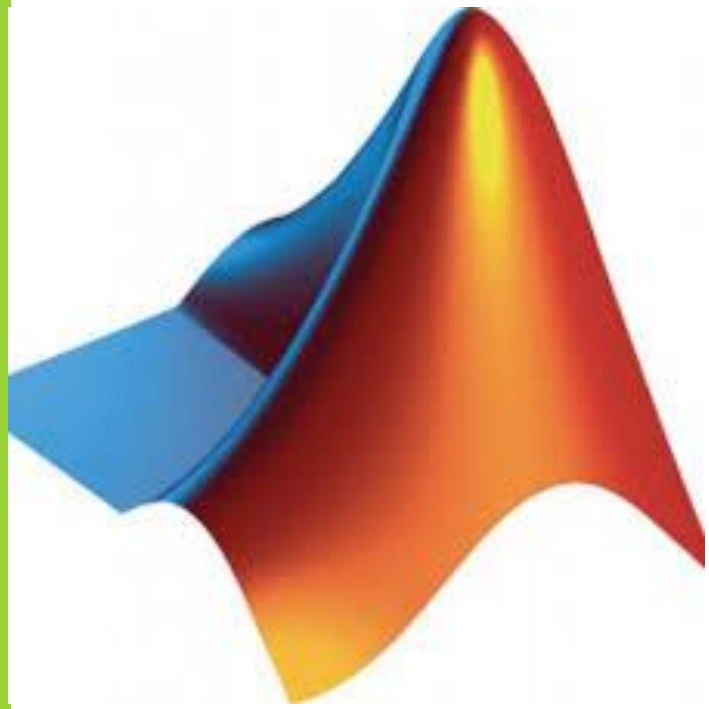
SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Queen's University Belfast

Introduction to Matlab

PhD Training Courses

Basics



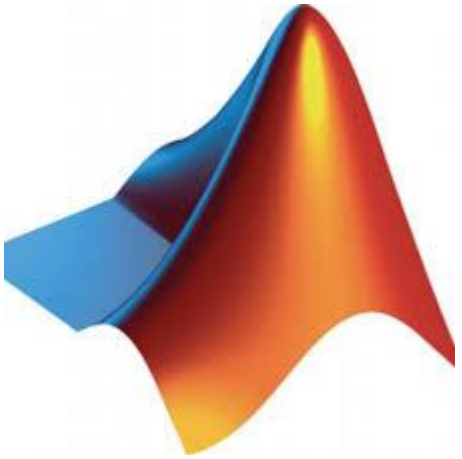
Jesús Martínez del Rincón

j.martinez-del-rincon@qub.ac.uk

SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
Queen's University Belfast

Matlab Tutorial

- Aim:
  - Learn the fundamentals of Matlab
- Assumptions:
  - Proficiency in any other programming language



Course details

- 2 Days Course
  - Day 1: Fundamentals of Matlab
  - Day 2: Machine learning using Matlab
- Delivery (each day):
  - 3h interactive lecture
  - 2h practical
- Feel free to ask questions, during and after lecture!
  - Open and play Matlab while I talk

Index

- What is Matlab and when to use
- Development Environment
- Programming basics
- Arrays
- Complex datatypes
- Visualization
- Input/output data
- Others

What is Matlab?

- Interpreted language for numerical computation.
  - generated scripts, do not need to be compiled.
  - memory to the variables is also allocated dynamically and can change easily.
  - As a disadvantage, Matlab can be slow on execution
    - poor programming practices can make it unacceptably slow and high memory consuming.
  - Operating systems:
    - Microsoft Windows, Linux, and OS X
- It allows:
  - performing numerical calculations
  - visualize the results

without the need for complicated and time consuming programming.

When shall I use Matlab?

- Matlab allows its users to:
  - accurately solve problems,
  - produce graphics easily
  - and produce code efficiently (not efficient code) → **fast prototyping**
- Large amount of toolboxes and programs available
  - perfect tool for fast and simple prototyping before moving on to the final implementation in Java or C.

- Wide range of applications:
  - Signal processing and communications, Image and video processing, Control systems, Test and measurement, Computational finance, Computational biology, etc..

SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Queen's University Belfast

IDE

Editor

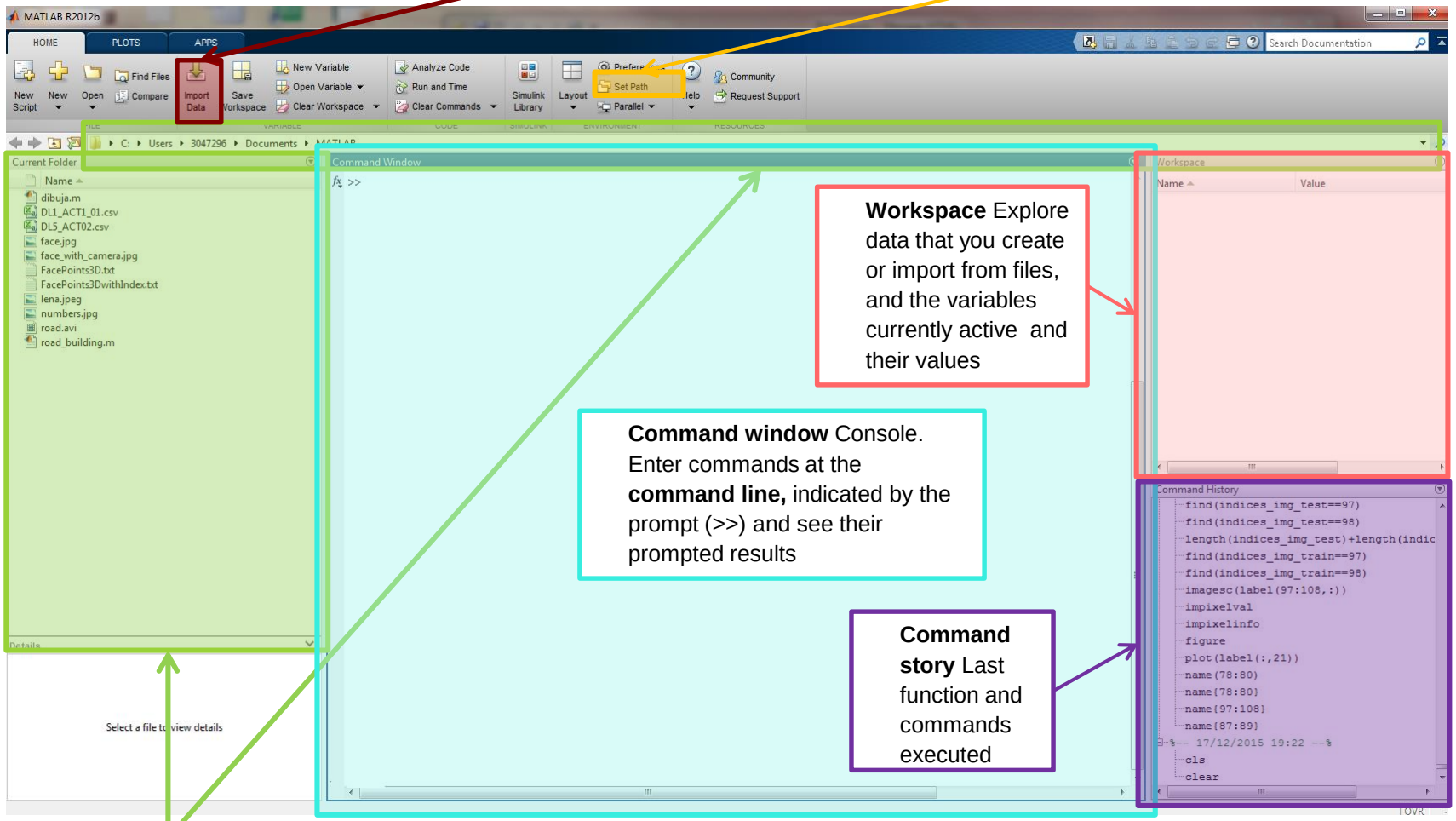
Other windows

INTEGRATED DEVELOPMENT ENVIRONMENT

Desktop

Import data: load data in matlab from other program/sources (excel, csv, txt)

Set path: add new folders and toolboxes containing files that will be accessible



Workspace Explore data that you create or import from files, and the variables currently active and their values

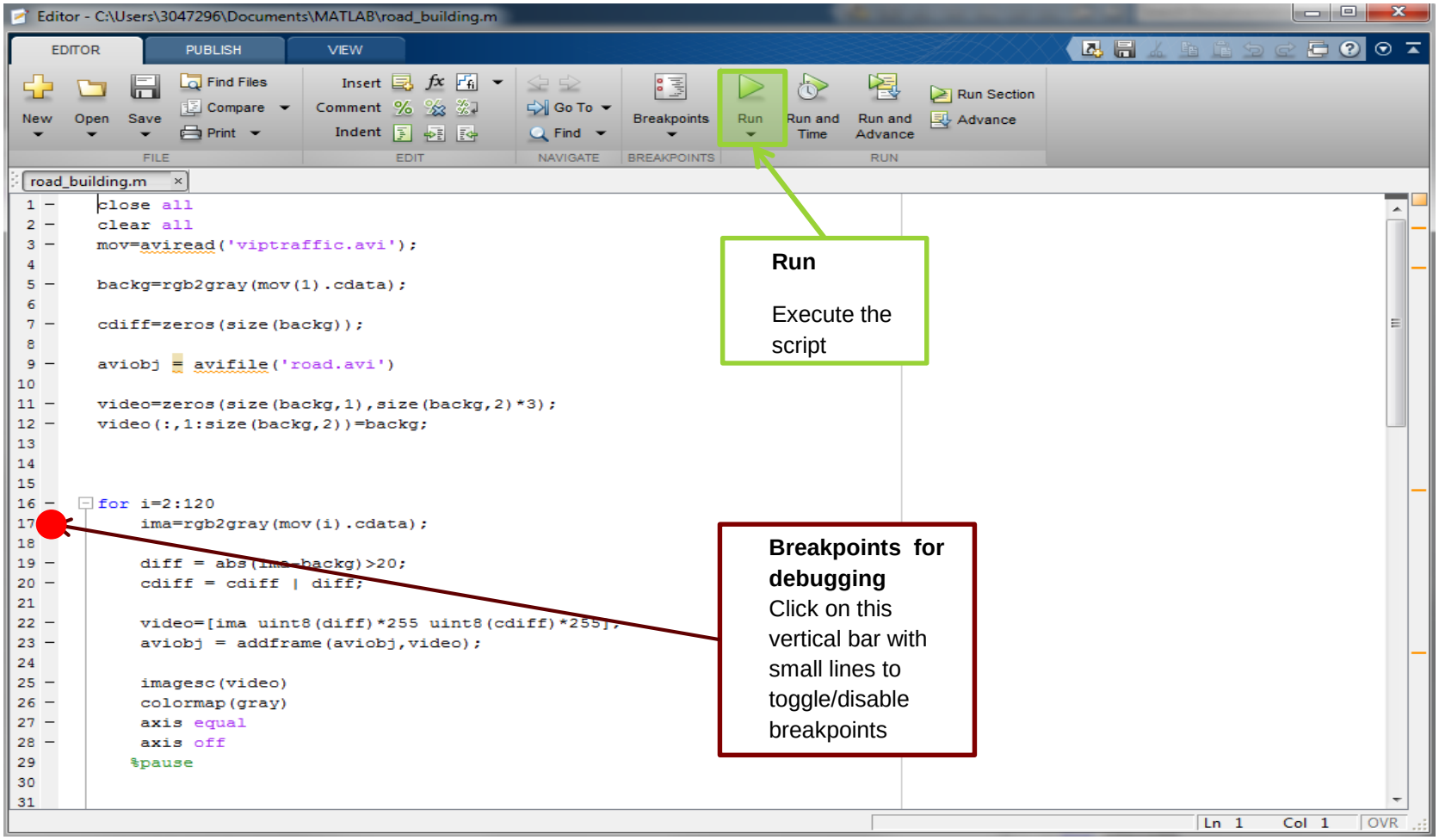
Command window Console. Enter commands at the **command line,** indicated by the prompt (>>) and see their prompted results

Command story Last function and commands executed

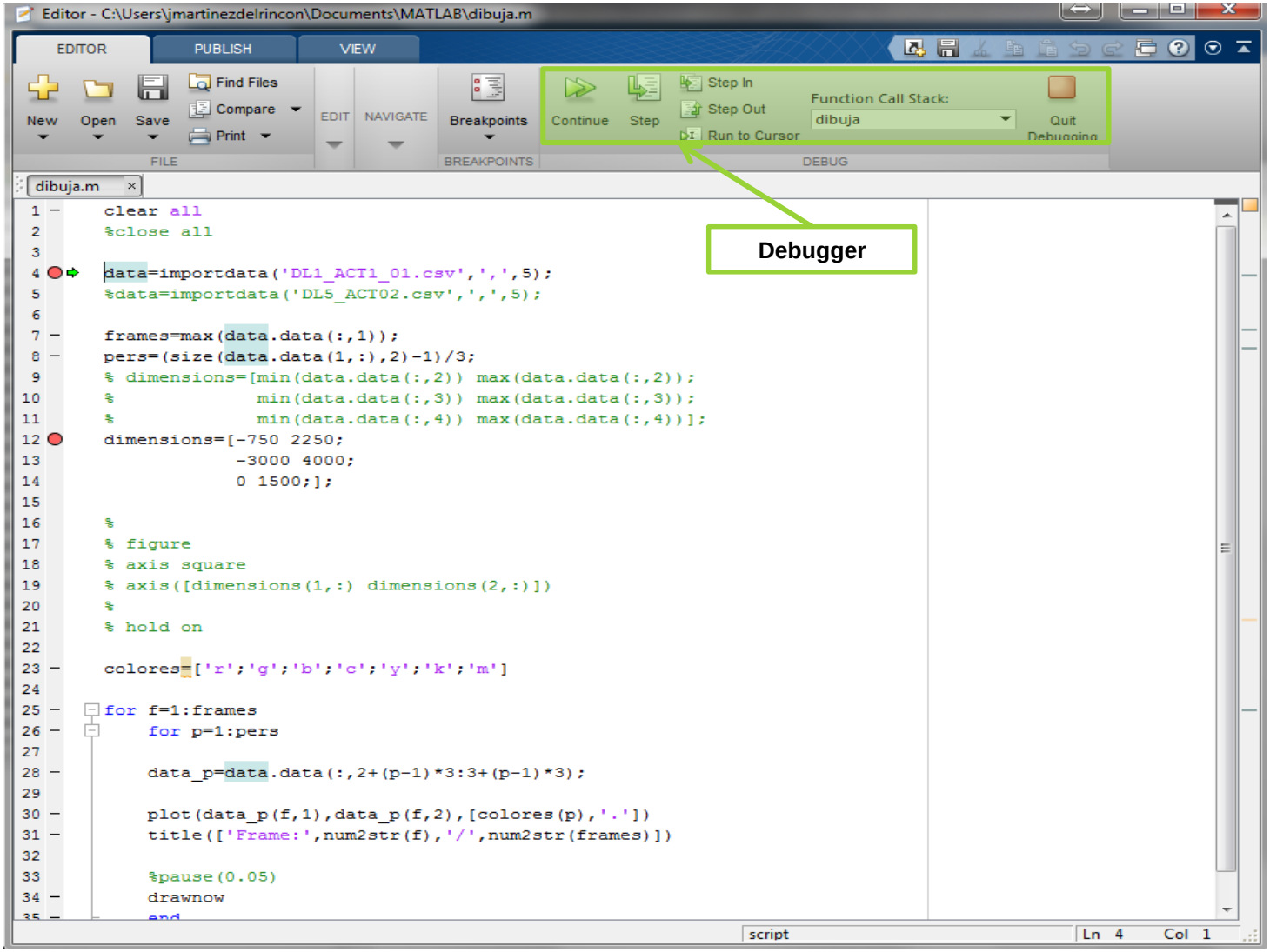
Current Folder Access your files

- Desktop layout can be configure to your liking
  - It can be reset it by clicking on *Layout → Default*

Editor Window



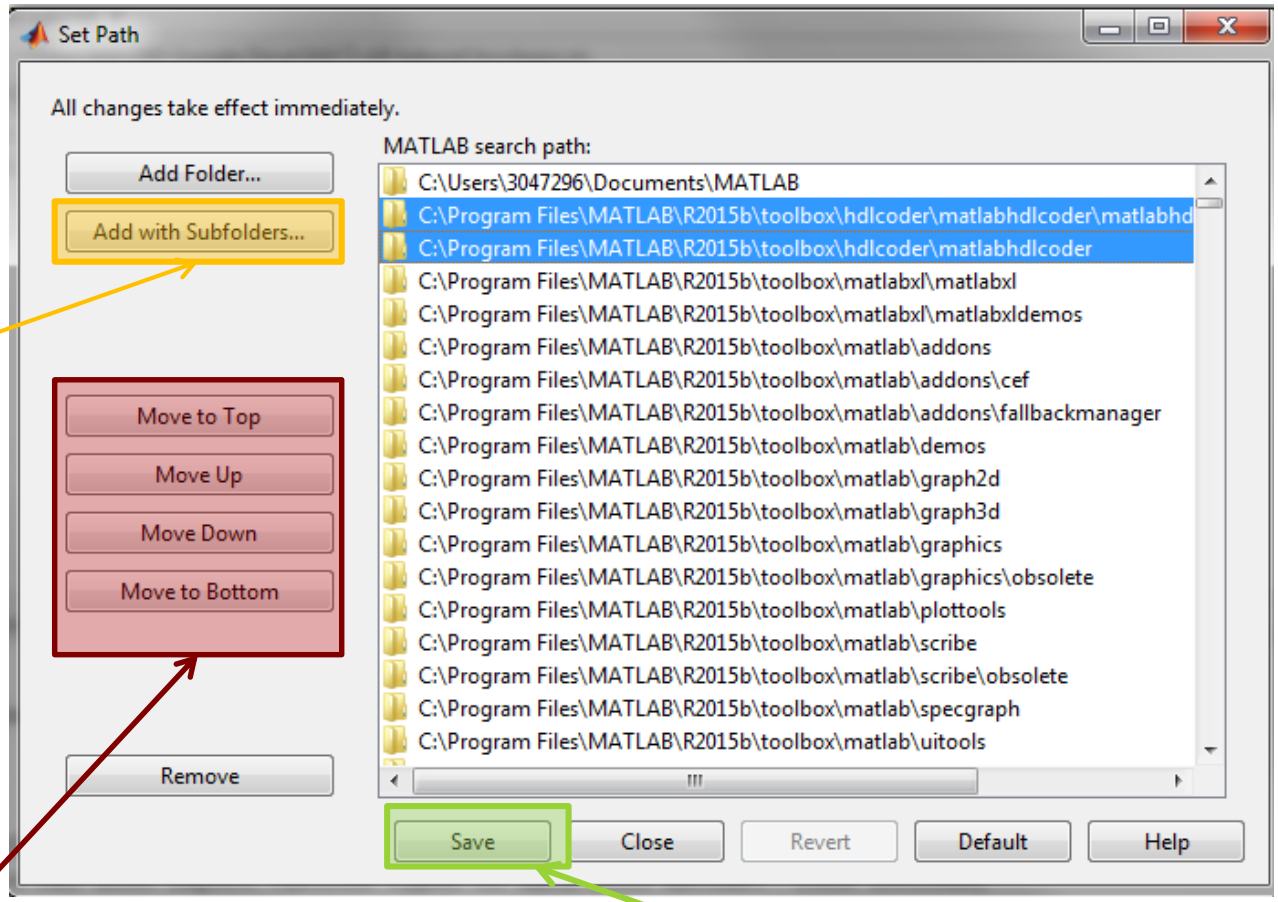
Editor/debugger Window



Set Path

For the functions/scripts in Matlab to be recognised when called, the containing folder should be in the path

Always use it when you download a new toolbox

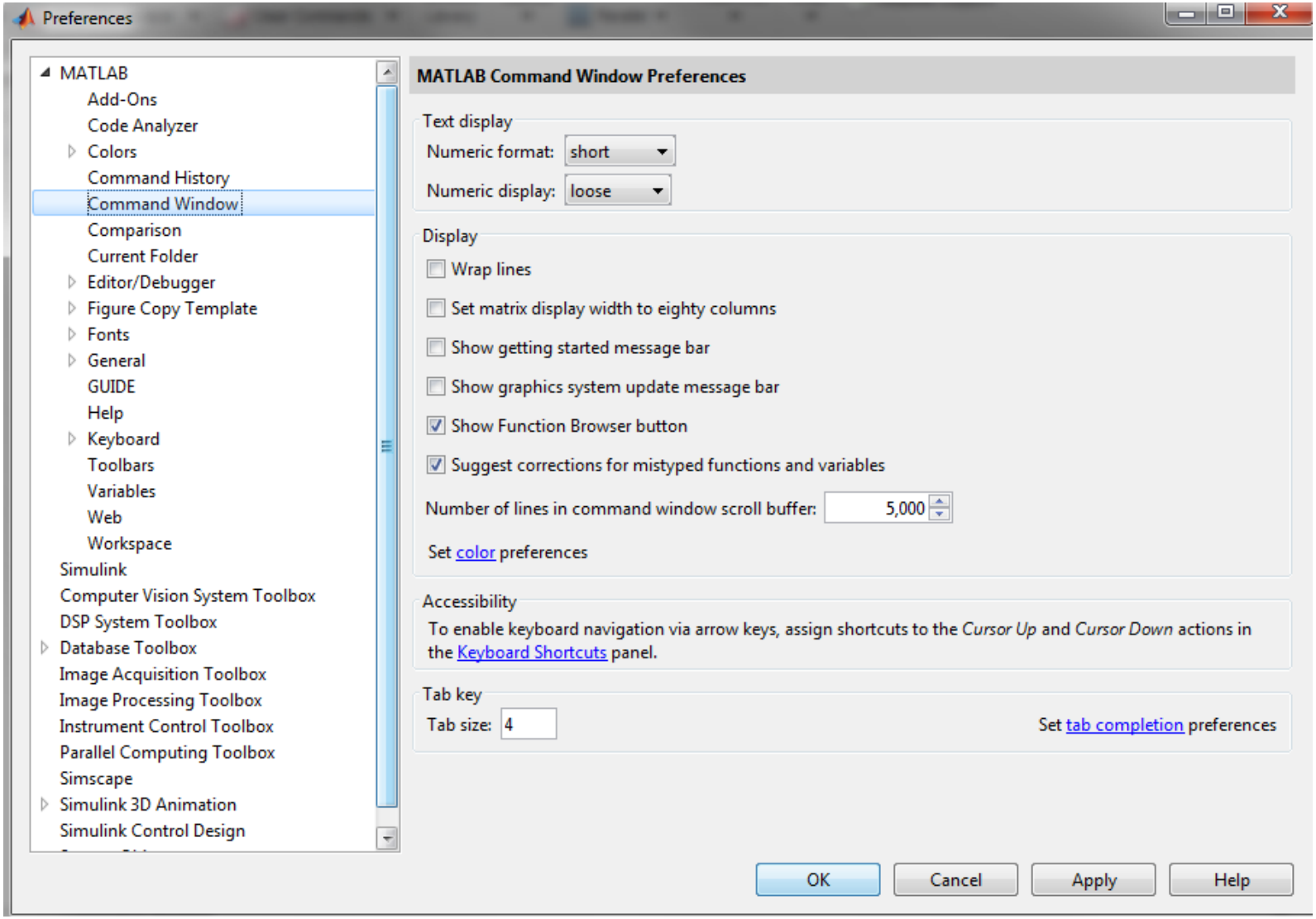


Careful with the order: if two functions/files have the same name, the one that will be used is determined by this ordering

Very common when downloading several toolboxes on the same topic

Remember to click before closing the window

Preferences



SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Queen's University Belfast

Scripts

Functions

Variables

PROGRAMMING BASICS

Creating a Script

- Command line
  - As a resource, for a simple checking or during debugging
  - Uncomfortable and does not keep a proper record
- Instead, MATLAB editor allows you to create new files *.m
  - scripts functions.
- A Script is a collection of commands
  - Good practice: first 2 statements of every script should be:

    `clear all` → delete all the variables in your workspace

    `close all` → close any open window/interface

    - To ensure every execution start from scratch and avoid errors or conflicts with variables previously initialised

Creating a Function

- Encapsulating code into functions or methods is always good idea
  - More readable and reusable code.
- To create a function we will be using the keyword *function:*

```
function [ output_arg1, output_arg2, … ] = function_name( input_arg1, input_arg2, …
```

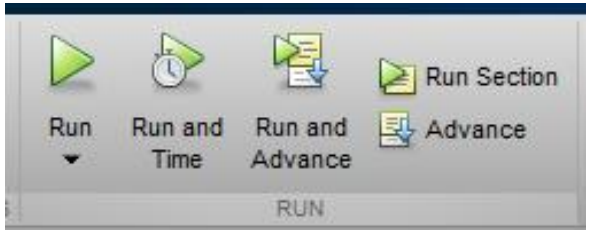
  - Write each function in a new file
  - Name of the file should match the name of the function

- To open an existing function:

```
>> open function_name
```

Running Functions and Scripts

- To run a script/function:
  - Click on Run in the editor (script)
  - Write the name of your script/function file in the command line



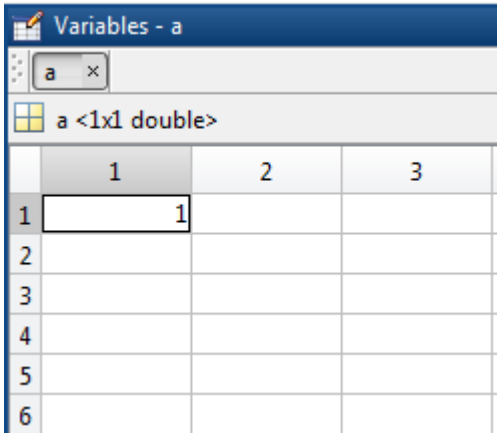
- By default, all the new files will be stored in the *current folder.*
  - To run a script, the file must be in the current folder or in the PATH
  - You can check and/or modify this folder by clicking on *Set Path*.

```
>> script
>> function1(arg1, arg2)
>> out = function1(arg1)
```

Creating a variable

```
a = 1

b = 2

c = a + b

d = cos(a)
```

- MATLAB adds a variable a to the workspace and displays the result in the Command Window.
  - If you do not want to display the result in the command window, add ; `a = 1;`
- You do not need to declare the type
  - Matlab will detect it for you
  - By default all variables are a **matrix of double**



Creating a variable

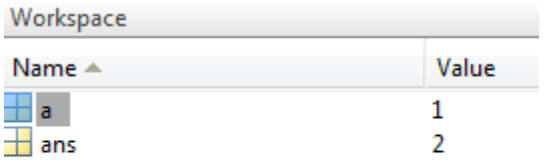
```
myText = 'Hello, world';

letter = 'a';
```

- A *character string* is a sequence of any number of characters enclosed in single quotes. Matlab also allow you to create string variables

Variable `ans` and Values

```
a + 1
```

```
>> a+1

ans =

     2
```



- When you do not specify an output variable, MATLAB uses the variable `ans` (short for answer) to store the results of your calculation.

- Some interesting values that you can assign/obtain

```
b = NaN;
```
→ Not a Number (e.g. /0)

```
c = Inf;
```
→ Infinity (maximum possible value)

```
d = [];
```
→ Empty variable

Complex Numbers

- To represent the imaginary part of complex numbers, use either i or j.

```
c = 3+4i;        ➔   3.0000 + 4.0000i
d = 4+3j;        ➔   4.0000 + 3.0000i
e = -i;          ➔   0.0000 - 1.0000i
f = 10j;         ➔   0.0000 +10.0000i
g = sqrt(-1);    ➔   0.0000 +1.0000i
```

Square root function

SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Queen's University Belfast

1D arrays

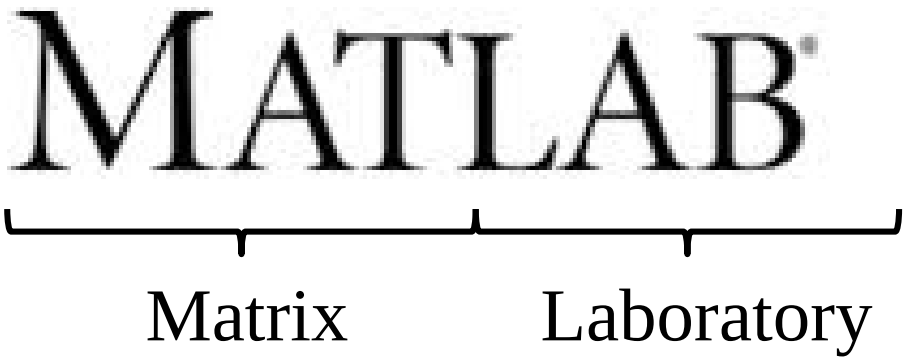
2D arrays: Matrices

Working with arrays

ARRAYS

Arrays and Matrices

- Matlab was designed to manage and operate with arrays and matrices.
  - Dynamic size change
  - Simple indexing
  - Speed-up matrix operations.



- **All MATLAB variables are multidimensional arrays**, no matter what type of data.
  - `a=1;` `a` is considered as a 1x1 matrix.

Arrays and Matrices

- To create an array or a matrix, we use []

```
v1 = [1 2 3 4]
```
→ row vector

```
v1 =
     1     2     3     4
```

```
v2=[1; 2; 3; 4]
```
→ column vector

```
v2 =
     1
     2
     3
     4
```

```
m = [1 2 3; 4 5 6; 7 8 9]
```
→ matrix

```
m =
     1     2     3
     4     5     6
     7     8     9
```

  - a space (or a comma ,) separates elements into columns, a semicolon (;) separates elements into rows

- Another way to create a matrix is to use a function provided by Matlab:
  - `ones()`, `zeros()`, or `rand()`, `randn()`
  - syntax: `zeros(rows, columns)`

Matrix Indexing

- To access to a single element of our matrix/vector:
  - First element is index 1 (not 0 like Java or C)
  - `name(row,column)`

    or
  - `name(element number)` → **Even for matrices!!!!**

    Maybe convenient for loops and searching elements
- To refer to multiple elements of an array:
  - colon operator :
    - To specify a range of the form `start:end` ← **Reserved words:**

      They give you access to the first or last element directly

      ```
      name(r1:r2,c1:c2)
      ```
    - By itself, specifies all of the elements in that dimension.

      ```
      name(r1,:)
      ```
- To refer to all the elements of an array:
  - `name(:)` → Return all the elements of the matrix as a single array

Colon Operator :

- You can also create arrays in a compact form by using the colon operator

```
v1 = [1:4]
```

```
v1 =

     1     2     3     4
```

- By default, it increases the initial value in steps of **value 1**, until the end value
  - Useful for indexes

- But we can control the steps manually
  - Even negative values (if `endValue<iniValue`)

```
array=[iniValue:step:endValue]
```

```
array=[iniValue:(iniValue-endValue)/(numbSteps-1):endValue]
```

Matrix Dynamic Size

- Check the size of our matrix/vector by using:

```
nelements = length(name)
[rows columns]=size(name)
        or
nelements = size(name,dim)
```

Only for arrays

Both arrays and matrices

- Dynamic size:
  - Matlab allocate them memory dynamically
  - Sizes can change easily during execution without t… destroying and creating the variable

```
M=ones(3,3)     M(3,4)=2
```

    - Non-specify elements are filled with zeros
  - Easy to concatenate several arrays or matrices to make larger ones.

```
M=[M; rand(1,4)]
```

    - The pair of square brackets [ ] is the concatenation operator.
    - And to add/delete rows/columns `M(2,:)=[]`

It may hide bugs difficult to find

Matrix Operators

- Conventional operators and functions to process all of the values in a matrix

  `+` sum
  `-` subtraction
  `*` multiplication
  `'` transpose (swap rows and columns)
  `^` potentiation
  `/` division
  `inv()` inverse matrix
  `det()` determinant

- **WARNING!!** The multiplication, division, and power operators will perform proper matrix operations

$$\mathbf{A} = \begin{pmatrix} a & b & c \\ x & y & z \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} \alpha & \rho \\ \beta & \sigma \\ \gamma & \tau \end{pmatrix}, \quad \Rightarrow \quad \mathbf{AB} = \begin{pmatrix} a & b & c \\ x & y & z \end{pmatrix} \begin{pmatrix} \alpha & \rho \\ \beta & \sigma \\ \gamma & \tau \end{pmatrix} = \begin{pmatrix} a\alpha + b\beta + c\gamma & a\rho + b\sigma + c\tau \\ x\alpha + y\beta + z\gamma & x\rho + y\sigma + z\tau \end{pmatrix},$$

- To be applied element-wise rather than matrix-wise:
  - use a dot before the operator

    `.*` `./` `.^`

Logical and Relational Operators

`A < B`	smaller than	`A & B`	AND
`A > B`	greater than	`A | B`	OR
`A <= B`	smaller or equal than	`~A`	NOT
`A >= B`	smaller or equal than	`xor(A,B)`	exclusive-OR
`A == B`	equal to	`A && B`	AND with short-circuiting behaviour
`A ~= B`	different from	`A || B`	OR with short-circuiting behaviour

- Matrix operators
  - Element-by-element comparisons between two arrays.
  - They return a logical array of the same size.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} == \begin{bmatrix} 5 & 2 \\ 4 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

- `true, false` ➔ reserved words for logical `1` and `0`
- **WARNING!!** Since in Matlab you do not need to define types explicitly, anything different than 0 will be considered as logical 1

```
-1 & 2   ➔  1          0 & 400 ➔  0
```

Other useful functions

- `find()`
  - Returns the linear indices corresponding to the nonzero entries of the array/matrix.

```
[rows columns]=find(name);          indexes=find(name);
```

  - `name` may be a logical expression.

```
[rows columns]=find(name==10);
```

- `sum(), cumsum()`
  - Returns the sum/sumulative sum of the elements of the array/matrix over the rows or the columns

```
result = sum(name,dimension)        result = sum(name)
```

If `A = [0 1 2; 3 4 5]` then `sum(X,1)` is `[3 5 7]` and `sum(X,2)` is `[3; 12]`

  - For applying it to the whole matrix.

```
sum(sum(name))        sum(name(:));
```

Other useful functions

- `mean()`
  - Returns the mean value or average of the elements of the array/matrix over the rows or the columns

```
result = mean(name,dimension)        result = mean(name)
```

- `max(), min()`
  - Returns the largest/smaller element from each row/column and their index/position in the original matrix

```
[result index]= min(name,[],dimension)
```

- `fliplr()`
  - Flip array/matrix in left/right direction.

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

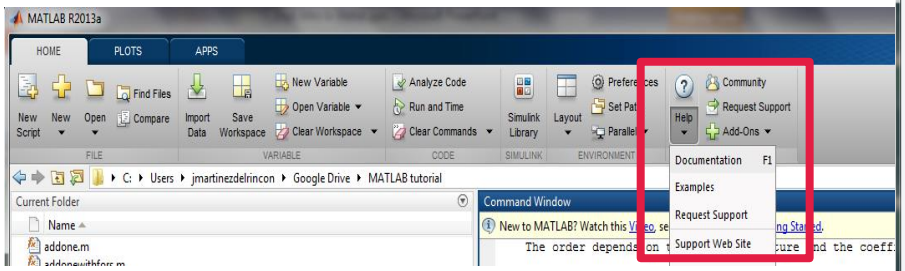
- `sort()`
  - Sorts each column/row in ascending/descending order.

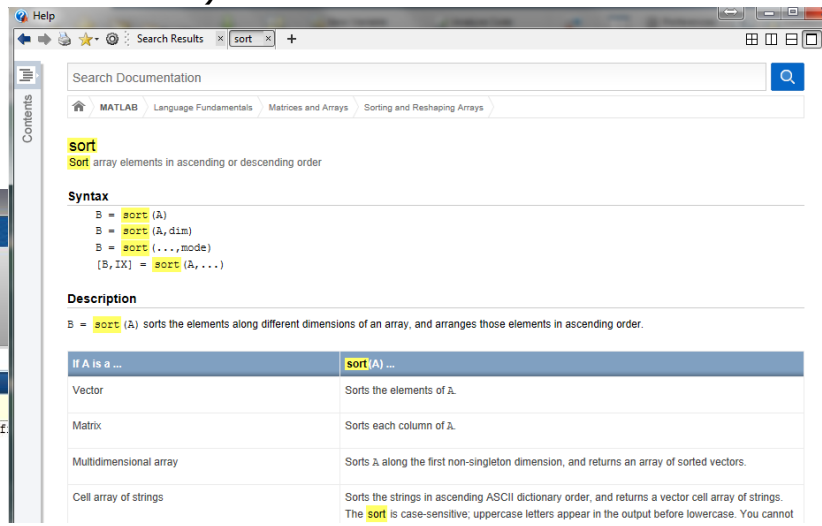
```
[sorted, index] = sort(name,dimension,mode)
```

- `floor(),ceil(), round()`: Round down/up/to nearest decimal or integer

Help

- How to use some of the functions?
  - Functionality? Input, output parameters?
  - Most functions are overloaded (default values)
  - Matlab, external toolboxes, etc...
- Documentation





- Or you can go old school:
  - `help function_name`
    - It will display in the command window a brief explanation of the function and the parameters, as well as some examples of use for that specific function_name
  - `lookfor concept`
    - It will display in the command window a list of available Matlab functions related with the concept of interest

Loops and Conditions

- Fundamental in any programming language

if-else-statement

```
if a==1


else


end
```

while statement

```
while a==1


end
```

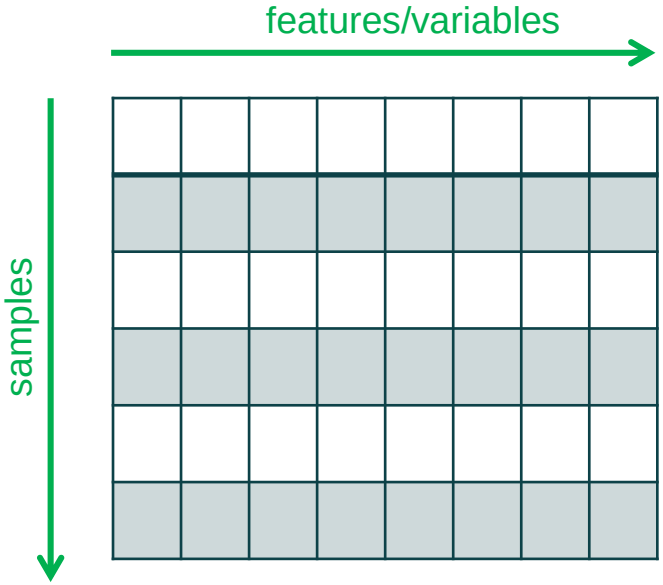
for statement

```
for index=1:10


end
```

- But do not forget, Matlab has been optimised for matrix operations
  - Minimise the use of loops as much as possible
  - Think mathematically

Exercise – part 1

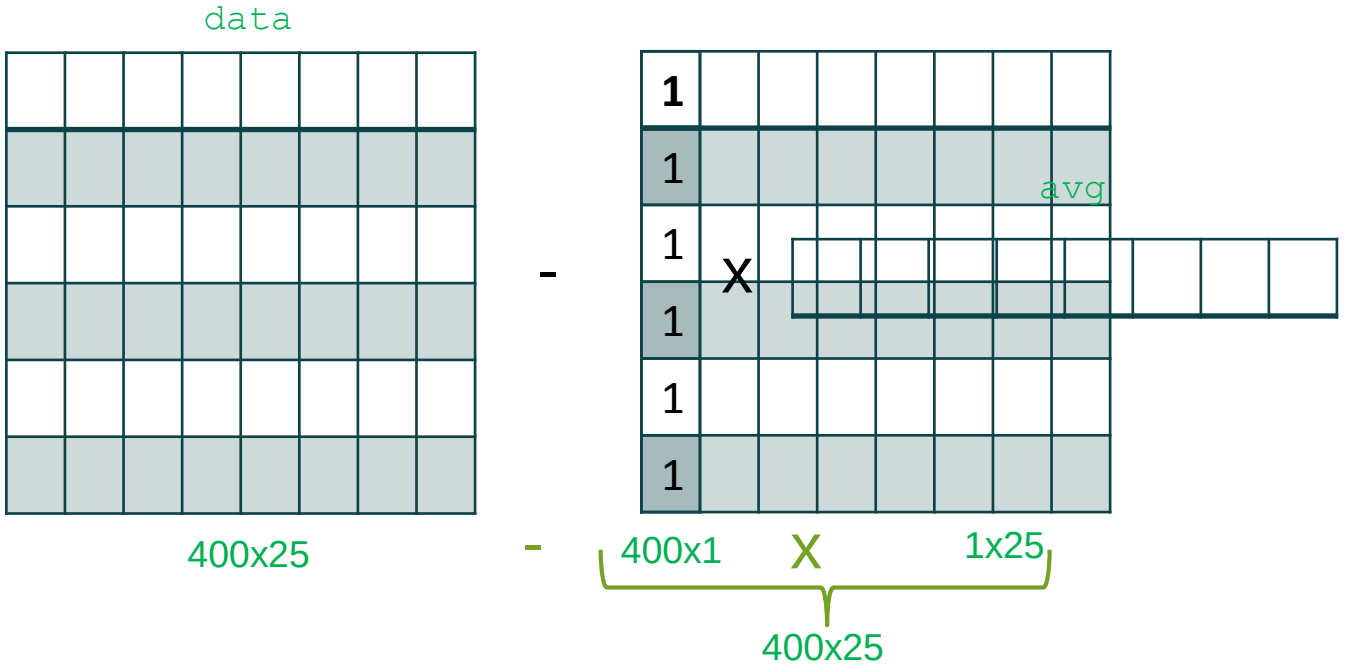
- Create a matrix `data` of size 400x25 with random numbers



- Calculate its average over the rows (over the samples) and store it in a vector `avg`
- Subtract the average to every sample in data using loops

Exercise – part 2

- Let's do the same using matrix operations



- Implement the average subtraction without using loops
  - Using the matrix operations an a column array fulll of ones

Measuring time

- Tic-toc

```
tic
 instructions
toc
```

Or saved to a variable (`time=toc;`)

```
time =
    4.2423
```

In the command line:

```
Elapsed time is 0.003975 seconds
```

- Profile

```
profile on
 instructions
profile viewer
```

Profiler

File Edit Debug Window Help

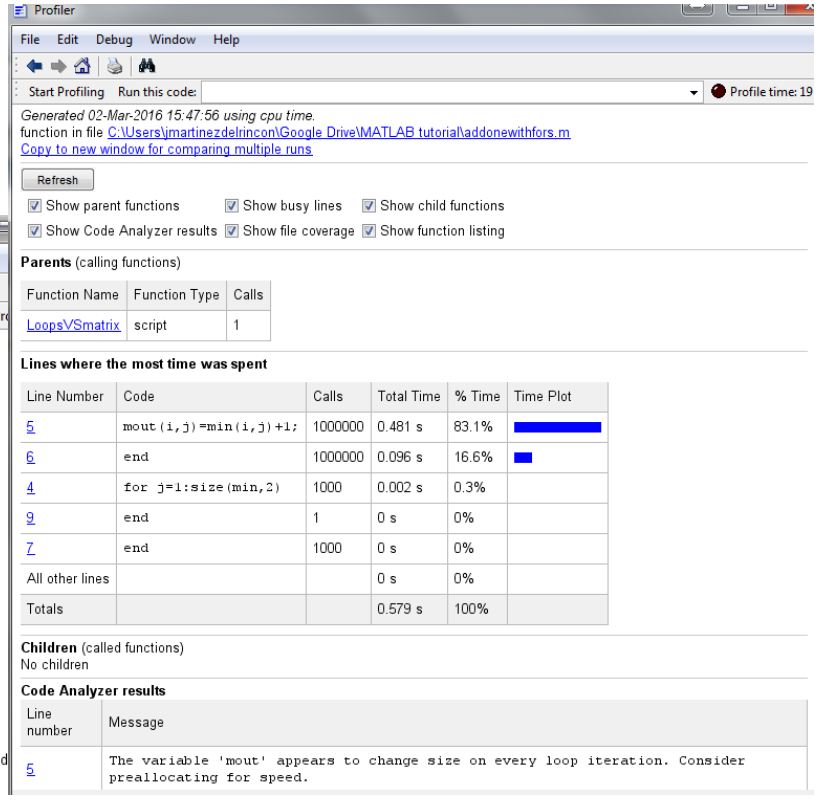
Start Profiling Run this code:

Profile Summary

Generated 02-Mar-2016 15:45:34 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
LoopsVSmatrix	1	1.025 s	0.049 s	
addonewithfors	1	0.579 s	0.579 s	
addonewithfors2	1	0.382 s	0.382 s	
close	1	0.013 s	0.013 s	
addone	1	0.002 s	0.002 s	
close>getEmptyHandleList	1	0 s	0.000 s	
close>safegetchildren	1	0 s	0.000 s	
close>checkfigs	1	0 s	0.000 s	
close>request_close	1	0 s	0.000 s	

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead from the process of profiling.



Saving time

- Always priorize matrix operations against loops
  - Opposite to other programming language
- Drawing images/graphs and plotting in the console are some of the most consuming tasks for Matlab
  - Deactivate them for running in batch mode

```
clear all
close all
drawing=0;
   .
   .
   .
if drawing
        plot(…)
end
```

variable at the beginning of my script

Every displaying/drawing bit in my code

SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Queen's University Belfast

Multidimensional Arrays (N>2)

Structures

Text

Cells

Tables

Classes

COMPLEX TYPES

SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING AND COMPUTER SCIENCE
Queen's University Belfast

Multidimensional arrays (N>2)

- Any arrays with more than two indexes (subscripts).
  - Allow to represent 3d or multidimensional data
    - Videos, evolving data over time, 3D coordinates, colour images, etc…
- Created by simple extending the notation
  - By calling `zeros`, `ones`, etc.. with more than two arguments

```
A=zeros(3,3,3)
```

  - By adding extra indexes

```
B(:,:,2)=[1 2 3; 4 5 6]        C(5,3,2,8)=11
```

- **WARNING!!** Some functions will work on 3D or ND data without problems

```
[x, y, z] = size(B)  ➔  ans = 2 3 2

sum(M,3)                mean(B,3)
```

  - However, other more complex will not (e.g. toolboxes, *,…)
    - Do a for loop for the 3rd index and beyond and operate with the 2D

Text

- String and character variables

```
myText = 'Hello, world';

letter = 'a';
```

- They are multidimensional arrays of characters
  - Sames rules, operations and indexing works

```
myText(3) ➔ ans = l

size(myText) ➔ ans = 1   12

longText = [myText,' - ',otherText]
```

- **Warning!!** Careful when concatenating/creating matrices of text

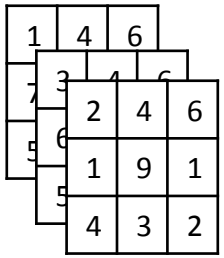
```
[myText; letter]
```
➔ ERROR! Dimensions do not agree

```
colours =['b', 'r', 'g', 'c']
```
➔ colours = brgc

```
colours =['b'; 'r'; 'g'; 'c']
```
➔
```
colours =
b
r
g
c
```

Cells

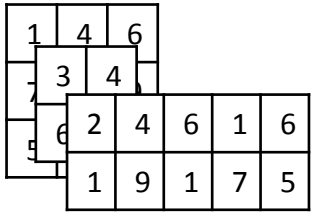
- Multidimensional arrays whose elements are copies of other arrays.
  - Allow to combine different type of data, elements of different sizes, or miscellaneous collection of things



3D arrays  **vs**  **cell arrays**

You can use three-dimensional arrays to store a sequence of matrices of the ***same*** size.

Cell arrays can be used to store a sequence of matrices of ***different*** sizes and **types**.



2	3	5	2	4	6	1	2
1	3	5	6	7	9	0	1
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
9	7	6	3	4	5	6	7

Y
N
Y
Y
Y
N

2	3	5	2	4	6	1	2	Y
1	3	5	6	7	9	0	1	N
.	.	.	.	.	.	.	.	Y
.	.	.	.	.	.	.	.	Y
.	.	.	.	.	.	.	.	Y
9	7	6	3	4	5	6	7	N

Cells

- To create a cell array, we use {}

```
C = {A sum(A) prod(prod(A))}        D = {[1 2 3 4] 'label'}
```

- {} are also use to access the content on specific indexes

```
D{1} ➔  ans = 1 2 3 4          D{2} ➔ ans = 'label'
```

  - Multidimensional!!!

```
DD={{[1 2 3 4] 'label'} {[3 4 5] 'labels2'}}

      DD{1,2},  DD{1,:}
```

  - We can concatenate the indexing (addressing!!)

```
D{1}(2) ➔      2              DD{1}{1}(2) ➔ 2
```

  - Careful on its use:

It returns the content of the cell. → `DD{1}{1}`

`DD{1}(1)` ← It returns the object cell

  - Cell arrays contain *copies* of other arrays, not *pointers* to those arrays.
    - If you subsequently change one, nothing happens to the other.

Structures

- Similar to any other programming language
  - multidimensional arrays with elements accessed by textual designator
- To create a structure, we use . or `struct()`

```
Student.name = 'Jesus';

Student.ID = 2436879;

Student.grades = [50 45 67];
```

```
Student(2) = struct('name', 'Mark',
'ID',768875,
'grades',[55 78 46])
```

- . are also use to access the content

```
Student.ID ➔  ans =  2436879        ans = 768875
```

- Multidimensional!!!
  - Easy way to recover a list of values from a specific field

```
IDs=[Student.ID] ➔ IDs =      2436879      768875

Names={Student.name} ➔ Names =      'Jesus'    'Mark'

[N1 N2] = Student.name; ➔ N1 = Jesus
                          N2 = Mark
```

Tables

3x4 table

- Merging all previous concepts we can create higher entity data structures such as tables
  - We use `table()`
  - Not very common or handy, but for display and/or input to a function/toolbox is used

table names the variables/columns with the workspace variable names

```
LastName = {'Smith';'Williams';'Jones'};
Age = [38;43;38];
Height = [71;69;64];
Weight = [176;163;131];
BloodPressure = [124 93; 109 77; 125 83];
```

```
T =

                Age    Height    Weight    BloodPressure
                ___    ______    ______    _____________

Smith           38     71        176       124      93
Williams        43     69        163       109      77
Jones           38     64        131       125      83
```

```
T = table(Age,Height,Weight,BloodPressure,'RowNames',LastName)
```

```
T = table(Age,Height,Weight,BloodPressure,...
 'RowNames',LastName,'VariableNames', {'Edad' 'Altura' 'Peso' 'Presion'} )
```

```
T = table(Age,Height,Weight,BloodPressure,'VariableNames', {'Edad' 'Altura' 'Peso'
'Presion'} )
```

```
T = table;
T.Age = Age;
T.Height = Height;
T.Weight = Weight;
T.BloodPressure = BloodPressure;
```

Classes

- Object oriented programming is possible in Matlab
  - Is it worthy?
    - Matlab is great for fast algorithm prototyping, where it shows its best potential
    - Maybe if you are developing a Matlab toolbox
- Use `classdef` to create a class ⇒ Save it in a file by itself (same name than the class name)

```
classdef BasicClass
   properties
      Value
   end
   methods
      function r = roundOff(obj)
         r = round([obj.Value],2);
      end
      function r = multiplyBy(obj,n)
         r = [obj.Value] * n;
      end
   end
end
```

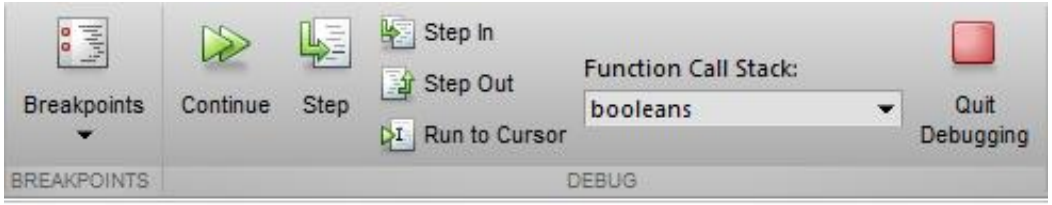
Constructor:

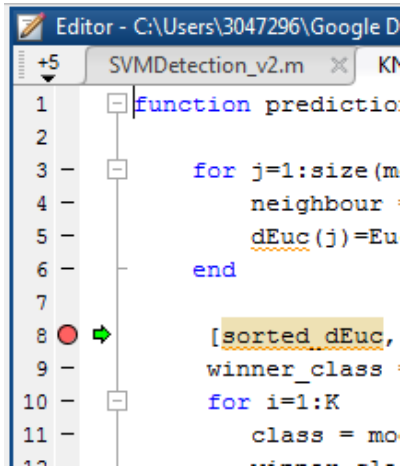
```
function obj = BasicClass(val)
      if nargin > 0
         if isnumeric(val)
            obj.Value = val;
         else
            error('Not a number')
         end
      end
   end
```

- To create an object `a = BasicClass;` `a = BasicClass(1.56);`
- To access properties/member class variables `a.Value = 2.3424;`
- To call a method `roundOff(a);` `multiplyBy(a,2);` `a.multiplyBy(2);`

Debugging your code

- Breakpoints
  - In both scripts and functions
  - The editor provides you the usual functionalities





- Ctrl+C
  - If you want to stop the execution before the end of the program
    - Infinite loops

- **Warning!** They may not work for second level functions
  - Depending of your Matlab version
  - Add breakpoints in first level functions and step in

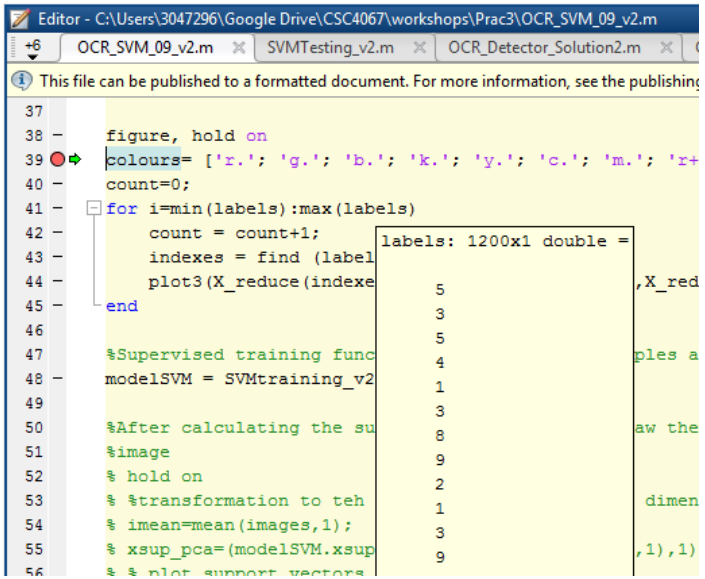
Debugging your code

- During the debugging, the command line changes

```
K>>
```

  - Allows you to explore the variable values:

```
K>> labels
```



- Other commands that allow you to navigate within the debugger:

`dbup` → Go out of the current function to the function/script that call it

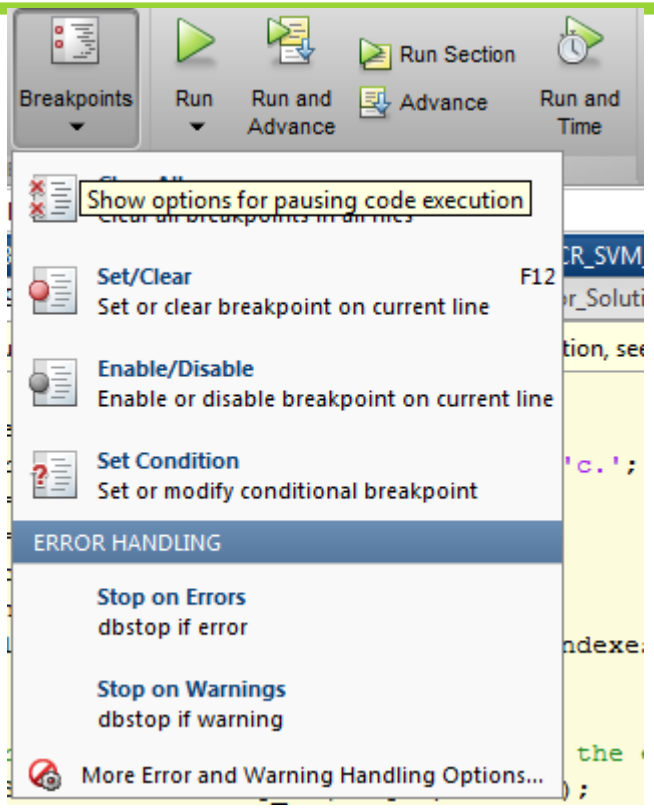
`dbquit` → Exits the debugger

Debugging your code

- Error Handling

```
clear all

close all

dbstop if error
```

  - Stops just before the error
  - Opens the debugger
  - Helpful to analyse what happened



- Saving full workspace and or variables

```
save trial1
save output1  varName1 varName2
```

  - Variables/workspace can be later loaded using `load`

```
load trial1
```

Commenting your code

- 3 purposes:
  - To give explanations of the code → Crucial for other people using your code
  - To provide documentation through `help` → Will plot in command line the block of commenst at the beginning of the script/function
  - To temporally disable some lines of code
- To write comments, we use `%`
  - Single `%`: line is ignored during running

```
%This is a comment
This is not a comment
```

```
a=1; %Initialization
```

  - `%{` : Comment a block of code/text

```
%}
```

  - `...` : Useful to fit a long statement in the screen
    - But also to comment part of a statement
    - Eve thing after the ellipsis will be considered a comment

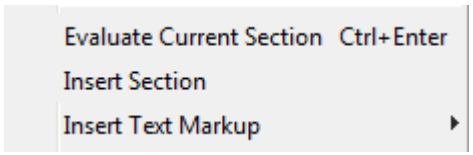
```
form = ['last name',...
        'first name',...
         'title'];
```

```
form = ['last name',...
         'first name',...
         ...'middle name',...
         'title'];
```

Commenting your code

- Double `%%`: Like `%` but it also creates a *Section*
  - More organised code
  - Script is still run as a whole
  - But each section can also be run independently → **right click**

```
58    % set(h,'lineWidth',5)
59
60
61    %% testing
62
63    % Loading testing labels and testing examples of handwritten digits from MN
64    % It is very important that this images are different from the ones used ir
65    % training or our results will not be reliable
66 -  images = loadMNISTImages('test-images',sampling);
67
68 -  for i=1:size(images,1)
69
70 -      testnumber= images(i,:);
71
72 -      classificationResult(i,1) = SVMTesting_v2(testnumber,modelSVM);
73
74 -  end
75
76
77    %% Evaluation
78
79 -  labels = loadMNISTLabels('test-labels',sampling);
```



- `Save/load` between sections can save you hours of debugging

SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Queen's University Belfast

Plot and 3D Plot

Surfaces

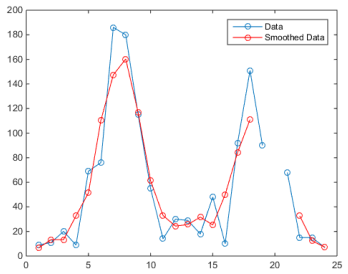
Histograms

Images

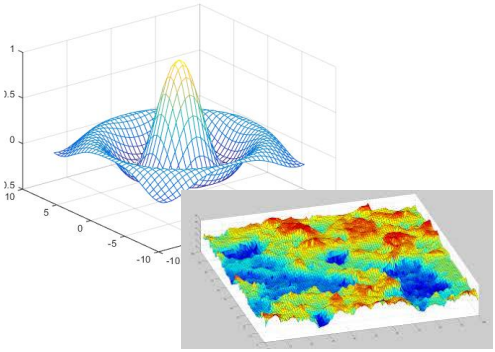
VISUALIZATION

Plotting and Visualising

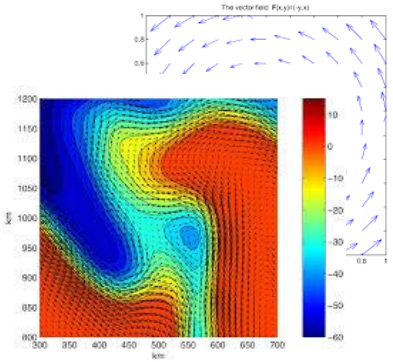
- One of the main features of Matlab
  - Native visualization functions
- Graphs:



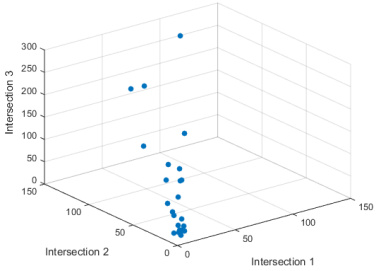
Line plots



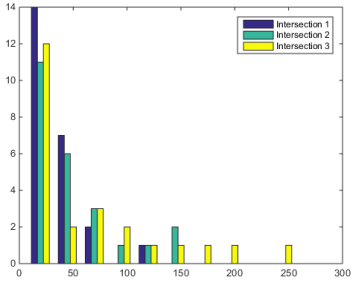
Surface plots



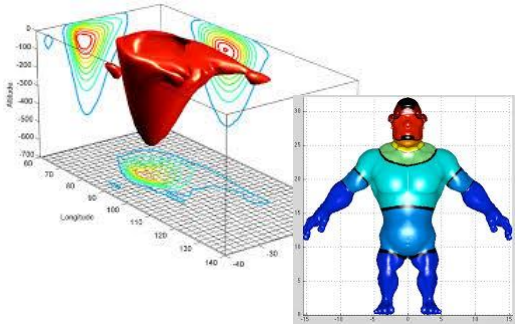
Vector fields



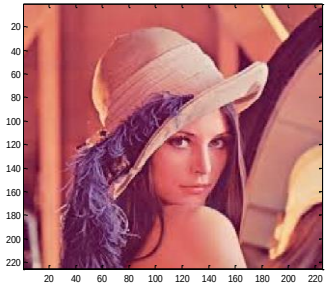
Scatter plots



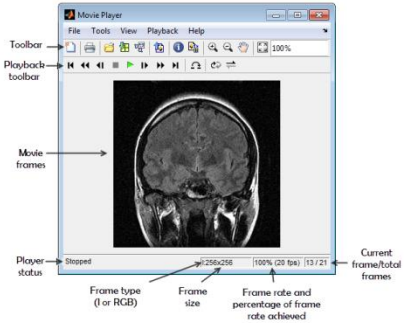
Histograms



3D graphs



Images



Videos

Figures

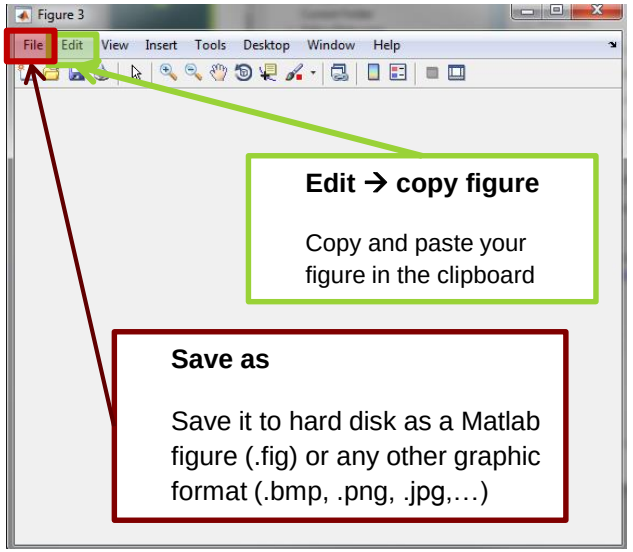
```
figure
```

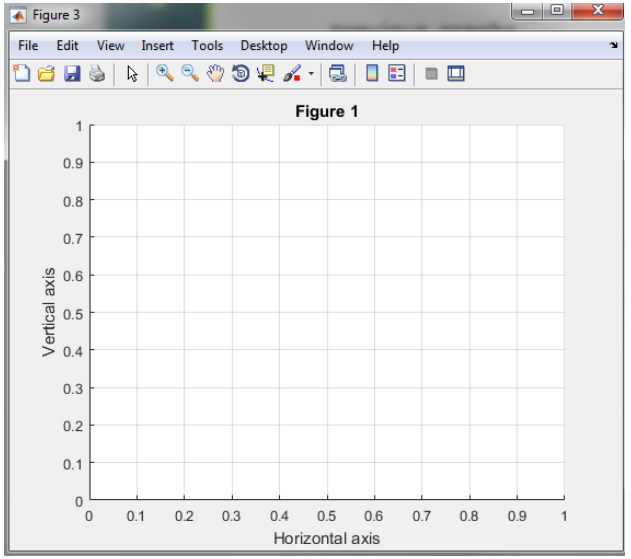
- Open a new figure window
  - It can plot one or multiple of the previous graphs
- You can select a specific figure

```
figure(number)
```

- Customizable:

```
title('Figure 1')
xlabel('Horizontal axis')
ylabel('Vertical axis')

grid on, grid off
```



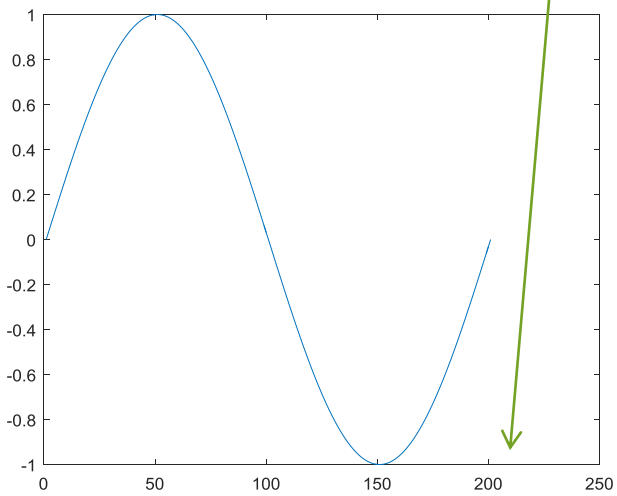


Plotting

- Function `plot(x_coord,y_coord,style)`
  - mathematical functions, curves, 2D points, 2D geometrical figures

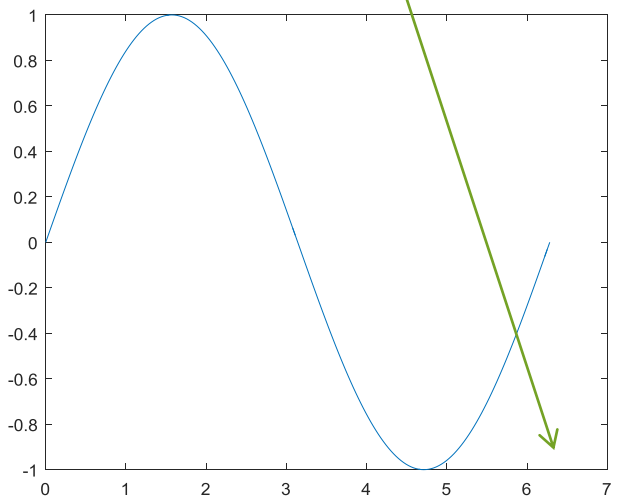
```
x = 0:pi/100:2*pi;
y = sin(x);
plot(y)
```

Plots an array vs its index



Plots an array vs the values in the other array
(dimensions should be the same)

```
plot(x,y)
```



- Colour and line styles can be changed

```
plot(x,y,'r:+')
```

Colour | Line type | Marker type

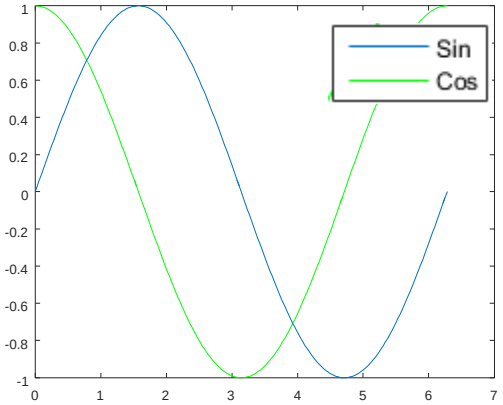
8 colours
5 line styles
13 marker types

help plot

Plotting

- Multiple functions can be plotted together

```
hold on
y2 = cos(x);
plot(x,y2,'g')
hold off
legend('Sin','Cos')
```



  - `hold on`: To avoid current figure content to be wipe-out
  - `hold off`: Remember to deactivate it, or it will redraw previous content forever ➡ Program gets slower and slower with the iterations

- When drawing inside a loop, we may not have time to see it
  - Matlab will wait to update figures to the end of the program

```
for i=1:100
        pause
        pause(time)
        drawnow
end
```

➡ Pause the program till user presses any key

➡ Pause the program for a few seconds/miliseconds

➡ Force redrawing and updating of the figure , as quick as Matlab is able, to see the updates on the screen immediately.

Subplot

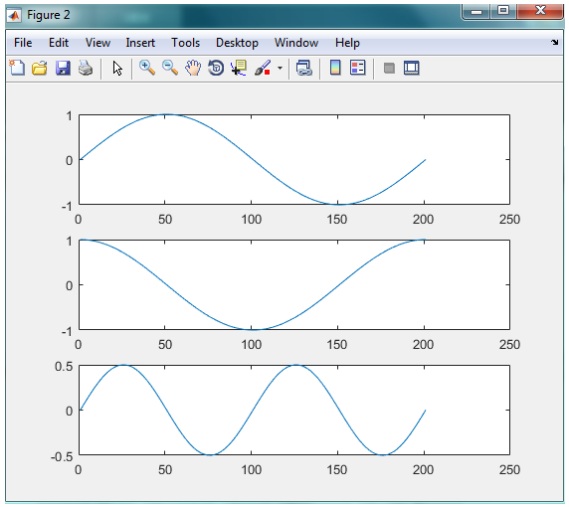
- We can also plot several graphs in the same window, but not in the same graph

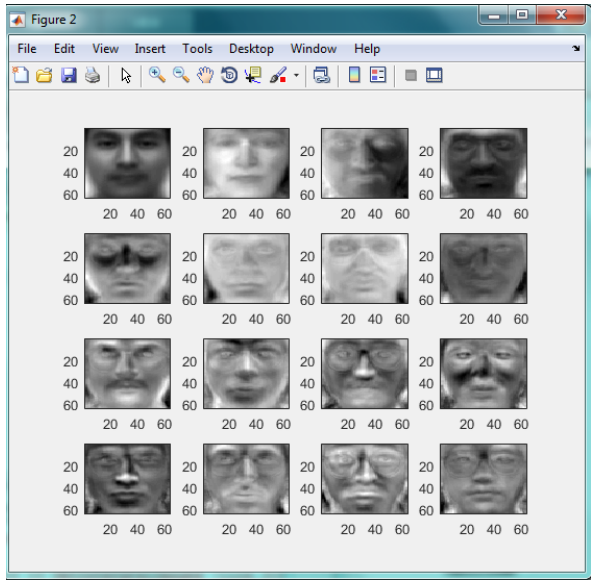
Where we will plot now

```
subplot(numberRows,numberColumns,activeSubplot)
```

- Example:

```
subplot(3,1,1); plot(sin(x))
subplot(3,1,2); plot(cos(x))
subplot(3,1,3); plot(sin(x).*cos(x))
```

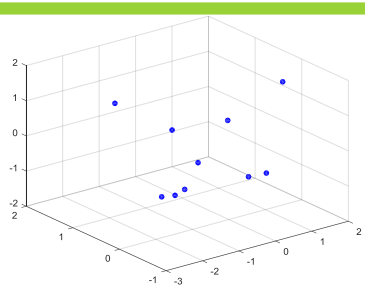




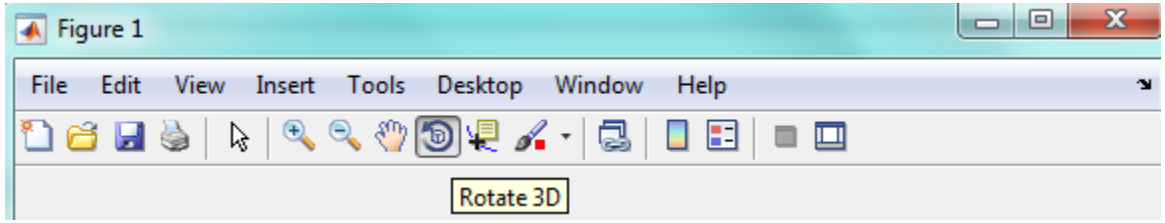
3D Plotting

- Plotting in 3D is a straight forward extension:

```
plot3(x_coord,y_coord,z_coord,style)
```



- Figure window allow you to rotate it in 3D using mouse



- To automatize it in the code or select a given viewpoint:

```
view([azimuth elevation]) or view([x y z])
```

view angle in Cartesian coordinates

  - Or some predefine view:
    - `view(2)` ➔sets the default 2-D view, AZ = 0, EL = 90.
    - `view(3)` ➔ sets the default 3-D view, AZ = -37.5, EL = 30.
  - `[AZ,EL] = view;` ➔ returns the current azimuth and elevation.

Axis

- We have full control on axis scaling and appearance.

```
axis([XMIN XMAX YMIN YMAX])

axis([XMIN XMAX YMIN YMAX ZMIN ZMAX])
```

- Some interesting ways of call it:

`axis TIGHT` ➔ sets the axis limits to the range of the data.

`axis EQUAL` ➔ sets the aspect ratio of x,y,z axis to be equal in size (increments on each axis are equal).

`axis IMAGE` ➔ same as `axis EQUAL` but the plot box fits tightly around the data.

`axis SQUARE` ➔ makes the current axis box square in size.

`axis OFF/ON` ➔ turns off/on all axis labeling

Images

- An image is a 3D array of 8 bit integers (0- 255)
  - Same indexing and matrix operators are valid
  - rows x columns ➔ image file resolution.
  - Colour images: rows x columns x RGB channels
- To load an image into Matlab:

```
I=imread('filename.jpg');
```

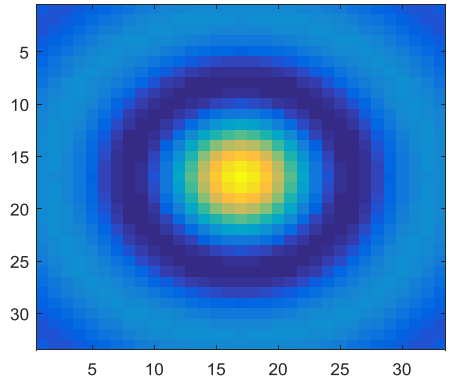
I 225x225x3 uint8

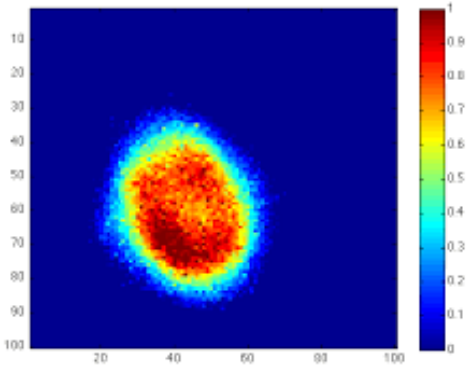
- Most standard image files (.tiff, .jpg, .jpeg, .png, .bmp)

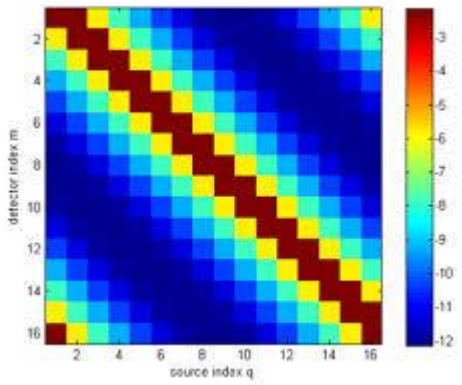
Displaying Images

- To display an image:
  - `image(I)` → displays matrix I as an image
  - `imagesc(I)` → scale data in matrix I to use the full colormap and display it as image
  - `imshow(I)` → More specific to images, optimizing figure, axes, and property settings for image display

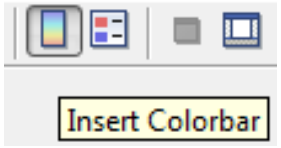
- Not only for images:
  - They can also be used to show 2D distributions





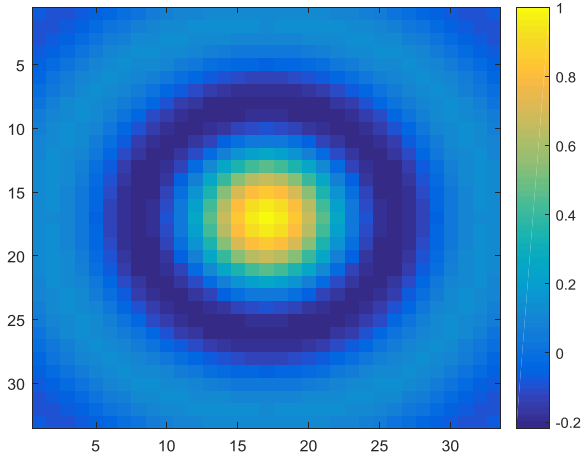


Displaying Images

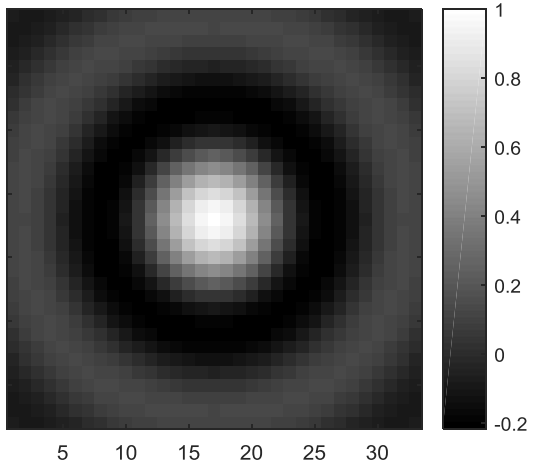


- We can change the colormap:
  - Look-up table of colours (palette) to be used

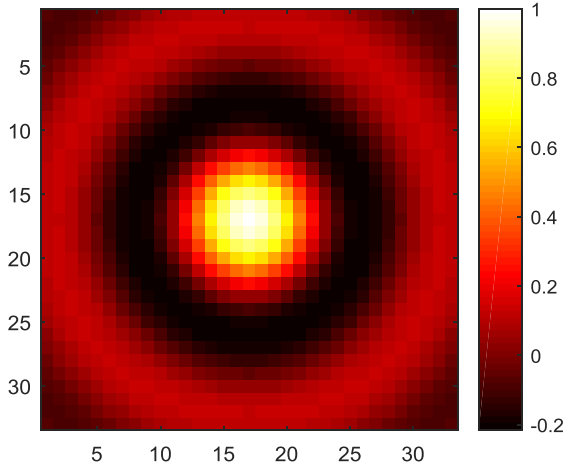
    `colormap(map)`
  - Define your own or use one of the multiple available



`colormap('default')`



`colormap(gray(256))`



`colormap(hot)`

You can also defines how many different colour you want

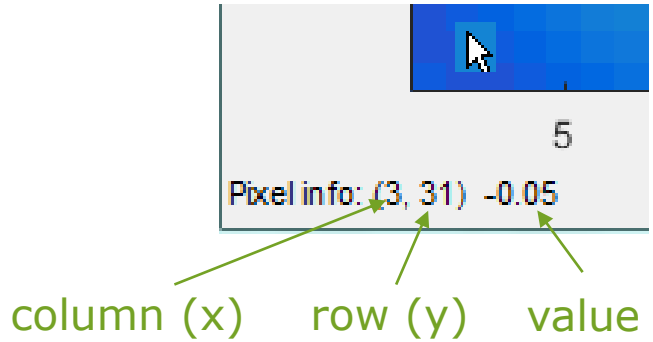
Modifying Images

- An image is a multidimensional array
  - Pixel values can be accessed and modified as usual

```
I(100,100)
I(100,100,2)
I2=I+50;
```

  - Values can also be check with the mouse:

```
impixelinfo
```



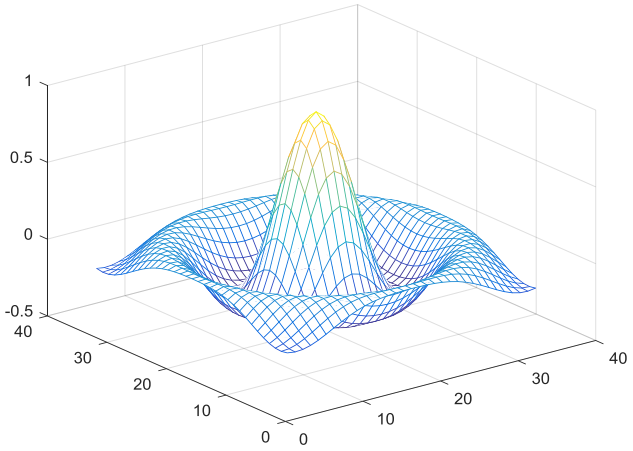
- After modification, we can save them as images

```
imwrite(I,'filename.jpg','JPEG');
```

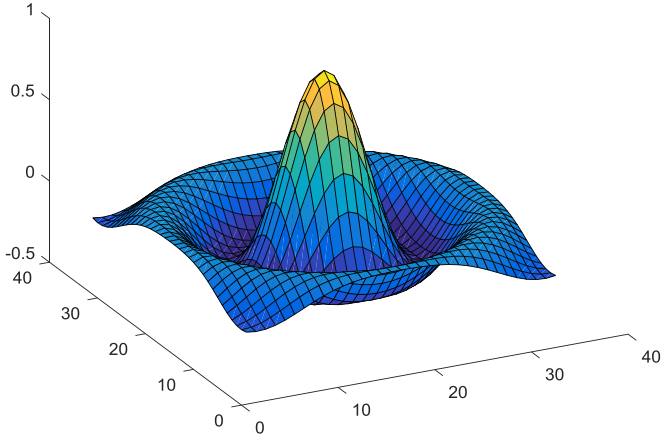
Surfaces

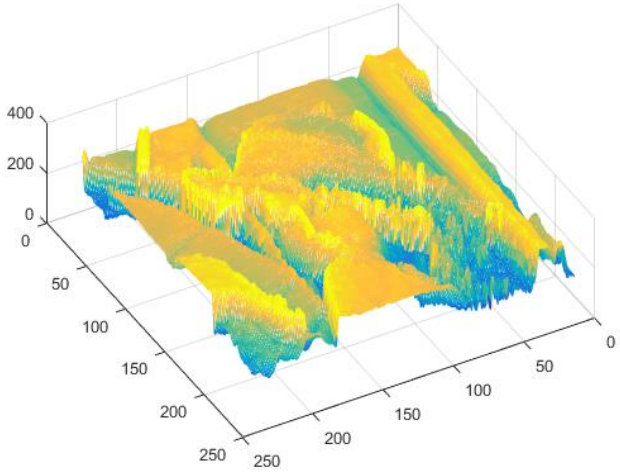
- 2D distributions can also be plotted as surfaces in 3D

`mesh(xcoord,ycoord,Z)`



`surf(xcoord,ycoord,Z)`

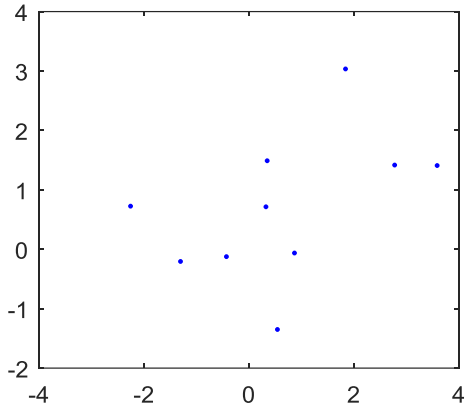




Plotting arbitrary things

- Scatter plots:
  - Simply deactivate the line type

```
plot(X,Y,'b.')
```



- Geometric shapes:
  - Points: `plot(x,y,z,'b.')`
  - Lines: `line(X,Y,Z)`
  - Rectangle, elipse, circle:

```
rectangle('Position',[x y w h],'Curvature',[0 0])
```

`w = h +`

`[1 1]`

  - Poligons:

```
line([x(1) ... x(n) x(1)], [y(1) ... y(n) y(1)])
```

  - Text: `text(x,y,zstringText)`

Augmented an image

- On the top of images:
  - `hold on/off`
  - **Warning!** y axis direction
- We can catch the drawn object handle:

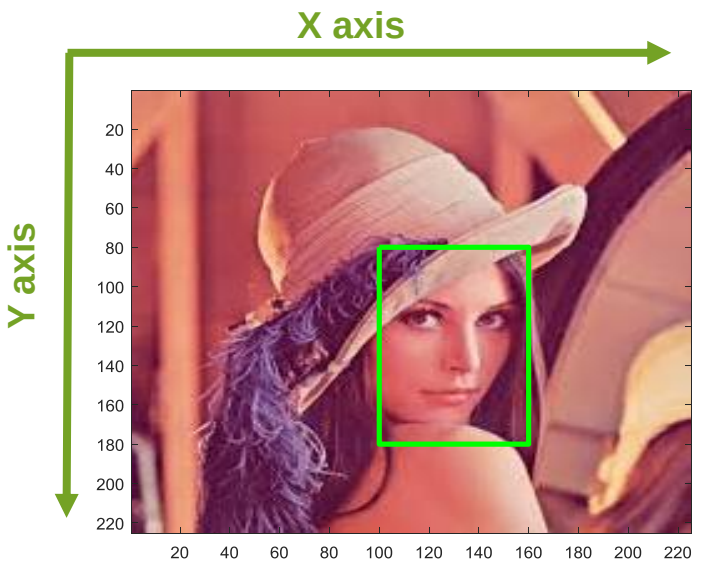
```
h=line([100 160 160 100 100],
[ 80 80 180 180 80]);
```

- Getters and setters:

`get(h)` → to see a list of object properties and current values

```
set(h,'LineWidth',3,'Color','g')
```

→ to change the values of the object properties

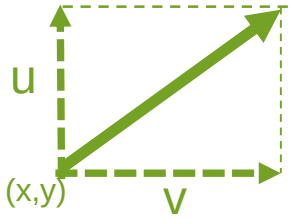


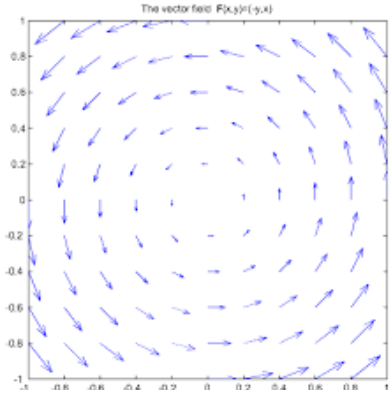

```
AlignVertexCenters: 'off'
        Annotation: [1x1 matlab.graphics.eventdata.Annotation]
     BeingDeleted: 'off'
       BusyAction: 'queue'
    ButtonDownFcn: ''
         Children: [0x0 GraphicsPlaceholder]
         Clipping: 'on'
            Color: [0 0.4470 0.7410]
         CreateFcn: ''
         DeleteFcn: ''
       DisplayName: ''
  HandleVisibility: 'on'
          HitTest: 'on'
    Interruptible: 'on'
         LineJoin: 'round'
        LineStyle: '-'
        LineWidth: 0.5000
           Marker: 'none'
  MarkerEdgeColor: 'auto'
  MarkerFaceColor: 'none'
       MarkerSize: 6
           Parent: [1x1 Axes]
  PickableParts: 'visible'
         Selected: 'off'
SelectionHighlight: 'on'
              Tag: ''
```

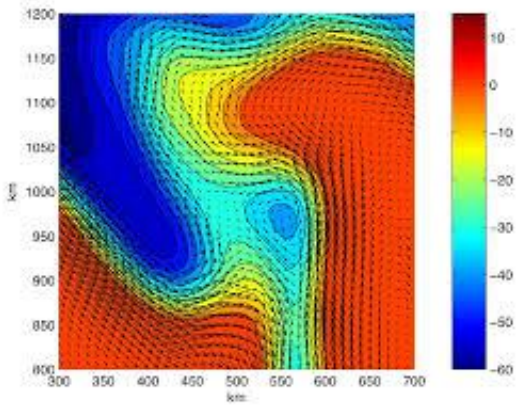
Vector fields

- Similar to plot vector fields

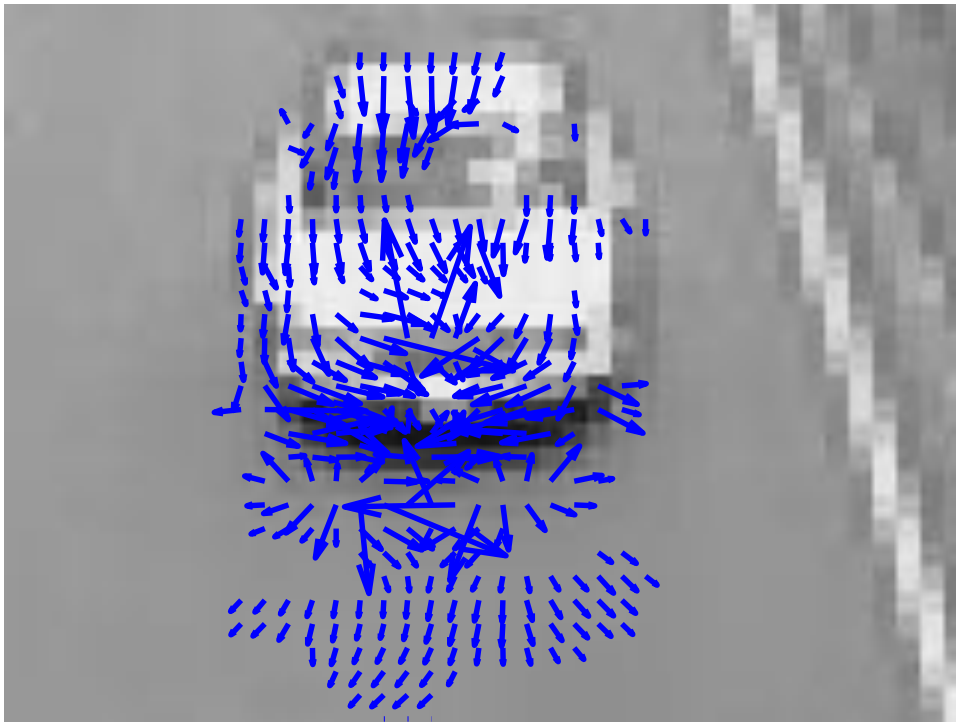
```
quiver(u,v)
quiver(x,y,u,v)
```







Filtered Optical Flow (small vectors are removed)



Histograms

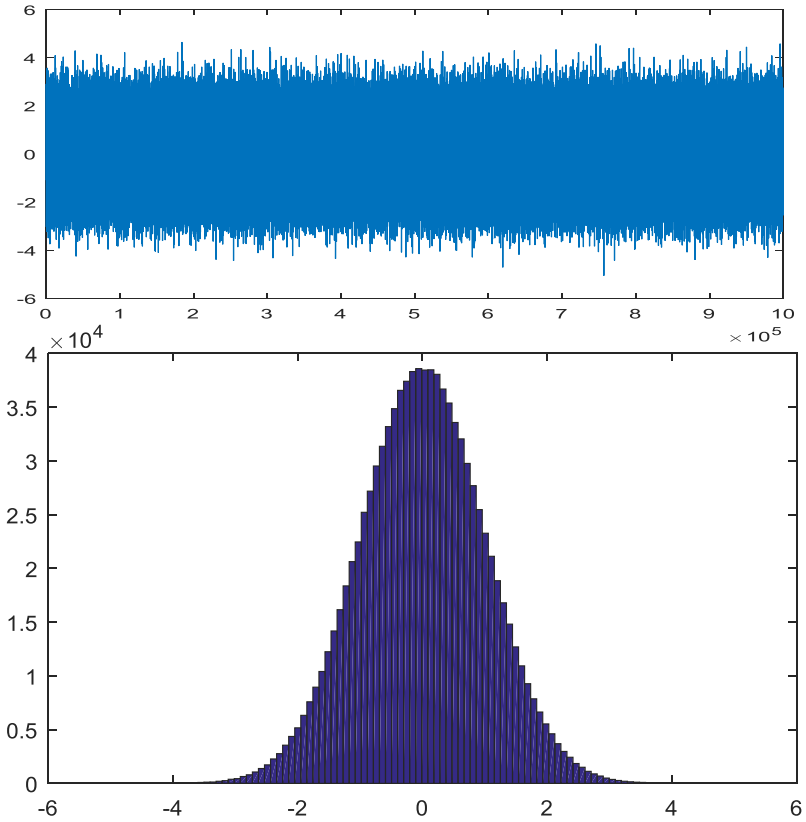
- We can also calculate the histogram of a distribution and plot it:

```
H = hist(dist,numberBins)
```

It also returns the histogram as an array

- Example;

```
noise=randn(1,1000000);

figure

plot(noise)

figure

hist(noise,100)
```



SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Queen's University Belfast

Input, ginput

Import

Reading files

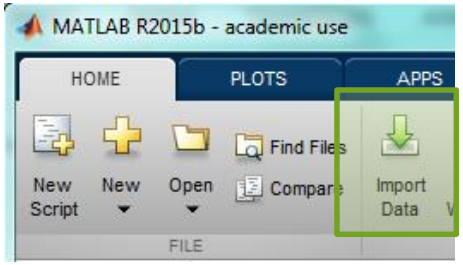
Text

GUI

I/O DATA

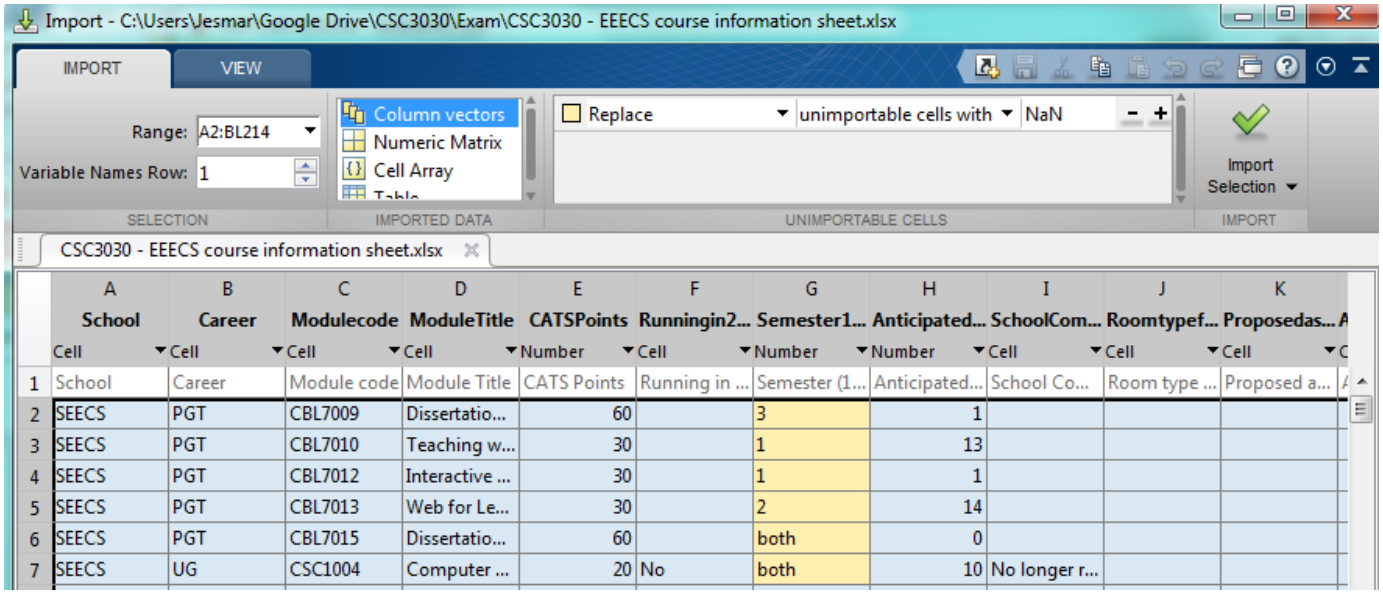
Import

- Import a file into one or several variables in the workspace




```
import()
```

  - Almost anything:
    - .xlsx, .csv, .txt, .dat, audio, images, videos,



	A	B	C	D	E	F	G	H	I	J	K
	School	Career	Modulecode	ModuleTitle	CATSPoints	Runningin2...	Semester1...	Anticipated...	SchoolCom...	Roomtypef...	Proposedas...
	Cell	Cell	Cell	Cell	Number	Cell	Number	Number	Cell	Cell	Cell
1	School	Career	Module code	Module Title	CATS Points	Running in ...	Semester (1...	Anticipated...	School Co...	Room type ...	Proposed a...
2	SEECS	PGT	CBL7009	Dissertatio...	60		3	1			
3	SEECS	PGT	CBL7010	Teaching w...	30		1	13			
4	SEECS	PGT	CBL7012	Interactive ...	30		1	1			
5	SEECS	PGT	CBL7013	Web for Le...	30		2	14			
6	SEECS	PGT	CBL7015	Dissertatio...	60		both	0			
7	SEECS	UG	CSC1004	Computer ...	20	No	both	10	No longer r...		

  - Sometimes a bit too manual for full automatization

Working with binary files

- To open a file:

```
fileID = fopen(filename,permissions);
```

- To close it:

```
fclose(fileID)
```

  - Warning! You will not be able to open the file again till you call `fclose` or Matlab closes

Reading/writing files

`fgetl`	Read line from file, removing newline characters
`fgets`	Read line from file, keeping newline characters
`fileread`	Read contents of file into string
`fread`	Read data from binary file
`frewind`	Move file position indicator to beginning of file
`fscanf`	Read data from text file
`fseek`	Move to specified position in file
`ftell`	Position in open file
`fwrite`	Write data to binary file
`fprintf`	Write data to text file

- E.g.:

```
A = fscanf(fileID,'%f');
fwrite(fileID,A,precision)
```

Interface through console

- Input

```
answ = input(promptText)
answ = input(promptText,'s')
```

  - Displays the PROMPT string on the screen
  - Waits for input from the keyboard (until Enter key)
  - Returns the value (number or string) in output value
  - E.g.:

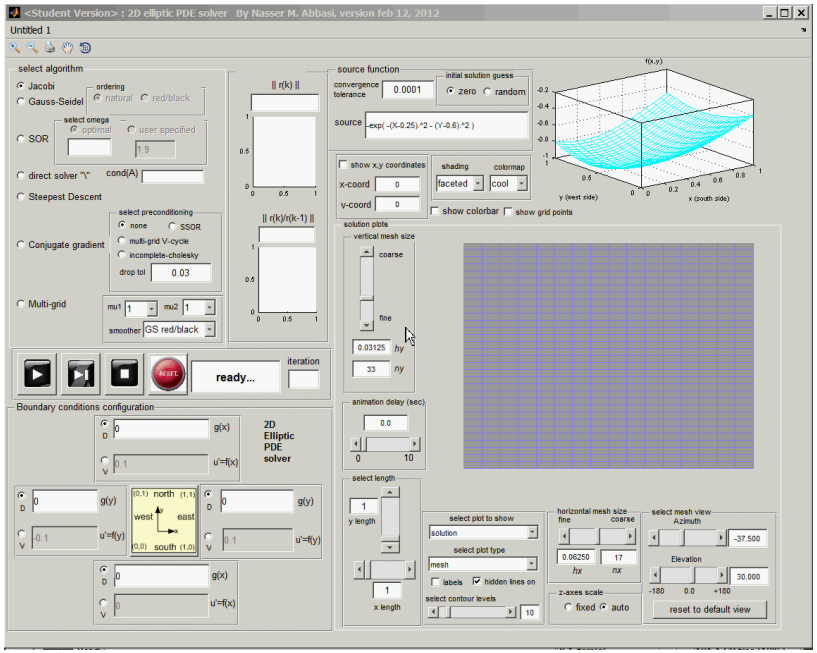
```
reply=input('Do you want more? Y/N:','s');
```

- Output

```
disp(promptText)
```

Other interfaces

- `[x, y, button] = ginput();`
  - Capture mouse position on a figure
- `load/save` .mat files
  - Intermediate results, alternative versions, etc…
- Graphic User Interfaces (GUI)



Data conversión

Runtime

Common problems and mistkes

Other languages

EXTRA NOTES

Data Conversion

- Every numeric value is a double by default
  - Matlab is a scientific software ➔ Precision is a priority
- But we can work with any other datatype:
  - Save time and memory

`double()`	Convert to double precision
`single()`	Convert to single precision
`uint8(), int8()`	Convert to unsigned/signed 8-bit integer
`uint16(), int16()`	Convert to unsigned/signed 16-bit integer
`uint32(), int32()`	Convert to unsigned/signed 32-bit integer
`uint64(), int64()`	Convert to unsigned/signed 64-bit integer

- `double() / uint8()`
  - Allow modifying images and/or save results as images

WARNING! You will not notice the overflow in an image, it will simply capped the value to 255 and carry on

Data Conversion

- Convert numbers to/from its ASCII representation

```
num2str()

str2num()
```

- I/O data:
  - To/from the workspace to the user interface

- Run-time changing behaviour

```
disp(['I found ',num2str(var),' solutions'])
```

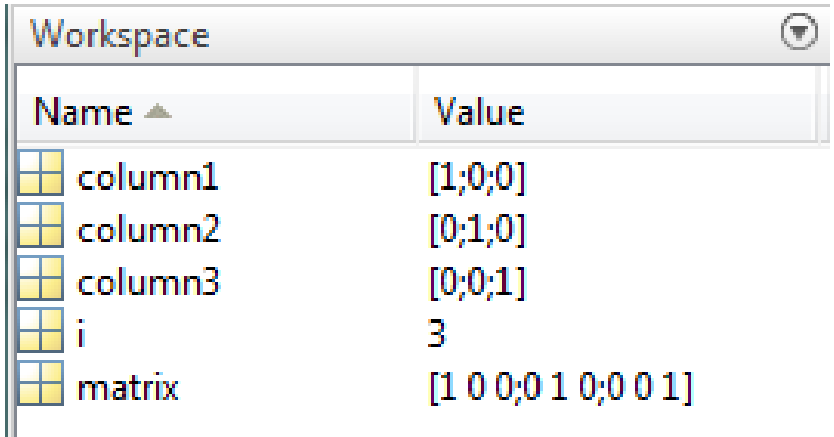
Runtime Dynamic Behaviour

- Execute MATLAB expression in text string

```
eval(string)
```

- E.g.:

```
for i=1:size(matrix,1)
    eval(['column',num2str(i),'=matrix(:,i)'])
end
```



More about functions

- Matlab functions are heavily overloaded
  - Multiple versions of the same function with different output/input arguments ⟹ **Always check documentation!**
  - Input arguments: Variable number
    - Default values
    - Different behaviour with different data type
  - `nargin`
    - Returns the number of function input arguments

```
function out=fName(ar1,ar2)
if nargin<2
    arg2=0;
end
```

  - Outputs are easily discarded by not assigning them to a variable

Common problems and mistakes

- Overloading is possible at all levels
  - At class level
  - At level function
  - At variable level
    - **Warning!** You can name a variable with the name of an existing function!!

```
A=eye(3);
find(A)
```

```
find=15;
.
.
A=eye(3);
find(A)
```

  - Errors everywhere
    - Subscript indices must either be real positive integers or logicals.
    - Matrix dimensions must agree

Common problems and mistakes

- Do not clear variables in the script
  - You will notice that the second time you try to run the script it behaves differently or gives an error
- No not reset dynamic size variables at each iteration
  - Matrix dimensions must agree

Memory problems

- If your multidimensional arrays are too big, you may not have memory to use them
  - Specially in 32 version/systems

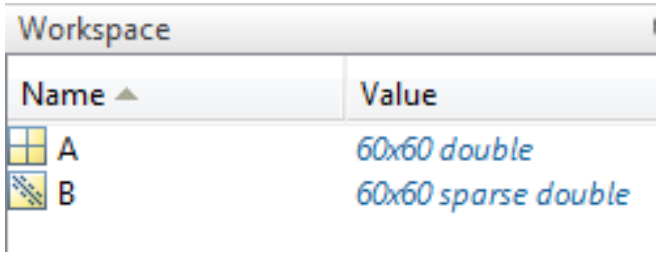
```
A=zeros(100000)    A=zeros(1000,1000,1000)
```

Error using zeros
Requested 100000x100000 (74.5GB) array exceeds maximum array size preference. Creation of arrays greater than this limit may take a long time and cause MATLAB to become unresponsive.

Error using zeros
Out of memory. Type HELP MEMORY for your options.

- Solution: use sparse matrices

```
B=sparse(A)
```



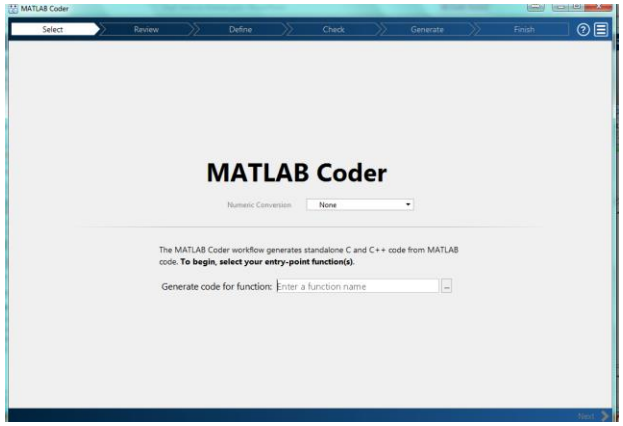
  - All of MATLAB's built-in arithmetic, logical and indexing operations can be applied to sparse matrices

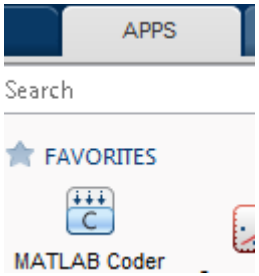
C/C++ compiler

- Convert you code into C

http://blogs.mathworks.com/loren/2011/11/14/generating-c-code-from-your-matlab-algorithms/

- Matlab Coder App






```
1     function c = simpleProduct(a,b)
2 -     c = a*b;
```

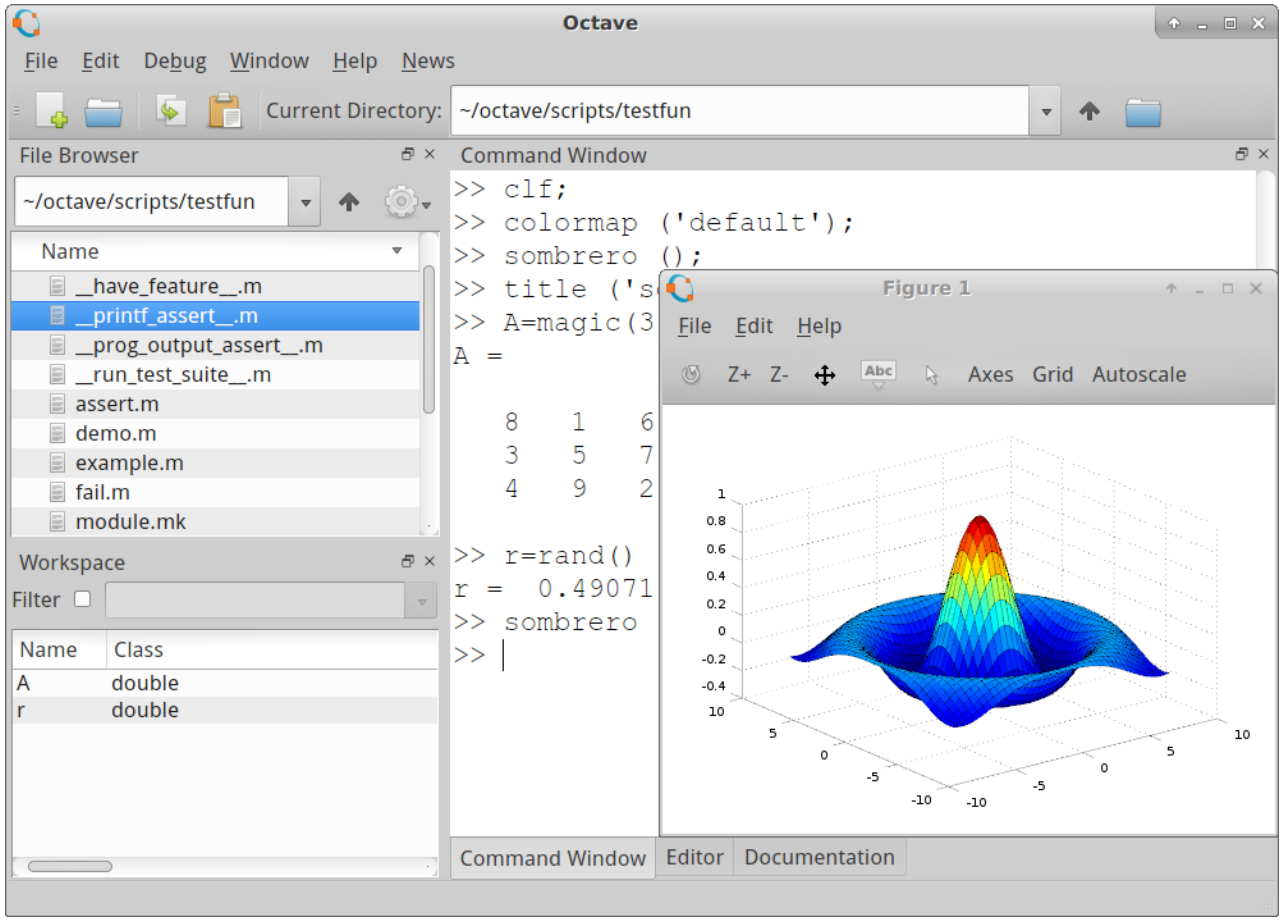
```
/* Include Files */
#include "rt_nonfinite.h"
#include "simpleProduct_fixpt.h"

/* Function Definitions */
/*
 * Arguments    : unsigned short a
 *                unsigned short b
 * Return Type  : unsigned int
 */
unsigned int simpleProduct_fixpt(unsigned short a, unsigned short b)
{

  return (unsigned int)a * b;
}
```

Calling external functions

- Using Mex files
- Allow calling functions from C/C++ to run in Matlab with the rest of your code
- Also for:
  - Java, C/C++, .NET, Fortan and Python

Alternatives

- **Octave**



- Others:
  - Scilab, Freemat, Mathscript(Labview)
  - Python + libraries, etc…

Questions?

Recommended Resources

- *Matlab Get Start Guide* **Mathworks**
- Language Reference Manual **Mathworks**
- Matlab central: file exchange (web)
- *Aprende Matlab como si estuviera en primero* **J. García, J. Rodríguez, J. Vidal**
- And many others