

## Image Processing Matlab tutorial 2 MATLAB PROGRAMMING

### 1. Objectives:

Last week, we 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!

### 2. Matlab resources:

#### Ø 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.

#### Ø Environment

- Command window. You are now familiar with it
- Editor
- Debugger
- Profiler (evaluate performances)

#### Ø Mathematical libraries

- Vast collection of functions

#### Ø API

- Call C functions from matlab

- Call Matlab functions from C programs
- Ø **GUI tool**
  - Allows to design easily Graphical User Interfaces to your programs.

### 3. 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.

### 4. 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, use 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
```

### 5. 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.

#### 5.1 if statement:

the if statement is written as follows:

```
if (condition1)
    expression1
```

```
elseif (condition2)
    expression2
else
    expression3
end
```

## 5.2 Logical operators:

<code>A==B</code>	<b>equality for scalars</b>
<code>isequal(A,B)</code>	<b>equality for vectors, matrices, structures</b>
<code>A~=B</code>	<b>difference</b>
<code>A&gt;B</code>	<b>superior. Will return an array of boolean for arrays with 1 for elements where the condition is satisfied and 0 elsewhere.</b>
<code>A&lt;B</code>	<b>inferior. Will return an array of boolean for arrays with 1 for elements where the condition is satisfied and 0 elsewhere.</b>
<code>A&gt;=B</code>	
<code>A&lt;=B</code>	
<code>isempty(A)</code>	<b>Check if the variable exists and is not empty.</b>
<code>isinf(A)</code>	<b>Returns 1 if A is Inf (infinity), 0 otherwise.</b>
<code>isnan(A)</code>	<b>Returns 1 if A is not a number (NaN), 0 otherwise<sup>1</sup>.</b>

## 5.3 Switch and case:

```
switch (expression)
    case value1
        action1;
    case value2
        action2;
    otherwise
        default action;
end
```

## 5.4 For loops<sup>2</sup>

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

Example:

```
Try
    ima = zeros(100,100);
    for i = 1:100
        for j = 1:100
            ima(i,j) = i+j;
        end
    end
```

---

<sup>1</sup> Note that NaN and Inf are defined. Example `A = Inf`

<sup>2</sup> They are slow in matlab and can sometime be avoided by vectorizing your programs.

```

imagesc(ima)      % imshow is buggy and sometimes does not
                  % work properly. In that case use imshow
                  % then imagesc or imagesc directly.

```

## 5.5 while loops<sup>3</sup>

```

while (expression)
    Action
end

```

## 5.6 Functions

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 or from the command line.

The syntax of a function is as follows:

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

### Tips:

```

nargin:      gives the number of input arguments
                allow default parameters
                allow variable number of arguments

```

**% This is the symbol for comments**

As an example, here is a basic function that reads in an image, displays 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.

```

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
    imshow(ima);                % 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!

### Try

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

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

## 6. 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:

---

<sup>3</sup> They are slow in matlab and can sometime be avoided by vectorizing your programs.

```
>>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`.

## 7. Function/Command Duality

### Command line

```
load August17.dat
save January25.dat
cd H:\matlab\
```

### In function or script

```
load('August17.dat')
save('January25.dat');
cd('H:\matlab')
```

## 8. General Tips

### 8.1 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!**

### 8.2Preallocation:

```

r = zeros(32,1);
for n = 1 : 32
    r(n) = n;
end

```

**5 times faster!**

### 8.3 Matrices

<code>zeros(m,n)</code>	Creates a mxn matrix filled with zeros
<code>ones(m,n)</code>	Creates a mxn matrix filled with ones
<code>rand(m,n)</code>	Creates a mxn matrix following a uniform distribution in [0,1]
<code>randn(m,n)</code>	Creates a mxn matrix following a normal Gaussian distribution (mu = 0, sigma = 1)

## 9. Full exercises to attempt:

### Exercise 1: Image inversion:

Example program that reads in an image and invert it.

### Exercise 2: Fourier Transform:

Example program that takes the Fourier transform and display it.

Same for the inverse Fourier Transform.

These programs must be downloaded from the Image Processing module on webct:

<http://webct.eps.hw.ac.uk>

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.

### 9.1 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)

% First convert ima to double
ima = double(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);
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('kids.tif');
>> ima_out = invert(ima);

```

*Note in this example, the use of ndims to find out the number of dimensions of the input array.*

### Additional Exercise:

Could you alter this script to invert each colour and generate a inverted RGB image?

Try it in your spare time...

## 9.2 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(fftn(single((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 display: Do it
```

### Try using this function with various images.

```
>> ima = imread('peppers.png');
>> ima = 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. These functions, as always in matlab work directly on arrays.

### Additional Exercise:

#### 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.

Try

```
>> ima = imread('circuit.tif');
>> ima1 = fft2d(ima);
>> ima2 = fft2d(imrotate(ima,30));
```

## 9.2.1 Inverse Fourier Transform

This function will perform the inverse Fourier Transform. Try to take first the Fourier Transform and then the inverse Fourier Transform to return to the original image.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% 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 display: Do it
```

## 9.2.2 Generic Fourier Transform Function

This is simply to illustrate how matlab functions can be called from other functions. This function can perform either the Fourier Transform or the inverse, depending on the setting of the direction argument.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% 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