**Medix Workshop: Tutorial**

|  |  |
| --- | --- |
| Daniela Raicu  [**draicu@cs.depaul.edu**](mailto:draicu@cs.depaul.edu) | Valerie Simonis  [**valerie.simonis@gmail.com**](mailto:valerie.simonis@gmail.com) |

School of Computing

College of Computing and Digital Media

DePaul University, Chicago, IL 60604

Contents

[2 MATLAB and toolboxes 2](#_Toc418245505)

[3 Development Environment: Command Window 3](#_Toc418245506)

[4 Help and Basics 3](#_Toc418245507)

[5 Basic Programming 4](#_Toc418245508)

[6 Basic Image Manipulations 7](#_Toc418245509)

[7 Working image formats 9](#_Toc418245510)

[How to convert between double and uint8 10](#_Toc418245511)

[How to read files 10](#_Toc418245512)

[Loading and saving variables in MATLAB 11](#_Toc418245513)

[How to display an image in MATLAB 11](#_Toc418245514)

[8 Image enhancement 12](#_Toc418245515)

[9 Segmentation 12](#_Toc418245516)

# MATLAB and toolboxes

MATLAB (Matrix Algebra laboratory), distributed by The MathWorks, is a technical computing environment for high-performance numeric computation and visualization.  It integrates numerical analysis, matrix computation, signal processing, and graphics in an easy-to-use environment.

MATLAB also features a family of application-specific solutions called toolboxes. Toolboxes are comprehensive collections of MATLAB functions that extend its environment in order to solve particular classes of problems. The table below lists several toolboxes available for MATLAB:

|  |  |  |
| --- | --- | --- |
| Communications | Image Processing | System Identification |
| Control System | Instrument Control | Wavelet |
| Data Acquisition | Mapping | MATLAB Compiler |
| Database | Neural Network | MATLAB C/C++ Graphics Library |
| Datafeed | Optimization | MATLAB C/C++ Math Library |
| Filter Design | Partial Differential Equation | MATLAB Report Generator |
| Financial | Robust Control | MATLAB Runtime Server |
| Frequency Domain System Identification | Signal Processing | MATLAB Web Server |
| Fuzzy Logic | Statistics | Simulink |
| Higher-Order Spectral Analysis | Spline | Symbolic/Extended Math |

# Development Environment: Command Window

To perform a simple computation type a command and next press the *Enter* or *Return* key. For instance,

>> s = 1 + 2

Note that the results of these computations are saved in variables whose names are chosen by the user. If they will be needed during your current MATLAB session, then you can obtain their values typing their names and pressing the Enter or Return key.

For instance,

>> s

s =

3

Variable name begins with a letter, followed by letters, numbers or underscores. MATLAB recognizes only the first 31 characters of a variable name.

To close MATLAB type exit in the Command Window and next press *Enter* or *Return* key. A second way to close your current MATLAB session is to select Exit in the MATLAB's toolbar. ***All unsaved information residing in the MATLAB Workspace will be lost.***

# Help and Basics

* To get help type “help” (will give you a list of help topics) or “help topic”.
* If you don't know the exact name of the topic or command you are looking for, type "lookfor keyword" (e.g., "lookfor regression")
* When writing a long MATLAB statement that exceeds a single row use “...” to continue statement to next row.
* When using the command line, a ";" at the end means MATLAB will not display the result. If ";" is omitted then MATLAB will display result.
* Use the up-arrow to recall commands without retyping them (and down arrow to go forward in commands)
* The symbol "%" is used in front of a comment.

# Basic Programming

% Adapted from:

% http://www.cns.nyu.edu/~eero and http://www.cs.dartmouth.edu/~farid/teaching/cs88/MATLAB.intro.html

**%% Objects in MATLAB: scalars, vectors and matrices**

% Scalars

N = 5 % a scalar

% Vectors

v = [1 0 0] % a row vector

v = [1;2;3] % a column vector

v = v' % transpose a vector

v = [1:.5:3] % a vector in a specified range:

v = pi\*[-4:4]/4 % [start: stepsize: end]

v = [] % empty vector

% Matrices

m = [1 2 3; 4 5 6] % a matrix: 1ST parameter is ROWS

% 2ND parameter is COLS

m = zeros(2,3) % a matrix of zeros

v = ones(1,3) % a matrix of ones

m = eye(3) % identity matrix

v = rand(3,1) % rand matrix (see also randn)

% Accessing elements

v = [1 2 3]; % access a vector element

v(3) % vector(number)

m = [1 2 3; 4 5 6] % access a matrix element

m(1,3) % matrix(rownumber, columnnumber)

m(2,:) % access a matrix row (2nd row)

m(:,1) % access a matrix column (1st row)

% size of elements

size(m) % size of a matrix

size(m,1) % number rows

size(m,2) % number of columns

m1 = zeros(size(m)) % create a new matrix with size of m

who % list of variables

whos % list/size/type of variables

**%% Simple operations on vectors and matrices**

*% (A) Pointwise (element by element) Operations:*

% addition of vectors/matrices and multiplication by a scalar are done "element by element"

a= [1 2 3 4]; % vector

2 \* a % scalar multiplication

a / 4 % scalar multiplication

b = [5 6 7 8]; % vector

a + b % pointwise vector addition

a - b % pointwise vector addition

a .^ 2 % pointwise vector squaring (note .)

a .\* b % pointwise vector multiply (note .)

a ./ b % pointwise vector multiply (note .)

log( [1 2 3 4] ) % pointwise arithmetic operation

round( [1.5 2; 2.2 3.1] ) % pointwise arithmetic operation

*% (B) Vector Operations (no for-loops needed)*

% Built-in MATLAB functions operate on vectors, if a matrix is given, then the function operates on each % column of the matrix

a = [1 4 6 3] % vector

sum(a) % sum of vector elements

mean(a) % mean of vector elements

var(a) % variance

std(a) % standard deviation

max(a) % maximum

a = [1 2 3; 4 5 6] % matrix

mean(a) % mean of each column

max(a) % max of each column

max(max(a)) % to obtain max of matrix

max(a(:)) % or...

*% (C) Matrix Operations:*

[1 2 3] \* [4 5 6]' % row vector 1x3 times column vector 3x1

% results in single number, also

% known as dot product or inner product

[1 2 3]' \* [4 5 6] % column vector 3x1 times row vector 1x3

% results in 3x3 matrix, also

% known as outer product

a = rand(3,2) % 3x2 matrix

b = rand(2,4) % 2x4 matrix

c = a \* b % 3x4 matrix

a = [1 2; 3 4; 5 6] % 3 x 2 matrix

b = [5 6 7]; % 3 x 1 vector

b \* a % matrix multiply

a' \* b' % matrix multiply

**%% Relations and control statements**

% Example: given a vector v, create a new vector with values equal to v if they are greater than 0, and equal to 0 if they less than or equal to 0.

v = [3 5 -2 5 -1 0] % 1: FOR LOOPS

u = zeros( size(v) ); % initialize

for i = 1:size(v,2)

if( v(i) > 0 )

u(i) = v(i);

end

end

u

v = [3 5 -2 5 -1 0] % 2: NO FOR LOOPS

u2 = zeros( size(v) ); % initialize

ind = find( v>0 ) % index into >0 elements

u2(ind) = v( ind )

# Basic Image Manipulations

**%% Working with Images**

[I,map]=imread('trees.tif'); % read a TIFF image

figure, imshow(I,map) % display it as indexed image

I2=ind2gray(I,map); % convert it to grayscale

figure

imagesc(I2,[0 1]) % scale data to use full colormap

% for values between 0 and 1

colormap('gray') % use gray colormap

axis('image') % make displayed aspect ratio %proportional

% to image dimensions

I=imread('photo.jpg'); % read a JPEG image into 3D %array

figure

imshow(I)

rect=getrect; % select rectangle

I2=imcrop(I,rect); % crop

I2=rgb2gray(I2); % convert cropped image to grayscale

imagesc(I2) % scale data to use full colormap

% between min and max values in I2

colormap('gray')

colorbar % turn on color bar

pixval % display pixel values interactively

truesize % display at resolution of one %screen pixel

% per image pixel

truesize(2\*size(I2)) % display at resolution of two %screen pixels

% per image pixel

I3=imresize(I2,0.5,'bil'); % resize by 50% using bilinear

% interpolation

I3=imrotate(I2,45,'bil','same'); % rotate 45 degrees and crop to

% original size

I3=double(I2); % convert from uint8 to double, to %allow

% math operations

imagesc(I3.^2) % display squared image (pixel-wise)

imagesc(log(I3)) % display log of image

# Working image formats

If an image is stored as a JPEG-image on your disc we first read it into MATLAB. However, in order to start working with an image, for example perform an enhancement operation on the image, we must convert it into a different format. This section explains four common formats.

**Intensity image (gray scale image)**

This is the equivalent to a "gray scale image" and this is the image we will mostly work with in this course. It represents an image as a matrix where every element has a value corresponding to how bright/dark the pixel at the corresponding position should be colored. There are two ways to represent the number that represents the brightness of the pixel: **The double class (or data type).** This assigns a floating number ("a number with decimals") between 0 and 1 to each pixel. The value 0 corresponds to black and the value 1 corresponds to white. **The other class is called uint8** which assigns an integer between 0 and 255 to represent the brightness of a pixel. The value 0 corresponds to black and 255 to white. The class uint8 only requires roughly 1/8 of the storage compared to the class double. On the other hand, many mathematical functions can only be applied to the double class. We will see later how to convert between double and uint8.

**Binary image**

This image format also stores an image as a matrix but can only color a pixel black or white (and nothing in between). It assigns a 0 for black and a 1 for white.

**Indexed image**

This is a practical way of representing color images. (In this course we will mostly work with gray scale images but once you have learned how to work with a gray scale image you will also know the principle how to work with color images.) An indexed image stores an image as two matrices. The first matrix has the same size as the image and one number for each pixel. The second matrix is called the *color map* and its size may be different from the image. The numbers in the first matrix is an instruction of what number to use in the color map matrix.

**RGB image**

This is another format for color images. It represents an image with three matrices of sizes matching the image format. Each matrix corresponds to one of the colors red, green or blue and gives an instruction of how much of each of these colors a certain pixel should use.

**How to convert between different formats**

The following table shows how to convert between the different formats given above. *All these commands require the Image processing tool box!*

|  |
| --- |
| **Image format conversion** (Within the parenthesis you type the name of the image you wish to convert.) |
| **Operation:** | **MATLAB command:** |
| Convert between intensity/indexed/RGB format to binary format. | dither() |
| Convert between intensity format to indexed format. | gray2ind() |
| Convert between indexed format to intensity format. | ind2gray() |
| Convert between indexed format to RGB format. | ind2rgb() |
| Convert a regular matrix to intensity format by scaling. | mat2gray() |
| Convert between RGB format to intensity format. | rgb2gray() |
| Convert between RGB format to indexed format. | rgb2ind() |

The command mat2gray is useful if you have a matrix representing an image but the values representing the gray scale range between, let's say, 0 and 1000. The command mat2gray automatically re scales all entries so that they fall within 0 and 255 (if you use the uint8 class) or 0 and 1 (if you use the double class).

**How to convert between double and uint8**

When you store an image, you should store it as a uint8 image since this requires far less memory than double. When you are processing an image (that is performing mathematical operations on an image) you should convert it into a double. Converting back and forth between these classes is easy.

I=im2double(I); %converts an image named I from uint8 to double.

I=im2uint8(I); %converts an image named I from double to uint8.

**How to read files**

When you encounter an image you want to work with, it is usually in form of a file (for example, if you down load an image from the web, it is usually stored as a JPEG-file). Once we are done processing an image, we may want to write it back to a JPEG-file so that we can, for example, post the processed image on the web. This is done using the imread and imwrite commands. *These commands require the Image processing tool box!*

|  |
| --- |
| **Reading and writing image files** |
| **Operation:** | **MATLAB command:** |
| Read an image.  (Within the parenthesis you type the name of the image file you wish to read. Put the file name within single quotes ' '.) | imread() |
| Write an image to a file. (As the first argument within the parenthesis you type the name of the image you have worked with. As a second argument within the parenthesis you type the name of the file and format that you want to write the image to. Put the file name within single quotes ' '.) | imwrite( , ) |

Make sure to use semi-colon; after these commands, otherwise you will get LOTS OF number scrolling on you screen.

**Loading and saving variables in MATLAB**

This section explains how to load and save variables in MATLAB. Once you have read a file, you probably convert it into an intensity image (a matrix) and work with this matrix. Once you are done you may want to save the matrix representing the image in order to continue to work with this matrix at another time. This is easily done using the commands save and load. Note that save and load are commonly used MATLAB commands, and works independently of what tool boxes that are installed.

|  |
| --- |
| **Loading and saving variables** |
| **Operation:** | **MATLAB command:** |
| Save the variable X . | save X |
| Load the variable X . | load X |

**How to display an image in MATLAB**

Here are a couple of basic

|  |
| --- |
| **Displaying an image given on matrix form** |
| **Operation:** | **MATLAB command:** |
| Display an image represented as the matrix X. | imagesc(X) |
| Adjust the brightness. s is a parameter such that -1<s<0 gives a darker image, 0<s<1 gives a brighter image. | brighten(s) |
| Change the colors to gray. | colormap(gray) |

Sometimes your image may not be displayed in gray scale even though you might have converted it into a gray scale image. You can then use the command colormap(gray) to "force" MATLAB to use a gray scale when displaying an image.

# Image enhancement

* histogram doc imhist
* histogram equalization doc histeq
* contrast stretching

# Segmentation

**%% Creating functions using m-files, ex: threshholding**

% Functions in MATLAB are written in m-files. Create a file called 'thres.m' In this file put the following:

function res = thres( v )

u = zeros( size(v) ); % initialize

ind = find( v>0 ) % index into >0 elements

u(ind) = v( ind )

% Use

v = [3 5 -2 5 -1 0]

thres( v ) % call from command line

I = imread('ClosingHline.jpg');

J = im2bw(I);

imshow(J);

r = regionprops(J);

hold on;

rectangle('position', r(1).BoundingBox, 'FaceColor', 'r')

scatter(r(1).Centroid(1), r(1).Centroid(2))