**HERIOT-WATT UNIVERSITY**

**DEPARTMENT OF COMPUTING AND ELECTRICAL ENGINEERING**

# B35SD2 – Matlab tutorial 2
# MATLAB PROGRAMMING

## Objectives:

Remember last week? Well you'd better.
We have introduced you to the basic use of matlab as a program to perform on-line simple calculations, analysis and display. But it is much more than a glorified calculator as we will see today.

During this session, you will learn:

1- Basic flow control and programming language
2- How to write scripts (main functions) with matlab
3- How to write functions with matlab
4- How to use the debugger
5- How to use the graphical interface
6- Examples of useful scripts and functions for image processing

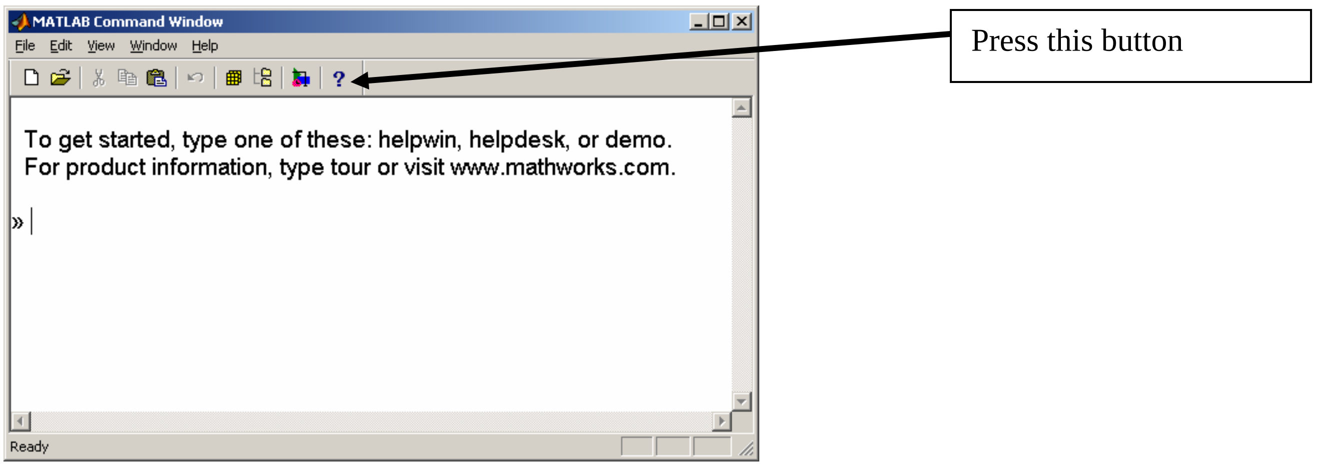
After this hour, you will know most of what you need to know about Matlab and should definitely know how to go on learning about it on your own. So the "programming" aspect won't be an issue any more, and you will be able to use matlab as a tool to help you with you math, electronics, signal & image processing, statistics, neural networks, control and automation….Programming languages don't come any easier!

Next page is a reminder of last week main points:

### How to start Matlab:

Press the matlab Icon on your desktop.

### How to get help:



```
To get started, type one of these: helpwin, helpdesk, or demo.
For product information, type tour or visit www.mathworks.com.

» 
```

Or type help in the command line

Or type lookfor ‘string’ in the command line

Or type Matlab help desk in the help window

Or  go to http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.shtml

### How to get an overview of matlab:

Use the **demo** tool typing demo in the command line. This can be used as a self-training tool.

### How to see what’s in your workspace:

```
>> who
>> whos                    % more detailed
```

### How to save and load a workspace

use the save and load commands to save results and data.
see help save for more details.

## Matlab ressources:

- Language: High level matrix/vector language with
    - Scripts and main programs
    - Functions
    - Flow statements (for, while)
    - Control statements (if,else)
    - data structures (struct, cells)
    - input/ouputs (read,write,save)
    - object oriented programming.
- Environment
    - Command window. You are now familiar with it
    - Editor
    - Debugger
    - Profiler (evaluate performances)
- Mathematical libraries
    - Vast collection of functions
- API
    - Call C functions from matlab
    - Call Matlab functions form C programs
- **GUI tool**
    - Allows to design easily Graphical User Interfaces to your programs.

## Scripts and main programs

In matlab, scripts are the equivalent of main programs. The variables declared in a script are visible in the workspace and they can be saved. Scripts can therefore take a lot of memory if you are not careful, especially when dealing with images. To create a script, you will need to start the editor, write your code and run it.

## How to use the editor

Just type edit in the command line and the editor starts.
This should start the editor with a new file. Write the following in the file and save the file as readimage.m in your local directory. To go to your local directory, ude the cd command as seen last week.

```
% This is my first script in matlab
% This script reads in an image, displays it and changes the colormap

% First read an image from matlab images
ima = imread('moon.tif');
% Now display it
clf;            % Close figure 1
figure(1);      % First create a figure
image(ima) ;    % Display but do not resize or rescale. Useful only of image is in [0,256]
disp('Press a key to continue');
pause;          % Waits until a key is pressed to continue
imagesc(ima); % Display and rescale the image. Useful for floating point images.
colormap(gray); %Uses the gray colormap
pause(5);       % Wait 5 seconds and display the image
```

```
imshow(ima);  % Normal matlab function from image processing toolbox
% Now write the image to disk
disp(‘Now writing image in jpeg format’);
imwrite(ima, ‘moon.jpg’,’jpg’);  % Write the image in JPEG format
% various formats are supported (see help imwrite)

%CAREFUL IMSHOW ONLY WORKS WITH UINT8 IMAGES
```

## Scripts and Functions:

You've already written your first elementary script. Standard branching and looping instructions can be used as well: "for" loops, "if" statements etc… To find out about their syntax, type "help" for these different instructions.

### <u>if statement:</u>

the if statement is written as follows:

**if (*condition1*)**
        ***expression1***
**elseif (*condition2*)**
        ***expression2***
**else**
        ***expression3***
**end**

### <u>Logical operators:</u>

| | |
|---|---|
| **A==B** | **equality for scalars** |
| **isequal(A,B)** | **equality for vectors, matrices, structures** |
| **A~=B** | **difference** |
| **A>B** | **superior. Will return an array of boolean for arrays with 1 for elements where the condition is satisfied and 0 elsewhere.** |
| **A<B** | **inferior. Will return an array of boolean for arrays with 1 for elements where the condition is satisfied and 0 elsewhere.** |
| **A>=B** | |
| **A<=B** | |
| **isempty(A)** | **Check if the variable exists and is not empty.** |
| **isinf(A)** | **Returns 1 if A is Inf (infinity), 0 otherwise.** |
| **isnan(A)** | **Returns 1 if A is not a number (NaN), O otherwise[1].** |

### <u>Switch and case:</u>

**switch (*expression*)**
        **case *value1***
                ***action1;***
        **case *value2***
                ***action2;***
        **otherwise**

[1] Note that NaN and Inf are defined. Example A = Inf

```
        default action;
    end
```

**<u>For loops</u>**[2]

```
for index = 1 : m
    r(index) = index;
end
```

Example:

```
    Do
        ima = zeros(100,100);
        for i = 1:100
            for j = 1:100
                ima(i,j) = i+j;
            end
        end
        imagesc(ima)            % imshow is buggy and sometimes does not work
                                % properly. In that case use imshow then imagesc or
                                % directly imagesc.
```

**<u>while loops</u>**[3]

```
while (expression)
    Action
end
```

**<u>Functions</u>**

Functions differ from scripts in that they take explicit input and output arguments. Type "help" function to know more and see examples. As all other matlab commands, a function can be called within a script of from the command line.

The synthax of a function is as follows:

***function* [out1,out2,...,outn] = *function_name*(in1,in2,...,inn)**

**Tips:**

| | |
|---|---|
| **nargin:** | **gives the number f input arguments** |
| | **allow defaults parameters** |
| | **allow variable number of arguments** |

**% This is the symbol for comments**

As an example, here is a basic function that read in an image, display it and finds the min and max of the image. **The function must be saved into a file of the same name as the function to be accessible from the command line.** In our case we save it as min_max.m.

[2] They are slow in matlab and can sometime be avoided by vectorizing your programs.
[3] They are slow in matlab and can sometime be avoided by vectorizing your programs.

```
function [mini,maxi] = min_max(imageName)
        ima = imread(imageName);    % reads in an image of name imageName
        imshow(ima);
        ima = rgb2gray(ima);         % Converts image to grey image
        inagesc(ima); axis image;    % Display the image
        mini = min(min(ima));        % Get the min of the image
        maxi = max(max(ima));        % Get the max of the image
```

Nothing else is required!

**Do**

```
        >>[mini,maxi] = min_max('peppers.png');
```

Try to modify this function to get the maximum and minimum of each color

## Debugging a program

Debugging is easy in matlab as it is an interpreted language. In order to start the debug mode, you just have to type in the command line:

```
>>dbstop if error               % Will stop for errors
>>dbstop if warning             % Will stop for warnings
```

Alternatively, you can go into the editor and set the options and breakpoints where you want. In any case, the editor will be invoked and show the line where the error has occurred.
In this case the K symbol appears in the command line.
As matlab is interpreted you can change the values of variables and continue the execution using:

```
K>>dbstep          %continues by one line
K>>dbcont          % continues until next stop
```

To see all the options of the debugger, use help dbstop.

To quit the debugger, use dbquit
To remove breakpoints and debugger options, use dbclear

Finally you can use the keyboard functions. This does not start the debugger but stops the execution where the keyboard has been typed. You can see the variables and resume execution typing dbcont.

## Function/Command Duality

| Command line | In function or script |
|---|---|
| load August17.dat | load('August17.dat') |
| save January25.dat | save('January25.dat'); |
| cd H:\matlab\ | cd('H:\matlab') |

## General Tips

### Vectorization:

```
x = 0;
for k = 1 : 1001
   y(k) = log10(x);
   x = x +0.01;
end
plot(x,y);
```

```
x = 0:0.01:10;
y = log10(x);
```

**10 times faster!**

**Preallocation:**

```
r = zeros(32,1);  ←————————————  5 times faster!
for n = 1 : 32
   r(n) = n;
 end
```

**Matrices**

| | |
|---|---|
| **zeros(m,n)** | **Creates a mxn matrice filled with zeros** |
| **ones(m,n)** | **Creates a mxn matrice filled with ones** |
| **rand(m,n)** | **Creates a mxn matrice following a uniform distribution in [0,1]** |
| **randn(m,n)** | **Creates a mxn matrice following a normal gaussian distribution (mu = 0, sigma = 1)** |

## Full examples to do today:

**Image inversion:**

Example  program that reads in an image and invert it.

**Fourier Transform:**

Example program that takes the Fourier transform and display it.
Same for the inverse Fourier Transform.

These programs must be downloaded from:
http://www.ece.eps.hw.ac.uk/~ceeyrp/WWW/Teaching/B39SD2/B39SD2.html

**Please stick this page into your bookmarks as it will be useful for the remainder of the course.**

**Please download them and try them out.**

### Image inversion

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Image inversion
%
%       ima_out = invert(ima)
%
% This function inverts the input image
%
%       ima : real input image
%       ima_out : output real
%
%
% Yvan Petillot, December 2000
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function ima_out = invert(ima)

  % Checks that ima is a gray level image
  if ndims(ima) > 2
    % RGB image?
    if ndims(ima) == 3
      ima = double(rgb2gray(uint8(ima));
    else
      display('Unknown format, Cannot guarantee result');
    end
  end
  % Create new figure
  figure(1)
  % Display ima
  imagesc(ima);
  % Convert ima to double
  ima = double(ima);
  colormap(gray);
  axis(image);
  % Rescales ima between [0, 255]
  mini = min(min(ima));
  maxi = max(max(ima));
  ima_out = (ima-mini)/(maxi-mini)*255;
  % Invert ima_out
  ima_out = 255-ima_out;
  % Display result
  % Create new figure
  figure(2)
  imagesc(ima_out);
  colormap(gray);
  axis image;
  axis off;
```

Try using this function with various images.
>> ima = imread('peppers.png');
>> ima_out = invert(ima);

*Note the use of ndims to find out the number of dimensions of the input array.*

Could you invert each color and generate a inverted RGB image?
Try it in you spare time...

**Fourier Transform**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% 2 Dimensional Fourier Transform
%
%       [imafft,mag,phi] = fft2d(ima_in)
%
% This function calculates the complex Fourier transform
% and the associated magnitude and phase (mag, phi) of the input real
% image ima_in.
%
%       ima_in : real input image
%       imafft : output complex image
%                real and imaginary part can be accessed through real
%                and imag function (see help real, imag)
%
%       mag,phi: magnitude and phase of the fourier transform
%
% Yvan Petillot, December 2000
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [imafft,mag,phi] = fft2d(ima_in)

  imafft = fftshift(fft2(fftshift(ima_in)));
  mag = abs(imafft);
  phi = angle(imafft);
  %Display results
  clf;       %Clear display
  figure(1); %Create figure
  title('Demo of the fourier transform')
  subplot(1,3,1);  %Separate figure in 3 and selects 1st partition
  imagesc(ima_in); % Scale image to [0 255] range and display it
  title('Original image');
  axis image;     % Display equal on x and y axes
  colormap(gray);  % Sets gray scale colormap;
  axis off;
  subplot(1,3,2);  % Selects the second partition
  imagesc(log(mag+1));    % Display magnitude
  title('Magnitude of the spectrum');
  axis image;     % Display equal on x and y axes
  axis off;
  subplot(1,3,3);  % Selects the second partition
  imagesc(phi);    % Display phase
  axis image;     % Display equal on x and y axes
  title('Phase of the spectrum');
  axis off;
  drawnow;         % Ready to dipslay: Do it
```

**Try using this function with various images.**
**>> ima = imread(‘peppers.png’);**
**>> ima = double(rgb2gray(ima));**
**>> ima_out = fft2d(ima);**

*Note that ima_out is an array of complex numbers stored as real and imaginary parts. real(ima_out) returns the real part while imag(ima_out) returns the imaginary part. Magnitude and phase can be obtained from the real and imaginary part using the abs and angle functions. This functions, as always in matlab work directly on arrays.*

**What would happen if the image was rotated?**
**Try it using the imrotate function to rotate the image with and without the crop parameter (see help imrotate) on the ‘circuit.tif’ image.**
**Do**

**>> ima = imread(‘circuit.tif’);**
**>> ima1 = fft2d(ima);**
**>> ima2 = fft2d(imrotate(ima,30));**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% 2 Dimensional Inverse Fourier Transform
%
%       [ima] = ifft2d(ima_in)
%
% This function calculates the complex Inverse Fourier transform
%  of the complex image ima_in.
%
%       ima_in : complex input image
%       ima : output (possibly complex) image
%             real and imaginary part can be accessed through real
%             and imag function (see help real, imag)
%
%
% Yvan Petillot, December 2000
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function ima = ifft2d(ima_in)

  ima = fftshift(ifft2(fftshift(ima_in)));
  %Display results
  clf;       %Clear display
  figure(1); %Create figure
  imagesc(abs(ima)); % Scale image to [0 255] range and display it
  axis square;     % Display equal on x and y axes
  colormap(gray);  % Sets gray scale colormap;
  drawnow;         % Ready to dipslay: Do it



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% 2 Dimensional generic Fourier Transform
%
%       [ima_out] = fft2(ima_in,direction)
%
```

```
% This function calculates the complex Fourier transform
% of the input image ima_in. If direction is set to 1, the direct fft
% is performed, otherwise, the inverse one performed.
%
%       ima_in : input image. Can be real or imaginary
%       direction: optional parameter. By default, direct Fourier transform
is assumed
%       imafft : output image
%
%
% Yvan Petillot, December 2001
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ima_out] = fft(ima_in,direction)

if nargin == 1
   direction = 1;
end

if (direction == 1)
   [ima_out,mag,phi] = fft2d(ima_in);
else
   ima_out = ifft2d(ima_in);
end
```

**This is essentially a demo of the nargin parameter**