lines of code here!

- *If the conditional expression is true, MATLAB runs the lines of code that are between the line with* **if** *and the line with* **end**. *Then it continues with the code after the* **end** *line.*

The if-end Structure : Example

Example of a simple if statement (script is saved as **water.m**):

```
temp = input('Enter the temperature: ');
if temp >= 100
  fprintf ('WARNING! Boiling water\n');
end
```

Run the program (script) named **water**

```
>> water
Enter the temperature: 66
>> water
Enter the temperature: 102
WARNING! Boiling water
>>
```

Run it again!

The if-else-end Structure

if-else-end structure lets you execute one section of code if a condition is true and a different section of code if it is false.

Syntax:

```
if condition
     statement(s) if condition is true
else
     statement(s) if condition is false
end
```

The if-else-end Structure : Example

Example of a simple if statement with two alternatives:

```
temp = input('Enter the temperature: ');
if temp >= 20
    fprintf ('Temperature is warm.\n');
else
    fprintf ('Temperature is cool.\n');
end
```

```
Enter the temperature: 16
The temperature is cool.
>>
```

Remember that you, the programmer, have to arrange the condition comparison values to make this work as you want it to do!

This concludes our overview of MATLAB and a taste of things to come!

End of Lesson