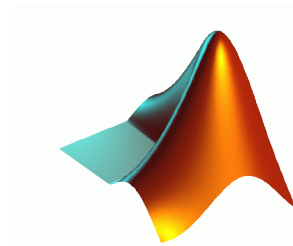




COMS W3101-2

# Programming Languages: MATLAB



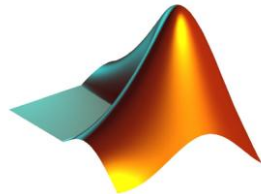
## Lecture 1

Spring 2010

Instructor: Michele Merler

# Course Information – Instructor

- ▶ Michele Merler
  - Email: [mmerler@cs.columbia.edu](mailto:mmerler@cs.columbia.edu)
  - Office : 624 CEPSR
  - Office Hours: TDB
- ▶ 3<sup>rd</sup> year PhD Student in CS Department
- ▶ Research Interests:
  - Image & Video Processing
  - Multimedia
  - Computer Vision



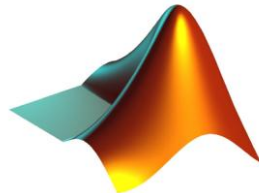
# Course Information – TA

## ▶ Daniel Miao

- Email: [dm2701@columbia.edu](mailto:dm2701@columbia.edu)
- Office : TA room
- Office Hours: Mon 10am – 12pm

## ▶ Rohit Sethi

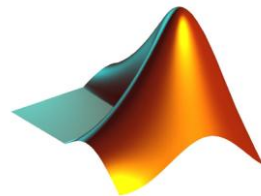
- Email: [rs2990@columbia.edu](mailto:rs2990@columbia.edu)
- Office: TA room
- Office Hours: Wed 3.30pm – 5.30pm



# Course Information – Goals

Learn how to use MATLAB for:

- ▶ Solve problems in Science and Engineering
- ▶ Perform Matrix and Vector Operations
- ▶ Compute Complex Mathematical Functions
- ▶ Plotting and Visualization
- ▶ Perform Simulations and Prototyping



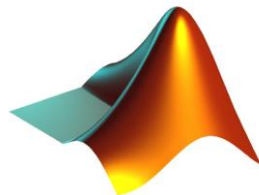
# Course Information – Syllabus

## ▶ Week 1 – March 2

- Data Structures (Variables, Vectors, Matrices)
- Types (int, double, single)
- Operators
- Basic Plotting
- Scripts

## ▶ Week 2 – March 9

- Plotting (continued)
- Control flow (if\_else, for, while, loops)



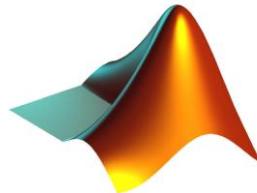
# Course Information – Syllabus

## ▶ Week 3 – ~~March 16~~ March 23

- I/O (from files, images, loading/saving variables)
- User input
- Advanced data structures (cell, struct)
- Debugging
- Functions

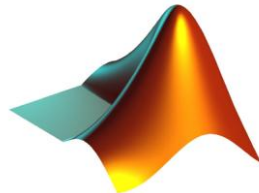
## ▶ Week 4 – March 30

- Figures
- Images
- Videos



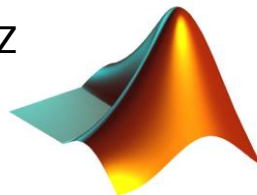
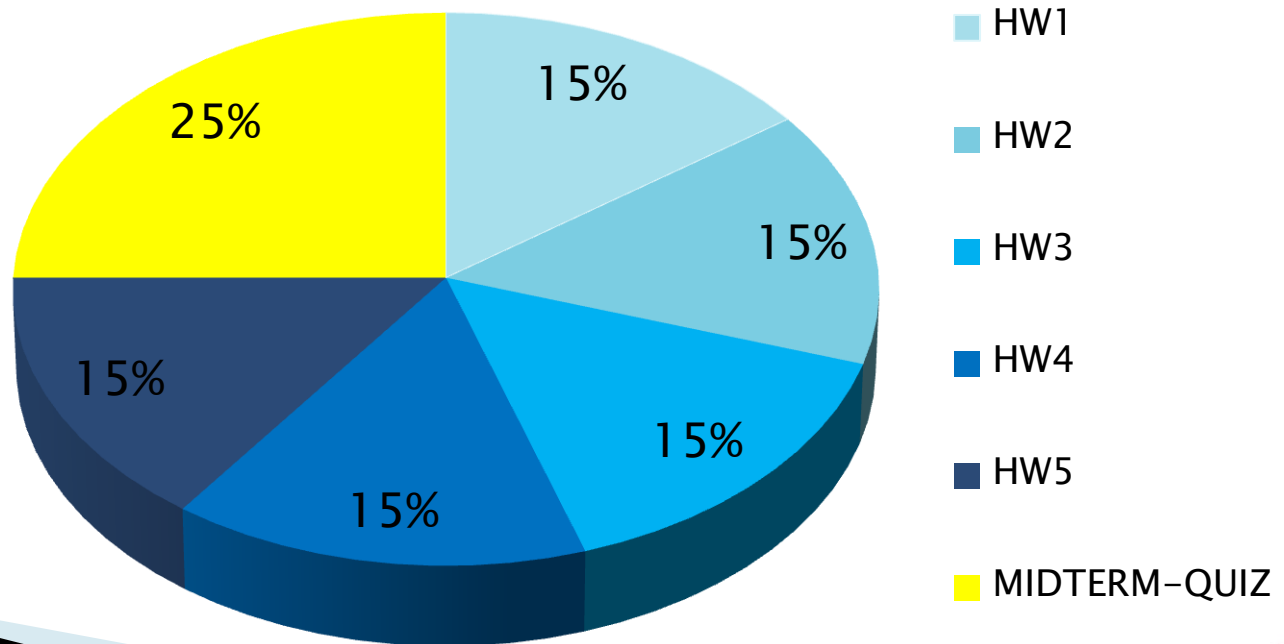
# Course Information – Syllabus

- ▶ Week 5 – April 6
  - Math and Linear Algebra
  - Solving Equations, basic statistics
- ▶ Week 6 – April 13
  - Final Useful things
  - Object Oriented Programming
  - GUI
  - Simulink & other Toolboxes



# Course Information – Grading

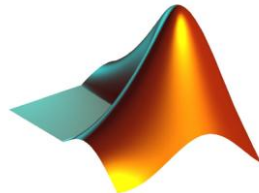
- ▶ 5 Homeworks (15%, 15%, 15% , 15% , 15%)
- ▶ 1 Midterm Quiz (25%) In class March. 30





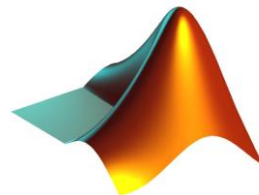
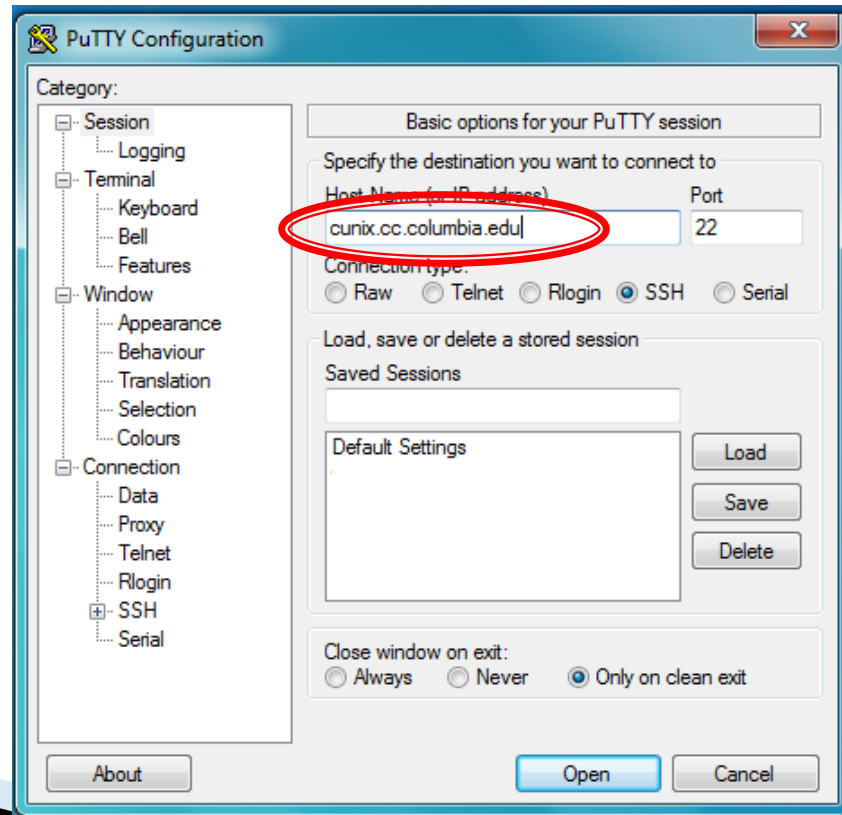
# Technical Details

- ▶ Download Xming and Putty (for Windows)
  - <http://sourceforge.net/projects/xming/>
  - <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>



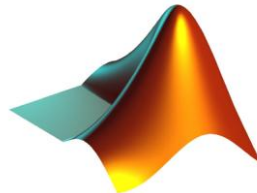
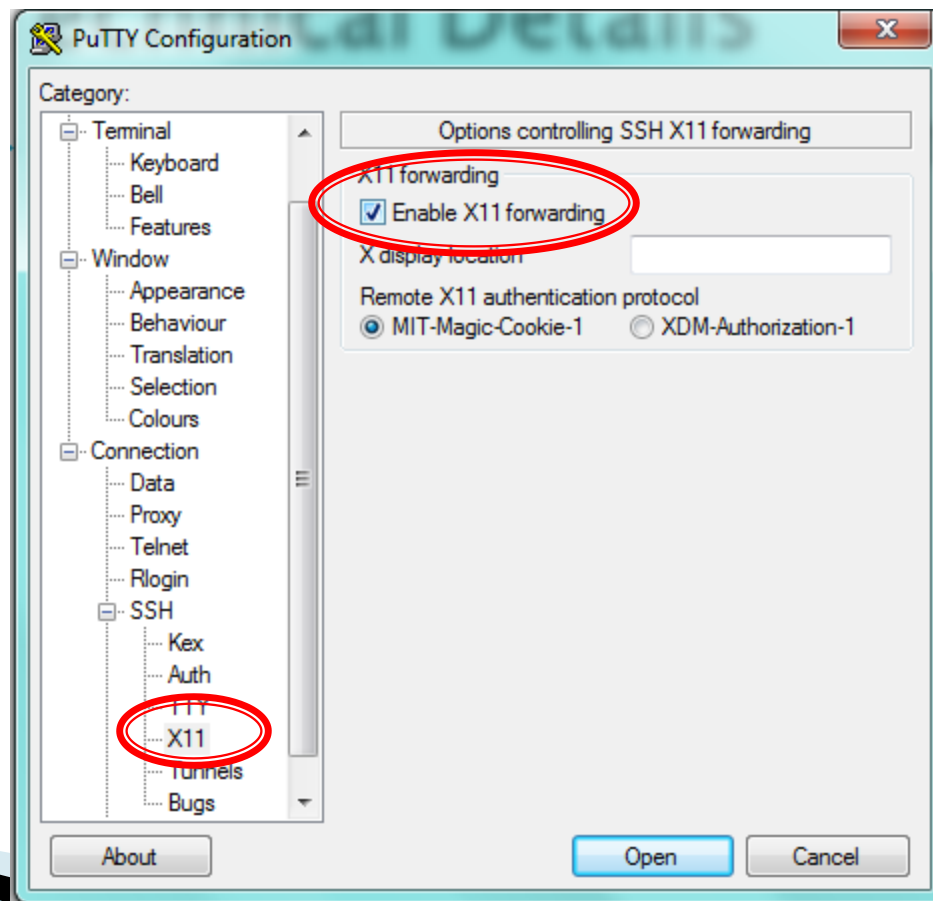
# Technical Details

- ▶ Launch Xming
- ▶ Open a session in putty with Host Name
  - `cunix.cc.columbia.edu`



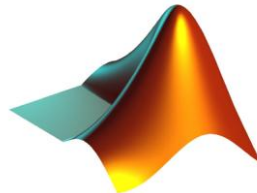
# Technical Details

- ▶ Make sure the X11 option of the SSH category is enabled



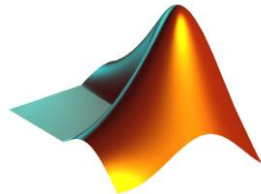
# Technical Details

- ▶ Enter your cunix credentials
- ▶ Type
  - \$ matlab &



# What is MATLAB?

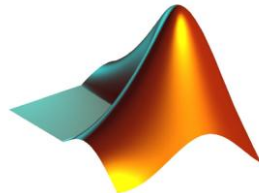
- ▶ Programming Environment
- ▶ Calculator
- ▶ Programming Language
- ▶ The solution to all your problems



# What is MATLAB?

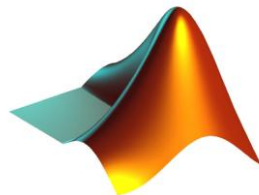
MATLAB® is a high-level **language** and **interactive environment** that enables you to perform computationally intensive tasks ~~faster~~ than with traditional programming languages such as C, C++, and Fortran

<http://www.mathworks.com/products/matlab/>



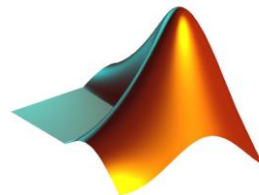
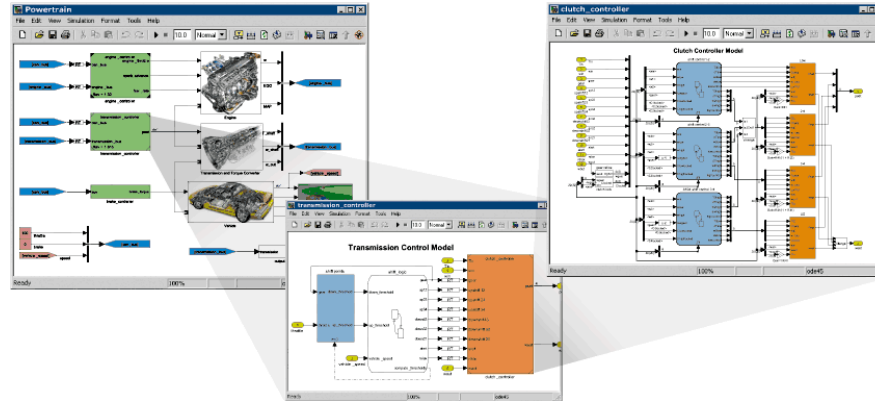
# What can you do with Matlab?

- ▶ Design
- ▶ Compute
- ▶ Visualize



# What can you do with Matlab?

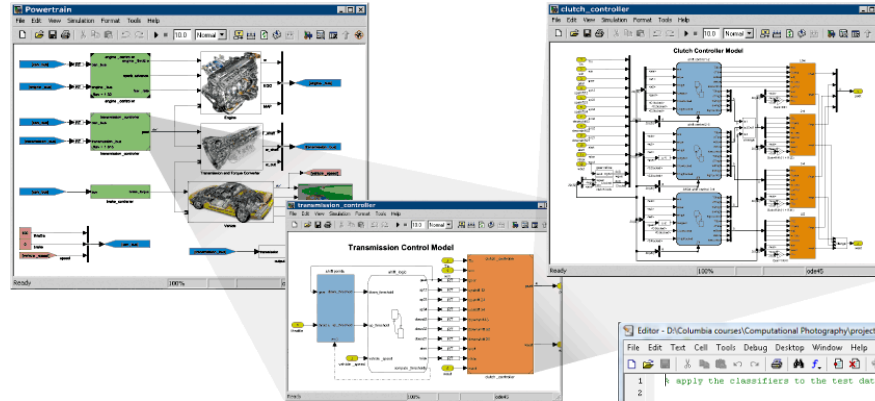
- ▶ Design
- ▶ Compute
- ▶ Visualize





# What can you do with Matlab?

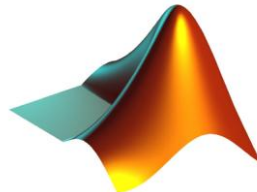
- Design



- Compute

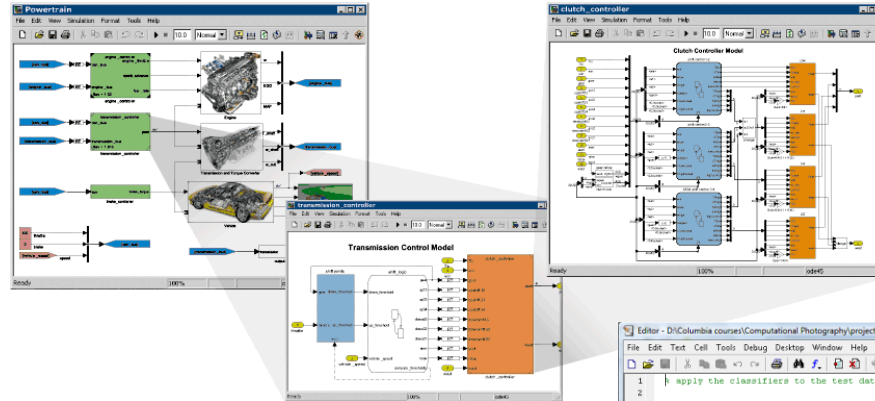
```
1 apply the classifiers to the test data
2
3
4 clear all
5 clc
6
7 featureTypes = ('edgeHistogramFeatures','colorMomentsFeatures','gistFeatures','colorHistogramFeatures');
8 modelTypes = ('edgeHistogram','colorMoments','gist','colorHistogram');
9
10 featureType = featureTypes(1);
11 modelType = modelTypes(1);
12
13 indexFeatures = 50;
14
15 useKnn = 0;
16 k = 1;
17
18 % read categories names
19 letters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
20 load concept;
21 clf = load('characterClassifier');
22
23 for i=1:200
24     1
25     conceptsToFind = [];
26     folder = sprintf('./TestImages/challenge%d',i);
27     fid = fopen(sprintf('%s/challenge%d.txt',folder,i));
28     for j=1:2
29         tline = fgetl(fid);
30     end
31 end
```

- Visualize



# What can you do with Matlab?

► Design

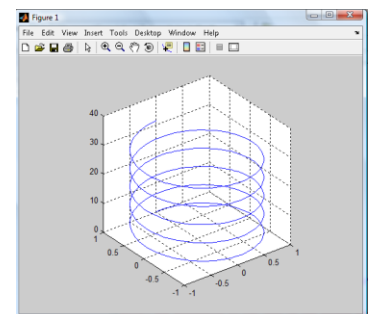
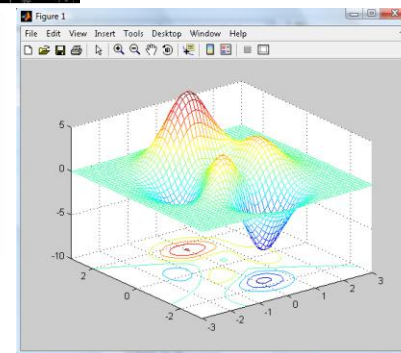
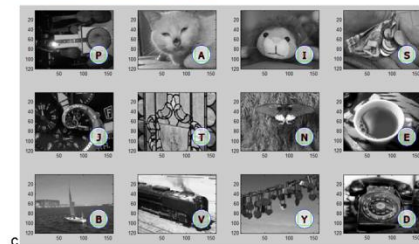


► Compute

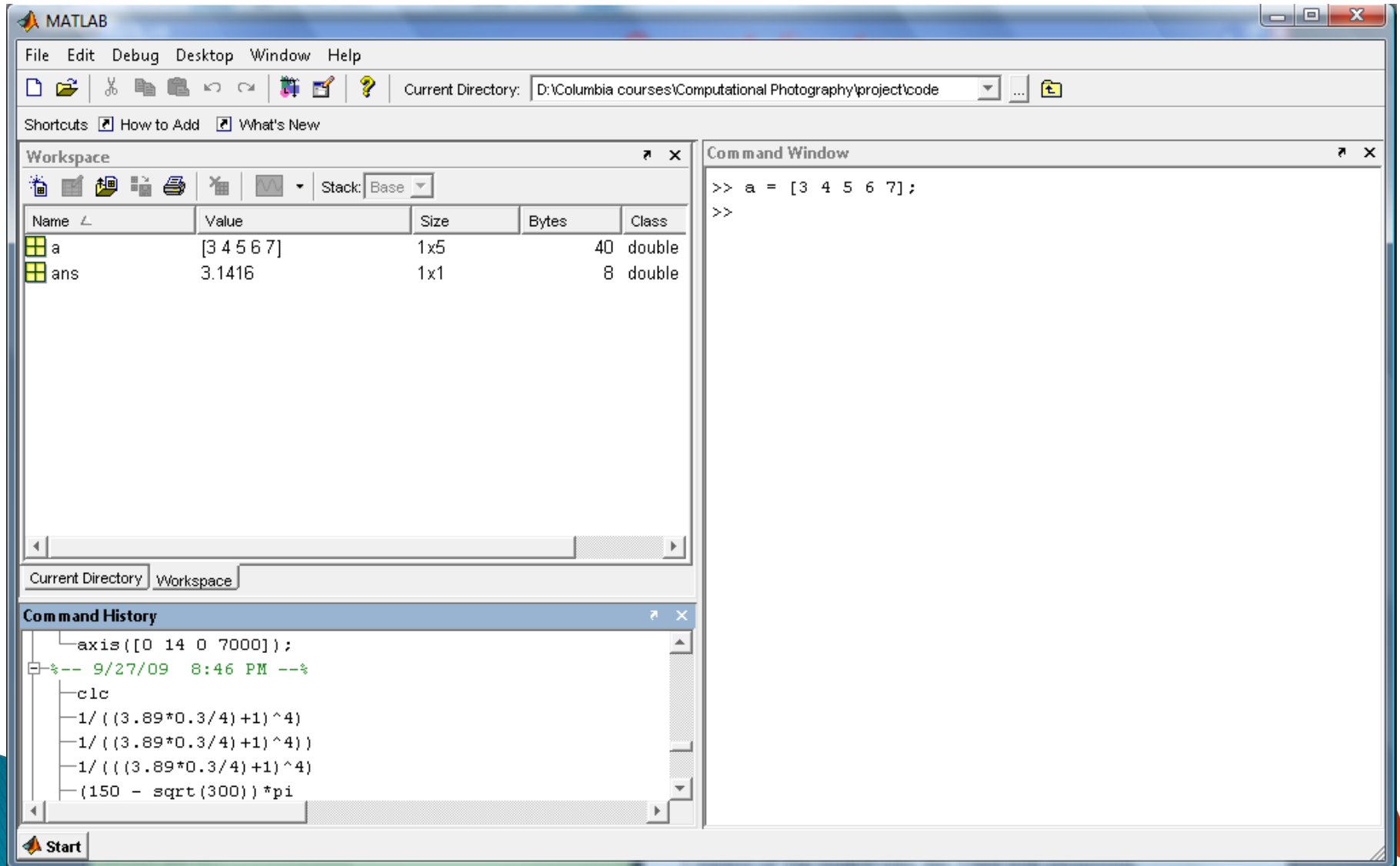


```
1 % apply the classifiers to the test data
2
3
4 clear all
5 clc
6
7 featureTypes = {'edgeHistogramFeatures','colorHistogramFeatures','gistFeatures','colorHistogramFeatures'};
8 modelTypes = {'edgeHistogram','colorHistogram','gist','colorHistogram'};
9
10 featureType = featureTypes(1);
11 modelType = modelTypes(1);
12
13 indexFeatures = 50;
14
15 useKnn = 0;
16 k = 1;
17
18 % read categories names
19 letters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
20 load concept
21 clf = load('characterClassifier');
22
23 for i=1:200
24     1
25     conceptToFind = [];
26     folder = sprintf('./TestImages/challenge%d',i);
27     fid = fopen(sprintf('%s/challenge%d.txt',folder,i));
28     for j=1:2
29         tline = fgetl(fid);
30     end
31 end
```

► Visualize

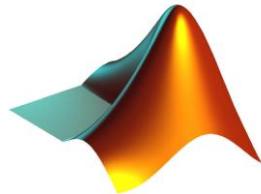


# MATLAB Interface



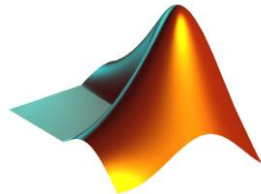
# Basics

- ▶ MATLAB records in the workspace and command history everything you write in the command window, so:
- ▶ `clear variable`
  - deletes variable from memory (and workspace)
- ▶ `clear all`
  - deletes all variables from memory (and workspace)
- ▶ `clc`
  - cleans command window



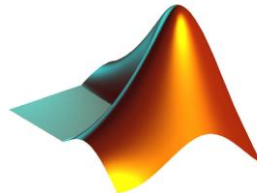
# Basics

- ▶ MATLAB's command window works like a Linux terminal
- ▶ Some example commands:
  - `cd`
  - `mkdir, rmdir`
  - `ls`
  - `⋮`



# Basics

- ▶ Some commands used to interact with MATLAB
  - `what`
    - returns the MATLAB files (.m , .mat) in the current directory
  - `who`
    - returns the variables in your workspace
  - `whos`
    - returns the variables in the workspace with additional info (size, dimensions)

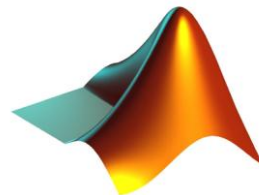
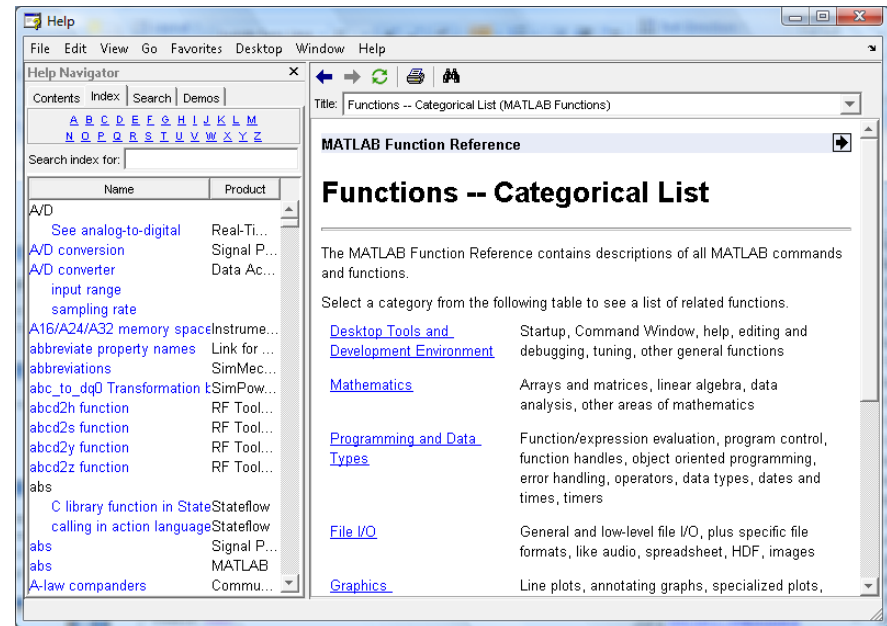


# Help

Meet your best friend...

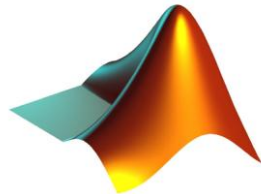
- ▶ Start → Help
- ▶ Press **?** in interface
- ▶ Type `doc name_function`

... what about `help name_function` ?



# Data Structures – Variables

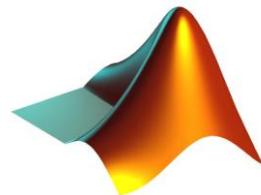
- ▶ MATLAB does not use explicit type initialization like other languages
  - ▶ Just assign some value to a variable name, and MATLAB will automatically understand its type
    - `int x`
    - `x = 3`
    - `x = 'hello'`
- `double`  
`char` } Most common types
- ▶ We can assign mathematical expressions to directly create variable
    - `x = (3 + 4)/2`
  - ▶ `;` operator prevents the variable to be printed in the command window
    - `x = 3;`
  - ▶ `disp` prevents `ans=` from being displayed
    - `disp(x)`





# Data Structures – Variables

- ▶ MATLAB does not use explicit type initialization like other languages
  - ▶ Just assign some value to a variable name, and MATLAB will automatically understand its type
    - ~~int~~ x
    - x = 3
    - x = 'hello'
- double  
char } Most common types
- ▶ We can assign mathematical expressions to directly create variable
    - x = (3 + 4)/2
  - ▶ ; operator prevents the variable to be printed in the command window
    - x = 3;
  - ▶ disp prevents ans= from being displayed
    - disp(x)



# Data Structures – Variables

## ▶ Naming Conventions

- Letter case matters

A = 2  
a = 4       $\searrow$   
                  $\nearrow$  These are 2 different variables!

- Avoid using functions names for variables

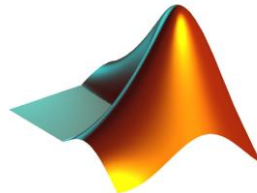
Example: `sin = 2`  
            `a = sin(0.5)`



`sin` cannot be used as  
a function any more!

## ▶ Built-in Variables

- `i` and `j` indicate complex numbers
- `pi` = 3.1415926...
- `ans` = last unassigned value
- `Inf` and `-Inf` = positive and negative infinity
- `NaN` = 'Not a Number'



# Data Structures – Arrays and Matrices

- ▶ This is really what MATLAB is all about!

- ▶ Row vectors

- `r = [2 3 5 7];`
- `r = [2, 3, 5, 7];`

[1x4]

2	3	5	7
---	---	---	---

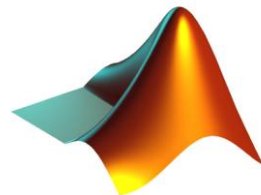
- ▶ Column vectors

- `c = [2; 3; 5; 7];`
- `c = [2 3 5 7]';`

[4x1]

2
3
5
7

Transpose operator



# Data Structures – Vectors

## ► Special Vectors Constructors

- `:` operator

- `x = 1:3:13;`

Spacing, default = 1

[1x5]

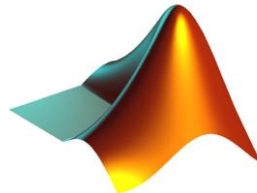
1	4	7	10	13
---	---	---	----	----

- `linspace()`

- `x = linspace(0,10,100);`

Creates a vector of 100 elements with values equally spaced between 0 and 10 (included)

- Equivalent notation with `:` operator?



# Data Structures – Matrices

## ► Explicit Definition

- $M = [2 \ 4; \ 3 \ 6; \ 8 \ 12];$

[3x2]

2	4
3	6
8	12

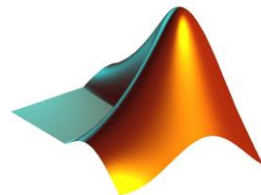
## ► Concatenation of vectors

- $r1 = [2 \ 4];$
- $r2 = [3 \ 6];$
- $r3 = [8 \ 12];$
- $M = [r1; \ r2; \ r3];$

## ► Concatenation of vectors and matrices

- $r1 = [2 \ 4];$
- $m1 = [3 \ 6; \ 8 \ 12];$
- $M = [r1; \ m1];$

Dimensions and Type must coincide!



# Data Structures – Matrices

## ► Some Predefined Matrix Creation Functions

- double {
- `M = zeros(2,3);` [3x2] matrix of zeros  
                    rows columns
  - `M = ones(2,3);` [3x2] matrix of ones
  - `M = eye(2);` [2x2] identity matrix
  - `M = rand(2,3);` [2x3] matrix of uniformly distributed random numbers in range [0,1]
  - `M = randn(2,3)` [2x3] matrix of normally distributed random numbers (mean 0, std dev. 1)

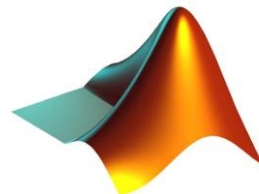
0	0	0
0	0	0

1	1	1
1	1	1

1	0
0	1

0.2	0.86	0.1
1	0	0.33

-1.2	-0.86	0.1
1.256	0.435	-1.33



# Data Structures – Matrices

## ► Replicating and concatenating matrices

### ◦ `repmat`

- `X = [1 2 3; 4 5 6];`
- `Y = repmat(X,2,4);`

X

1	2	3
4	5	6

Y

1	2	3	1	2	3	1	2	3	1	2	3
4	5	6	4	5	6	4	5	6	4	5	6
1	2	3	1	2	3	1	2	3	1	2	3
4	5	6	4	5	6	4	5	6	4	5	6

### ◦ `vertcat`

- `x1 = [2 3 4];`
- `x2 = [1 2 3];`
- `X = vertcat(x1,x2);`

x1

2	3	4
---	---	---

x2

1	2	3
---	---	---

X

2	3	4
1	2	3

### ◦ `horzcat`

- `x1 = [2; 3; 4];`
- `x2 = [1; 2; 3];`
- `X = horzcat(x1,x2);`

x1

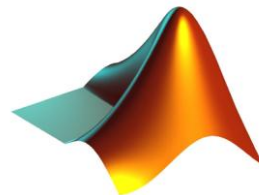
2
3
4

x2

1
2
3

X

2	1
3	2
4	3



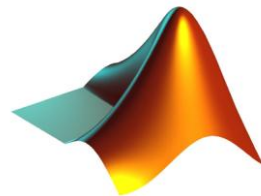
# Data Structures – Matrices

## ► Getting the size of the matrix

- `M = [2 3 4; 3 4 55];`

- `[r c] = size(M);`
- `r = size(M,1);`
- `c = size(M,2);`

} `r = 2;`  
`c = 3;`





# Data Structures – Matrices

## ▶ Accessing Elements of Matrix M

- Matrix indexing starts with **1** !

- Explicit access

- `element = M(2,3);`
  - `element = M(5);`

- **:** operator

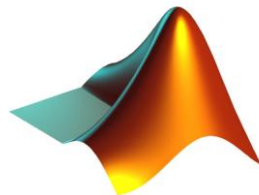
- `element = M(1,1:2);`
  - `element = M(:,1);`

- **end** operator

- `element = M(1,2:end);`

M

-1.2	-0.86	0.1
1.256	0.435	-1.33



# Data Structures – Matrices

## ▶ Accessing Elements of Matrix M

- Matrix indexing starts with **1** !

- Explicit access

- element = M(2,3);
- element = M(5);

- **:** operator

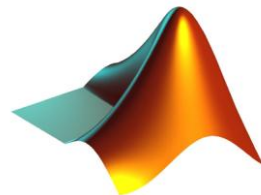
- element = M(1,1:2);
- element = M(:,1);

- **end** operator

- element = M(1,2:end);

M

-1.2	-0.86	0.1
1.256	0.435	-1.33



# Data Structures – Matrices

## ▶ Accessing Elements of Matrix M

- Matrix indexing starts with **1** !

- Explicit access

- `element = M(2,3);`
- `element = M(5);`

- `:` operator

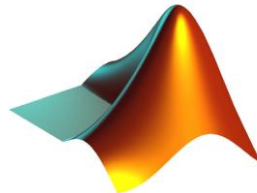
- `element = M(1,1:2);`
- `element = M(:,1);`

- `end` operator

- `element = M(1,2:end);`

M

-1.2	-0.86	0.1
1.256	0.435	-1.33



# Data Structures – Matrices

## ▶ Accessing Elements of Matrix M

- Matrix indexing starts with **1** !

- Explicit access

- `element = M(2,3);`
- `element = M(5);`

- `:` operator

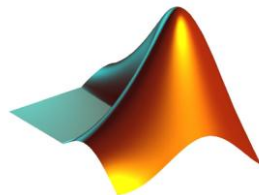
- `element = M(1,1:2);`
- `element = M(:,1);`

- `end` operator

- `element = M(1,2:end);`

M

-1.2	-0.86	0.1
1.256	0.435	-1.33



# Data Structures – Matrices

## ▶ Accessing Elements of Matrix M

- Matrix indexing starts with **1** !

- Explicit access

- `element = M(2,3);`
  - `element = M(5);`

- `:` operator

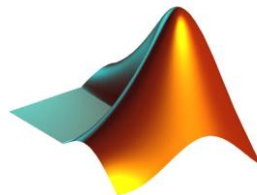
- `element = M(1,1:2);`
  - `element = M(:,1);`

- `end` operator

- `element = M(1,2:end);`

M

-1.2	-0.86	0.1
1.256	0.435	-1.33



# Data Structures – Matrices

## ▶ Accessing Elements of Matrix M

- Matrix indexing starts with **1** !

- Explicit access

- `element = M(2,3);`
  - `element = M(5);`

- `:` operator

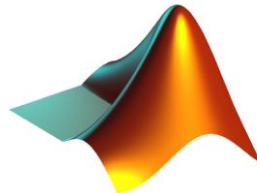
- `element = M(1,1:2);`
  - `element = M(:,1);`

- `end` operator

- `element = M(1,2:end);`

M

-1.2	-0.86	0.1
1.256	0.435	-1.33

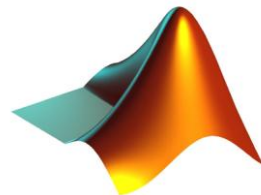


# Types

Type name	bits	Example
double	64	<code>x = 32</code>
char	16	<code>x = 'as'</code>
(u)int8	8	<code>x = (u)int8(32)</code>
(u)int16	16	<code>x = (u)int16(32)</code>
(u)int32	32	<code>x = (u)int32(32)</code>
(u)int64	64	<code>x = (u)int64(32)</code>
single float	32	<code>x = single(32)</code>
complex	128 (64+64)	<code>x = complex(2,1)</code>
logical	1	<code>x = true, x = logical([1 0 1])</code>

► Note on complex numbers:

- `x = 3 + 4j;`
- `x = complex(3,4);`



# Operators

## ▶ Basic Mathematical Operators

- `+` `-` `*` `/` `\` `^`

## ▶ Some more complex mathematical functions

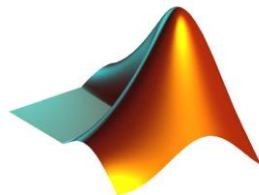
- `sqrt()`
- `log()`, `exp()`
- `sin()`, `cos()`, `tan()`, `atan()`
- `abs()`, `angle()`
- `round()`, `floor()`, `ceil()`
- `conj()`, `imag()`, `real()`
- `sign()`

## ▶ Logical Operators

- `&` `|` `~`

## ▶ Relational Operators

- `>` `<` `>=` `<=` `==` `~=`





# Operators

## ► Operators on matrices

- `X = [2 3 4; 5 4 6];`

- `Y = [1 2 3; 3 3 3];`

- `Rplus = X + Y;`

- `Rminus = X - Y;`

- `Rmult = X * Y;` ??? Error using ==> mtimes  
Inner matrix dimensions must agree.

- `X2 = X' ;`

- `Rmult = X2 * Y;`

- `Rpoint_mult = X .* Y;`

`Rpoint_mult`

X

2	3	4
5	4	6

Y

1	2	3
3	3	3

Rplus

3	5	7
8	7	9

Rminus

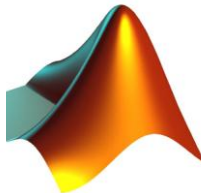
1	1	1
2	1	3

Rmult

4	9	16
25	16	36

2	6	12
15	12	18

Some operators, like `+` and `-`, are always element wise !  
Other operators, like `*` and `/`, must be disambiguated with `.` !



# Operators

## ► Operators on matrices

- `R = X ^ 2` ??? Error using ==> mpower  
Matrix must be square
- `X2 = [1 2 3; 3 4 5; 1 1 1];`
- `Rsquare = X2 ^ 2;`

$$\text{Rsquare} = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 10 & 13 & 16 \\ 20 & 27 & 34 \\ 5 & 7 & 9 \end{bmatrix}$$

- `Rdot = X .^ 2`

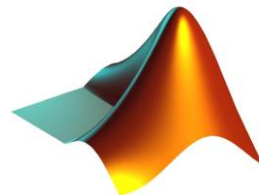
Rdot

4	9	16
25	16	36

X	2	3	4
	5	4	6

Y	1	2	3
	3	3	3

X2	1	2	3
	3	4	5
	1	1	1



# Operators

`X = [1 2 3; 4 5 6];`

X	1	2	3
	4	5	6

## ► Special Functions for Matrices

### ◦ `sum(), prod()`

- `SumCols = sum(X);`
- `SumRows = sum(X,2);`
- `SumTot = sum(sum(X));`

SumCols 

5	7	9
---	---	---

SumRows 

6
15

SumTot = 21

### ◦ `mean()`

- `MeanCols = mean(X);`
- `MeanRows = mean(X,2);`
- `MeanTot = mean(mean(X));`

MeanCols 

2.5	3.5	4.5
-----	-----	-----

MeanRows 

2
5

MeanTot = 3.5

### ◦ `max(), min()`

- `MaxVal = max(max(X));`
- `minCols = min(X);`
- `minRows = [min(X(1,:));min(X(2,:))];`
- `minRows2 = min(X,2)≡min(X,2*ones(size(X)))`

MaxVal = 6

minCols 

1	2	3
---	---	---

minRows 

1
4

`min`

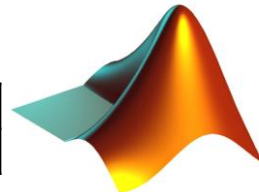
1	2	3
4	5	6

 , 

2	2	2
2	2	2

 → 

1	2	2
2	2	2



# Operators

`X = [1 2 13 4 5 6];`

X 

1	2	13	4	5	6
---	---	----	---	---	---

## ► Special Functions for Matrices

### ◦ `max()`, `min()` – continued

- `[maxVal maxLoc] = max(X);`      `maxVal = 13, maxLoc = 3`

MATLAB also tells us the location  
of the maximum value!

### ◦ `sort()` – orders the elements of a vector in ascending (default) or descending order

- `xAsc = sort(X);`

xAsc 

1	2	4	5	6	13
---	---	---	---	---	----

- `[xDes order] = sort(X, 'descend');`

xDes 

13	6	5	4	2	1
----	---	---	---	---	---

order 

3	6	5	4	2	1
---	---	---	---	---	---

### ◦ `find()`

- `R = find(X > 4);`

R 

3	5	6
---	---	---

- `R = find(X == 13);`

R = 3

`X = [1 2 13; 4 5 6];`

X 

1	2	13
4	5	6

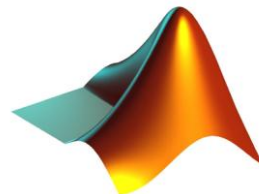
- `R = find(X >= 2 & X < 6)';`

R 

2	3	4
---	---	---

- `[r c] = find(X == 6);`

r=2 c=3



# Matrix indexing

- ▶ If we want to define the position of element 1 within the matrix M, we can do it with a single index or with the indexes of row and column

- `M = [2 4; 3 6; 5 1; 8 12];`
- `index = find(M==1);`

- ▶ `ind2sub`

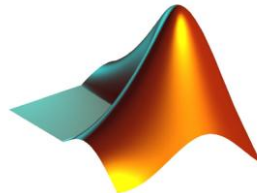
- `[r c] = ind2sub(size(M), index);`

- ▶ `sub2ind`

- `newIndex = sub2ind(size(M), r, c);`

[4x2] M

2	4
3	6
5	1
8	12



# Matrix indexing

- ▶ If we want to define the position of element 1 within the matrix M, we can do it with a single index or with the indexes of row and column

- `M = [2 4; 3 6; 5 1; 8 12];`
- `index = find(M==1);`

7

## ▶ `ind2sub`

- `[r c] = ind2sub(size(M), index);`

3 2

## ▶ `sub2ind`

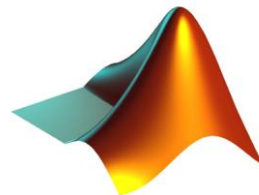
- `newIndex = sub2ind(size(M), r, c);`

7

[4x2] M

2	4
3	6
5	1
8	12

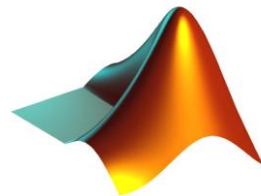
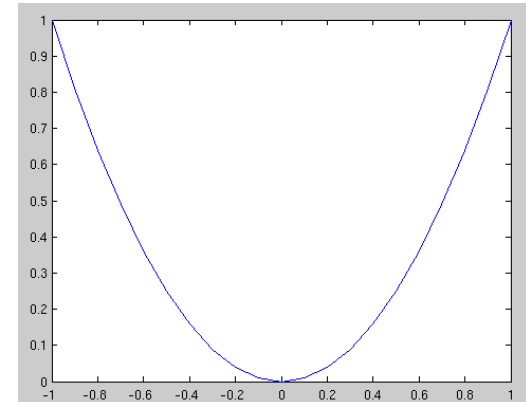
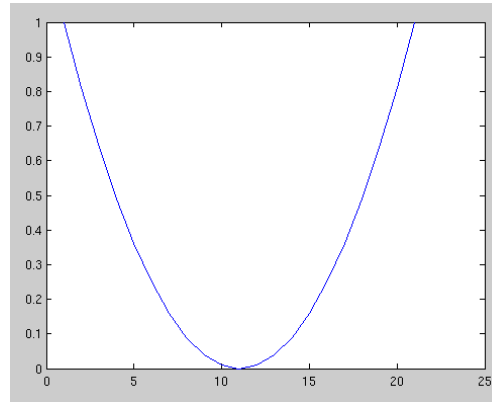
It's necessary to provide  
the size of the matrix!



# Basic Plotting

## ► `plot()`

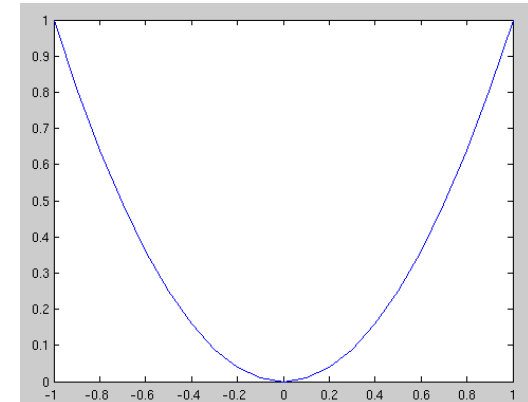
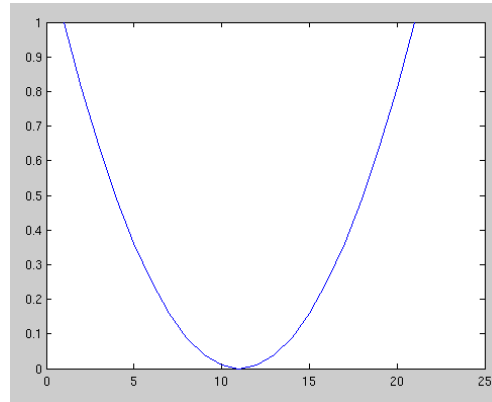
- `x = [-1:0.1:1];`
- `y = x.^2;`
- `plot(y);`
- `plot(x,y);`



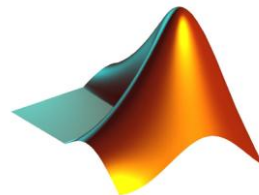
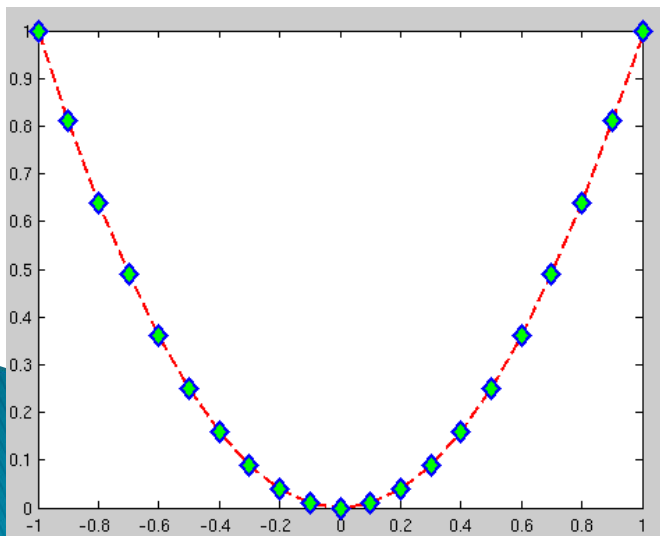
# Basic Plotting

## ► `plot()`

- `x = [-1:0.1:1];`
- `y = x.^2;`
- `plot(y);`
- `plot(x,y);`



- `plot(x,y,'--rd','LineWidth',2,...  
'MarkerEdgeColor','b',...  
'MarkerFaceColor','g',...  
'MarkerSize',10);`

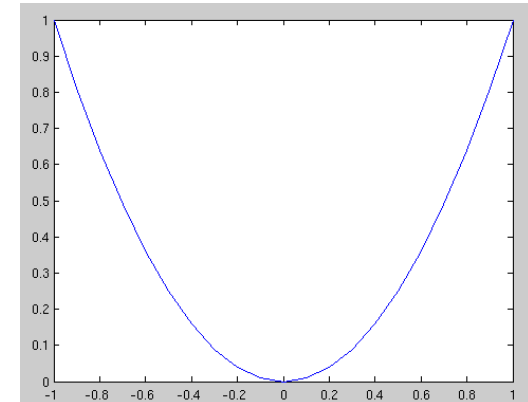
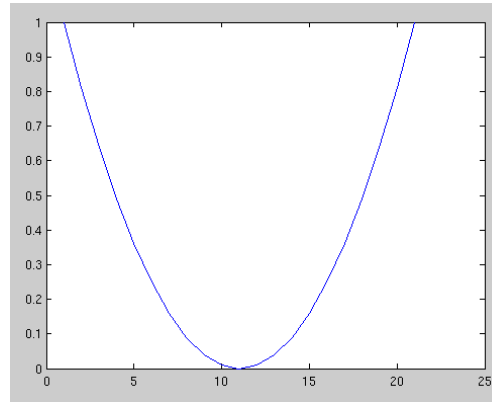




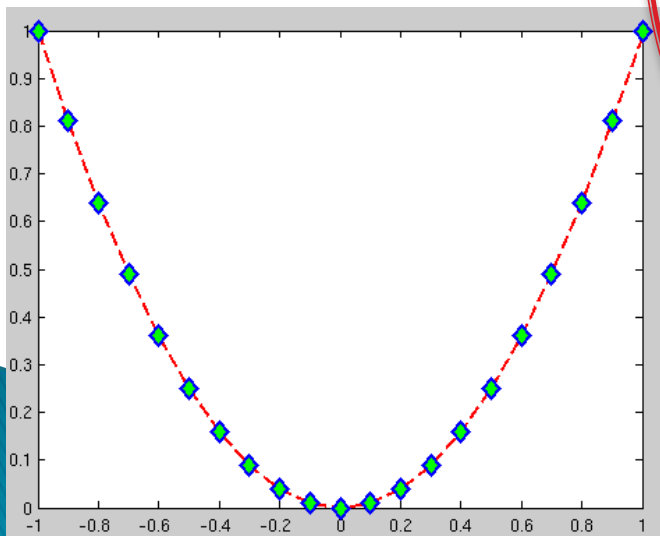
# Basic Plotting

## ► `plot()`

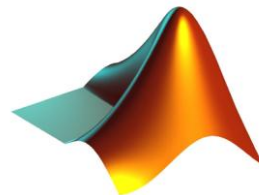
- `x = [-1:0.1:1];`
- `y = x.^2;`
- `plot(y);`
- `plot(x,y);`



- `plot(x,y,'--rd','LineWidth',2,...`  
`'MarkerEdgeColor','b',...`  
`'MarkerFaceColor','g',...`  
`'MarkerSize',10);`



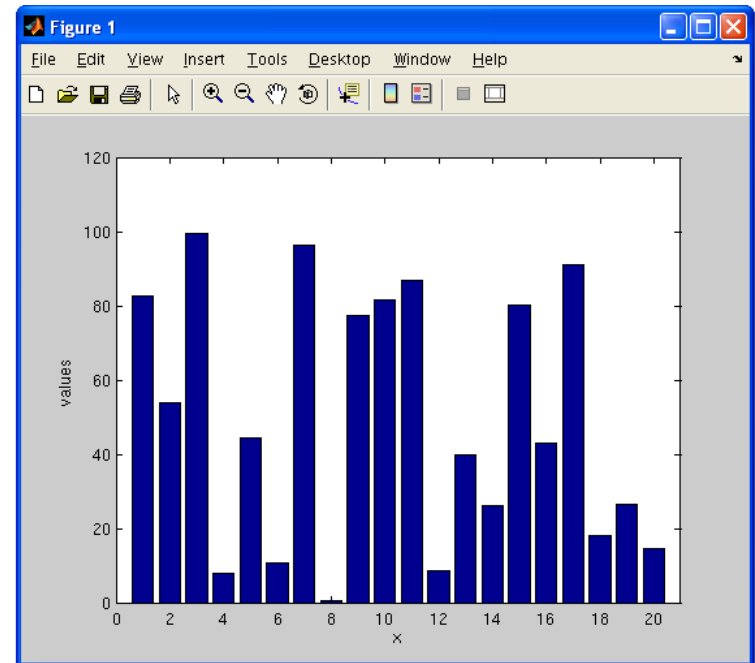
- Line style --
- Line color 'red'
- Marker Type 'diamond'



# Basic Plotting

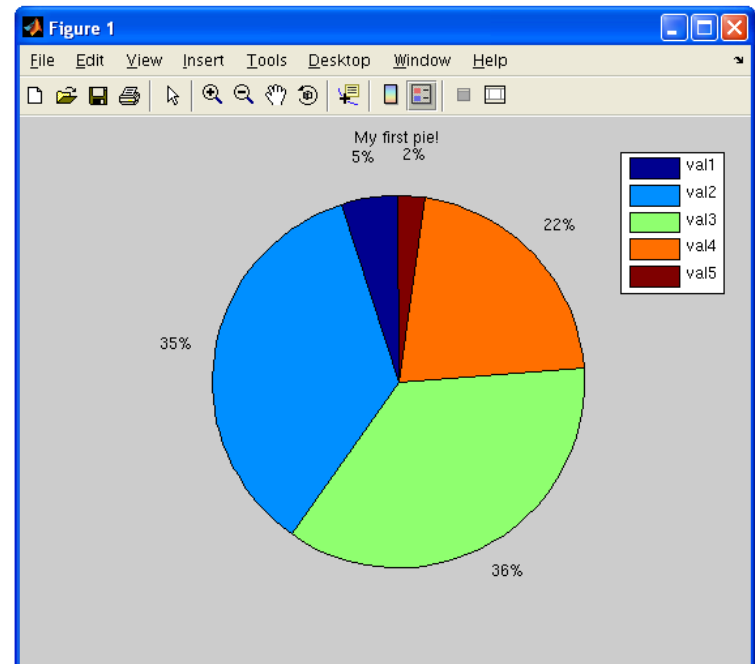
## ► bar()

- `x = 100*rand(1,20);`
- `bar(x);`
- `xlabel('x');`
- `ylabel('values');`
- `axis([0 21 0 120]);`  
                    x range y range
- `xlim([0 21]); ylim([0 120]);`



## ► pie()

- `x = 100*rand(1,5);`
- `pie(x);`
- `title('My first pie!');`
- `legend('val1','val2',...  
      'val3','val4','val5');`



# Basic Plotting

## ► figure

- To open a new Figure and avoid overwriting plots

- `x = [-pi:0.1:pi];`

- `y = sin(x);`

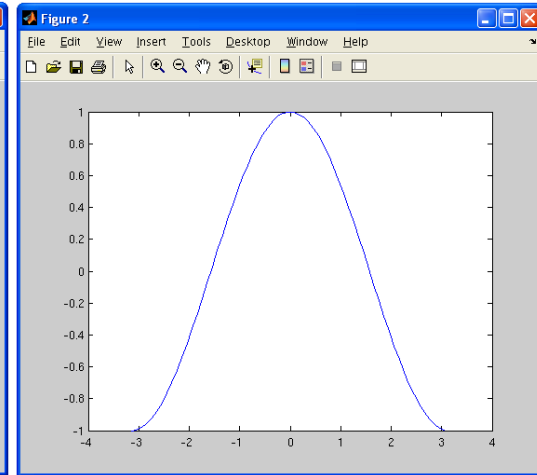
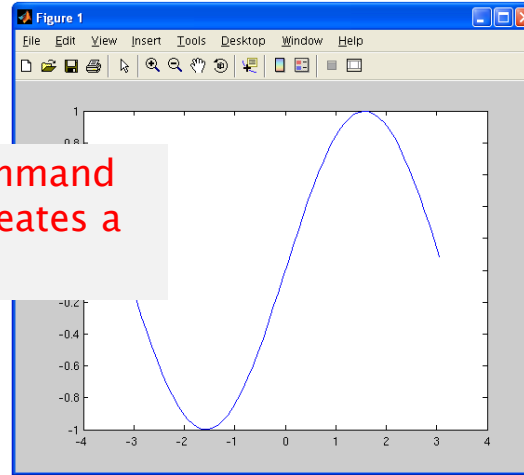
- `z = cos(x);`

- `plot(x,y);`

- `figure`

- `plot(x,z);`

The first plot command automatically creates a new Figure!



## ► Close figures

- `close 1`

- `close all`

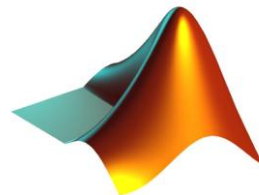
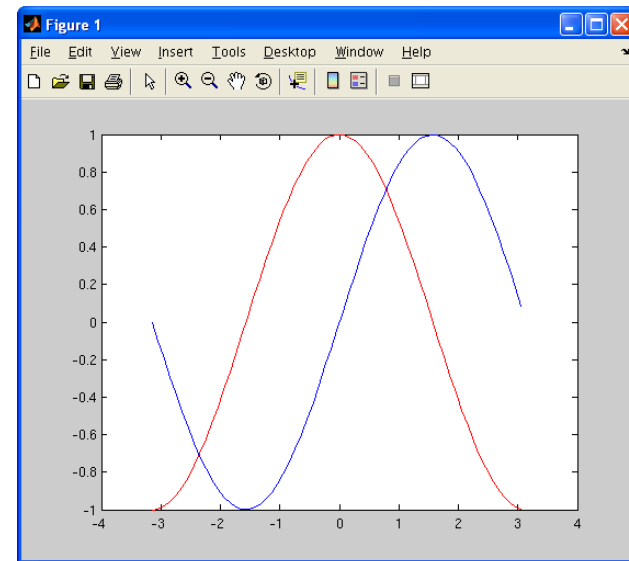
## ► Multiple plots in same Graph

- `plot(x,y);`

- `hold on`

- `plot(x,z,'r');`

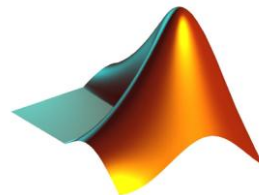
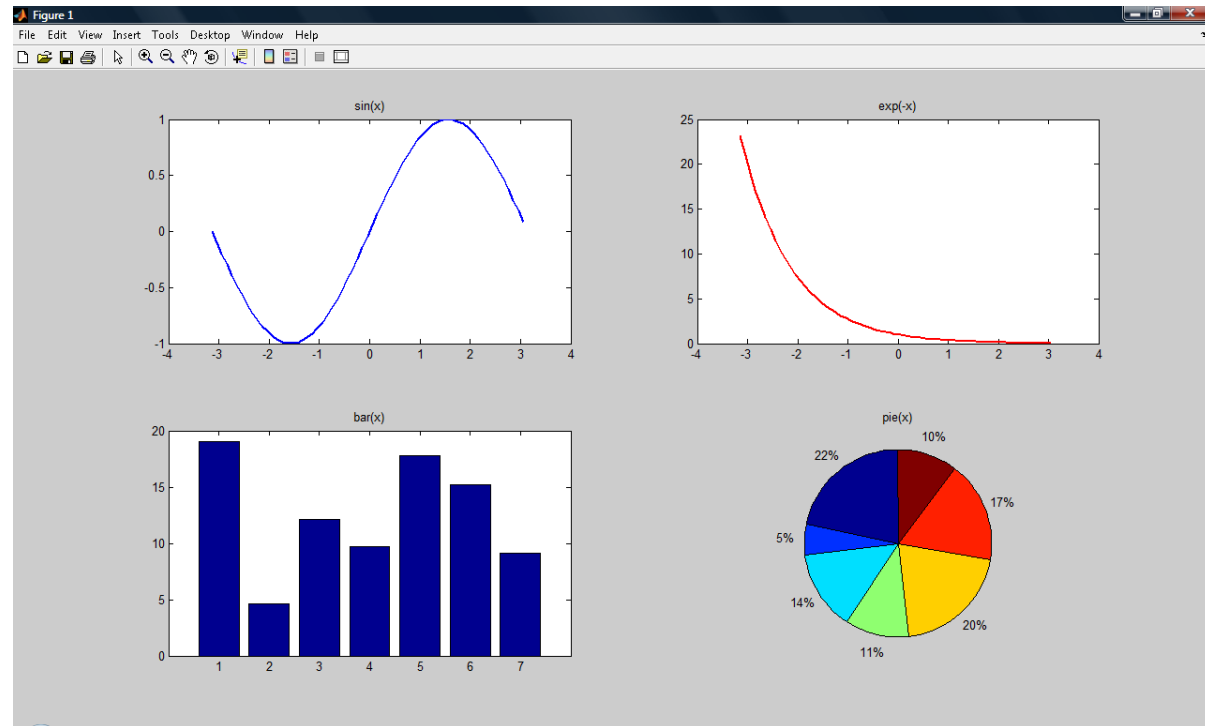
- `hold off`



# Basic Plotting

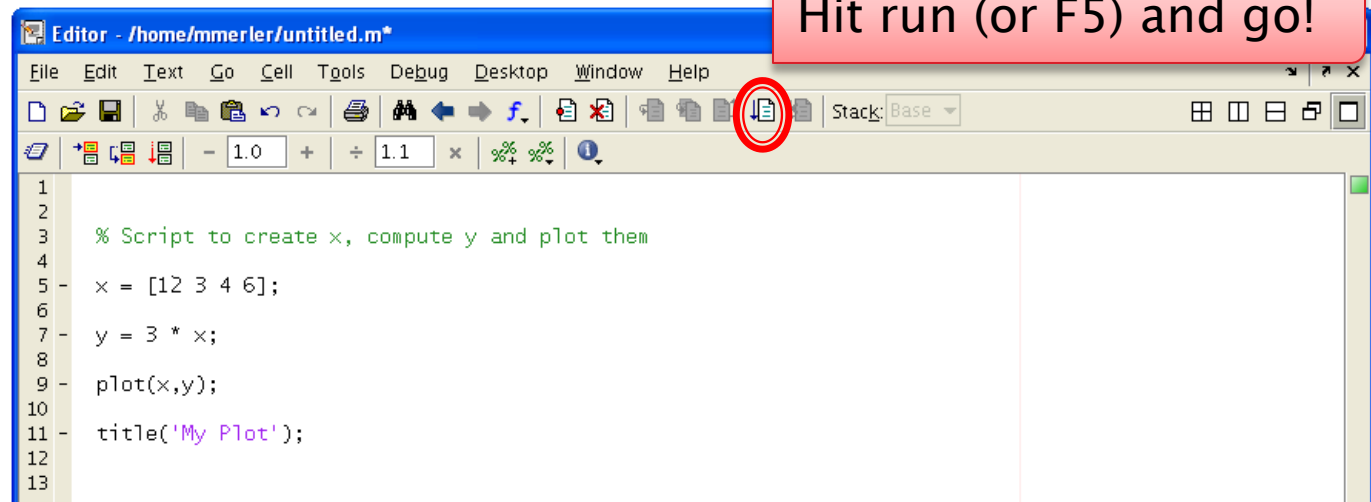
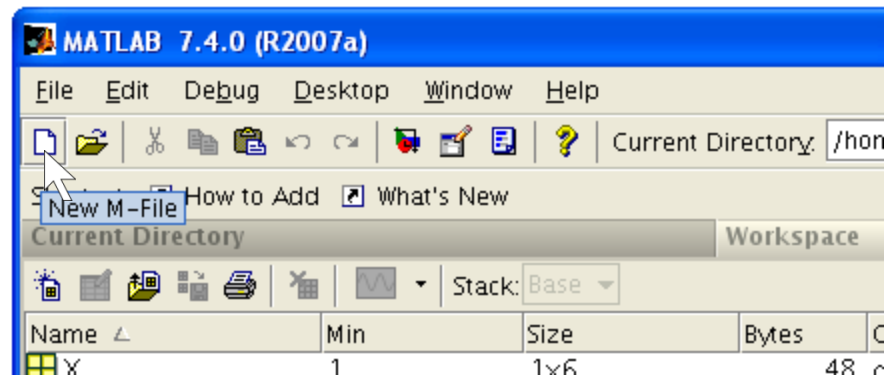
## ► Multiple plots in same Figure

- `figure(1)`
- `subplot(2,2,1)`
- `plot(x,y);`
- `title('sin(x)');`
- `subplot(2,2,2)`
- `plot(x,z,'r');`
- `title('exp(-x)');`
- `subplot(2,2,3)`
- `bar(x);`
- `title('bar(x)');`
- `subplot(2,2,4)`
- `pie(x);`
- `title('pie(x)');`

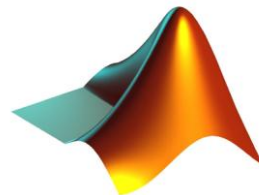


# Scripts

- ▶ Like a notebook, but for code!
- ▶ M-files are MATLAB specific script files, they are called *namefile.m*

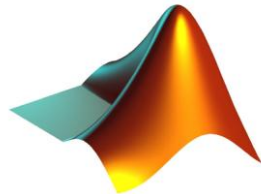


- ▶ You can open scripts from command window too, just type `open scriptname`



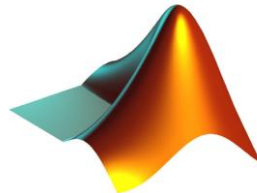
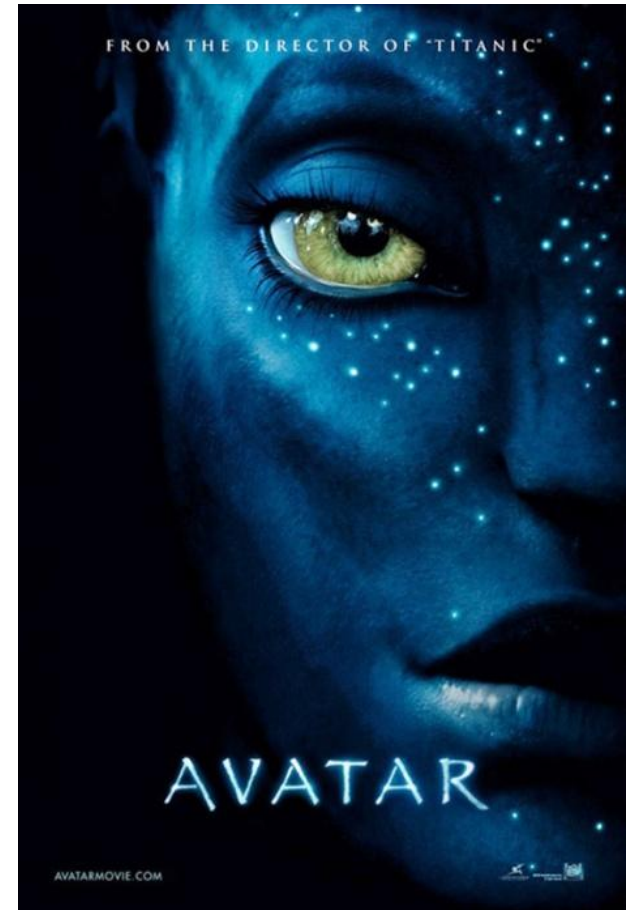
# Comments

- ▶ Adding comments to your code is a very healthy habit
- ▶ Think about other people who have to read and understand 3000 lines of your code!
- ▶ MATLAB comments, the % operator
  - `x = [1 2 3 4];`
  - `% this is a comment`
  - `bar(x);`
  - `title('bar(x)');`
- ▶ When you type `help namefunction` in the command window, what you get is the comments on top of the *namefunction.m* script



# Homeworks policy

- ▶ Due at beginning of class, no exceptions
- ▶ Put your code (.m files) and additional files in a single folder, name it *youruni\_hw\_X* and zip it
- ▶ Upload the zipped folder to CourseWorks
- ▶ Bring a printout of your code to class
- ▶ Good luck and have fun !!!



# Conclusion

- ▶ MATLAB is also a philosopher!
- ▶ Try typing `why` in the command window...  
you'll get the answers!!!

