Computer Vision 1: MATLAB Tutorial (Optional)

University of Amsterdam

Thursday 5th September, 2019

This tutorial aims to make you familiar with the programming environment that will be used throughout the course. If you have experience with MATLAB and feel comfortable, you can skip the exercises; otherwise, we strongly suggest that you complete them all, as they are meant for getting hands-on experience. **You do not need to submit anything.**

1 Introduction

MATLAB is an interactive system for performing numerical computations. It stands for MATrix LABoratory and the software is built up around vectors and matrices, which makes it extremely useful for linear algebra and image processing tasks. It is also fast in a way that you do not need to compile your code, which enables you to perform computationally intensive tasks faster than with traditional programming languages.

When you start MATLAB, you will see 3 main panels as shown in below Figure 1. From left to right; **Current Folder**, where the files in the current working directory are displayed, **Command Window**, where you write your commands, and **Workspace**, where your data are shown. As you work in MATLAB, you issue commands that create variables and call functions by using the *Command Window*, so that variables are added to the *Workspace* results are displayed in the *Command Window*.

To view the online documentation and get help, click on the **?** icon from the **Home** panel. It is good to note it is not possible to cover MATLAB's every aspect in this tutorial. As a result, you need to discover it by yourself most of the times. Throughout this process, your best friend will be *help*, which displays information on a specific function and *doc*, which opens the online version of the help manual. Now, go ahead and type in the following commands to see how *help* and *doc* works:

```
> help sin
> doc sin
```

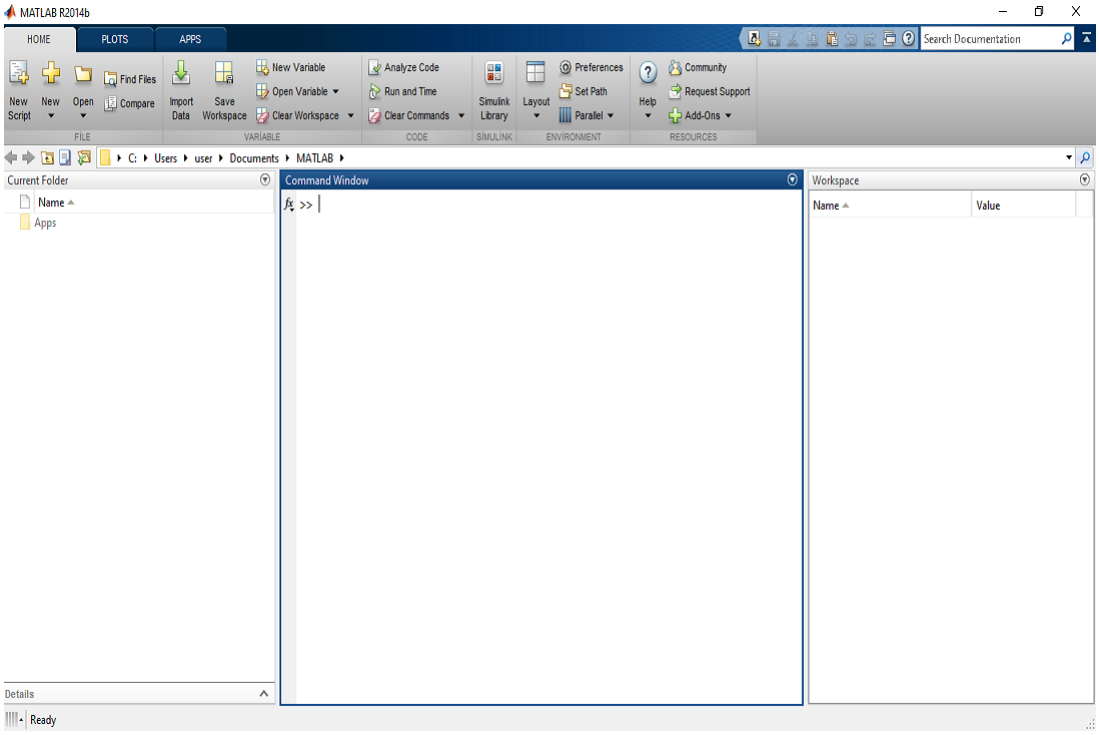

Figure 1: MATLAB User Interface

1.1 Scalar Variables

Type in the following variables and obverse the outputs and the workspace.

(a) s1 = 5; % this is how you write comments

(b) s2 = 5 % without the semicolon, the variable will be printed

(c) $s3 = 6.02 \times 10^{23}$ % Avogadro's number

(d) $s4 = ln(e^4)$ (Hint: Use *doc* to find out how.)

(e) s5 = $\sqrt{s4}$

(f) s6 = s2 / s5

1.2 Vector Variables

Type in the following variables, obverse the outputs and the workspace, and try to understand each case.

(a) v1 = [1, 2, 3, 4]

(b) v2 = [1; 2; 3; 4] % use size(v1) and size(v2) to see the difference

(c) v3 = v2' % transpose

(d) v4 = 0:2:10

(e) v5 = 10:-2:0

(f) v6 = 'I am a string'

1.3 Matrix Variables

(a) Create a 10 x 10 matrix filled with 3s by using the built in function *ones* or *zeros*. (Hint: Type in *help ones* or *help zeros*)

(b) m2 = 5-by-5 identity matrix. (Hint: Check *doc* to use *eye*)

(c) Create the following matrix by using *diag* function:

$$\text{m3} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 33 & 0 \\ 0 & 0 & 0 & 0 & 333 \end{bmatrix}$$

(d) m4 = [10, 20; 30, 40]
m5 = [2, 4; 3, 5]

(e) m6 = m4 / m5 % matrix division

(f) m7 = m4 ./ m5 % element-wise division

(g) m8 = $|m7|$, where $|\cdot|$ denotes the determinant operator. (Hint: use *doc* or try to figure out by yourself this time :-))

(h) m9 = m7' % transpose

(i) m10 = m9(:)

1.4 Indexing & Assignments

Given the matrix A = $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ and B = $\begin{bmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \\ 70 & 80 & 90 \end{bmatrix}$, type in the following expressions and obverse the outputs.

(a) i1 = [A, B; B, A]

(b) i2 = A(1:2, 1)

(c) i3 = A(:, 1)

(d) A(1, :) = B(2, :)

(e) A(2, 1) = 33

(f) B(5, 5) = 55

(g) i4 = B(1:2:5)

(h) B(1:3, :) = B(1:3, :) + 100

(i) i5 = B(end)

1.5 Vector and matrix operations

MATLAB is very well-known for efficient vector and matrix operations. Instead of running a for loop, a specific operator can be used by placing a period " **.** " in front of the traditional " * " or " / " or " ^ ". This operator is much faster than a for loop and will help make your code faster.

1.6 Exercises

1. Create a vector of the even whole numbers between 10 and 49.
2. Let $\mathbf{X}$ = [2 3 1 9],
   (a) Add 16 to each element.
   (b) Sum over all the elements in the vector.
   (c) Sort the vector in the ascending order.
   (d) Sum over the elements with odd indices.
3. Let $\mathbf{X}$ = [5; 3; 1; 8] and $\mathbf{Y}$ = [4 1 7 5],
   (a) Raise each element of $\mathbf{X}$ to the power specified by the corresponding element in $\mathbf{Y}$.
   (b) Divide each element in $\mathbf{X}$ by the corresponding element in $\mathbf{Y}$. Use vectorization.
4. Given the vector t = 1:0.1:2.0, write down the MATLAB expressions using vectorization that will compute the following:
   (a) $ln(2 + t + t^2)$
   (b) $e^{1+cos(3t)}$
   (c) cos^2(t) + sin^2(t)
   (d) tan^{-1}(t)
5. Given $\mathbf{x}$ = [3 1 5 7 9 2 6], observe the output of the commands (do it by hand before typing them in).
   (a) x(3)
   (b) x(1:7)
   (c) x(1:end)
   (d) x(1:end-1)

(e) x(6:-2:1)

(f) x([1 6 2 1 1])

6. Given the matrix **A** = [2 4 1; 6 7 2; 3 5 9], provide the commands needed to

   (a) assign the first row of **A** to a vector called **x**

   (b) assign the last 2 rows of **A** to a matrix called **y**

   (c) compute the sum over the columns of **A**

   (d) compute the sum over the rows of **A**

   (e) compute the mean over all the elements of **A**

   (f) compute the standard deviation over all the elements of **A**

7. Given **x** = [6 1 -8], **y** = [1 1 2] and **A** = [-3 0 3; 6 -2 6], determine which of the following statements will correctly execute and provide the result. Try to understand why it fails.

   (a) x + y

   (b) x + A

   (c) x'+ y

   (d) B = A - [x' y']

   (e) B = [x; y']

   (f) B = A + A'

   (g) A + 1

8. Given that **x** = [1 5 2 8 9 0 1] and **y** = [5 2 2 6 0 0 2], execute and try to understand the results of the following commands:

   (a) x < y

   (b) y <= x

   (c) y ~ = x

   (d) x == y

   (e) x | y

   (f) x & y

9. Given that **A** = [2 7 9 7; 3 1 5 6; 8 1 2 5], try to explain the results of the following commands.

   (a) A(:, [1 4])

   (b) A([2 3], [3 1])

   (c) A(:)

% some of the commonly used built-in functions. Use *doc* for details

(i) reshape(A, 2, 6)

(ii) flipud(A)

(iii) fliplr(A)

(iv) size(A)

(v) max(max(A))

2 Plotting

This chapter briefly explains the basic concepts of creating plots in MATLAB environment.

2.1 Plotting in 2D

To create 2D line plots, use MATLAB's built-in *plot* function. There are other built-in plotting functions available as well that may come handy in the future, such as *loglog*. Let us now have a look at an example of plotting the value of the sine function from 0 to 2π. After typing in the following statements, you should see the Figure 2 displayed on the screen. Try it also with different x values, such as x^2, -x and x + π, then observe the changes in the plot.

```
x = 0:pi/100:2*pi; % defines the interval of the function
y = sin(x); % defines the function we want to plot
plot(x, y)
title('2-D Line Plot', 'FontSize', 20) % title of the figure
xlabel('x', 'FontSize', 20) % label for x axis
ylabel('sin(x)', 'FontSize', 20) % label for y axis
```

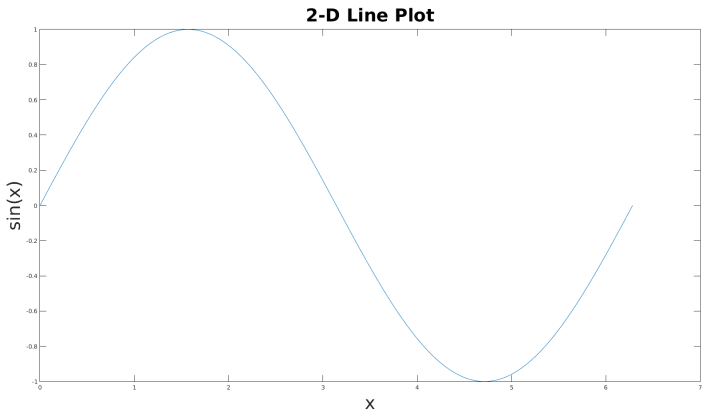

Figure 2: 2D Plot Example

2.2 Plotting in 3D

3D plots can be produced using the functions *surf*, *plot3* or *mesh*. Check also *doc* for more information. Let us have a look at a surface plot example and its output in Figure 3 :

```
% range of x and y values
x = -2:0.2:2;
y = -2:0.2:2;
[X, Y] = meshgrid(x, y); % generates the actual grid of x and y values
Z = X .* exp(-X.^2 - Y.^2); % function we want to plot
surf(X, Y, Z)
title('3-D Surface Plot', 'FontSize', 30)
xlabel('x', 'FontSize', 30)
ylabel('y', 'FontSize', 30)
zlabel('z', 'FontSize', 30)
```

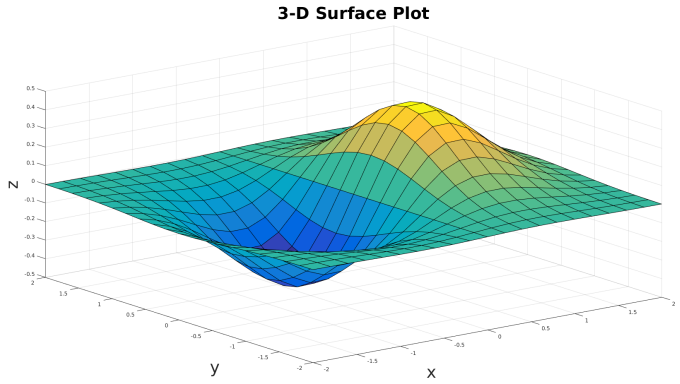

Figure 3: 3D Plot Example

2.3 Exercises

1. Plot the function Y = $5x^3 - 3x^2 + 7x - 2$ for x $\in [-4, 4]$.
2. Given the function Z = $(sin(Y^2 + X) - cos(Y - X^2))$, where X and Y lies between $[0, \pi]$, visualize it by using *mesh* function.

3 Image Processing

MATLAB's image processing toolbox provides a broad set of standardized algorithms and functions for image processing, analysis, visualization, and algorithm development. It is extremely helpful as we can represent images in multidimensional matrices. In this chapter, you will learn the very basics of MATLAB's image processing toolbox, such as how to read, write, display and process images.

3.1 Reading, Displaying & Writing Images

```
I = imread('peppers.png'); % reads the image into marrix I
size(I) % 384 x 512 x 3 = row x columns x color channels (RGB in this case)
figure(1) % creates a figure on the screen
imshow(I) % displays the matrix I as an image
imwrite (I, '/myNewLocation/myimage.png'); % writes image data I to the file
```

3.2 Basic Image Processing

In this part, you will observe some of the basic image processing operations.

3.2.1 Converting RGB Image to Grayscale

```
im1 = imread('peppers.png'); % reads the image into matrix I
im2 = im2double(im1); % converts image to double precision
% converts RGB image to grayscale: 0.2989 * R + 0.5870 * G + 0.1140 * B %
im3 = rgb2gray(im2);
figure(2) % creates a figure on the screen
imshow(im3) % displays the grayscale image
```

3.2.2 Image Histogram & Improving Image Contrast

```
J = imread('pout.tif'); % read a grayscale image into matrix I
figure(3) % creates a figure on the screen
imhist(J) % displays the distribution of image pixel intensities
% Notice how the histogram indicates that the intensity range of the image is %
rather narrow. The range does not cover the potential range of [0, 255], and % is
missing the high and low values that would result in good contrast.
J2 = histeq(J); % spreads the intensity values over the full range of the image
figure(4) % creates a figure on the screen
imshow(J2) % displays contrast enhanced image
figure(5) % creates a figure on the screen
imshow(J) % displays original image
```

3.2.3 Image Scaling

```
P = imread('cameraman.tif'); % read an image into matrix P
figure(6) % creates a figure on the screen
imshow(P) % display the original image
S = imresize(P, 0.5) % scale the original image into its half size
figure(7) % creates a figure on the screen
imshow(S) % display the scaled image
```

4 Functions

Functions are routines that accept input arguments and return output arguments. Each function has its own area of workspace, separated from the global one. The first line of a function starts with the keyword *function*. It gives the function name and order of arguments. Let us have a look at a simple example, which has one output and two input arguments:

```
function [ result ] = mySum(num1, num2)
%Returns the sum of the input arguments.

result = num1 + num2;

end
```

You can increase the number of input and output arguments, or you can even write a function without any input or output argument. Moreover, you can put anything inside a function, such as image processing routines.

4.1 Exercises

1. Write a function that gets a vector as its input and outputs its mean, median and standard deviation as the output.

2. Write a function that gets an image as its input and displays it as grayscale. (Try it on your favorite image to see if it works)

5 Final Exercises

1. Write a function that takes an image as its input, divides the image into 4 pieces and create a new image by changing the order of the pieces of the original image as shown in below Figure 4. In the end, the function should display both the original and the newly created image separately. (Hint: use *subplot*).

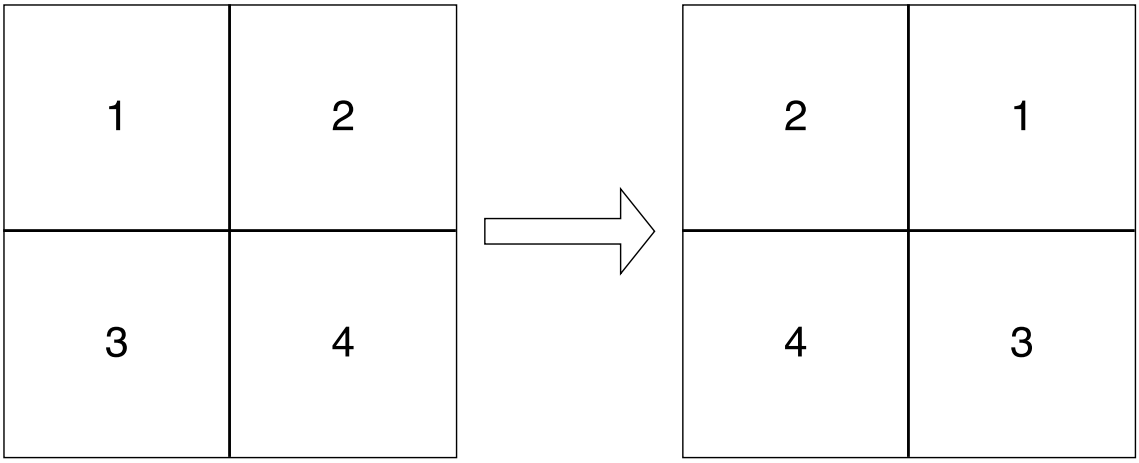

Figure 4: Creating the new image

2. Write a function that takes an RGB image as its input. Divide the image into red, green and blue components as matrices R, G and B. Create a new image by reordering the colors as BGR (Hint: use *cat*). Display the original image and the new one in a 2-by-1 layout (Hint: use *subplot*).

3. Create a 100 x 100 matrix of random integer numbers and name it A (Hint: use *randi*). Move a window of size 3 x 3 over the matrix A and compute the average of the 9 elements within this window. Store the computed averaging results in matrix B (Matrix B is expected to be of size 98 x 98, since you will exclude the first and the last rows and columns of A from your computations).

6 Extra Sources

1. `https://web.stanford.edu/class/ee262/software/getstart.pdf`
2. `https://web.stanford.edu/class/cme001/handouts/matlab.pdf`
3. `https://www.mccormick.northwestern.edu/documents/students/undergraduate/introduction-to-matlab.pdf`
4. `https://www.cs.tau.ac.il/~dcor/Graphics/cg-slides/MATLAB-tutorial.pdf`
5. `http://www.imageprocessingplace.com/downloads_V3/dipum2e_downloads/dipum2e_sample_book_material_downloads/DIPUM2E_Chapter02_Pgs_13-50.pdf`
6. `http://syllabus.cs.manchester.ac.uk/ugt/2017/COMP27112/doc/matlab.pdf`