WHAT IS MATLAB?
Matlab was originally developed to be a “matrix laboratory,’’ and is used as a powerful software package for interactive analysis and visualization via numerical computations. It is oriented to matrix analysis, hence the name, but it is quite capable of effectively dealing with scalars as well. This primer is a basic overview of the Matlab program and the image processing toolbox.

THE BARE ESSENTIALS OF MATLAB

Starting Matlab (4.2 or higher for Windows)
Use the mouse to “double left click” on the Matlab group icon. Double left click on the new icon labeled “MATLAB” to start the Matlab program. A Matlab window will now appear with the prompt >>. You can now key in various MATLAB commands as described below.

Quitting Matlab
The first thing you should know is how to get out. Enter either quit or exit to end the program and return to the Windows Program Manager.

Executing Commands
In this mode Matlab will try to execute anything typed in before you press enter, which might be data entry or a command. The results of the command will be displayed on the screen by default UNLESS you end the line with a semicolon “;”. If you wish to repeat a previous entry use the up arrow key as many times as necessary to bring the command to the display. When the command is displayed you can edit it using right and left arrows, the delete, insert key and the characters. When you press “enter” the revised command will be executed if possible. Note that upper and lower case characters are distinct in the default mode.

Printing Graphs
To print graphs from the graph window click on that window’s “file” menu and then click on the option “print”. Follow further instructions given by the window. Similarly, you can print a portion of your text screen by dragging the mouse over the text and clicking on “file” and on “print”.

On-line help
To obtain on-line help, type help followed by the command for which information is sought. For example, type: help plot to obtain help on the plot command. For information about the differences between versions 1 and 2, type the following command at the MATLAB prompt:

helpwin images/Readme

For a list of all the functions in the Image Processing Toolbox, type this command:

helpwin images/Contents

To determine if the image processing toolbox is installed on your system, type ver at the MATLAB command prompt.
MATRICES

Entering a Matrix
When entering a matrix by explicit list, the elements are entered in square brackets, with spaces or commas separating elements in a given row and semicolons or carriage-returns separating rows. For example:

\[
A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad B = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \quad D = \begin{bmatrix} 1 \\ 2 \\ 3 & 4 \end{bmatrix}
\]

To create a vector in which the elements vary incrementally, use:

\[
a = \text{starting value} \ : \ \text{increment} \ : \ \text{final value}
\]

or

\[
a = \text{linspace} \ (\text{starting value}, \ \text{final value}, \ \text{number of terms})
\]

gives

\[
a = [0 \ 0.5 \ 1.0 \ 1.5 \ 2.0]
\]

Matrix Operations
Operations can be performed on matrices with relative ease. The following examples describe some of the basic Matlab matrix operations.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>A = \begin{bmatrix} 1 &amp; 2 \ 3 &amp; 4 \end{bmatrix}</th>
<th>B = \begin{bmatrix} -1 \ 3 \ 5 \end{bmatrix}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>(A + B)</td>
<td>\begin{bmatrix} 0 &amp; 5 \ 8 &amp; 11 \end{bmatrix}</td>
<td></td>
</tr>
<tr>
<td>Subtraction</td>
<td>(A - B)</td>
<td>\begin{bmatrix} 2 &amp; -1 \ -2 &amp; -3 \end{bmatrix}</td>
<td></td>
</tr>
<tr>
<td>Multiplication</td>
<td>(A \cdot B)</td>
<td>\begin{bmatrix} 9 &amp; 17 \ 17 &amp; 37 \end{bmatrix}</td>
<td></td>
</tr>
<tr>
<td>Forward Division</td>
<td>(A / B = A \cdot B^{-1})</td>
<td>\begin{bmatrix} 0.1364 &amp; 0.2273 \ -0.0455 &amp; 0.5909 \end{bmatrix}</td>
<td></td>
</tr>
<tr>
<td>Backward Division</td>
<td>(A \backslash B = A^{-1} \cdot B)</td>
<td>\begin{bmatrix} 7 &amp; 1 \ -4 &amp; 1 \end{bmatrix}</td>
<td></td>
</tr>
<tr>
<td>Transpose</td>
<td>(A')</td>
<td>\begin{bmatrix} 1 &amp; 3 \ 2 &amp; 4 \end{bmatrix}</td>
<td></td>
</tr>
</tbody>
</table>
WHAT IS THE IMAGE PROCESSING TOOLBOX?

A collection of functions built on MatLab’s computing environment. The functions support

- import and export images,
- image display,
- geometric operations,
- neighborhood and block operations,
- filtering and filter design,
- analysis and enhancement,
- binary image operations,
- region of interest operations, and
- image transform.

SUMMARY OF CATEGORIES OF FUNCTIONS:

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image Display</td>
<td>Display single image, multiple images, or display images as a texture-mapped surface.</td>
</tr>
<tr>
<td>Image Conversion</td>
<td>Change any type of image (indexed, intensity, or RGB) into another type, or create an image deck.</td>
</tr>
<tr>
<td>Colormap Operations</td>
<td>Modify colormaps for optimal display results.</td>
</tr>
<tr>
<td>Filtering and Related Operations</td>
<td>Create and apply filters using a variety of filter types and techniques.</td>
</tr>
<tr>
<td>Geometric, Enhancement, and</td>
<td>Perform standard image processing operations, such as resizing, image</td>
</tr>
<tr>
<td>Analysis Operations</td>
<td>statistics, edge extraction, and contours.</td>
</tr>
<tr>
<td>Image Transforms</td>
<td>Perform Fourier transforms, discrete cosine transforms, and Radon transforms.</td>
</tr>
<tr>
<td>File I/O Operations</td>
<td>Read images from disk files, or write them to disk files.</td>
</tr>
</tbody>
</table>

UNDERSTANDING IMAGES IN MATLAB:

Basic image data type in Matlab is a rectangular matrix, an order set of real picture elements (or pixels). To select a single pixel from an image, use normal matrix subscripts as follows:

\[ X(n, m) \]

The above returns the value of the pixel at row \( n \) and column \( m \) of the image \( X \). (Note that Matlab is case sensitive, \( x \neq X \)). To display the size of an image \( X \),

\[ \text{size}(X) \]
**Data Type**
By default, MATLAB stores most data in arrays of "class double" (64 bit double precision floating point numbers). For image processing, this data processing is not always ideal. For example 1000x1000 or 1 million pixels would require about 8MB of memory. To reduce the memory requirements, MATLAB supports storing image data in arrays of "class unit8" (8-bit unsigned integer). This requires only 1/8 as much memory as data in double arrays.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>Double-precision, floating-point numbers in the approximate range (-10^{308}) to (10^{308}) (8 bytes per element).</td>
</tr>
<tr>
<td>uint8</td>
<td>Unsigned 8-bit integers in the range ([0, 255]) (1 byte per element).</td>
</tr>
<tr>
<td>uint16</td>
<td>Unsigned 16-bit integers in the range ([0, 65535]) (2 bytes per element).</td>
</tr>
<tr>
<td>uint32</td>
<td>Unsigned 32-bit integers in the range ([0, 4294967295]) (4 bytes per element).</td>
</tr>
<tr>
<td>int8</td>
<td>Signed 8-bit integers in the range ([-128, 127]) (1 byte per element).</td>
</tr>
<tr>
<td>int16</td>
<td>Signed 16-bit integers in the range ([-32768, 32767]) (2 bytes per element).</td>
</tr>
<tr>
<td>int32</td>
<td>Signed 32-bit integers in the range ([-2147483648, 2147483647]) (4 bytes per element).</td>
</tr>
<tr>
<td>single</td>
<td>Single-precision floating-point numbers with values in the approximate range (-10^{38}) to (10^{38}) (4 bytes per element).</td>
</tr>
<tr>
<td>char</td>
<td>Characters (2 bytes per element).</td>
</tr>
<tr>
<td>logical</td>
<td>Values are 0 or 1 (1 byte per element).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Converts Input to:</th>
<th>Valid Input Image Data Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>im2uint8</td>
<td>uint8</td>
<td>logical, uint8, uint16, and double</td>
</tr>
<tr>
<td>im2uint16</td>
<td>uint16</td>
<td>logical, uint8, uint16, and double</td>
</tr>
<tr>
<td>mat2gray</td>
<td>double (in range ([0, 1]))</td>
<td>double</td>
</tr>
<tr>
<td>im2double</td>
<td>double</td>
<td>logical, uint8, uint16, and double</td>
</tr>
<tr>
<td>im2bw</td>
<td>logical</td>
<td>uint8, uint16, and double</td>
</tr>
</tbody>
</table>

**Image Types**
Matlab supports four basic types of images:
- Indexed images (typical color images)
- Intensity images (gray-scale images)
- Binary images (black and white)
- RGB images (standard way of representing color data)
Indexed Images
Typical color images require two matrices, a colormap and an image matrix:

The colormap is an ordered set of values that represent the colors in the image. The size of the colormap is nx3 for an image containing n colors. Each row of the colormap matrix is a 1-by-3 red, green, blue (RGB) color vector

\[ \text{color} = [R \ G \ B] \]

that specifies the intensity of the red, green, and blue components of the color. R, G, and B are real scalars that range from 0.0 (black) to 1.0 (full intensity).

For each image pixel, the image matrix contains a corresponding index in the colormap. When Matlab displays an indexed image, it uses the values in the image matrix to look up the desired color in the colormap.

Example 1

load trees
 imshow (X, map)
 X(86, 198)

Example 2

Outside the Matlab, indexed images with n colors often contain values from 0 to n-1. The values are indices into a colormap with 0 as its first index. Since Matlab matrices start with index 1, each value in the image must be incremented to create an image that can be manipulated with the toolbox functions.

\[
X = \begin{bmatrix}
1 & 1 & 1 & 2 & 3 \\
1 & 1 & 2 & 3 & 1 \\
2 & 2 & 3 & 2 & 2 \\
1 & 3 & 2 & 1 & 1 \\
\end{bmatrix}; \\
\text{map} = [.4 \ .4 \ .4; .6 \ .1; 1 \ 0 \ 0]
\]

imshow (X, map)
**Intensity Images**

Store an intensity image as a single matrix, which contains double precision values ranging from 0.0 to 1.0, with each element of the matrix corresponding to an image pixel. Try the following example statements:

```matlab
load trees
I = ind2gray(X, map); % convert indexed image to gray scale.
imshow(I, 64)
```

To convert an intensity image to gray scale,

```matlab
ind2gray (X+1, gray(256)) % zero-based colormap.
ind2gray (X, gray(256)) % colormap beginning with index 1
ind2gray (X+1, map) or ind2gray (X, ,map) % if image comes with its own colormap, say map
```

**Binary Images**

A binary (black and white) image is a special kind of intensity image. Binary contain only two levels, 0(black) or 1(white). Try the following statements:

```matlab
load trees
I = ind2gray(X, map);
BW = edge(I); % command edge detects change in intensity
imshow(~BW, 2) % ~ invert black to white and vice verse.
```

The following command creates a 5x5 binary image:

```matlab
BW = [0 0 1 0 0
      0 1 1 1 0
      1 1 1 1 1
      0 1 1 1 0
      0 0 1 0 0 ];
imshow(BW, 2)
```

**RGB Images**

In Matlab, the red, green, and blue components of an RGB image reside in three separate intensity matrices, each having the same row and column dimensions as the original RGB image. The intensities of corresponding pixels from each matrix combine to create the actual pixelcolor at a given location. Try the following statements:

```matlab
load trees
[R,G,B] = ind2rgb(X, map); % or load trees
RGB= ind2rgb(X, map);
imshow([R,G,B])
```
Generate an intensity image \( I(x,y) \) using the NTSC standard for luminance
\[
I = 0.299 \times R + 0.587 \times G + 0.114 \times B.
\]

The following statements create an RGB image on the right:

```matlab
R = [ .4 .4 .4 0 1  
     .4 .4 0 1 .4  
     0 0 1 0 0  
     .4 1 0 .4 .4 ];  
G = [ .4 .4 .4 .6 0  
     .4 .4 .6 0 .4  
     .6 .6 0 .6 .6  
     .4 0 .6 .4 .4 ];  
B = [ .4 .4 .4 1 0  
     .4 .4 1 0 .4  
     1 1 0 1 1  
     .4 0 1 .4 .4 ];

rgb = 
```
```
Changing Image Types

<table>
<thead>
<tr>
<th>Image type</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexed images</td>
<td><code>imshow(X, map)</code></td>
</tr>
<tr>
<td>Intensity images</td>
<td><code>imshow(I, 64)</code></td>
</tr>
<tr>
<td>Binary images</td>
<td><code>imshow(BW, 2); 0(black), 1(white)</code></td>
</tr>
<tr>
<td></td>
<td><code>imshow(~BW, 2); 1(black), 0(white)</code></td>
</tr>
<tr>
<td>RGB images</td>
<td><code>imshow(R, G, B)</code></td>
</tr>
</tbody>
</table>

DISPLAYING IMAGES
**Importing and Exporting Images**

The MATLAB `imread` function can read these graphics file formats:

- `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)
- `xwd`  X Window Dump (XWD)

<table>
<thead>
<tr>
<th>Format Name</th>
<th>Description</th>
<th>Recognized Extensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIFF</td>
<td>Tagged Image File Format</td>
<td>.tif, .tiff</td>
</tr>
<tr>
<td>JPEG</td>
<td>Joint Photographic Experts Group</td>
<td>.jpg, .jpeg</td>
</tr>
<tr>
<td>GIF</td>
<td>Graphics Interchange Format</td>
<td>.gif</td>
</tr>
<tr>
<td>BMP</td>
<td>Windows Bitmap</td>
<td>.bmp</td>
</tr>
<tr>
<td>PNG</td>
<td>Portable Network Graphics</td>
<td>.png</td>
</tr>
<tr>
<td>XWD</td>
<td>X Window Dump</td>
<td>.xwd</td>
</tr>
</tbody>
</table>

† GIF is supported by `imread`, but not by `imwrite`.

To write image data from MATLAB to a file, use the `imwrite` function, which can write the same file formats that `imread` reads.

**Example 3**

```matlab
I=imread('trees.tif');  % import the image saturn.tif
imshow(I)              % display a grayscale intensity image, default [0 255]
imwrite(I, 'test.jpg'); % save the image into a new file test.jpg
```

**Example 4**

```matlab
[X,map]=imread('trees.tif'); % read the trees in tiff format to the memory (matlab workspace) as index image
imshow(X, map)             % once the image is on the MATLAB workspace, it can be shown on screen.
imwrite(X,map,'test.jpg')  % convert the image to jpg format and save it in a new file 'test.jpg.
% The colormap "map" is included in the imwrite command in order to retain the colormap.
imwrite(X,'testgray.jpg')  % convert the image to jpg format and save it in a new file 'testgray.jpg.
% if the colormap "map" is not included in the imwrite command, the image will be gray.
```
[C,map]=imread('test.jpg');
[G,map]=imread('testgray.jpg');
imshow(C, map)
figure, imshow(G, map)
% Images must be saved in the MATLAB workspace (memory) before they can be displayed.
% Note that the result of image(G,map) is the same as imshow(I) where I = imread('trees.tif');

<table>
<thead>
<tr>
<th>Image Format</th>
<th>Matlab commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphics Interchange Format</td>
<td>[X, map]=gifread('img.gif'); gifwrite(X, map ‘filename’)</td>
<td>Read/write img.gif from/to disk and return/save as indexed image matrix/colormap. Matlab version 5.0 or higher is no longer supporting the 'gif' format.</td>
</tr>
<tr>
<td>(GIF)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tagged Image File Format</td>
<td>[r,g,b]=tiffread(‘rgb.tiff’); tiffwrite(X, map, ‘test’)</td>
<td>Read rgb.tiff from disk and return the component matrices r, g, b.</td>
</tr>
<tr>
<td>(TIFF)</td>
<td></td>
<td>Write an indexed image X with a colormap named map to the file ‘test’</td>
</tr>
<tr>
<td>MS Windows format</td>
<td>[X, map]=bmpread(‘img.bmp’); bmpwrite(X, map ‘filename’)</td>
<td>Read/write img.bmp from/to disk and return/save as indexed image matrix/colormap.</td>
</tr>
<tr>
<td>(BMP)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zsoft Paint format</td>
<td>[X, map]=pcxread(‘img.pcx’); pcxwrite(X, map ‘filename’)</td>
<td>Read/write img.pcx from/to disk and return/save as indexed image matrix/colormap.</td>
</tr>
<tr>
<td>(PCX)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Matlab MAT-files</td>
<td>load filename</td>
<td>Load and save Matlab general data files</td>
</tr>
<tr>
<td></td>
<td>save filename</td>
<td></td>
</tr>
</tbody>
</table>

**COORDINATE SYSTEMS**

Result of `imshow(C,map)` or `imshow(C)`

Result of `imshow(G,map)` or `imshow(G)`
**Cartesian Coordinate System** - Original (0, 0) is in the lower left corner. Most of the matlab graphics routines use this system.

**Matrix Coordinate System** – row-column (i, j) coordinate system. The origin (1, 1) is in the upper left corner. Matrix coordinates are integer and range between 1 and the length of the row or column. The image processing toolbox uses this system for direct image matrix sub scripting.

**Pixel Coordinate System** – Similar to Cartesian except the y-components is increasing downward and the origin (1, 1) is in the upper left corner. Most commonly used in image processing.

**GEOMETRIC OPERATIONS**

Supported interpolation methods include:
- 'nearest' (default) - nearest neighbor interpolation.
  - The output is assigned the value of the pixel that the point falls within. No other pixels are considered.
- 'bilinear' bilinear interpolation
  - The output is a weighted average of the pixels in the nearest 2x2 neighborhood.
- 'bicubic' bicubic interpolation
  - The output is a weighted average of the pixels in the nearest 4x4 neighborhood.

**Image Resize**
The MATLAB command `imresize` resizes an image of any type using the specified interpolation method.

`B = IMRESIZE(A,M,METHOD)` returns an image that is M times the size of A. If M is between 0 and 1.0, B is smaller than A. If M is greater than 1.0, B is larger than A. If METHOD is omitted, IMRESIZE uses nearest neighbor interpolation.

**Example 5**

Y = imresize(X, 2); % double the number of pixels in image X.
Y = imresize(X, [100 150]); % resize the image to 100-by-150.

**Rotating an Image**
The MATLAB command `imrotate` rotate an image of any type counterclockwise using the specified interpolation method. Negative value rotates the image clockwise.

`B = IMROTATE(A,ANGLE,METHOD)` rotates the image A by ANGLE degrees in a counterclockwise direction, using the specified interpolation method.

**Example 6**

I = imread('ic.tif');
J = imrotate(I, 35, 'bilinear'); % rotate then 35 degrees using bilinear interpolation.
Imshow(I)
figure, imshow(J); % create new figure

**Cropping an Image**
The MATLAB command *imcrop* crops an image to a specified rectangle. In the syntaxes below, IMCROP displays the input image and waits for you to specify the crop rectangle with the mouse:

\[
I_2 = \text{IMCROP}(I)
\]

\[
X_2 = \text{IMCROP}(X, \text{MAP})
\]

\[
\text{RGB2} = \text{IMCROP}(	ext{RGB})
\]

**Example 7**

\[
\text{I} = \text{imread('ic.tif')};
\]
\[
I_2 = \text{imcrop(I,[60 40 100 90])};
\]
\[
\text{imshow(I), figure, imshow(I2)}
\]

**Example: help bwlabel**

\[
\text{BW} = \text{logical}([1 1 1 0 0 0 0 0 \\
1 1 1 0 1 1 0 0 \\
1 1 1 0 1 1 0 0 \\
1 1 1 0 0 0 1 0 \\
1 1 1 0 0 0 1 0 \\
1 1 1 0 0 0 1 0 \\
1 1 1 0 1 1 0 \\
1 1 1 0 0 0 0 0]);
\]
\[
[L, \text{num}] = \text{bwlabel}(\text{BW,4})
\]

\[
\begin{array}{c|c}
L & \begin{array}{cccccccc}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 2 & 2 & 0 & 0 \\
1 & 1 & 1 & 0 & 2 & 2 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 3 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 3 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 3 & 0 \\
1 & 1 & 1 & 0 & 0 & 3 & 3 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
\end{array} \\
\text{num} & 3
\end{array}
\]

**Example: help regionprops**
Label the connected pixel components in the text.png image, compute their centroids, and superimpose the centroid locations on the image.

```matlab
bw = imread('text.png');
L = bwlabel(bw);
s = regionprops(L, 'centroid');
centroids = cat(1, s.Centroid);
imshow(bw)
hold on
plot(centroids(:,1), centroids(:,2), 'b*')
hold off
```

**Example help edge**

Find the edges of the circuit.tif image using the Prewitt and Canny methods:

```matlab
I = imread('circuit.tif');
    BW1 = edge(I,'prewitt');
    BW2 = edge(I,'canny');
    figure, imshow(BW1)
    figure, imshow(BW2)
```
[BW,thresh,gv,gh] = edge(I,'sobel',...) returns vertical and horizontal edge responses to Sobel gradient operators.

```
I=imread('checker.jpg');
J=rgb2gray(I);
[BW,thresh,gv,gh]= EDGE(J,'sobel');
imshow(gv)
imshow(gh)
imshow(BW)
```

```
J
BW
```

```
gv
gh
thresh =
0.1860
```