MATLAB for Image Processing

October 3rd, 2019

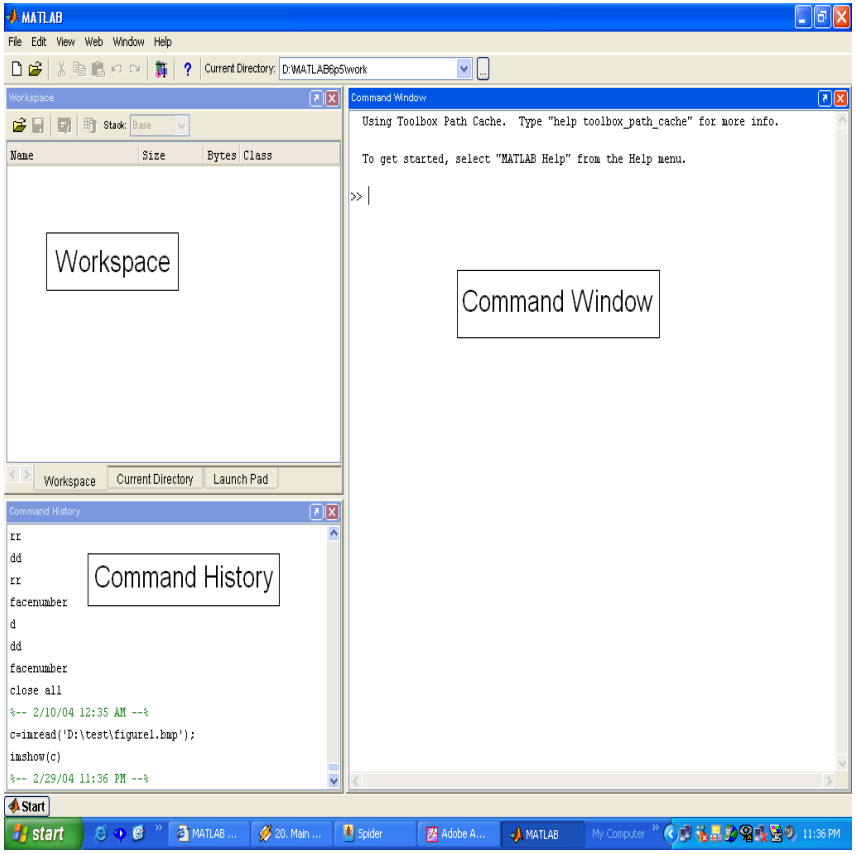
Outline

- **Introduction to MATLAB**
  - **Basics & Examples**
- Image Processing with MATLAB
  - Basics & Examples

What is MATLAB?

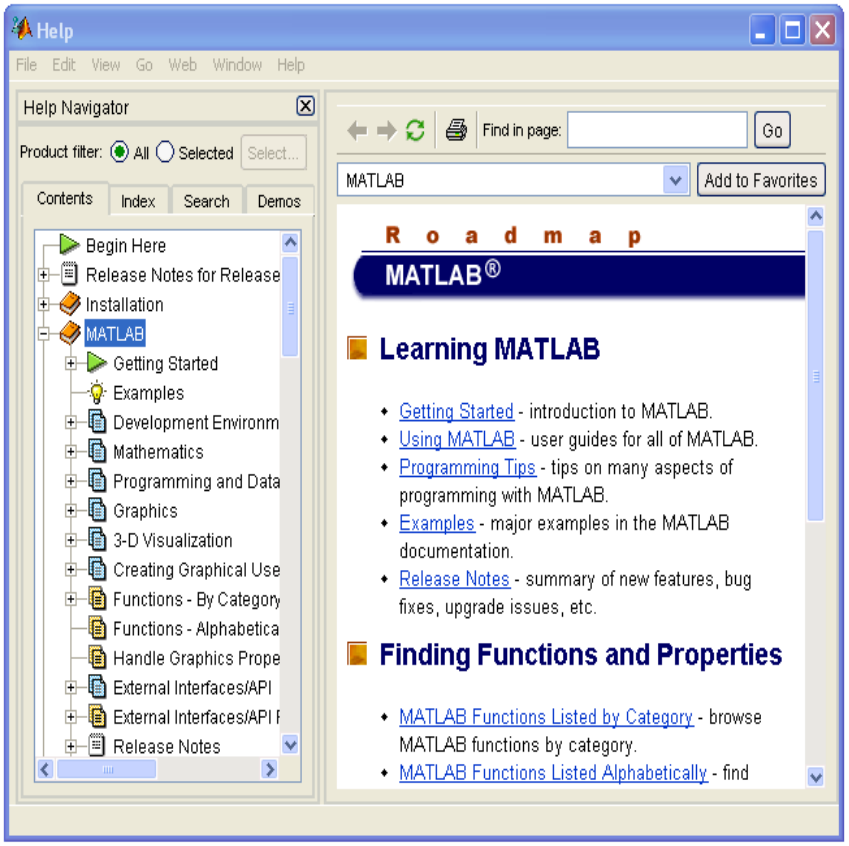
- MATLAB = Matrix Laboratory
- “MATLAB is a high-level language and interactive environment that enables you to perform computationally intensive tasks faster than with traditional programming languages such as C, C++ and Fortran.” (www.mathworks.com)
- MATLAB is an interactive, interpreted language that is designed for fast numerical matrix calculations

The MATLAB Environment



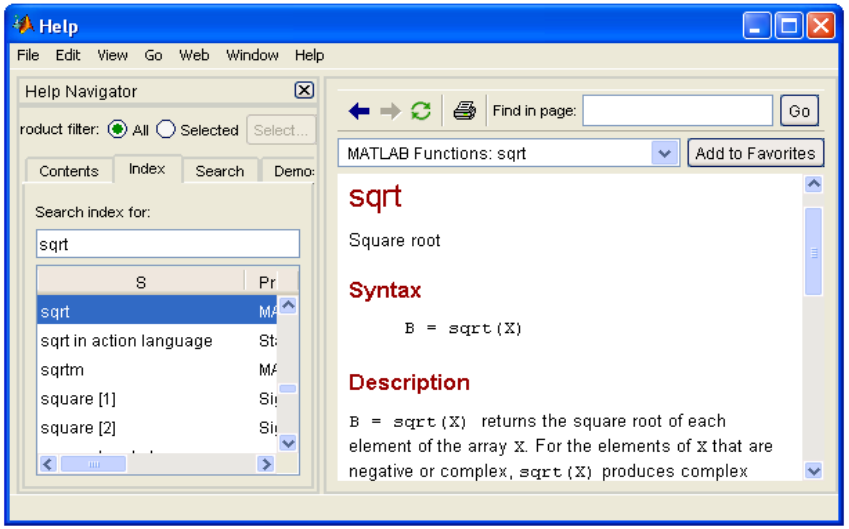
- MATLAB window components:
  - Workspace
    - > Displays all the defined variables
  - Command Window
    - > To execute commands in the MATLAB environment
  - Command History
    - > Displays record of the commands used
  - File Editor Window
    - > Define your functions

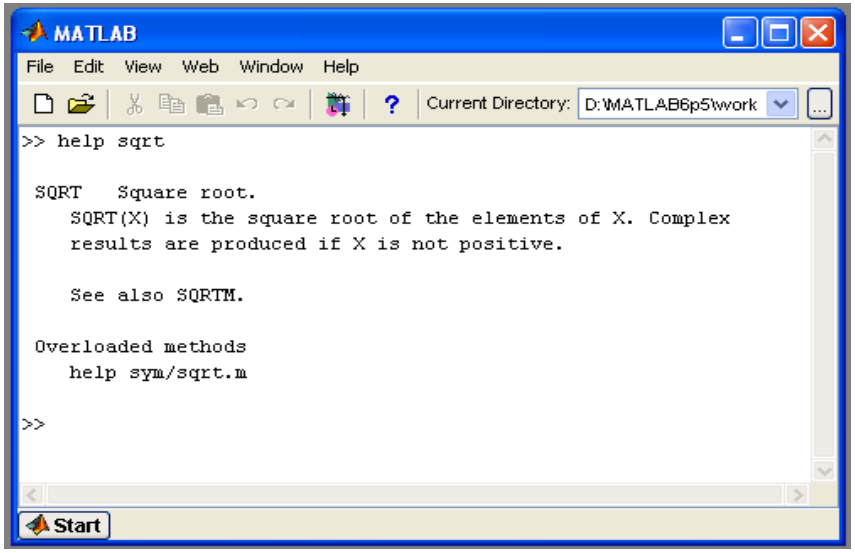
MATLAB Help



- MATLAB Help is an extremely powerful assistance to learning MATLAB
- Help not only contains the theoretical background, but also shows demos for implementation
- MATLAB Help can be opened by using the HELP pull-down menu

MATLAB Help (cont.)





- Any command description can be found by typing the command in the search field
- As shown above, the command to take square root (`sqrt`) is searched
- We can also utilize MATLAB Help from the command window as shown

More about the Workspace

- `who`, `whos` – current variables in the workspace
- `save` – save workspace variables to *.mat file
- `load` – load variables from *.mat file
- `clear` – clear workspace variables

Matrices in MATLAB

- Matrix is the main MATLAB data type
- How to build a matrix?
  - `A=[1 2 3; 4 5 6; 7 8 9];`
  - Creates matrix A of size 3 x 3
- Special matrices:
  - `zeros(n,m), ones(n,m), eye(n,m), rand(), randn()`
- `Numbers are always double (64 bits) unless you specify a different data type`

Basic Operations on Matrices

- All operators in MATLAB are defined on matrices: `+, -, *, /, ^, sqrt, sin, cos`, etc.
- Element-wise operators defined with a preceding dot: `.*, ./, .^`
- `size(A)` – size vector
- `sum(A)` – columns sums vector
- `sum(sum(A))` – sum of all the elements

Variable Name in Matlab

- Variable naming rules

  - must be unique in the first 63 characters

  - must begin with a letter

  - may not contain blank spaces or other types of punctuation

  - may contain any combination of letters, digits, and underscores

  - are case-sensitive

  - should not use Matlab keyword

- Pre-defined variable names
  - pi

Logical Operators

- `==, <, >,` (not equal) `~=`, (not) `~`
- `find('condition')` – Returns indexes of A's elements that satisfy the condition

Logical Operators (cont.)

- Example:

```
>>A=[7 3 5; 6 2 1], Idx=find(A<4)
   A=
      7 3 5
      6 2 1
   Idx=
      3
      4
      6
```

Flow Control

- MATLAB has five flow control constructs:
  - `if` statement
  - `switch` statement
  - `for` loop
  - `while` loop
  - `break` statement

if

- IF statement condition
  - The general form of the IF statement is

```
IF expression
   statements
ELSEIF expression
   statements
ELSE
   statements
END
```

  - CODE

switch

- SWITCH – Switch among several cases based on expression
- The general form of SWITCH statement is:

```
SWITCH switch_expr
   CASE case_expr,
      statement, …, statement
   CASE {case_expr1, case_expr2, case_expr3, …}
      statement, …, statement
      …
   OTHERWISE
      statement, …, statement
END
```

switch (cont.)

- Note:
  - Only the statements between the matching `CASE` and the next `CASE`, `OTHERWISE`, or `END` are executed
  - Unlike C, the `SWITCH` statement does not fall through (so `BREAK`s are unnecessary)

- CODE

for

- FOR repeats statements a specific number of times

- The general form of a FOR statement is:

```
FOR variable=expr
   statements
END
```

  - CODE

while

- WHILE repeats statements an indefinite number of times
- The general form of a WHILE statement is:

```
WHILE expression
   statements
END
```

- CODE

Scripts and Functions

- There are two kinds of M-files:
  - Scripts, which do not accept input arguments or return output arguments. They operate on data in the workspace
  - Functions, which can accept input arguments and return output arguments. Internal variables are local to the function

Functions in MATLAB (cont.)

- Example:
  - A file called STAT.M:

```
function [mean, stdev]=stat(x)
%STAT Interesting statistics.
n=length(x);
mean=sum(x)/n;
stdev=sqrt(sum((x-mean).^2)/n);
```

  - Defines a new function called STAT that calculates the mean and standard deviation of a vector. Function name and file name should be the SAME!
  - CODE

Visualization and Graphics

- `plot(x,y),plot(x,sin(x))` – plot 1D function
- `figure, figure(k)` – open a new figure
- `hold on, hold off` – refreshing
- `axis([xmin xmax ymin ymax])` – change axes
- `title('figure titile')` – add title to figure
- `mesh(x_ax,y_ax,z_mat)` – view surface
- `contour(z_mat)` – view z as topo map
- `subplot(3,1,2)` – locate several plots in figure

Saving your Work

- `save mysession`

  % creates mysession.mat with all variables

- `save mysession a b`

  % save only variables a and b

- `clear all`

  % clear all variables

- `clear a b`

  % clear variables a and b

- `load mysession`

  % load session

Outline

- Introduction to MATLAB
  - Basics & Examples
- **Image Processing using MATLAB**
  - **Basics & Examples**

Course Outline:

1. Working with Images in MATLAB
    a) Image types and classes
    b) Read/write images
    c) Display images
2. Basic Image Processing
    a) Image contrast and brightness enhancement
    b) Image arithmetic
3. Block Processing of Images
4. Image Restoration
    a) Noise reduction (filtering)
    b) Image alignment
5. Image Segmentation & Edge Detection
6. Case Studies

What is the Image Processing Toolbox?

- The Image Processing Toolbox is a collection of functions that extend the capabilities of the MATLAB's numeric computing environment. The toolbox supports a wide range of image processing operations, including:
  - Geometric operations
  - Neighborhood and block operations
  - Linear filtering and filter design
  - Transforms
  - Image analysis and enhancement
  - Binary image operations
  - Region of interest operations

Images in MATLAB

- MATLAB can import/export several image formats:
  - BMP (Microsoft Windows Bitmap)
  - GIF (Graphics Interchange Files)
  - HDF (Hierarchical Data Format)
  - JPEG (Joint Photographic Experts Group)
  - PCX (Paintbrush)
  - PNG (Portable Network Graphics)
  - TIFF (Tagged Image File Format)
  - XWD (X Window Dump)
  - raw-data and other types of image data

- **Typically switch images to double to perform any processing and convert back to unsigned integer**

- Data types in MATLAB
  - Double (64-bit double-precision floating point)
  - Single (32-bit single-precision floating point)
  - Int32 (32-bit signed integer)
  - Int16 (16-bit signed integer)
  - Int8 (8-bit signed integer)
  - Uint32 (32-bit unsigned integer)
  - Uint16 (16-bit unsigned integer)
  - Uint8 (8-bit unsigned integer)

Section Outline:

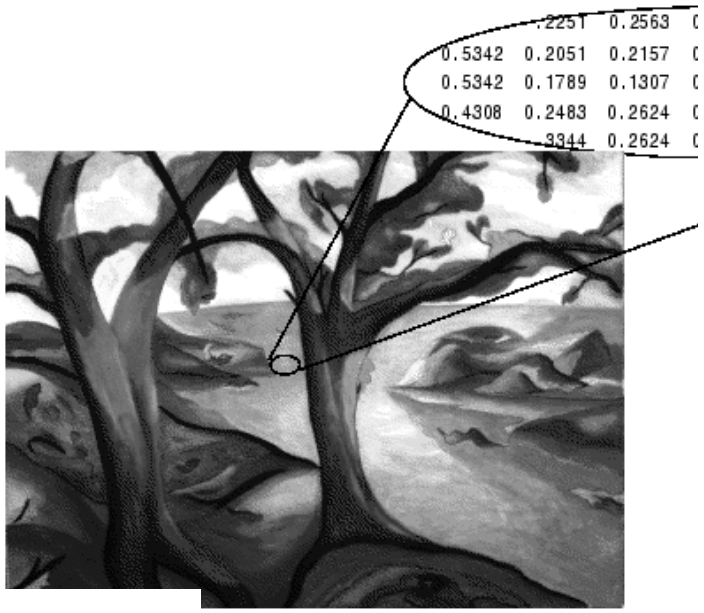
1. Image types
   - Index images
   - Intensity images
   - Binary images
   - RGB images

2. Importing and exporting images in MATLAB
   - `imfinfo`
   - `imread` and `imwrite`
   - `imshow`

3. Converting between image formats

Images in MATLAB

- Binary images : {0,1}
- Intensity images : [0,1] or `uint8`, `double` etc.
- RGB images : m × n × 3
- Multidimensional images: m × n × p (p is the number of layers)



```
>> load sampleImages
```

Image Types: MATLAB Data Types Used

- A wide array of different data types exist in MATLAB, but only a subset of the data types are used to represent images in MATLAB.

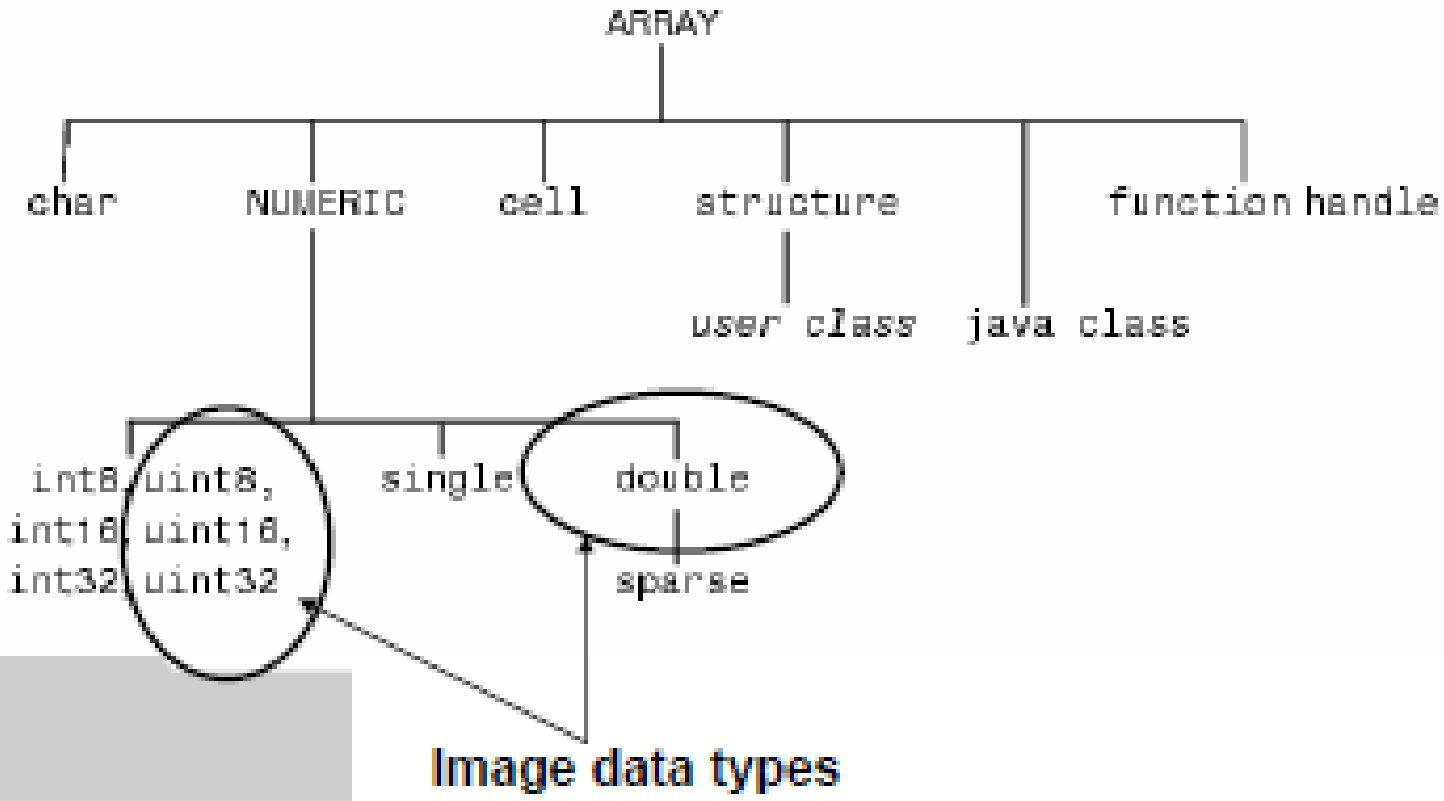


Image Processing Toolbox (4)

- Indexed Image (2 matrices – colormap and index)

```
>> imshow(indexImg, map)
```

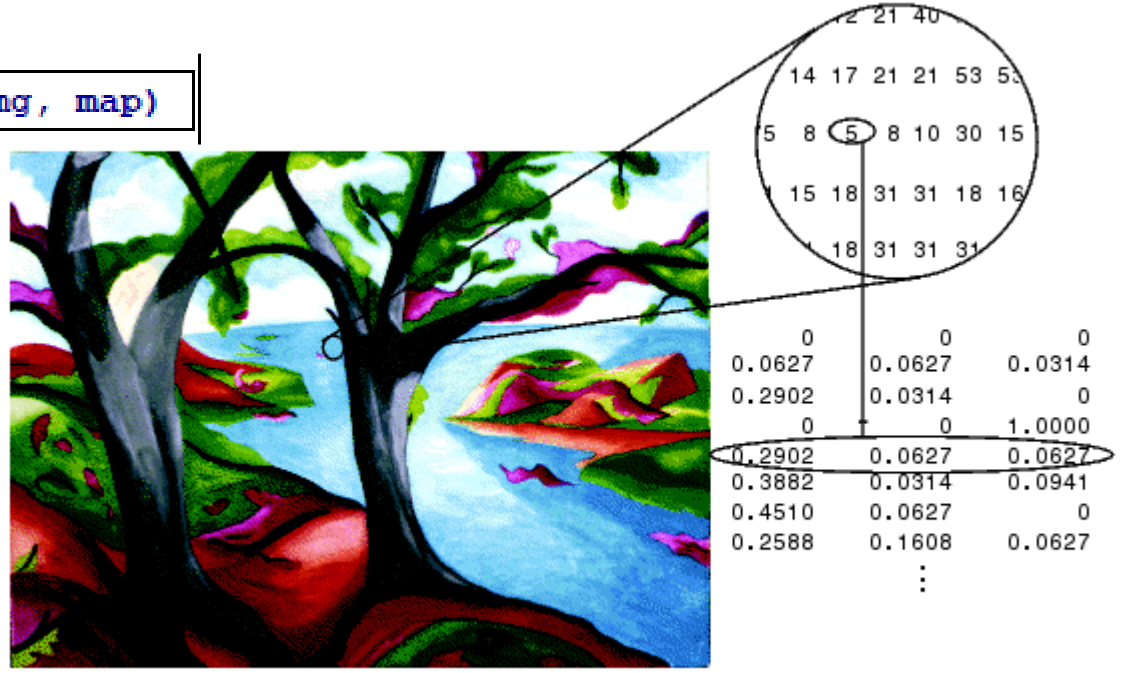
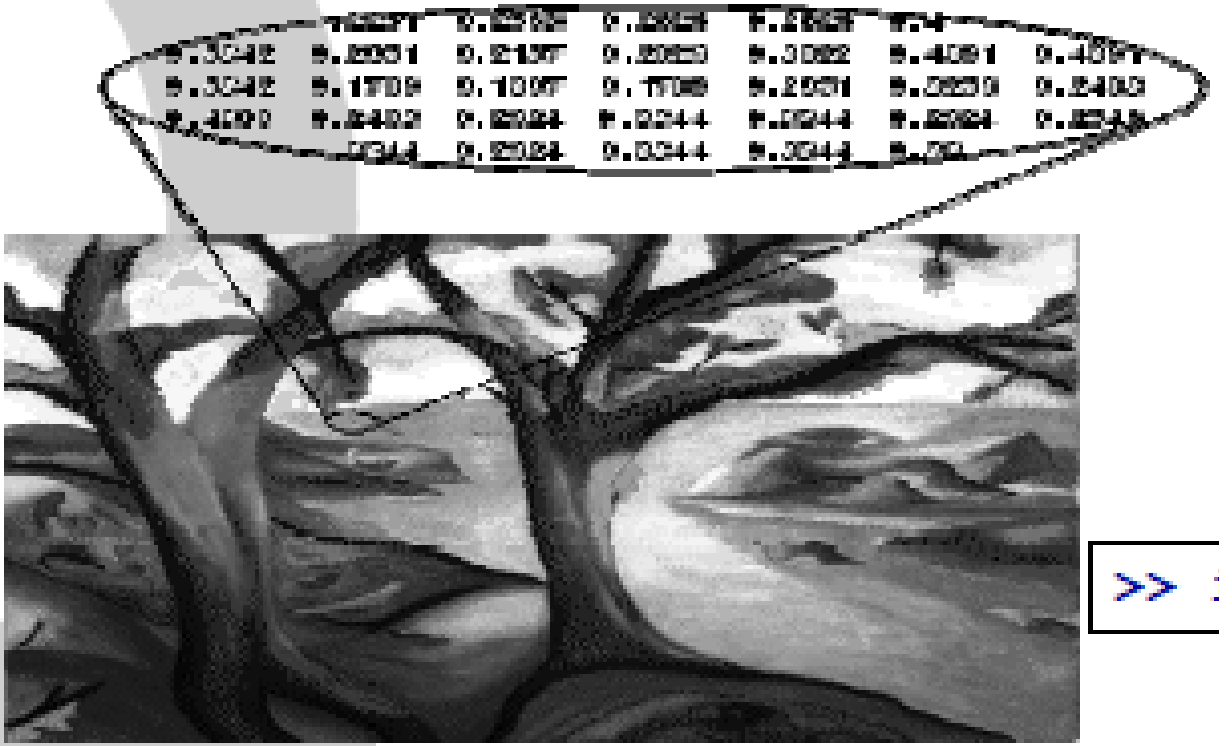


Image Courtesy of Susan Cohen

Image Types: Intensity Images

- An intensity image only consists of one matrix, I, whose values represent intensities within some range, for example [0 1] or `uint8`.



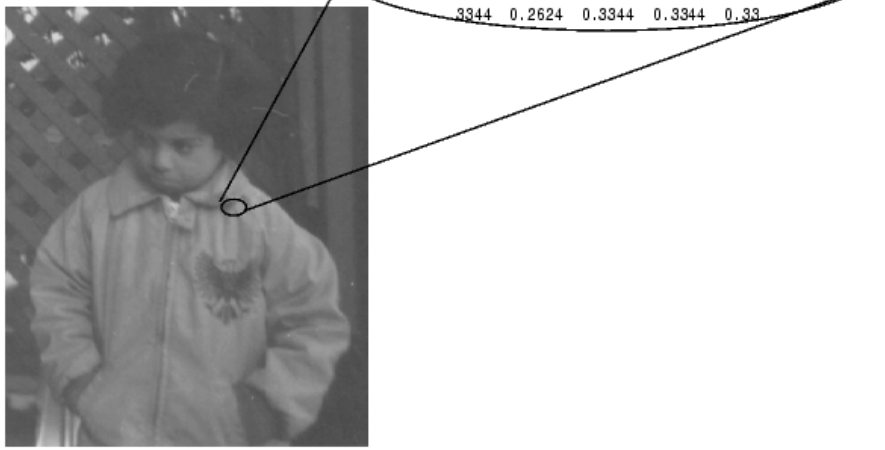
```
>> imshow(intenImg)
```

Image Processing Toolbox (2)

- Grayscale Image

  (row x col)

```
>> imshow(bwImg)
```



- Binary Image

  (row x col)

  = Grayscale with

  2 levels

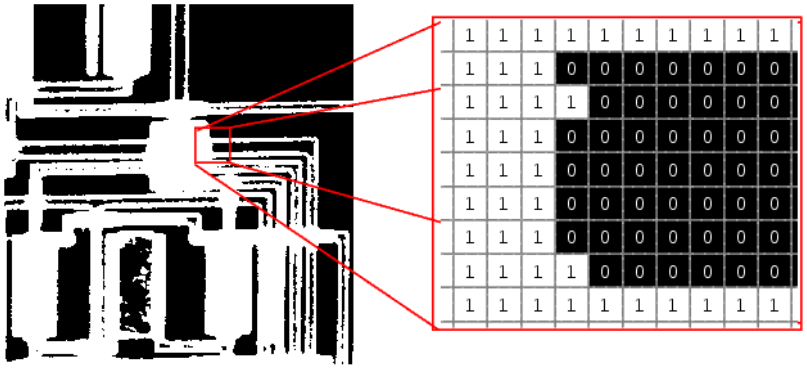
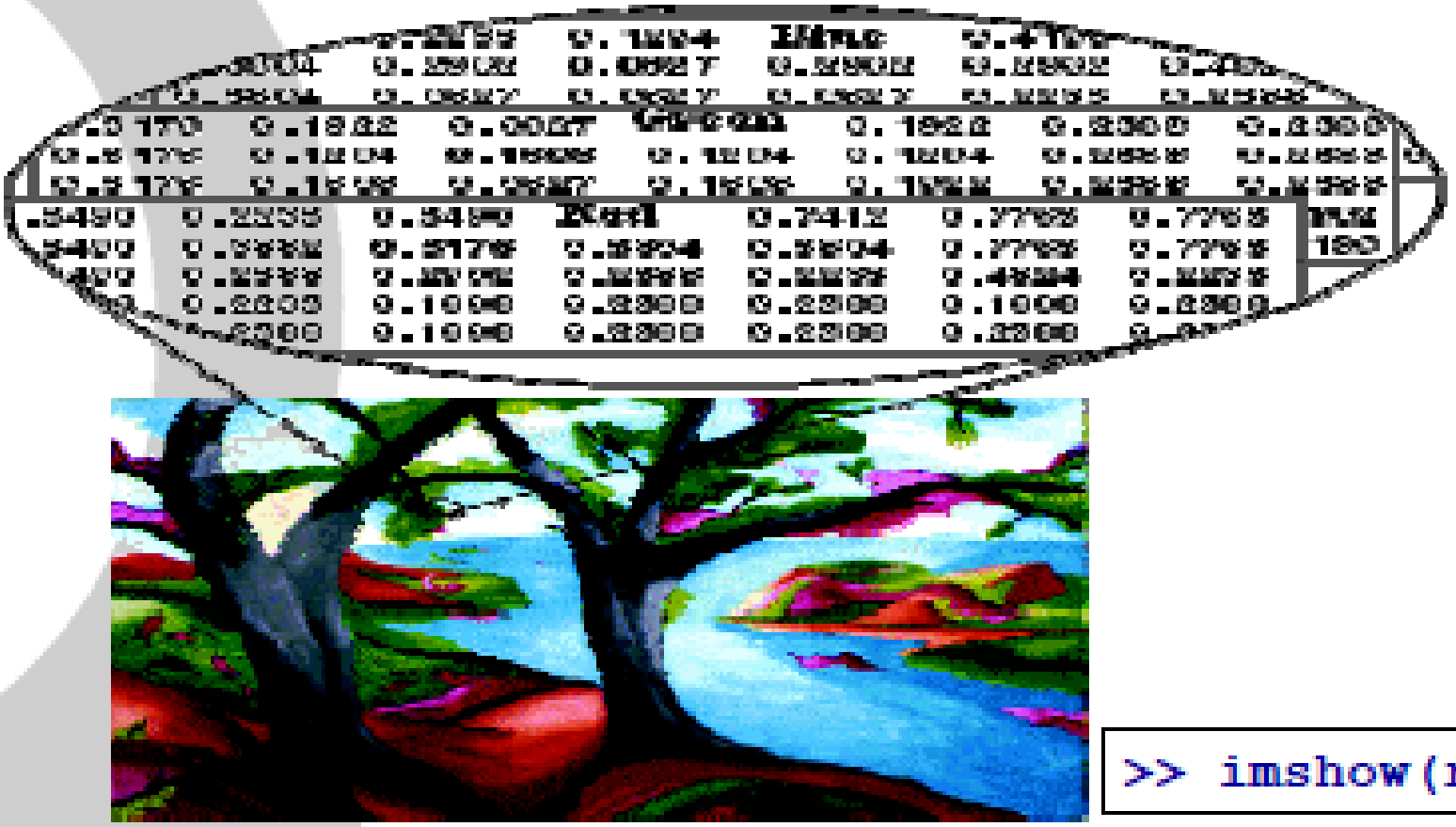


Image Types: RGB Images

- RGB image is stored in MATLAB as an m-by-n-by-3 data where each m-by-n page defines red (R), green (G) and blue (B) color components for each pixel.



```
>> imshow(rgbImg)
```

Image Processing Toolbox (3)

- RGB Image
  (row X col X 3)

Image Processing Toolbox (5)

- Reading an Image and storing it in matrix I:
  - `I = imread('pout.tif');`
  - `[I,map] = imread('pout.tif');` For indexed images
- Deals with many formats
  - JPEG, TIFF, GIF, BMP, PNG, PCX, … more can be added from Mathworks Central File Exchange
    - http://www.mathworks.com/matlabcentral/fileexchange

Image Processing Toolbox (6)

- After an image has been read we can convert from one type to another:
  - `ind2gray`, `gray2ind`, `rgb2gray`, `gray2rgb`, `rgb2ind`, `ind2rgb`
- Images are read into `uint8` data type. To manipulate the pixel values, they have to be first converted to `double` type using the `double(I)` function.

Image Processing Toolbox (7)

- Pixel values are accessed as matrix elements.
  - 2D Image with intensity values: `I(row,col)`
  - 2D RGB images `I(row,col,color)`
    - Color : Red = 1; Green = 2 ; Blue = 3
- Displaying images
  - `>>figure, imshow(I)`
- Displaying pixel position and intensity information
  - `pixval on`

Image Processing Toolbox (8)

- All arithmetic operations performed on matrices may be performed on images
- After processing, an image matrix can be written to an output image file with the `imwrite` function
  - `imwrite(I,map,'filename','fmt')`
- Without the map argument, the image data is supposed to be grayscale or RGB.
- The format 'fmt' needs to support the particular type of image

Images in Matlab

- Matlab is optimised for operating on matrices
- Images are matrices!
- Many useful built-in functions in the Matlab Image Processing Toolbox
- Very easy to write your own image processing functions

Loading and displaying images

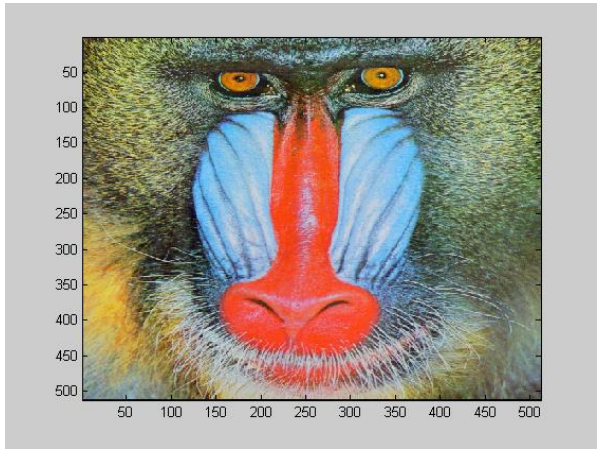
```
>> I=imread('mandrill.bmp','bmp');      % load image
```

Matrix with image data

image filename as a string

image format as a string

```
>> image(I)   % display image
>> whos I
```



```
Name        Size              Bytes       Class

  I         512x512x3         786432    uint8 array

Grand to                                32 bytes
```

Dimensions of I (red, green and blue intensity information)

Matlab can only perform arithmetic operations on data with class double!

Display the left half of the mandrill image

Representation of Images

- Images are just an array of numbers

>> I % ctrl+c to halt output!

- Intensity of each pixel is represented by the pixel element's value in the red, green and blue matrices

>> I(1,1,:) % RGB values of element (1,1)

ans(:,:,1) = **Red**

135

ans(:,:,2) = **Green**

97

ans(:,:,3) = **Blue**

33

Images where the pixel value in the image represents the intensity of the pixel are called **intensity images**.

Indexed images

- An indexed image is where the pixel values are indices to elements in a **colour map** or **colour lookup table**.
- The colour map will contain entries corresponding to red, green and blue intensities for each index in the image.

```
>> jet(20)  % Generate a jet colourmap for 20 indices
ans =
         0         0    0.6000
         0         0    0.8000
         0         0    1.0000
         0    0.2000    1.0000
         0    0.4000    1.0000
         0    0.6000    1.0000
         0    0.8000    1.0000
         0    1.0000    1.0000
    0.2000    1.0000    0.8000
    0.4000    1.0000    0.6000
    0.6000    1.0000    0.4000
    0.8000    1.0000    0.2000
    1.0000    1.0000         0
    1.0000    0.8000         0
    1.0000    0.6000         0
    1.0000    0.4000         0
```

RGB Entry for index value 3

Values can range from 0.0 to 1.0

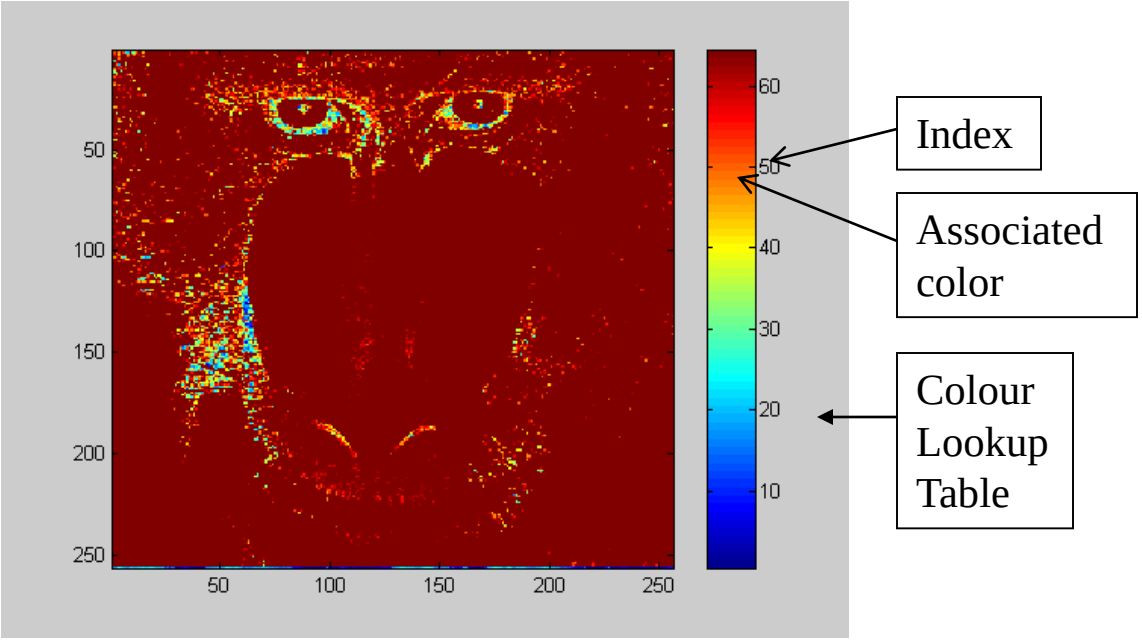
3	4	7	3	6	1 9	8	9	1	2
5	6	14	4	2	5	6	1	4	5
2	8	9	4	2	13	7	8	4	5
5	1	11	5	6	4	1	7	4	4
1	9	5	6	5	5	1 4	4	6	5
5	9	2	1	11	1	3	6	1	9
7	6	8	18	1	8	1	9	1 3	3
9	2	3	7	2	9	8	1 6	6	4
7	8	6	7	4	15	8	2	1	3
7	5	10	8	4	10	4	3	6	4

Red, green and blue intensities of the nearest index in the colourmap are used to display the image.

Displaying indexed images

```
>> I2=I(:,:,2);  % green values of I
>> image(I2)
>> colorbar  % display colourmap
```

Matlab considers I2 as an indexed image as it doesn't contain entries for red, green and blue entries



Displaying indexed images

(continued)

- change colourmap

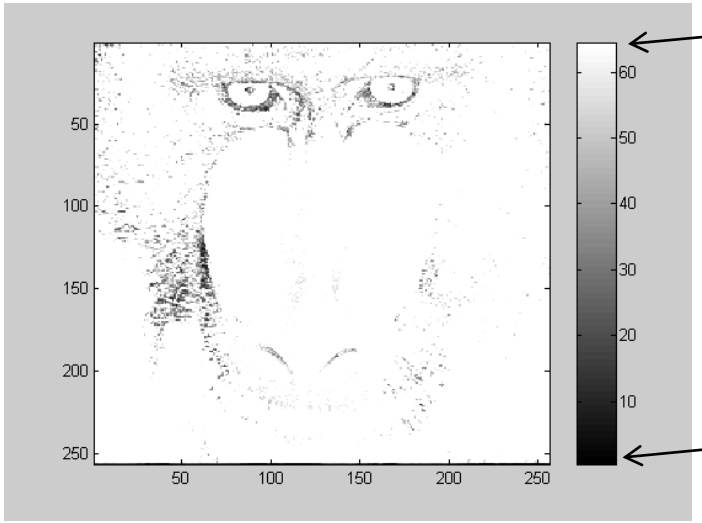
>> colormap(gray)

Type >>help graph3d to get a list of built-in colourmaps. Experiment with different built-in colourmaps.

Define your own colourmap mymap by creating a matrix (size m x 3) with red, green, blue entries. Display an image using your colourmap.

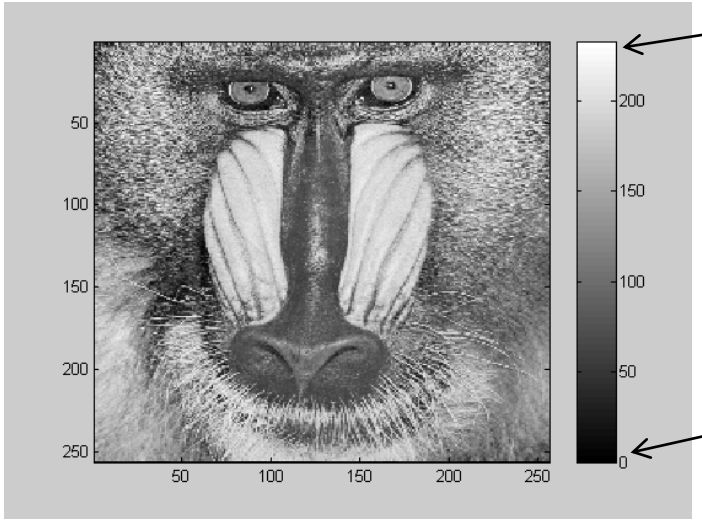
- scale colourmap

>> imagesc(I2)



Red =1.0, Green = 1.0, Blue =1.0, corresponds to index 64

Red =0.0, Green = 0.0, Blue = 0.0, corresponds to index 1

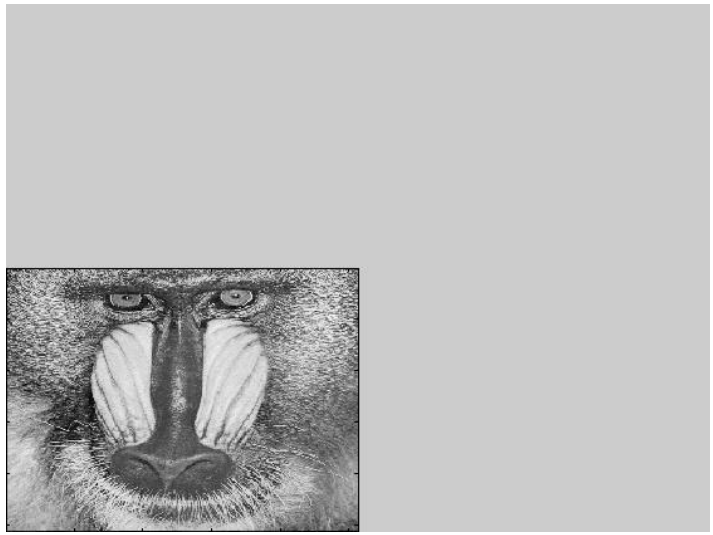


Red =1.0, Green = 1.0, Blue =1.0, corresponds to index 255

Red =0.0, Green = 0.0, Blue = 0.0, corresponds to index 0

Useful functions for displaying images

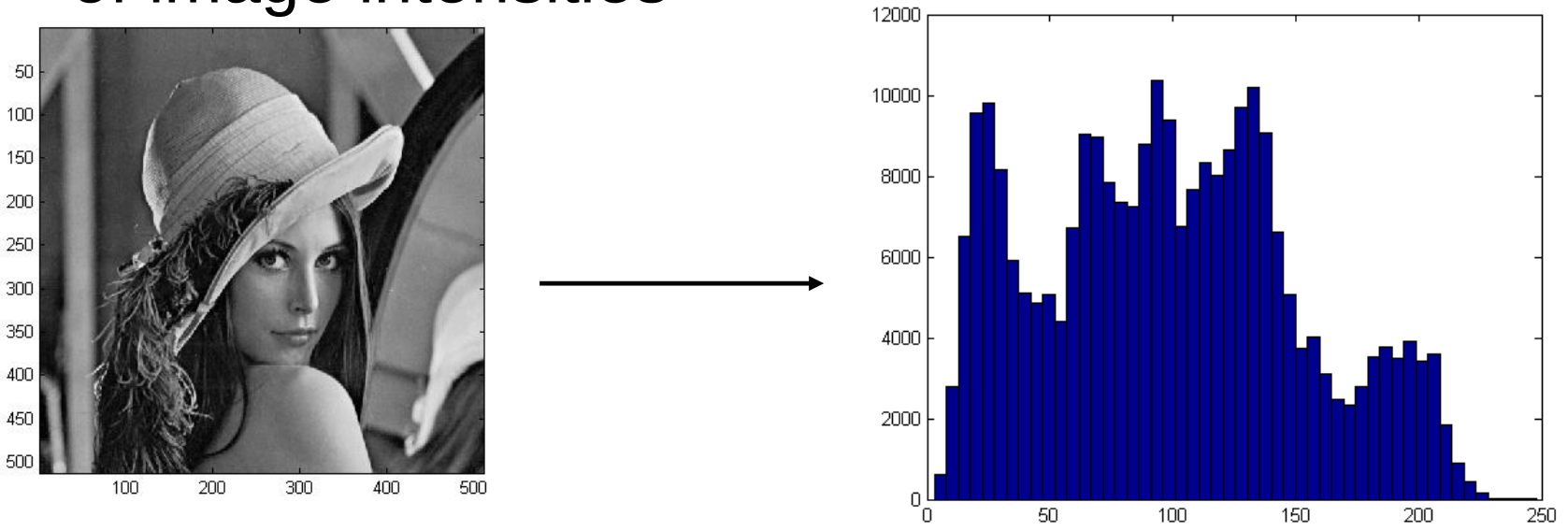
```
>> axis image   % plot fits to data
>> h=axes('position', [0 0 0.5 0.5]);
>> axes(h);
>> imagesc(I2)
```



Investigate axis and axes functions using Matlab's help

Histograms

- Frequency of the intensity values of the image
- Quantise frequency into intervals (called bins)
- (Un-normalised) probability density function of image intensities



Computing histograms of images in Matlab

```
>>hist(reshape(double(Lena(:,:,2)),[512*512 1]),50)
```

Histogram function

Convert image into a 262144 by 1 distribution of values

Number of bins

Generate the histograms of the green channel of the Lena image using the following number of bins : 10, 20, 50, 100, 200, 500, 1000

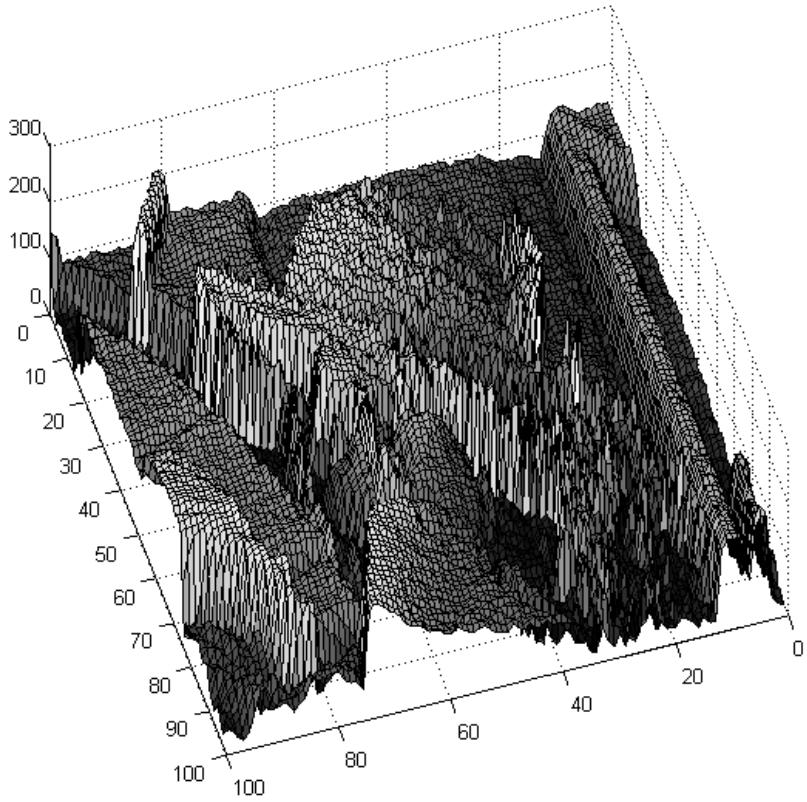
Histogram equalisation works by equitably distributing the pixels among the histogram bins. Histogram equalise the green channel of the Lena image using Matlab's **histeq** function. Compare the equalised image with the original. Display the histogram of the equalised image. The number of pixels in each bin should be approximately equal.

Visualising the intensity surface

```
>>surf(double(imresize(Lena(:,:,2),[50 50])))
```

Change type to double precision

Remember to reduce size of image!



Use Matlab's built-in **mesh** and **shading** surface visualisation functions

Useful functions for manipulating images

- Convert image to grayscale

```
>>Igray=rgb2gray(I);
```

- Resize image

```
>>Ismall=imresize(I,[100 100], 'bilinear');
```

- Rotate image

```
>>I90=imrotate(I,90);
```

Other useful functions

Convert polar coordinates to cartesian coordinates >>pol2cart(rho,theta)	Check if a variable is null >>isempty(I)	Trigonometric functions sin, cos, tan
Convert polar coordinates to cartesian coordinates >>cart2pol(x,y)	Find indices and elements in a matrix >>[X,Y]=find(I>100)	Fast Fourier Transform  fft2(I)
Get size of matrix >>size(I)	Change the dimensions of a matrix >>reshape(rand(10,10),[100 1])	Discrete Cosine Transform  dct(I)
Add elements of a Matrix (columnwise addition in matrices) >>sum(I)	Exponentials and Logarithms exp log log10	

Image Import and Export

- Read and write images in Matlab

```
img = imread('apple.jpg');
dim = size(img);
figure;
imshow(img);
imwrite(img, 'output.bmp', 'bmp');
```

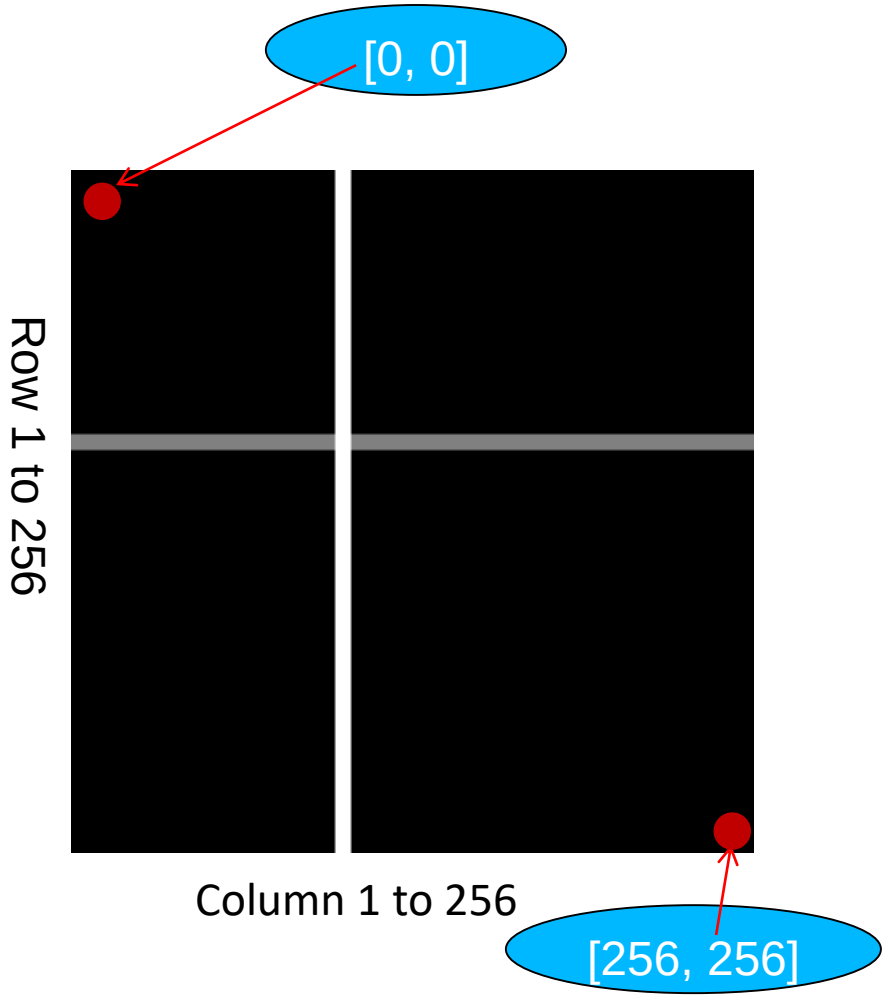
- Alternatives to `imshow`

```
imagesc(I)
imtool(I)
image(I)
```

Images and Matrices

How to build a matrix (or image)?
Intensity Image:

```
row = 256;
col = 256;
img = zeros(row, col);
img(100:105, :) = 0.5;
img(:, 100:105) = 1;
figure;
imshow(img);
```



Images and Matrices

Binary Image:

```
row = 256;
col = 256;
img = rand(row,
col);
img = round(img);
figure;
imshow(img);
size(img)
```

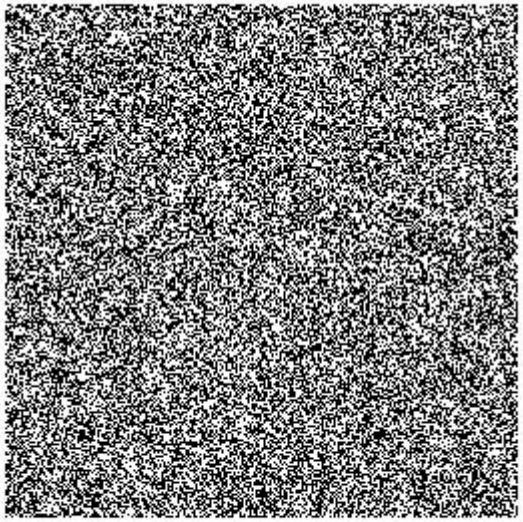


Image Display

- `image` - create and display image object
- `imagesc` - scale and display as image
- `imshow` - display image

Performance Issues

- The idea: MATLAB is
  - very fast on vector and matrix operations
  - Correspondingly slow with loops

- Try to avoid loops
- Try to vectorize your code

  http://www.mathworks.com/support/tech-notes/1100/1109.html

THE END

- Thank you ☺
- Questions?