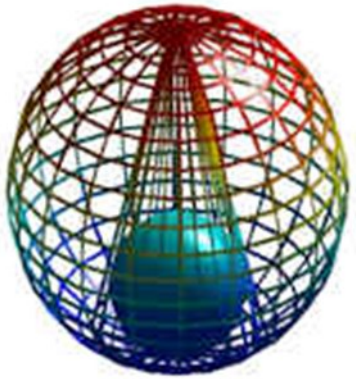


MATLAB TUTORIAL

MAT 220

Dr.C. Rajan (8113053359)
rajancv@am.amrita.edu Office: S109E last room

What is MATLAB?

- A high-performance language for technical computing (Mathworks, 1998)
- The name is derived from **MAT**rix **Lab**oratory

- Typical uses of MATLAB
  - Mathematical computations
  - Algorithmic development
  - Model prototyping
  - Data analysis and exploration of data (visualization)
  - Scientific and engineering graphics for presentation

Why Matlab?

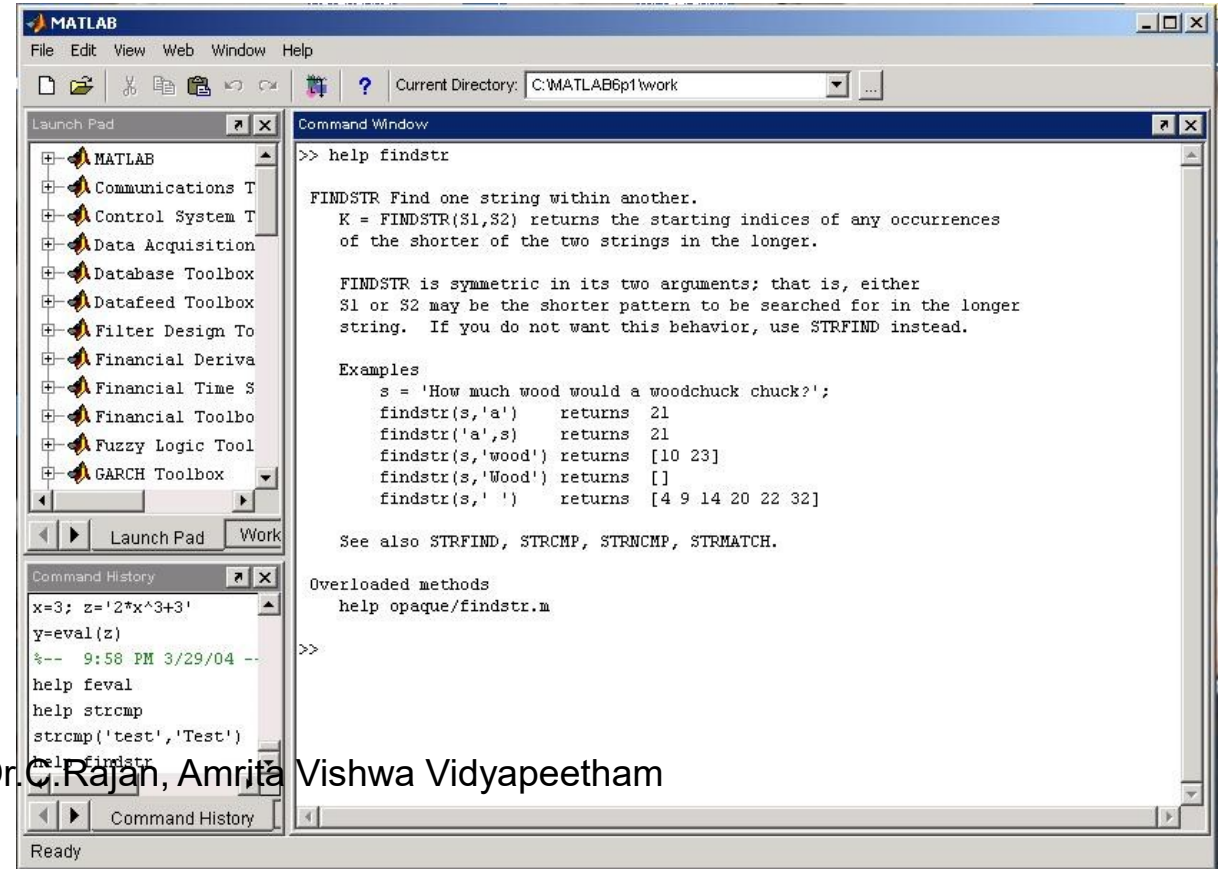
- Because it simplifies the analysis of mathematical models
- It frees you from coding in high-level languages (saves a lot of time - with some computational speed penalties)
- Provides an extensible programming/visualization environment
- Provides professional looking graphs

Why MATLAB?

- If MATLAB can process and visualize data, why ever use FORTRAN, C, or some other language?
- MATLAB is an ***interpreted*** **language**
  - It is not a compiled language
  - Therefore identical code executes more slowly, sometimes MUCH more slowly in MATLAB
  - MATLAB has more memory overhead than equivalent FORTRAN or C programs
  - MATLAB may be unsafe for some critical systems

Getting Help and Looking Up Functions

- To get help on a function type “help function_name”, e.g., “help plot”.
- To find a topic, type “**lookfor topic**”, e.g., “lookfor matrix”



Matlab's Workspace

- **who**, **whos** – current workspace vars.
- **save** – save workspace vars to *.mat file.
- **load** – load variables from *.mat file.
- **clear all** – clear workspace vars.
- **close all** – close all figures
- **clc** – clear screen
- **clf** – clear figure

Basic Commands

- **%** used to denote a comment
- **;** suppresses display of value (when placed at end of a statement)
- **...** continues the statement on next line
- **eps** machine epsilon
- **inf** infinity
- **NaN** not-a number, e.g., 0/0.

Other symbols

>>	prompt
. . .	continue statement on next line
,	separate statements and data
%	start comment which ends at end of line
;	(1) suppress output
	(2) used as a row separator in a matrix
:	specify range

Relational Operators

- MATLAB supports six relational operators.

Less Than	<
Less Than or Equal	<=
Greater Than	>
Greater Than or Equal	>=
Equal To	==
Not Equal To	~=

MATLAB Logical Operators

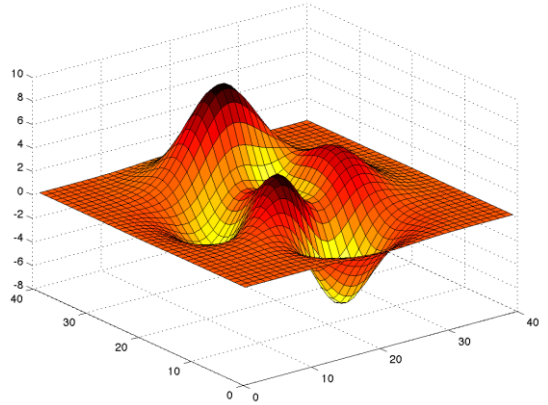
- MATLAB supports five logical operators.

not/~	element wise/scalar logical NOT
and/&	element wise logical AND
or / |	element wise logical OR
&&	logical (short-circuit) AND
||	logical (short-circuit) AND

Logical Functions

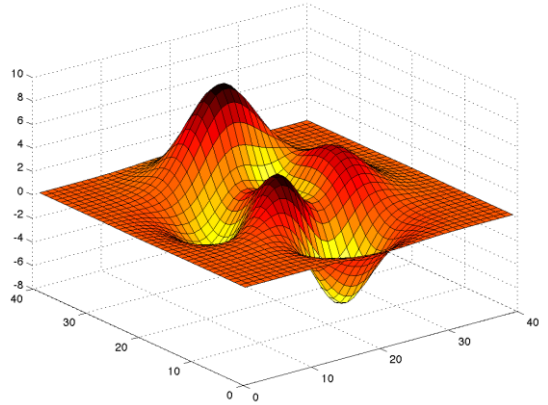
- MATLAB also supports some logical functions.
  - **xor** (a, b) exclusive or
  - **any**(x) returns 1 if any element of x is nonzero
  - **all**(x) returns 1 if all elements of x are nonzero
  - **isnan**(x) returns 1 at each NaN in x
  - **isinf**(x) returns 1 at each infinity in x
  - **finite**(x) returns 1 at each finite value in x
  - **find**(x) find indices and values of non zero elements

Matlab



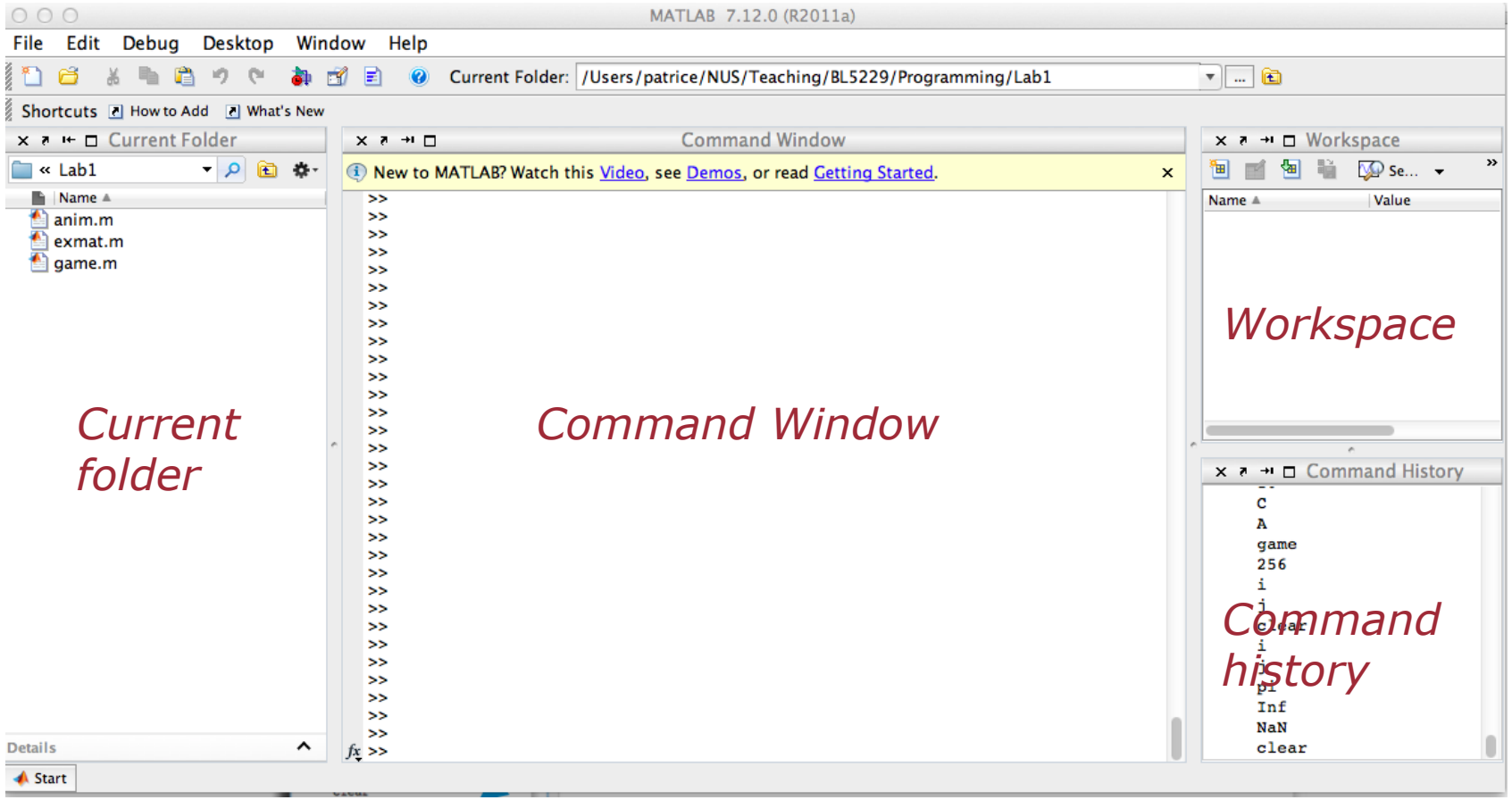
- The Matlab Environment
- Variables; operations on variables
- Programming
- Visualization

Matlab

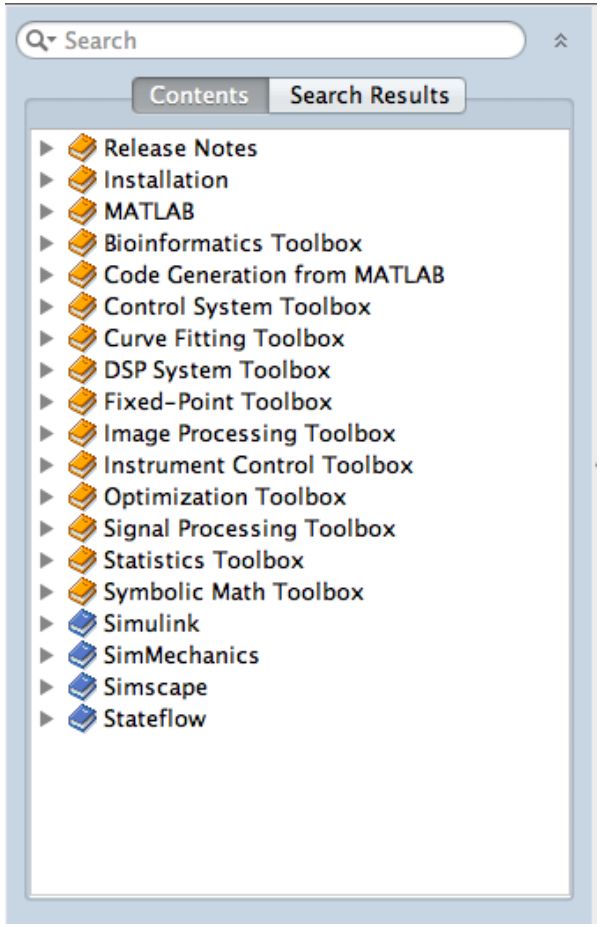


- The Matlab Environment
- Variables; operations on variables
- Programming
- Visualization

The Matlab Environment



Help in Matlab



Help Browser

-> Product Help

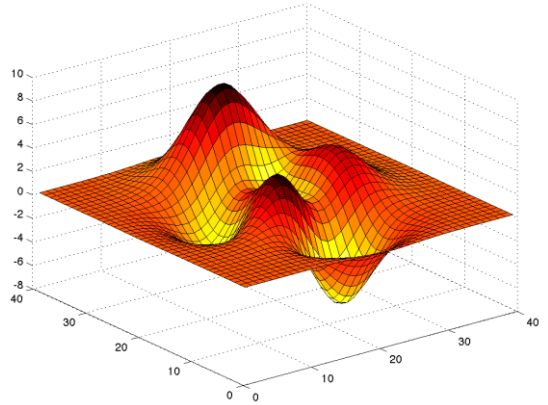
Command line:

>> help <command>

Example:

>> help sqrt

Matlab



- The Matlab Environment
- Variables; operations on variables
- Programming
- Visualization

Variables in Matlab

- Begin with an alphabetic character: a
- Case sensitive: a, A
- No data typing: a=10; a= ‘OK’ ; a=2.5
- Default output variable: ans
- Built-in constants: **pi i j Inf**
- **clear** removes variables
- **who** lists variables
- **whos** list variables and gives size
- Special characters : **[] () {} ; % : = ... @**

Special Variables

- Special variables:
  - ans : default variable name for the result
  - pi: π = 3.1415926……
  - eps: $\in$ = 2.2204e-016, smallest amount by which 2 numbers can differ.
  - Inf or inf : ∞, infinity
  - NaN or nan: not-a-number

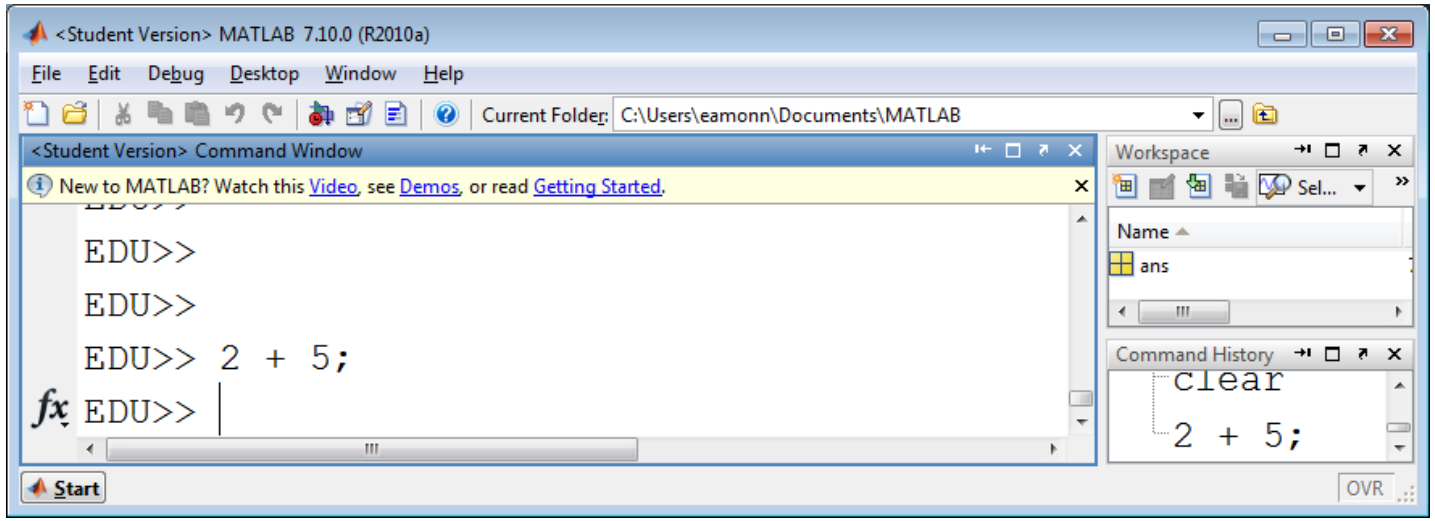
Vectors

Some useful commands:

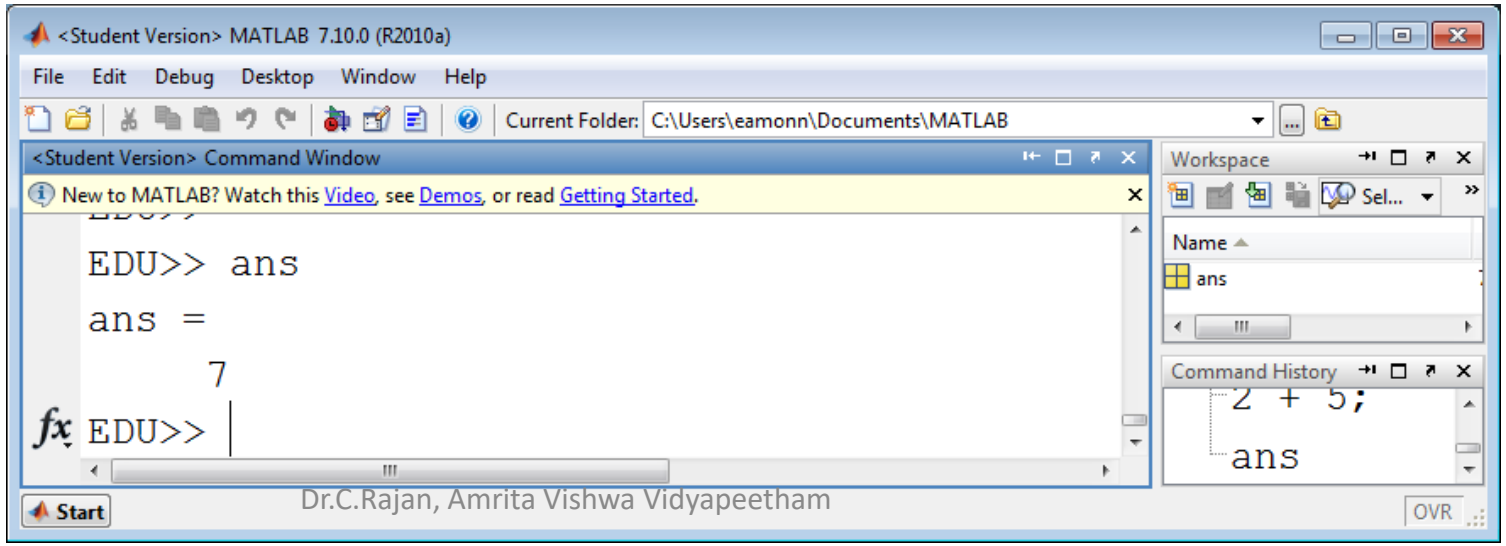
x = start:end	create row vector x starting with start, counting by one, ending at end
x = start:increment:end	create row vector x starting with start, counting by increment, ending at or before end
linspace(start,end,number)	create row vector x starting with start, ending at end, having number elements
length(x)	returns the length of vector x
y = x’	transpose of vector x
dot (x, y)	returns the scalar dot product of the vector x and y.

Using the semicolon to suppress echoing

;



We can examine the contents of **ans** at any time, simply by typing its name.



(arithmetic) Operators

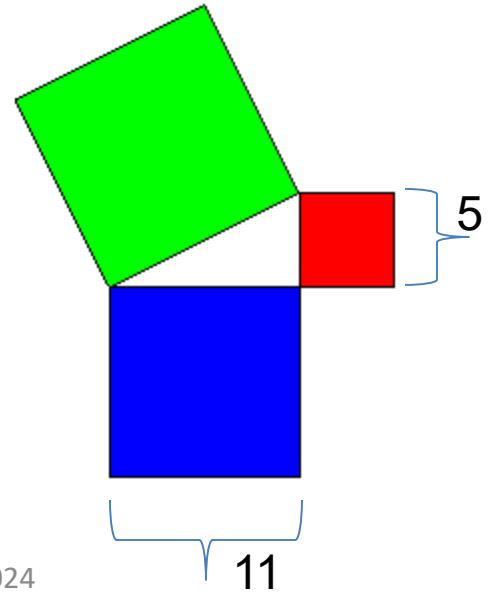
- Scalar arithmetic operations

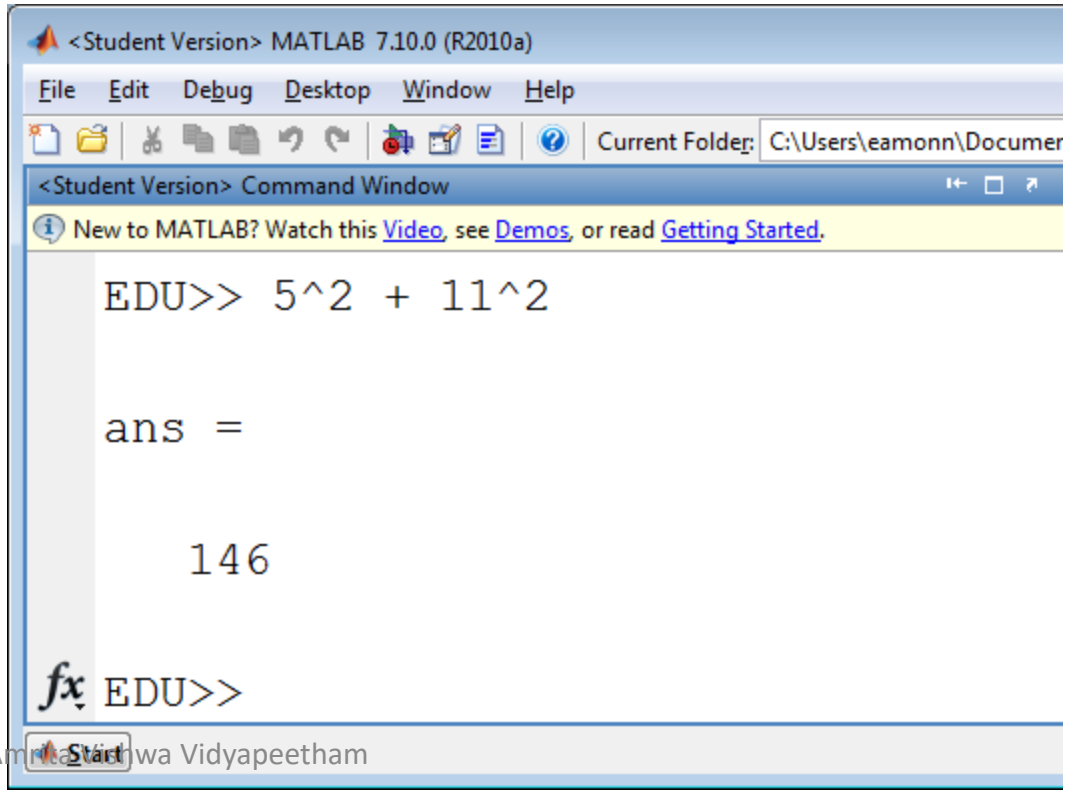
Operation			MATLAB form
– Exponentiation:	^	a^b	a^b
– Multiplication:	*	ab	a*b
– Right Division:	/	a / b = a/b	a/b
– Left Division:	\	a \ b = b/a	a\b
– Addition:	+	a + b	a+b
– Subtraction:	-	a – b	a-b

- MATLAB will ignore white space between variables and operators

Examples

- What is the area of the green square?
- We know it is the sum of the area of the two smaller squares (for a right triangle)
- So it is: $5^2 + 11^2$

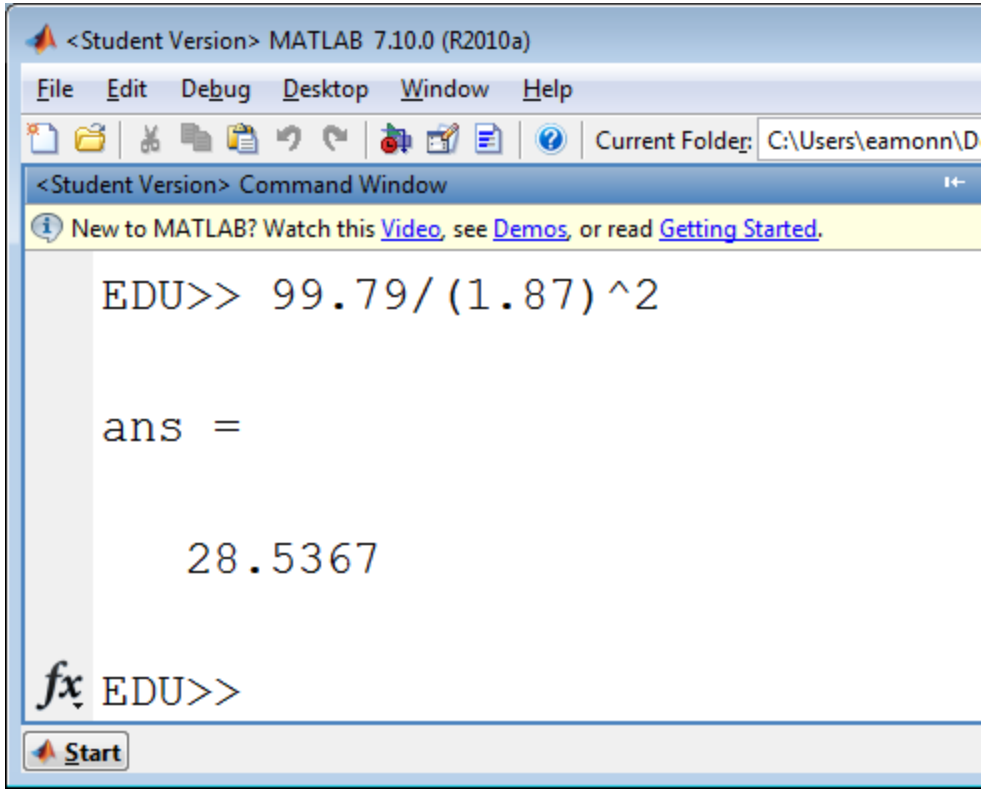




Examples

- What is Eamonn's BMI? He is 1.87m tall and 99.79 kg in weight

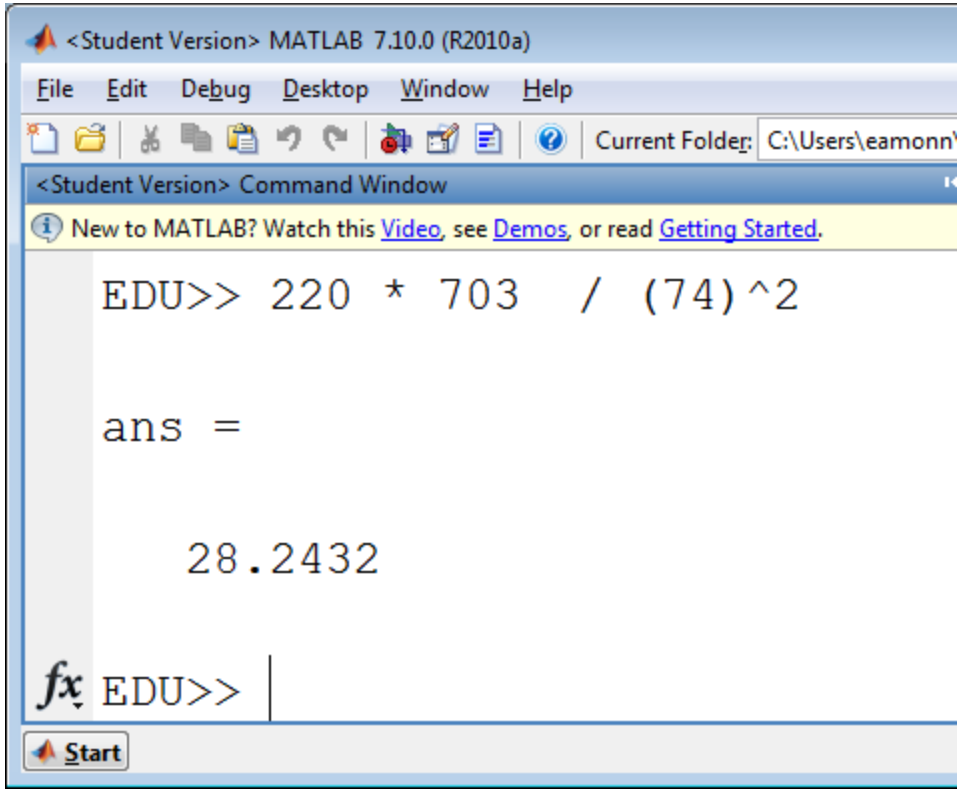
$$\mathrm{BMI} = \frac{\mathrm{mass\ (kg)}}{(\mathrm{height(m)})^2}$$



Examples

- What is Eamonn's BMI, using imperial units?
- He is 74 inches tall and 220 lbs in weight

$$\text{BMI} = \frac{\text{mass (lb)} \times 703}{(\text{height(in)})^2}$$



It is a little different to the metric version, due to *rounding*

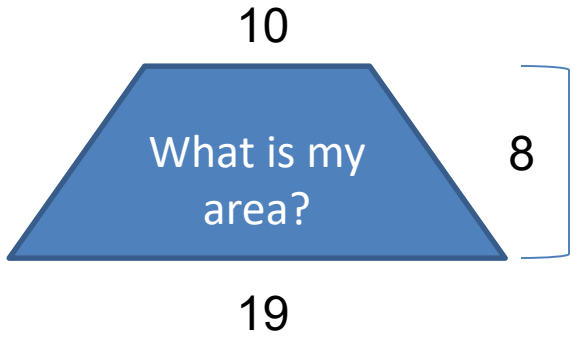
Precedence: Order of Operations

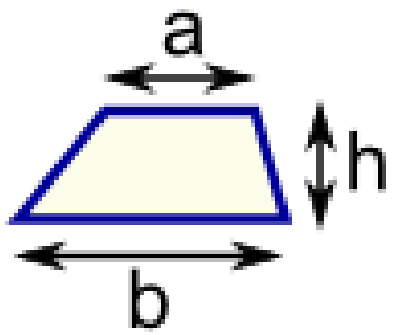
- Parentheses
- Exponentiation
- Multiplication and division have equal precedence
- Addition and subtraction have equal precedence
- Evaluation occurs from left to right
- When in doubt, use parentheses
  - MATLAB will help match parentheses for you

An expression inside parentheses is evaluated on its own before being used by operands outside the parentheses

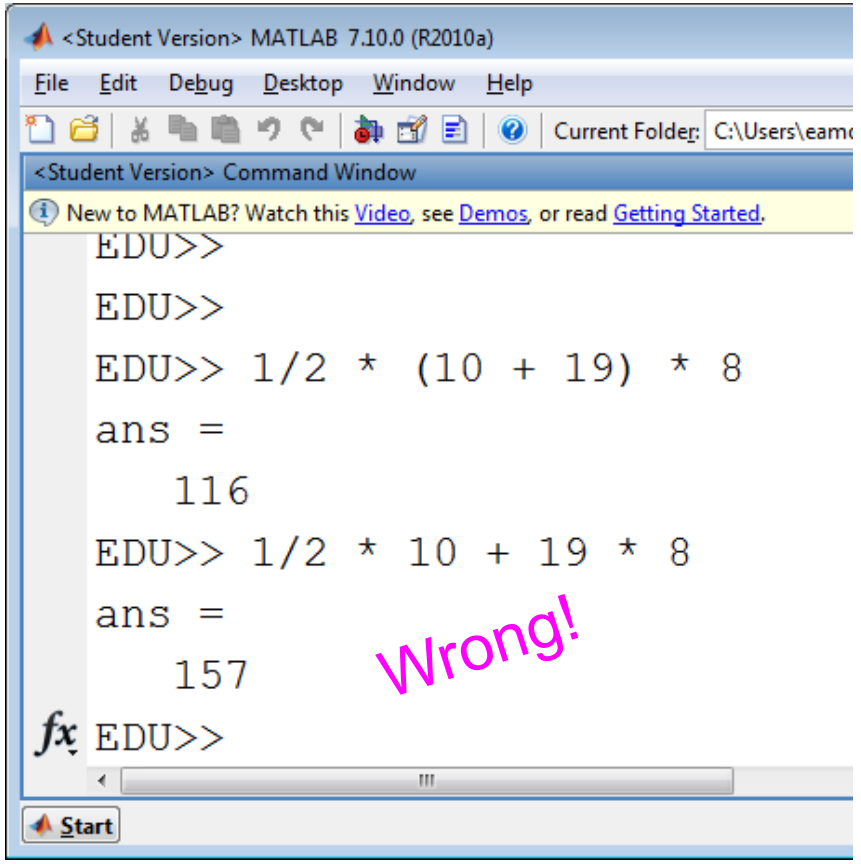
The acronym ***PEMDAS*** is common. It stands for *P*arentheses, *E*xponents, *M*ultiplication, *D*ivision, *A*ddition, *S*ubtraction.

Area of a Trapezoid





$$= \tfrac{1}{2}(a+b) \times h$$



In the wrong example Matlab did…

1/2 * 10 + 19 * 8

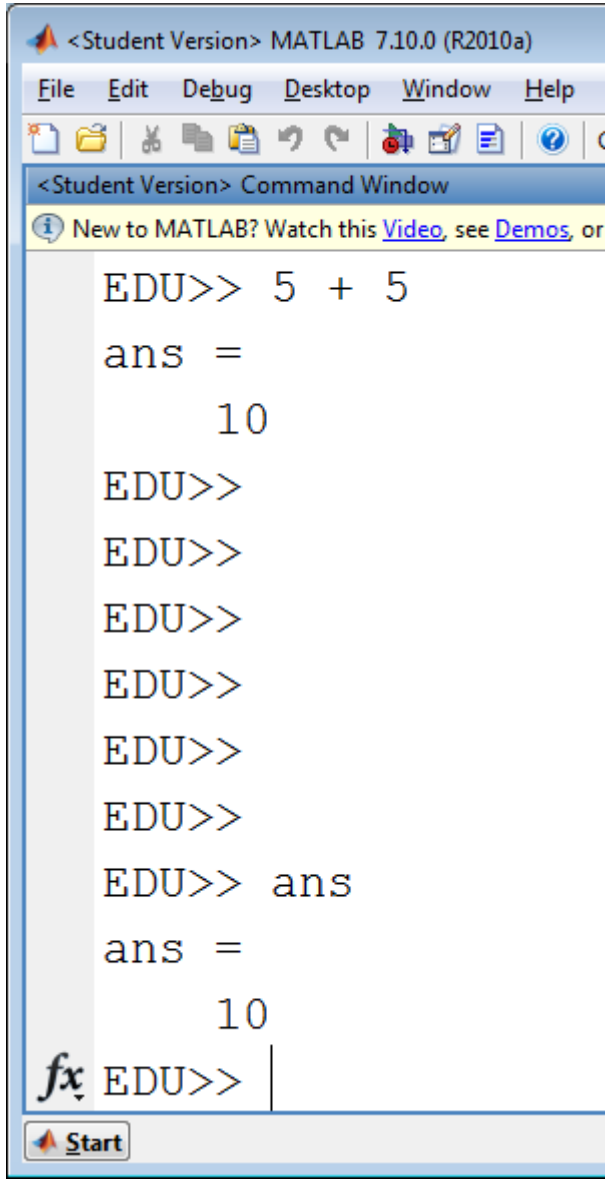
5 + 19 * 8

5 + 152

157

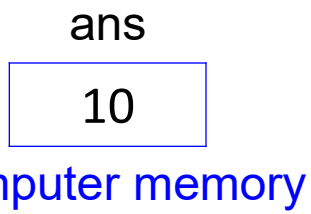
The value of ‘ans’, stays there forever, until we exit matlab, or we change it.

We can get the value of ‘ans’ at anytime simply by typing its name (and hitting <enter>)

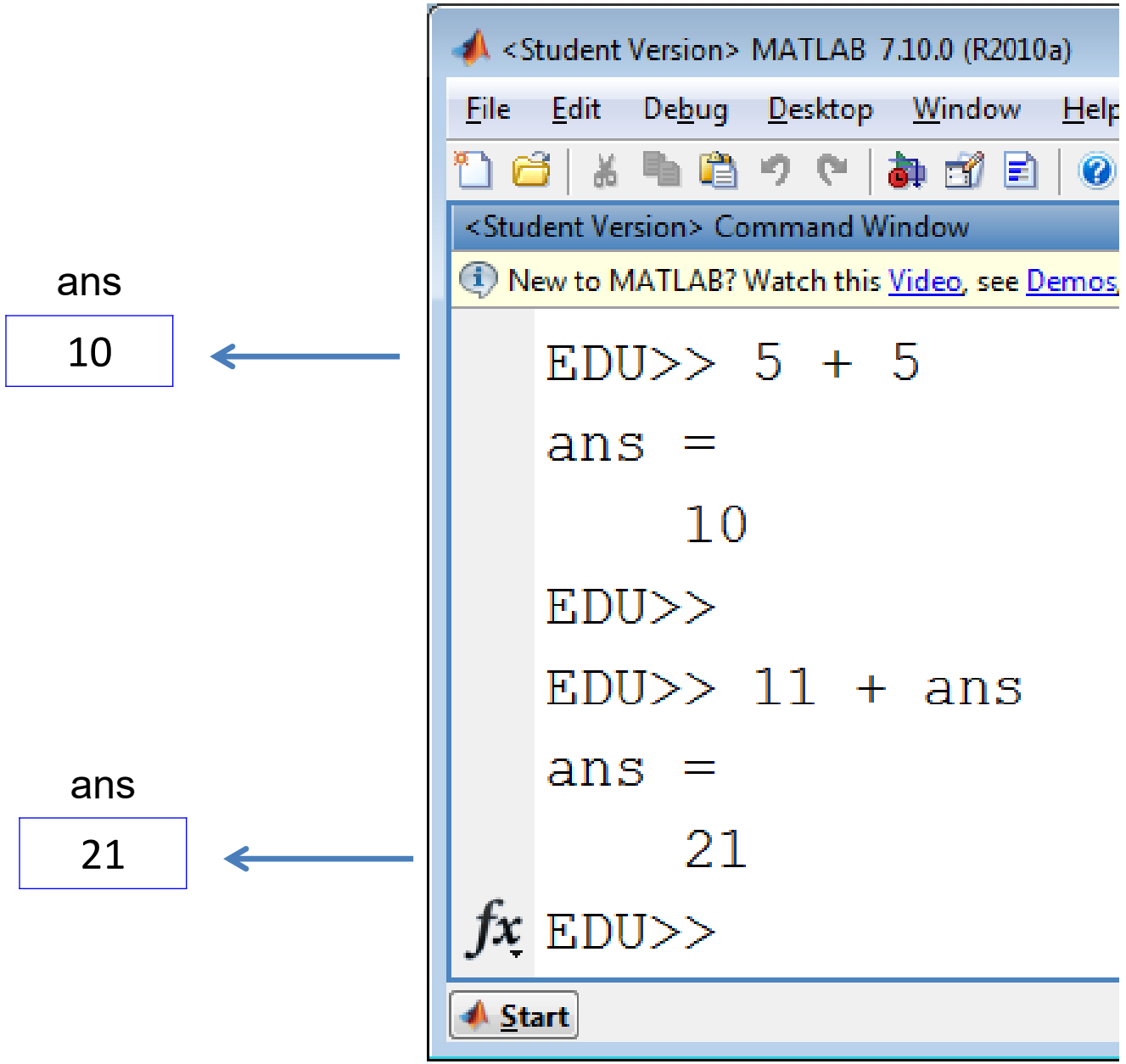


We can read the value in ans as many times as we want, without changing it

This is a *nondestructive read*



The variable ‘ans’, can appear in an expression

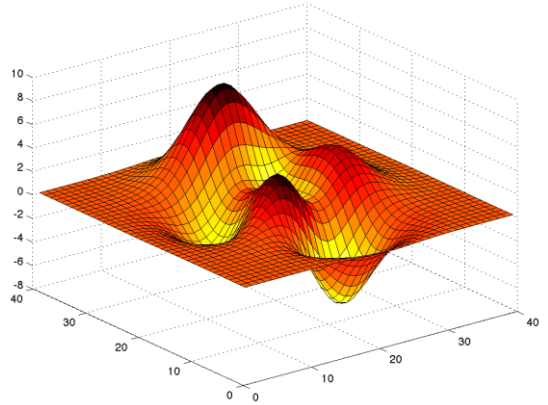


Element wise operations

Operators .*, ./, and .^

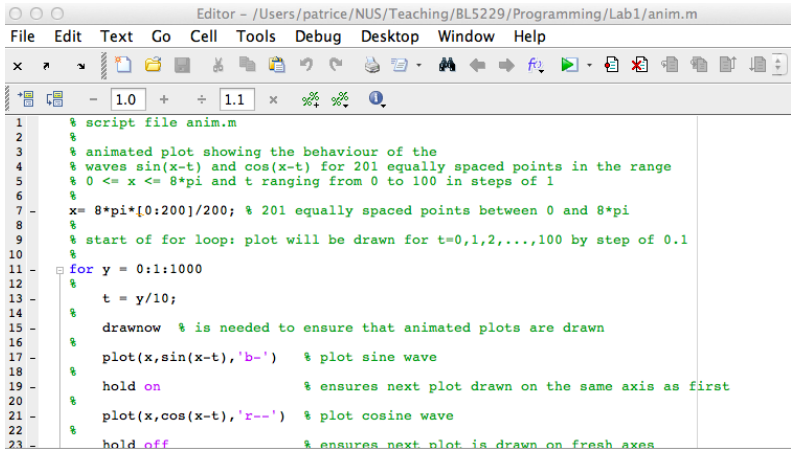
```
>> Z = [2 3 4]'

>> B = [Z.^2 Z Z.^0]

B=   4   2   1
     9   3   1
    16   4   1
```

Matlab



- The Matlab Environment
- Variables; operations on variables
- Programming
- Visualization

M-file programming



➢ Script M-Files

- Automate a series of steps.
- Share workspace with other scripts and the command line interface.

➢ Function M-Files

- Extend the MATLAB language.
- Can accept input arguments and return output arguments.
- Store variables in internal workspace.

M-file programming

- **Always has one script M-File**
- **Uses built-in and user-defined functions**
- **Created in MATLAB Editor**

>> edit model.m

- **Run from Command Line Window**

>> model

Example of script

Example: model.m

```
% Define input
T = [ 0 : 0.01 : 30]
% Compute model
Y = exp(-T);
% Plot model
plot (T, Y);
```

Example of function

Example: amodel.m

```
function Y = amodel(t, A, B, a, w, p)
% H1 line: AMODEL computes step response.
% Help text: appears when you type
% "help amodel" in command line window.

% Comment: function body is below.
Y = A * exp(-b.*t).*cos(w.*t + p) + B;
```

```
function DummyVar = CountToTen()

for i = 1 : 10
    disp(i)
end
```

```
EDU>> CountToTen
    1
    2
    3
    4
    5
    6
    7
    8
    9
   10
EDU>>
```

Input / Output

- Get input from command window:

```
>> num = input( ‘What is the altitude :’ )
>> str = input( ‘Enter name of the planet’ ,’ s’ )
```

- Display output in command window:

String

```
>> disp( ‘The answer is:’ )
```

String + number:

```
>> disp([ ‘The value of x is:  ‘ num2str(x)])
```

Example of a Function M-File

```
function answer = average3(arg1,arg2, arg3)

% A simple example about how MATLAB deals with user
% created functions in M-files.
% average3 is a user-defined function.  It could be named anything.
% average3 takes the average of the three input parameters: arg1,
% arg2, arg3
% The output is stored in the variable answer and returned to the
% user.
% This M file must be saved as average3.m

answer = (arg1+arg2+arg3)/3;
```

How to call a function in matlab

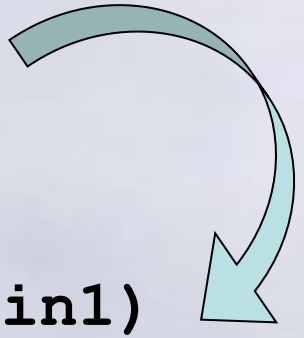
```
X = sin(pi);
% call the interior function of Matlab, sin

Y = input("How are you?")
% call the interior function of Matlab, input
% input is a function that requests information from
% the user during runtime


Y=average3(1, 2, 3);.
% average3 is a user-defined function.
% The average value of 1,2 and 3 will be stored in Y
```

Example: Calling a Function from within the Main program.

```
% Example1: This is a main program example to
  illustrate how functions are called.
% input is a function that requests information from
% the user during runtime.

VarA = input('What is the first number?');
VarB = input('What is the second number?');
VarC = input('What is the third number?');

VarAverage=average3(VarA, VarB, VarC);
% num2str converts a number to a string.
result=['The average value was', num2str(VarAverage)]
```

Functions

- A MATLAB function is very similar to a script, but:
  - Starts with a function declaration line
  - May have defined input arguments
  - May have defined output arguments



```
function [out1,out2]=function_name(in1)
```

  - Executes in its own workspace: it CANNOT see, or modify variables in the calling workspace – values must be passed as input & output variables

MATLAB function can return multiple values

```
function [x1,x2,x3]=powers1to3(n)
% calculate first three powers of [1:n]
% (a trivial example function)
x1=[1:n];
x2=x1.^2;
x3=x1.^3;
```

```
>> [x1,x2,x3]=powers1to3(4)
x1 =
     1     2     3     4

x2 =
     1     4     9    16

x3 =
     1     8    27    64
```

```
>> [x1,x2]=powers1to3(4)
x1 =
     1     2     3     4

x2 =
     1     4     9    16
```

If fewer output parameters used than are declared, only those used are returned.

Operators

- Arithmetic: `x+y;  A*B;  X.*Y;` etc.
- Logical
  - Element-wise AND: `a & b`
  - Element-wise OR: `a | b`
- Relational

  `a == 5;  a >= b;  b ~= 6;`
- Operator precedence

  `()  {}  []` -> Arithmetic -> Relational -> Logical

DIFFERENTIATION

Polynomial derivatives are easy to find

```
p(x) = x^2 –3*x + 5,
polyder(p)
```

p(x) = x^2 – 3*x + 5

is represented by the vector p = [1, -3, 5]

polyder(p) % returns coeff of derivative as 2 -3

polyval(polyder(p),1) % eval derivative at x=1

Diff is used for differentiation of arbitrary function

Program flow control: For

Simple program that sums the squares of all the elements of a matrix A:

```
N = 10;
M = 20;

A = rand(10,20)

Sum = 0;
for i = 1:N
    for j = 1:M
        Sum = Sum + A(i,j)^2;
    end
end
```

Note that this can be done in one line:

```
Sum2 = sum(sum(A.*A));
```

For loops

- **x = 0;**
  for i=1:2:5 % start at 1, increment by 2
  x = x+i; % end with 5.
  end

  This computes x = 0+1+3+5=9.

While loops

- **x=7;**
  while (x > = 0)
  x = x-2;
  end;

  This computes x = 7-2-2-2-2 = -1.

If statements

- **if (x == 3)**
  disp('The value of x is 3.');
  elseif (x == 5)
  disp('The value of x is 5.');
  else
  disp('The value of x is not 3 or 5.');
  end;

Program flow control: if

Simple program that compares two numbers a and b: set j to 1 if a>b, -1 if a<b, and 0 if a = b:

```
if a > b
    j = 1;
else if a < b
    j = -1;
else
    j = 0;
end
```

Break statements

- **break** – terminates execution of for and while loops. For nested loops, it exits the innermost loop only.

Switch statement

- **switch face**
  case {1}
  disp(‘Rolled a 1’);
  case {2}
  disp(‘Rolled a 2’);
  otherwise
  disp(‘Rolled a number >= 3’);
  end

- NOTE: Unlike C, ONLY the SWITCH statement between the matching case and the next case, otherwise, or end are executed. *(So breaks are unnecessary.)*

switch, case

Generic form:

```
switch statement
case value1
  statements;
case value2
  statements;
case value2
  statements;
otherwise
  statements;
end
```

example

```
switch A*B
case 0
  disp('A or B is zero')
case 1
  disp('A*B equals 1')
case C*D
  disp('A*B equals C*D')
otherwise
  disp('no cases match')
end
```

A case is matched when the switch statement equals the case value (may be the result of a statement). Only first matching case is executed, then switch statement exits, remaining cases are not tested.

continue, break, return

- **`continue`** – forces current iteration of loop to stop and execution to resume at the start of next iteration.
- **`break`** – forces loop to exit, and execution to resume at first line after loop.
- **`return`** – forces current function to terminate, and control to be passed back to the calling function or keyboard.

Cell arrays

- Cell arrays are arrays of arbitrary data objects, as a whole they are dimensioned similar to regular arrays/matrices, but each element can hold any valid matlab data object: a single number, a matrix or array, a string, another cell array…
- They are indexed in the same manner as ordinary arrays, but with curly brackets

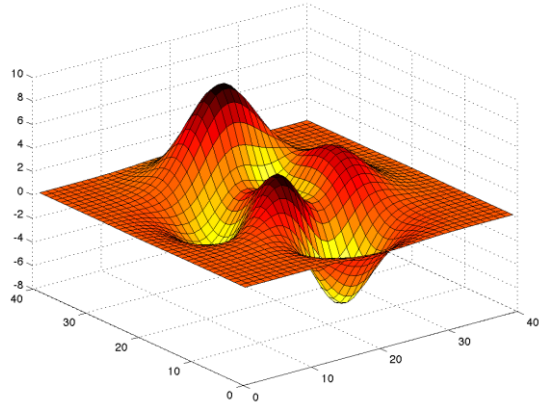
```
>> X{1}=[1 2 3 4];
   >> X{2}='some random text'
   X =
       [1x4 double]    'some random text'
```

Other useful commands

- **Workspace**
  - >> clear
  - >> who
  - >> whos
  - >> close

- **File operations**
  - >> ls
  - >> dir
  - >> cd
  - >> pwd
  - >> mkdir

Matlab



- The Matlab Environment
- Variables; operations on variables
- Programming
- Visualization

Plotting

- MATLAB will plot one vector vs. another. The first one will be treated as the abscissa (or x) vector and the second as the ordinate (or y) vector. The vectors have to be the same length.
- MATLAB will also plot a vector vs. its own index. The index will be treated as the abscissa vector. Given a vector "time" and a vector "dist" we could say:

>> plot (time, dist) % plotting versus time
>> plot (time + i*dist) % plotting versus time
>> plot (dist) % plotting versus index

- Sometime we want to see it with different color\line stile

>> plot (time, dist, line_characteristics)

- And sometimes we want to plot few functions in graphs

>> plot(…), hold, plot(…)
>> plot(t,d1,l_c1, t,d2, l_c2)

- To split page to several axes check use

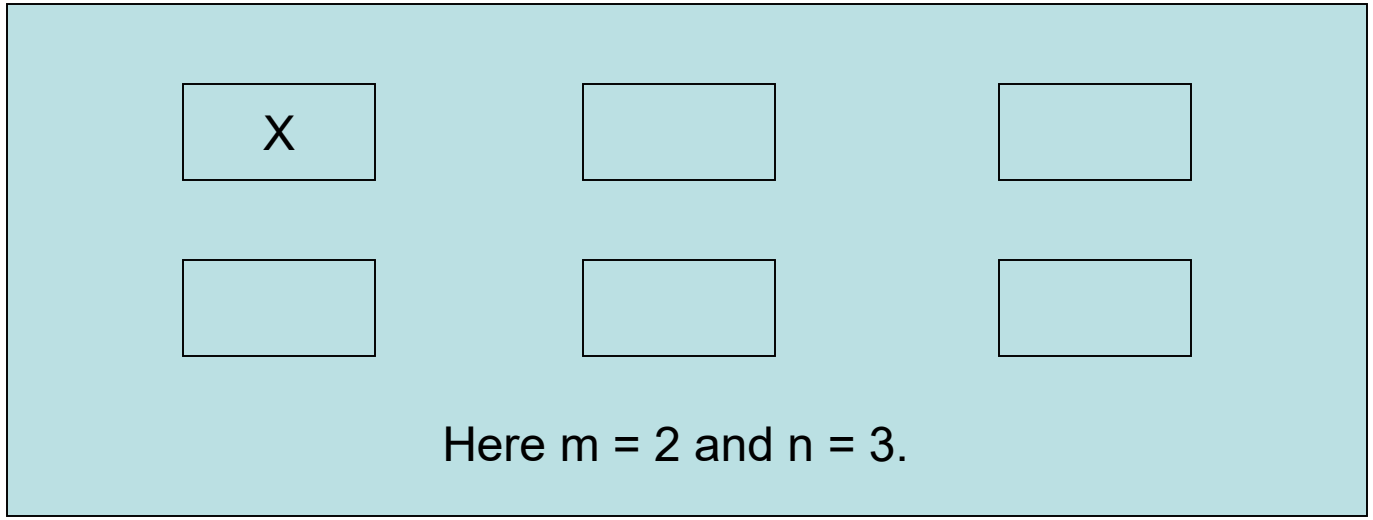
>> subplot (rows, cols, place)

Graphics

- **x = linspace(-1,1,10);**
- **y = sin(x);**
- **plot(x,y);** % plots y vs. x.
- **plot(x,y,'k-');** % plots a black line of y vs. x.
- **hold on;** % put several plots in the same figure window.
- **figure;** % open new figure window.

Graphics (2)

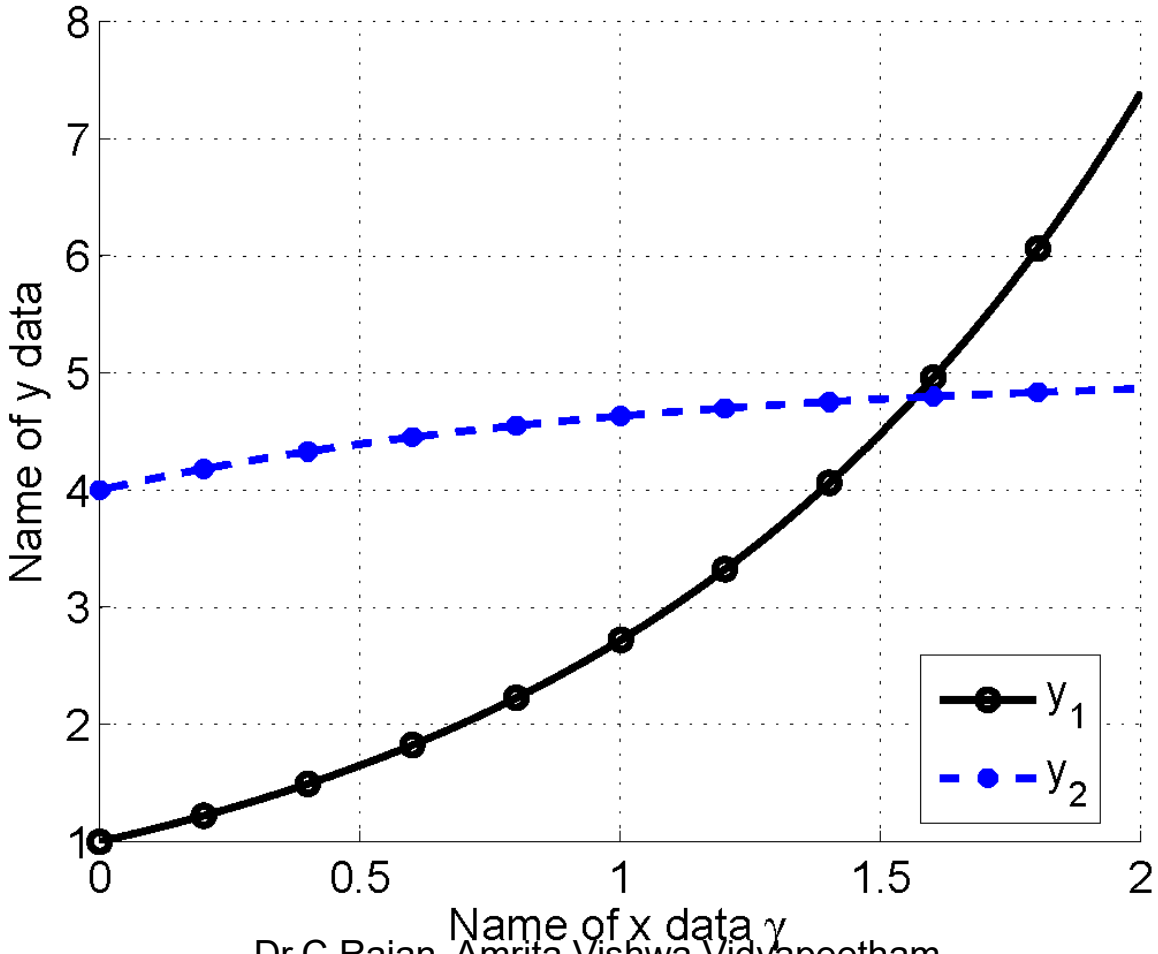
- **subplot(m,n,1)** % Makes an mxn array for plots. Will place plot in 1st position.



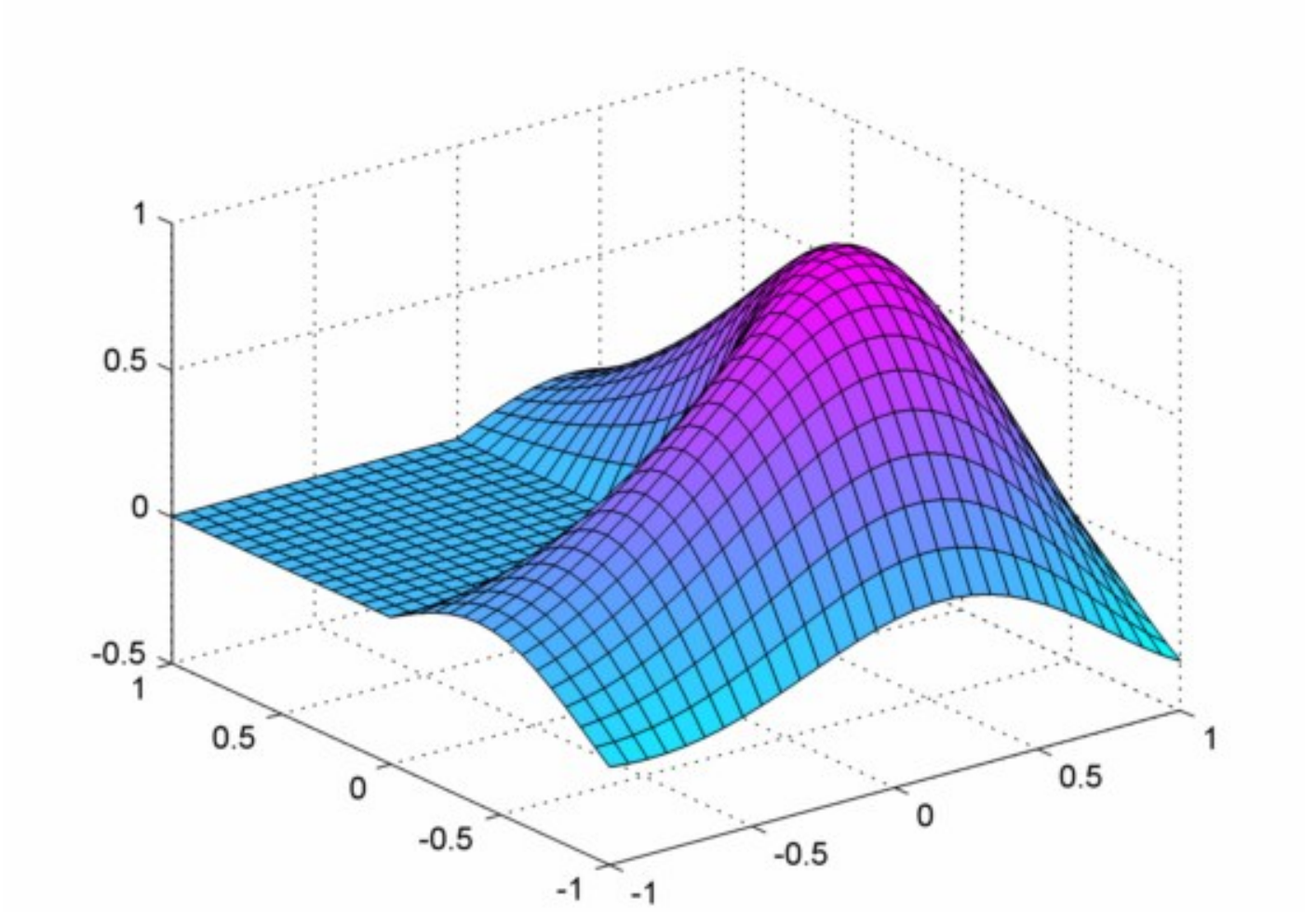
Graphics (3)

- **plot3(x,y,z)** % plot 2D function.
- **mesh(x_ax,y_ax,z_mat)** – surface plot.
- **contour(z_mat)** – contour plot of z.
- **axis([xmin xmax ymin ymax])** – change axes
- **title('My title');** - add title to figure;
- **xlabel, ylabel** – label axes.
- **legend** – add key to figure.

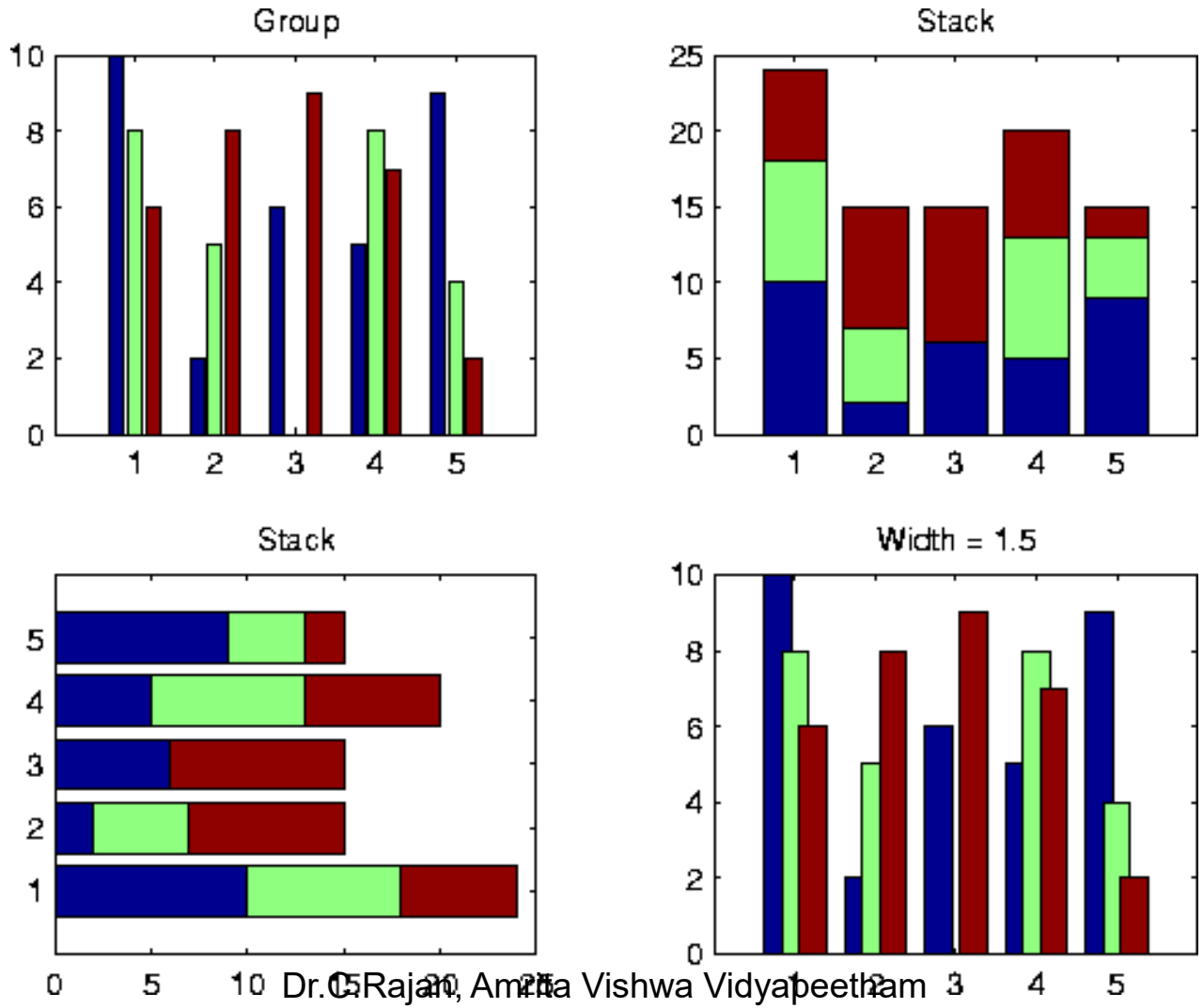
Examples of Matlab Plots



Examples of Matlab Plots



Examples of Matlab Plots



- Linear plots

```
>> plot (X, Y)
```

Plotting commands open the Figure editor.

- Multiple datasets on a plot

```
>> plot(xcurve, ycurve)
>> hold on
>> plot(Xpoints, Ypoints)
>> hold off
```

- Subplots on a figure

```
>> subplot(1, 2, 1)
>> plot(time, velocity)
>> subplot(1, 2, 2)
>> plot(time, acceleration)
```

- 2D linear plots: `plot`

```
>> plot (X, Y, 'r-')
```

Colors: b, r, g, y, m, c, k, w

Markers: o, *, ., +, x, d

Line styles: -, --, -., :

- Annotating graphs

```
>> plot (X, Y, 'ro')
>> legend ('Points')
>> title ('Coordinates')
>> xlabel ('X')
>> ylabel ('Y')
```

- Plot Edit mode: icon  in Figure Editor

Vectors in Matlab

- Row vectors

```
>> R1 = [1 6 3 8 5]

>> R2 = [1 : 5]

>> R3 = [-pi : pi/3 : pi]
```

- Column vectors

```
>> C1 = [1; 2; 3; 4; 5]

 >> C2 = R2'
```

Arrays and Matrices

- **v = [-2 3 0 4.5 -1.5];** % length 5 row vector.
- **v = v';** % transposes v.
- **v(1);** % first element of v.
- **v(2:4);** % entries 2-4 of v.
- **v([3,5]);** % returns entries 3 & 5.
- **v=[4:-1:2];** % same as v=[4 3 2];
- **a=1:3; b=2:3; c=[a b];** → c = [1 2 3 2 3];

Matrices in Matlab

- **Creating a matrix**

```
>> A = [1 2.5 5 0; 1 1.3 pi 4]
>> A =  [R1; R2]

>> A = zeros(10,5)

>> A = ones(10,5)

>> A = eye(10)
```

- **Accessing elements**

```
>> A(1,1)

>> A(1:2, 2:4)

>> A(:,2)
```

Matrix Operations

- **Operators + and –**

```
>> X = [1 2 3]
>> Y = [4 5 6]
>> A = X + Y
   A=   5  7   9
```

- **Operators *, /, and ^**

```
>> Ainv = A^-1 Matrix math is default!
```

Arrays and Matrices (2)

- **x = linspace(-pi,pi,10);** % creates 10 linearly-spaced elements from –pi to pi.
- **logspace** is similar.
- **A = [1 2 3; 4 5 6];** % creates 2x3 matrix
- **A(1,2)** % the element in row 1, column 2.
- **A(:,2)** % the second column.
- **A(2,:)** % the second row.

Arrays and Matrices (3)

- **A+B**, **A-B**, **2*A**, **A*B** % matrix addition, matrix subtraction, scalar multiplication, matrix multiplication
- **A.*B** % element-by-element mult.
- **A'** % transpose of A (complex-conjugate transpose)
- **det(A)** % determinant of A

Creating special matrices

- **diag(v)** % change a vector v to a diagonal matrix.
- **diag(A)** % get diagonal of A.
- **eye(n)** % identity matrix of size n.
- **zeros(m,n)** % m-by-n zero matrix.
- **ones(m,n)** % m*n matrix with all ones.

Matrices

Data in Matlab is always held as a **matrix**. Most are 2-D, in which individual elements are referenced by *row* and *column*. This is a 6×10 matrix:.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	12	15	12	18	19	20	19	17	16	15
2	13	14	61	19	11	9	10	12	14	19
3	13	15	18	17	19	20	23	11	10	12
4	14	15	14	14	17	20	21	10	9	12
5	12	13	13	19	30	35	36	15	19	15
6	15	16	17	18	45	40	38	16	15	12

If this matrix is given the name `m`, then `m(5,4) = 19`, for example.

Remember: ***row*** then ***column***.

In Matlab, rows and columns are always numbered starting at 1.

Matrices

Some commands

Transpose	B = A′
Identity Matrix	eye(n) ➔ returns an n x n identity matrix eye(m,n) ➔ returns an m x n matrix with ones on the main diagonal and zeros elsewhere.
Addition and subtraction	C = A + B C = A – B
Scalar Multiplication	B = αA, where α is a scalar.
Matrix Multiplication	C = A*B
Matrix Inverse	B = inv(A), A must be a square matrix in this case. rank (A) ➔ returns the rank of the matrix A.
Matrix Powers	B = A.^2 ➔ squares each element in the matrix C = A * A ➔ computes A*A, and A must be a square matrix.
Determinant	det (A), and A must be a square matrix.

A, B, C are matrices, and m, n, α are scalars.

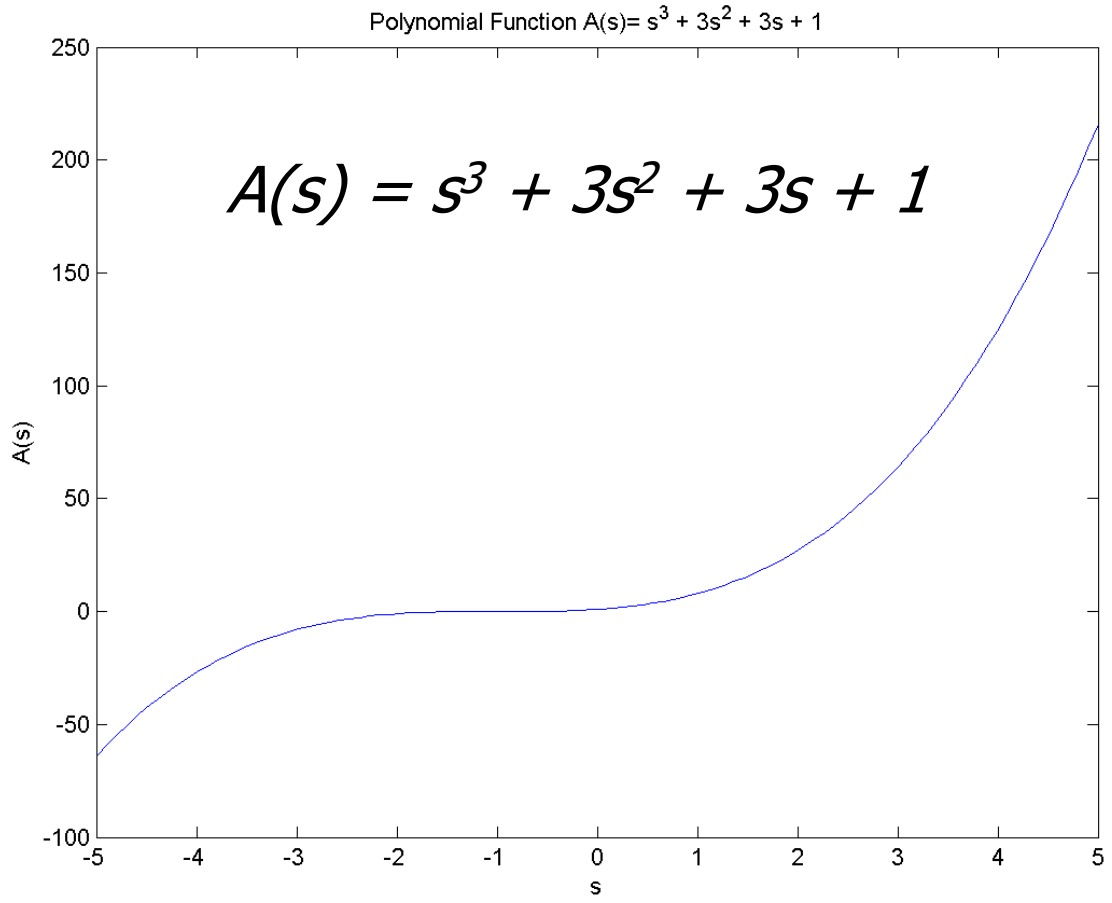
Matrices

Some operators must be handled with care:

```
A = [1 2 ; 4 5]

B = A * A     prints

 9   12
24   33

B = A .* A    prints

 1    4
16   25
```


Element by element multiplication

Visualization: Plotting



- Example:

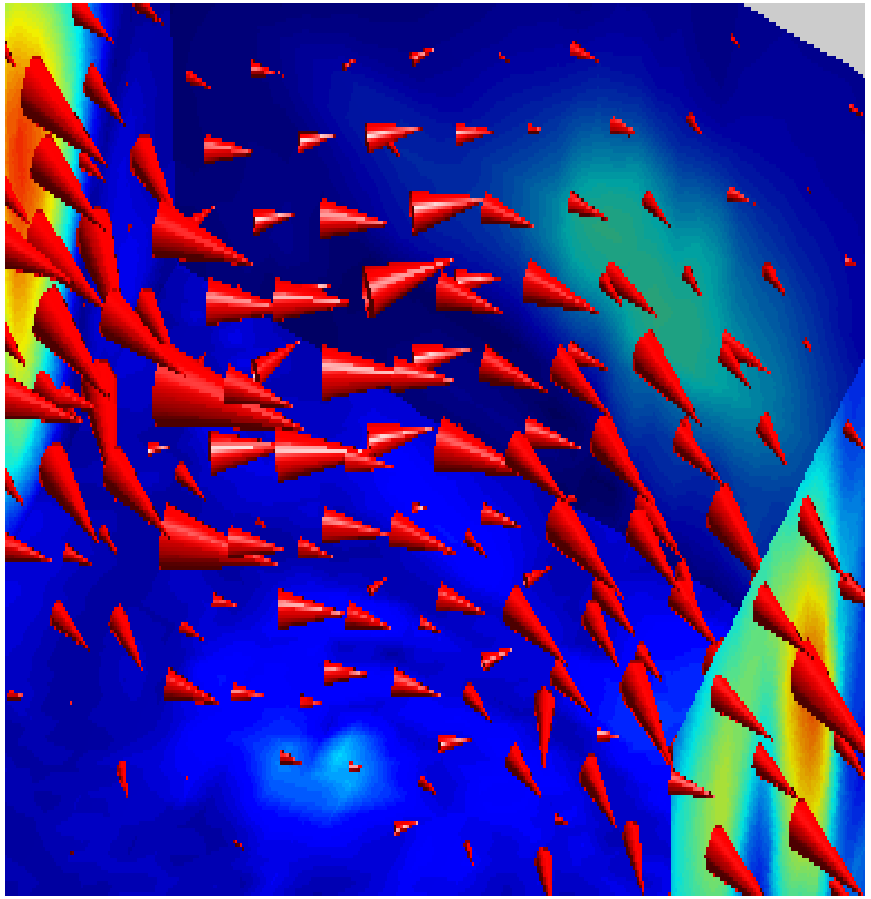
```
>> s = linspace (-5, 5, 100);
>> coeff = [ 1 3 3 1];
>> A = polyval (coeff, s);
>> plot (s, A),
>> xlabel ('s')
>> ylabel ('A(s)')
```

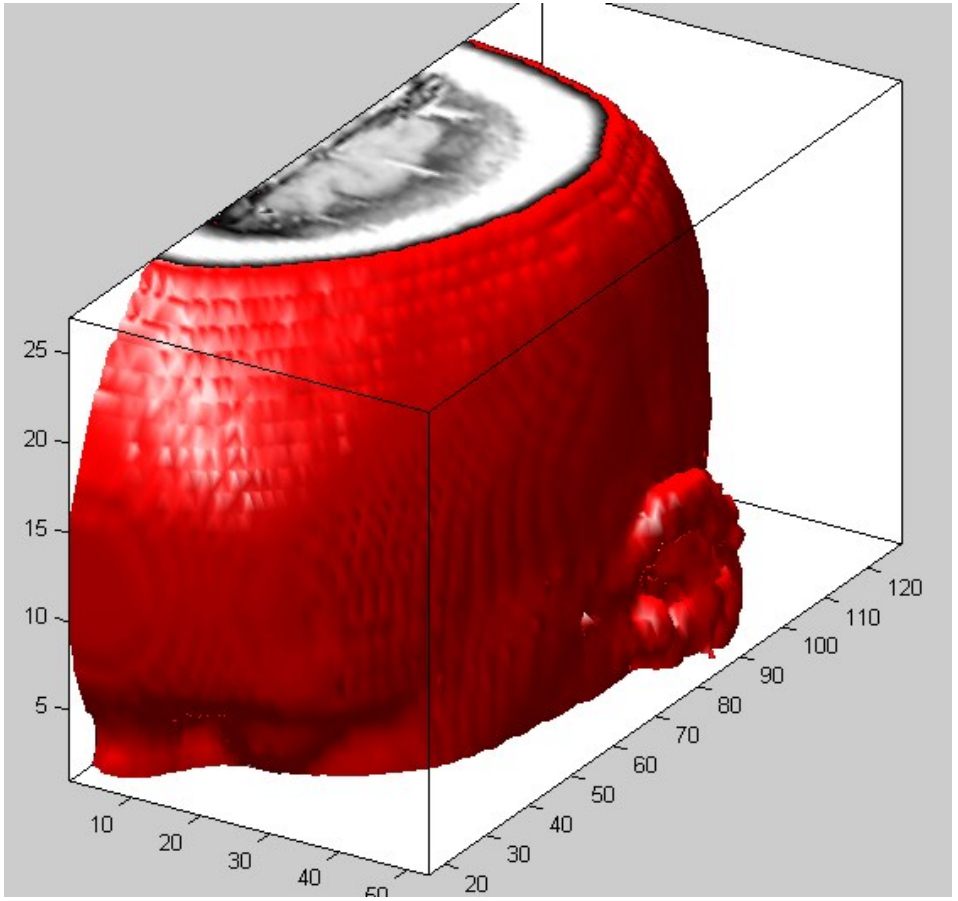
Plotting (con't)

- Plot a Helix

```
t = linspace (-5, 5, 101);
x = cos(t);
y = sin(t);
z = t
plot3(x,y,z);
box on;
```

Advanced Visualization





File Input/Output

- **fid = fopen('in.dat','rt');** % open text file for reading.
- **v = fscanf(fid,'%lg',10);** % read 10 doubles from the text file.
- **fclose(fid);** % close the file.
- **help textread;** % formatted read.
- **help fprintf;** % formatted write.

Example Data File

Sally	Type1	12.34	45	Yes
Joe	Type2	23.54	60	No
Bill	Type1	34.90	12	No

Read Entire Dataset

fid = fopen('mydata.dat', 'r'); % open file
for reading.

% Read-in data from mydata.dat.

[names,types,x,y,answer] =
textread(fid,'%s%s%f%d%s');

fclose(fid); % close file.

Read Partial Dataset

fid = fopen('mydata.dat', 'r'); % open file
for reading.

% Read-in first column of data from mydata.dat.
[names] = textread(fid,'%s %*s %*f %*d %*s');

fclose(fid); % close file.

Read 1 Line of Data

fid = fopen(‘mydata.dat’, ‘r’); % open file
% for reading.
% Read-in one line of data corresponding
% to Joe’s entry.
[name,type,x,y,answer] =… textread(fid,’%s%s%f%d%s’,1,…
’headerlines’,1);
fclose(fid); % close file.

Writing formatted data.

% open file for writing.

fid = fopen(‘out.txt’,’w’);

% Write out Joe’s info to file.

fprintf(fid,’%s %s %f %d…
%s\n’,name,type,x,y,answer);

fclose(fid); % close the file.

Keeping a record

- To keep a record of your session, use the **diary** command:

diary filename
x = 3
diary off

This will keep a diary called filename showing the value of x (your work for this session).

Timing

- Use **tic**, **toc** to determine the running time of an algorithm as follows:

  tic

  commands…

  toc

  This will give the elapsed time.