Matlab Tutorial

Prepared by Xiaojun Qi

1. What Is Matlab?

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Math and computation
- Algorithm development
- Data acquisition
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

Specifically, MATLAB is a matrix programming system with many mathematical methods implemented. It also has many toolboxes such as statistics, image processing, signal processing, wavelet, and so on. The MathWorks web site (http://www.mathworks.com/) has useful information including books describing the basic system and toolboxes. The system supports procedural programming and has some object-oriented programming capabilities.

2. How to Get Help From Matlab?

Matlab has a HELP facility that lists and describes all of Matlab's functions, operations, and commands. To obtain information on any of the Matlab command, one need only type help followed by the name of command. Another way is to go to menu bar "help" and select Matlab help. Follow the link "Getting Started" to read the introduction to Matlab.

3. Things to Know

1) Data Type

The language is not typed and the default data type is double precision floating point (This data type takes eight bytes of information). This may be significant for large data sets such as images.

Matlab does not require any type declarations or dimension statements. When Matlab encounters a new variable name, it automatically creates the variable and allocates the appropriate amount of storage. If the variable already exists, Matlab changes its contents and, if necessary, allocates new storage.

2) Basic Data Structure

A basic data structure is the array. MATLAB has many useful functions to process arrays. Multidimensional arrays are supported. A 1-D array may be referred to as a vector. A 2-D array is referred to as a matrix. The terms array and matrix are sometimes used interchangeably. There are built in functions for performing standard matrix operations as described in linear algebra topics. An image would be a 2-D array or matrix in this notation. The matrix operations are often the most efficient ways to implement algorithms since they have been optimized.

The system has another data structure called cell arrays where the elements are cells. A cell can hold other arrays of any size and type. It is a flexible and useful data structure.

Also in the course of executing a program, several copies of the data structure may be created (depending upon the care of the programmer) by a simple assignment statement.

3) Internal Storage
All numbers are stored internally using the long format specified by the IEEE floating-point standard. Floating-point numbers have a finite precision of roughly 16 significant decimal digits and a finite range of roughly 10^{-308} to 10^{308}.

4) Basic Help
To obtain a list of the elementary mathematical functions, type:
help elfun
To obtain a list of more advanced mathematical functions, type:
help specfun
To obtain a list of the elementary matrix functions, type:
help elmat

5) Suppressing Output
Type a statement and press Enter, MATLAB automatically displays the results on screen. This feature is useful for the debugging purpose. However, if you end the line with a **semicolon**, MATLAB performs the computation but does not display any output. This is particularly useful when you generate large matrices.

6) Entering a Long Command Line
If a statement does not fit on one line, use three periods, **...**, followed by Enter to indicate that the statement continues on the next line. For example,

```
s = 1 -1/2 + 1/3 -1/4 + 1/5 - 1/6 + 1/7 ...
  - 1/8 + 1/9 - 1/10 + 1/11 - 1/12;
```

7) Scripts and Functions
They are files that contain code in the Matlab language. They are called M-Files.

- Scripts: Do not accept input arguments or return output arguments. They operate on data in the workspace.
- Functions: Accept input arguments and/or return output arguments. Internal variables are local to the function.

4. Things to Master

1) Four ways to enter matrices into Matlab

- Enter an explicit list of elements;
- Load matrices from external data files;
- Generate matrices using built-in functions;
- Create matrices with your own functions in M-files.

2) The colon operator (i.e., :)

3) The concatenation operator (i.e., []) to build a bigger matrix

4) The building blocks of expressions: variables, numbers, operators, and functions.

5) The logical subscripting: The logical vectors created from logical and relational operations can be used to reference subarrays. Suppose X is an ordinary matrix and L is a matrix of the same size that is the result of some logical operation. Then X(L) specifies the elements of X where the elements of L are nonzero.

6) The find function: Find the indices of array elements that meet a given logical condition.

7) The saveas function: Save a figure to a desired output format.

8) The print function: Print a Matlab figure directly on a printer connected to your computer or export the figure to one of the standard graphic file format supported by Matlab.

9) The six control statements: if, switch and case, for, while, continue, and break.

5. Short Examples

Example 1: Create a script file

The Matlab system can be used as a calculator but a more powerful use is as a programming language. One can save commands in script file and execute them by calling the script file. An editor such as the MATLAB M-file editor is used to create script files. The name of the file must end with the extension 'm'. For example the commands

```
>> A = [1 2; 2 2];
>> B = [3 4; 5 5];
>> C = A + B;
```

can be put in a file **Test.m** and then executed by using the statement

Test

It will execute the commands in the M-file **Test.m**.

Example 2: Create a function file

Matlab allows one to add to its functionality by adding user-defined functions. The functions are usually placed in M-files so they can be easily edited and used as needed with other functions. The following function **Crossv** would be edited and put in the files **Crossv.m (Note: Function name is the same as the file name.)**

```
function vans = Crossv(ra, rb)
% r=[x y z] a row vector
% ra and rb must be row vectors with three elements
% cross product of ra and rb
vx = ra(2) * rb(3) - ra(3) * rb(2);
vy = ra(3) * rb(1) - ra(1) * rb(3);
```

```
vz = ra(1) * rb(2) - ra(2) * rb(1) ;
vans =[vx vy vz];
```

In the command window, one can issue the following commands to utilize the function.

```
>> u =[1 2 2];
>> v = [3 2 2];
>> w = Crossv(u, v);
>> w
w =
  0 4 –4
```

Example 3: Call by reference or call by value?
Arguments to functions are passed by reference (i.e., pointers) in most cases to conserve memory. If an argument is modified by the function, then the argument is passed by value, which means a copy of the argument is made.

For example the file Change.m contains the following:

```
function B = Change(A)
A = A + 10;
B=A;
```

This function modifies A, so A is copied into the function. The modified copied A is assigned to B and B is returned. Here, one has consumed twice as much memory in this case as required for A alone. If A is a very large data structure, this may affect the program performance.

6. Coordinate Systems in MATLAB Image Processing Toolbox

MATLAB has a toolbox for image processing. The system has a number of supplied functions that facilitate image processing. The system is also readily programmable which enables the user's implementation of additional image processing functionality.

Three coordinate systems are important in MATLAB image processing. These are

- Cartesian Coordinates
- Matrix Coordinates
- Pixel Coordinates

Cartesian Coordinates:
The Cartesian coordinate system is commonly used in mathematics. The origin is the point (0, 0). In this system, the first component x increases to the right, while the second component y increases upward. The origin (0, 0) is at the lower-left corner. It is the most common coordinate system for manipulating mathematical entities, such as parametric surfaces.

Matrix Coordinates:
The matrix coordinate system is commonly used for matrices and displays. The origin is at the upper left corner and usually starts with (1, 1). The matrix coordinate system, also called the row-column coordinate system, is MATLAB's coordinate system for matrices. MATLAB uses

this system for matrix subscript notation. In this system, the first component (row) increases downward, while the second component (column) increases to the right.

Pixel Coordinates:
The pixel coordinate system is the most common coordinate system for digital image processing in MATLAB. For pixel coordinates, the first component x increases to the right, while the second component y increases downward. The origin is at the upper left corner. These coordinates are typically integers. Note that if we compare it to the row-column system, then row corresponds to y and column corresponds to x. If we compare it to the Cartesian system, then the y axis is reversed.

MATLAB uses the matrix coordinate system for image matrix manipulation and the pixel coordinate system for everything else. It is helpful to use the indices r, c or x, y to distinguish these uses.

7. Images in MATLAB

MATLAB has several different image formats. These are

- Intensity images
- Binary images
- RGB (or truecolor) images
- Indexed images

Intensity images are stored as a simple matrix. The pixel values are of type double and range from 0.0 to 1.0. The intensity 0.0 is black and the intensity 1.0 is white. The image class **uint8** stores data as a byte with the range of numbers being from 0 to 255. This data format is convenient for storing large images.

A binary image contains only two gray-levels. The value 0 is black and 1 is white. The values are stored as double precision numbers. Binary images can represent logical quantities. This allows one to use logical operators such as >, = = to designate regions of the image.

RGB images are used for color images and are created from an m-by-n-by-3 data array containing valid RGB triples. R, G, and B represent the color components of red, green, and blue respectively. The three component values of each pixel are combined to create the pixel's color. Each component is a double type scaled between 0 and 1. An example would be A(x, y, :)=(0.1238, 0.9874, 0.2543) where the green component is the largest. Note that [0, 0, 0] is black, [1, 1, 1] is white, [1, 0, 0] is pure red, and [0.5, 0.5, 0.5] is gray.

An indexed image is another way to represent a color image. The value of the pixel is an index into a colormap matrix that gives the color for the pixel. The colormap matrix is a standard colormap with any m-by-3 array containing valid RGB data. For example, if an image data array A(x,y)=18, then colormap(18, :) gives the color components for the pixel. The default colormap has 64 indices. The value for index 18 is [0, 0.6250, 1.0] that are the RGB components of the pixel color. This implied that the image data is required to be in the range 1 and the length of the colormap.

8. Converting Between Image Types

MATLAB has functions for converting between the different image types.

1) The function **mat2gray**

It converts a matrix to an intensity image. The call is of the form

I = mat2gray(A, [amin amix])

converts the matrix A to the intensity image I. The variables A and I are of type double. The returned matrix I contains values in the range 0.0 (black) to 1.0 (full intensity or white). The quantities amin and amax are the values in A that correspond to 0.0 and 1.0 in I. The call **I = mat2gray(A)** sets the values of amin and amax to the minimum and maximum values in A.

2) The function **gray2ind**

It converts an intensity image to an indexed image. The function **gray2ind** scales and rounds an intensity image to produce an equivalent indexed image. The function call **[X, MAP] = gray2ind(I, N)** converts the intensity image I to an indexed image X with colormap gray(N).

If N is omitted, it defaults to 64. The input image I can be of class uint8 or double. The class of the output image X is uint8 if the colormap length is less than or equal to 256. If the colormap length is greater than 256, then X is of class double.

3) The function **ind2gray**

It converts an indexed image to an intensity image. The function call **I = ind2gray(X, MAP)** converts the image X with colormap MAP to an intensity image I. The function ind2gray removes the hue and saturation information while retaining the luminance. The variable X can be of class uint8 or double. I is of class double.

4) The function **rgb2gray**

It converts an RGB image or colormap to a grayscale image. The function **rgb2gray** converts RGB images to a grayscale image by eliminating the hue and saturation information while retaining the luminance. The function call **I = rgb2gray(RGB)** converts the color image RGB to the grayscale intensity image I. The function call **NEWMAP = rgb2gray(MAP)** returns a grayscale colormap equivalent to MAP. If the input is an RGB image, it can be of class uint8 or double; the output image I is of the same class as the input image. If the input is a colormap, the input and output colormaps are both of class double.

5) The function **ind2rgb**

It converts and indexed image to an RGB image. The function call **RGB = ind2rgb(X, MAP)** converts the matrix X and corresponding colormap MAP to RGB (truecolor) format. The variable X can be of class uint8 or double. RGB is an M-by-N-by-3 array of class double.

6) The function **uint8**

It converts to an unsigned 8-bit integer. The function call **I = uint8(X)** converts the X into an unsigned 8-bit integer. X can be any numeric object (such as a double). The elements of an uint8 range from 0 to 255. The result for any element of X outside this range is not defined (and may vary from platform to platform). If X is already an unsigned 8-bit integer, uint8 has no effect. The uint8 class is primarily meant to be used to store integer values. Hence most operations that manipulate arrays without changing their elements are defined (examples are

reshape, size, subscripted assignment and subscripted reference). No math operations are defined for the uint8 since such operations are ambiguous on the boundary of the set (for example they could wrap or truncate there).

9. Displaying Images

1) The function **imshow**

- The function call **imshow(I, n)** displays the intensity image I with n discrete levels of gray. If you omit n, **imshow(I)** uses 256 gray levels on 24-bit displays, or 64 gray levels on other systems.
- The function call **imshow(I, [low high])** displays image I as a grayscale intensity image and specifies the data range for I. The value low (and any value less than low) displays as black, the high (and any value greater than high) displays as white, and values in between display as intermediate shades of gray. If you use an empty matrix ([ ]) for [low high], imshow uses [min(I(:)) max(I(:))]; the minimum value in I displays as black, and the maximum value displays as white.
- The function call **imshow(BW)** displays the binary image BW. Values of 0 are displayed as black, and values of 1 are displayed as white.
- The function call **imshow(X, MAP)** displays the indexed image X with the colormap MAP.
- The function call **imshow(RGB)** displays the true color image RGB. The input image can be of class uint8 or double.

2) The function **image**

MATLAB displays an image by mapping each element in a matrix to an entry in the colormap of the figure. The function call image(C) displays matrix C as an image. Each element of C specifies the color of a display element in the output. This corresponds to a rectilinear patch in the displayed image. C can be a matrix of dimension M-by-N or M-by-N-by-3, and can contain double or uint8 data. When C is a 2D M-by-N matrix, the elements of C are used as indices into the current colormap (This current colormap can be set up by using colormap command) to determine the color. For matrices containing doubles, color intensities are on the range [0.0, 1.0]. For uint8 matrices, color intensities are on the range [0, 255].

3) The function **imagesc**

It scales and displays an image. It is similar to the function **image** except the data are scaled to use the full colormap.

Example 4: Create a script file called DisplayImage to display mandrill

```
load mandrill
% Other sample images coming along with the Matlab installation
%load clown
%load detail
%load durer
%load flujet
%load gatlin
[r, c] = size(X) ;
figure(1) ;
```

```
colormap(map) ;
imagesc(X) ;
axis('image') ;
axis('off') ;
title('Mandril Image') ;
```

Example 5: Create a script file called FullDisplay to display mandrill in the whole figure. This is an advanced example.

```
load mandrill
[r, c] = size(X) ;
figure('Units', 'Pixels', 'Position', [100 100 c r]) ;
image(X) ;
axis image off ;
set(gca, 'Position', [0 0 1 1]) ;
colormap(map)
```

10. Image Import and Export

MATLAB supports the import and export of images in the tagged image file format (TIFF), in the graphics interchange format (GIF), in the hierarchical data format (HDF) and various vendor specific formats. The functions are **gifread, gifwrite, tiffread, tiffwrite, hdfread,** and **hdfwrite**.

The function **imread** reads an image from a file. The function call **A = imread(filename, fmt)** reads the image in filename into **A whose class is of type uint8**. If the file contains a grayscale intensity image, then A is a two-dimensional array. If the file contains a truecolor (RGB) image, then A is a three-dimensional (M-by-N-by-3) array. The variable filename is a string that specifies the name of the graphics file, and fmt is a string that specifies the format of the file. The possible values for fmt include:

- 'bmp' Windows Bitmap (BMP)
- 'hdf' Hierarchical Data Format (HDF)
- 'jpg' or 'jpeg' Joint Photographic Experts Group (JPEG)
- 'pcx' Windows Paintbrush (PCX)
- 'tif' or 'tiff' Tagged Image File Format (TIFF)

…. ….

The function **imwrite** is similar except that it writes an image to a file. The function call **imwrite(A, filename, fmt)** writes the image A to filename. The variable filename is a string that specifies the name of the output file, and fmt is a string that specifies the format of the file. If A is an indexed image then it writes the image and its associated colormap MAP to filename. If A is of class uint8, the actual values are written to the file. If A is of class double, the values are scaled to 8 bit values between 0 and 255 using uint8(A-1).

11. One Programming Example to Use Matlab for Image Processing

1. Data files are stored in the ASCII format with an extension .ascii in the current directory.
2. Display and print images using MATLAB.
3. The source codes are stored in two M-files: TransImage.m and Translate.m.

Example MATLAB program:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PROGRAM NAME: TransImage.m
% PURPOSE:     To translate an image by a specified number of rows and columns
% FUNCTION USED:  Translate
% INPUT EXPECTED: An ASCII format input image filename
% OUTPUT GENERATED: An ASCII format output image
% REMARK: Coordinate system used is matrix coordinate system (row, col).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear;    % clear all variables
clf;       % clear the graph
load('tools_bit.ascii');  % read the ASCII file; Single quotation is used to indicate the file name
img = tools_bit;         % assign to matrix img
clear tools_bit;
disp('  *** Display Input Image  ****');
figure(1);                    % create a figure window
colormap(gray);        % set default colormap to gray
imagesc(img);           % display the input image
axis('image');             % set axis to image size
title('Input Image');
disp('Press any key to continue ...');
pause;                                 % wait till key pressed
no_rows = size(img, 1);       % get the number of rows
no_cols = size(img, 2);        % get the number of columns
disp('   *** Translating image ****');
out = Translate(img, 50, 25, no_rows, no_cols);   % call the translate function
disp('   *** Displaying Output Image ***');
figure(2);                          % create another figure window
colormap(gray);              % set default colormap to gray
imagesc(out);                  % display the output image
title('Output Image');
axis('image');
save trans.ascii out -ascii;       % Save the output image
```

Function Translate:

```
function out_img = Translate(in_img, rows, cols, total_rows, total_cols)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PROGRAM NAME: Translate.m
% FUNCTION NAME: Translate
% PURPOSE: Translate an image by a specified number of rows and columns
% PARAMETERS: input image, number of rows and columns to be translated by, total number
% of rows and columns
% CALLING CONVENTION: Calling arrays and integers
% OUTPUT RETURNED: An array containing the output translated image.
```

```
% REMARKS: Coordinate system used is matrix system (row, col).
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for r = 1 : total_rows
   for c = 1 : total_cols
     if (c+cols <= total_cols) & (r+rows <= total_rows)
          out_img(r+rows, c+cols) = in_img(r, c);
     end
   end
end
```