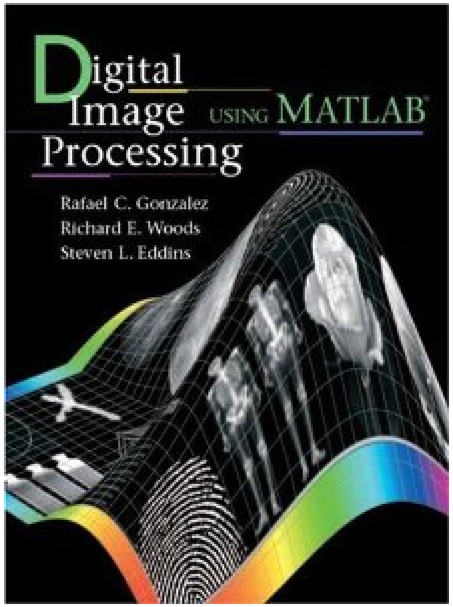
Digital Image Processing Using Matlab

Workshop



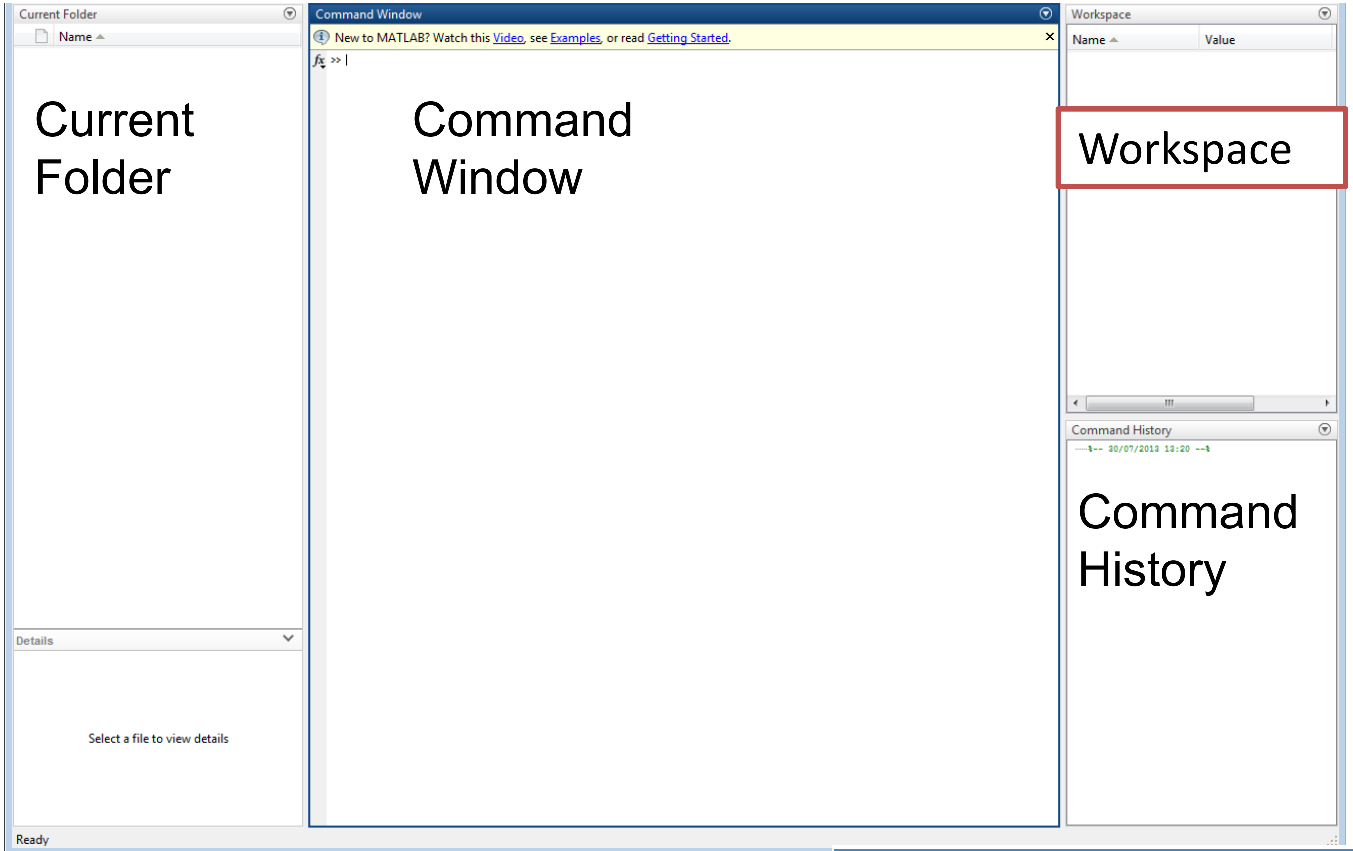
What is Matlab?

- 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.”

- MATLAB is an interactive, interpreted language that is designed for fast numerical matrix calculations

The MATLAB Environment



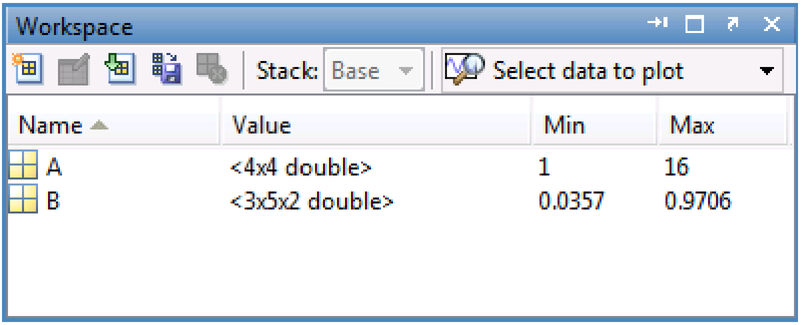
New to MATLAB? Watch this Video, see Examples, or read Getting Started.

```
fx >> |
```

```
%-- 30/07/2013 13:20 --%
```

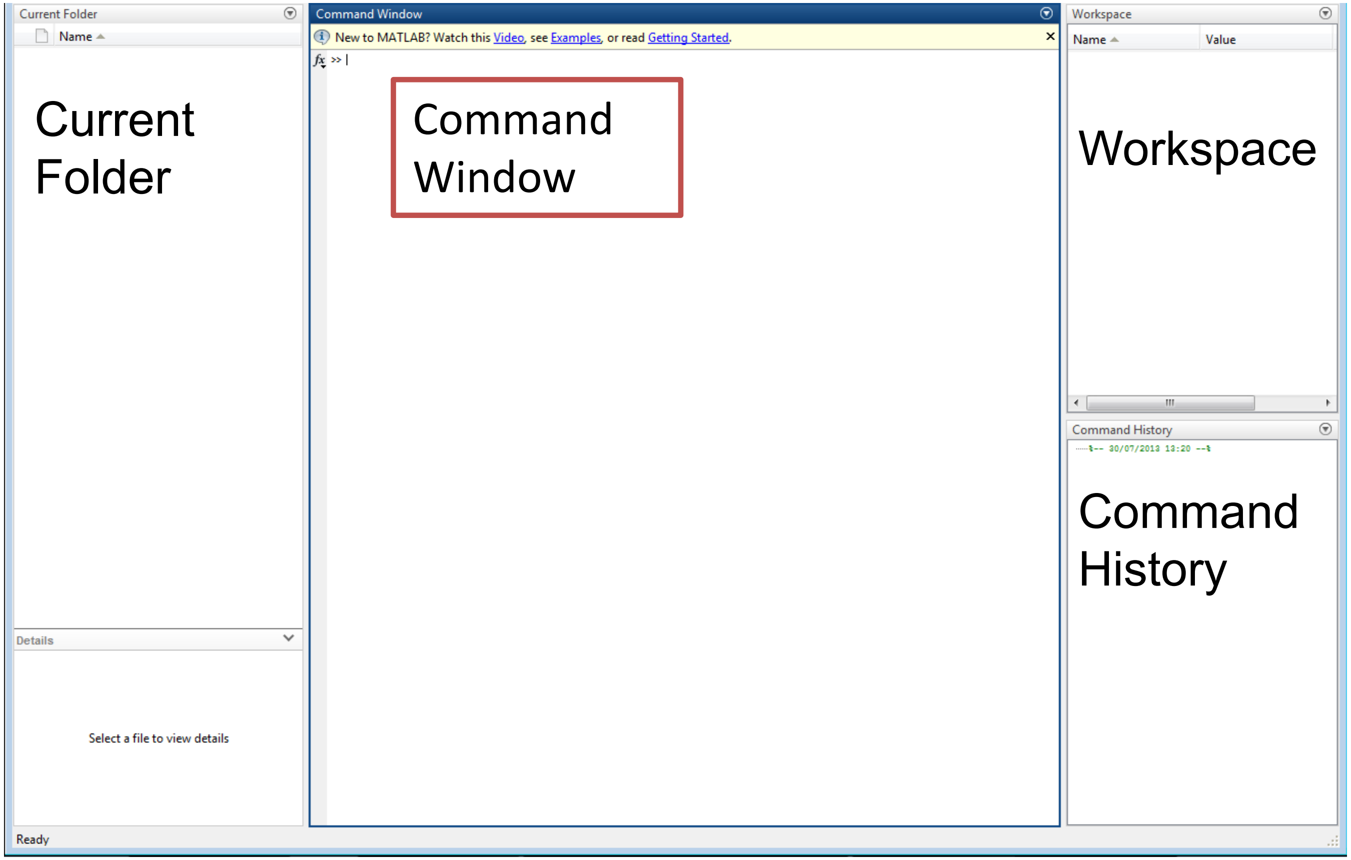
Select a file to view details

Workspace: Displays all the defined variables



Name	Value	Min	Max
A	<4x4 double>	1	16
B	<3x5x2 double>	0.0357	0.9706

The MATLAB Environment



New to MATLAB? Watch this Video, see Examples, or read Getting Started.

```
>>
```

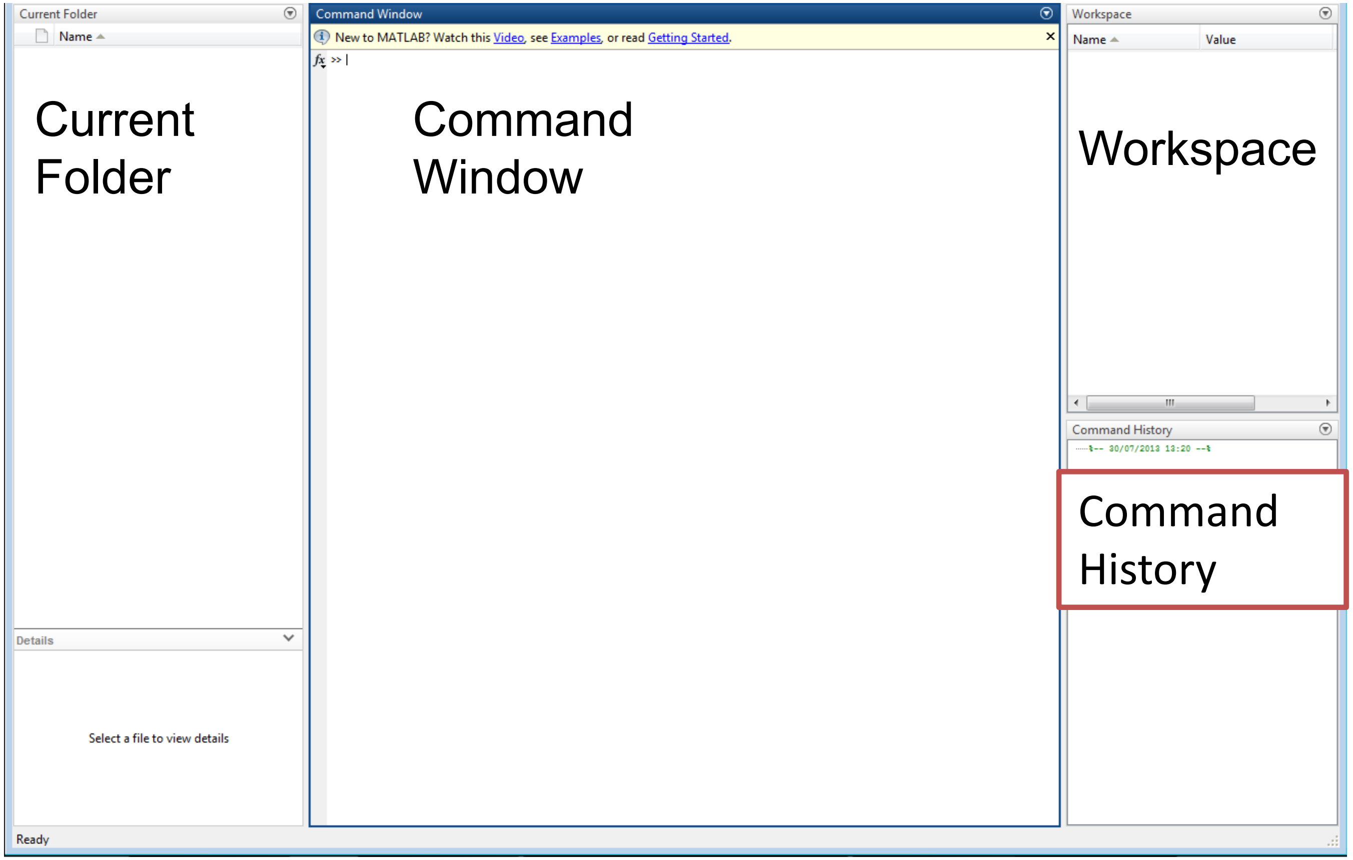
```
%-- 30/07/2013 13:20 --%
```

Select a file to view details

Ready

Command Window: Run Matlab statements

The MATLAB Environment



New to MATLAB? Watch this Video, see Examples, or read Getting Started.

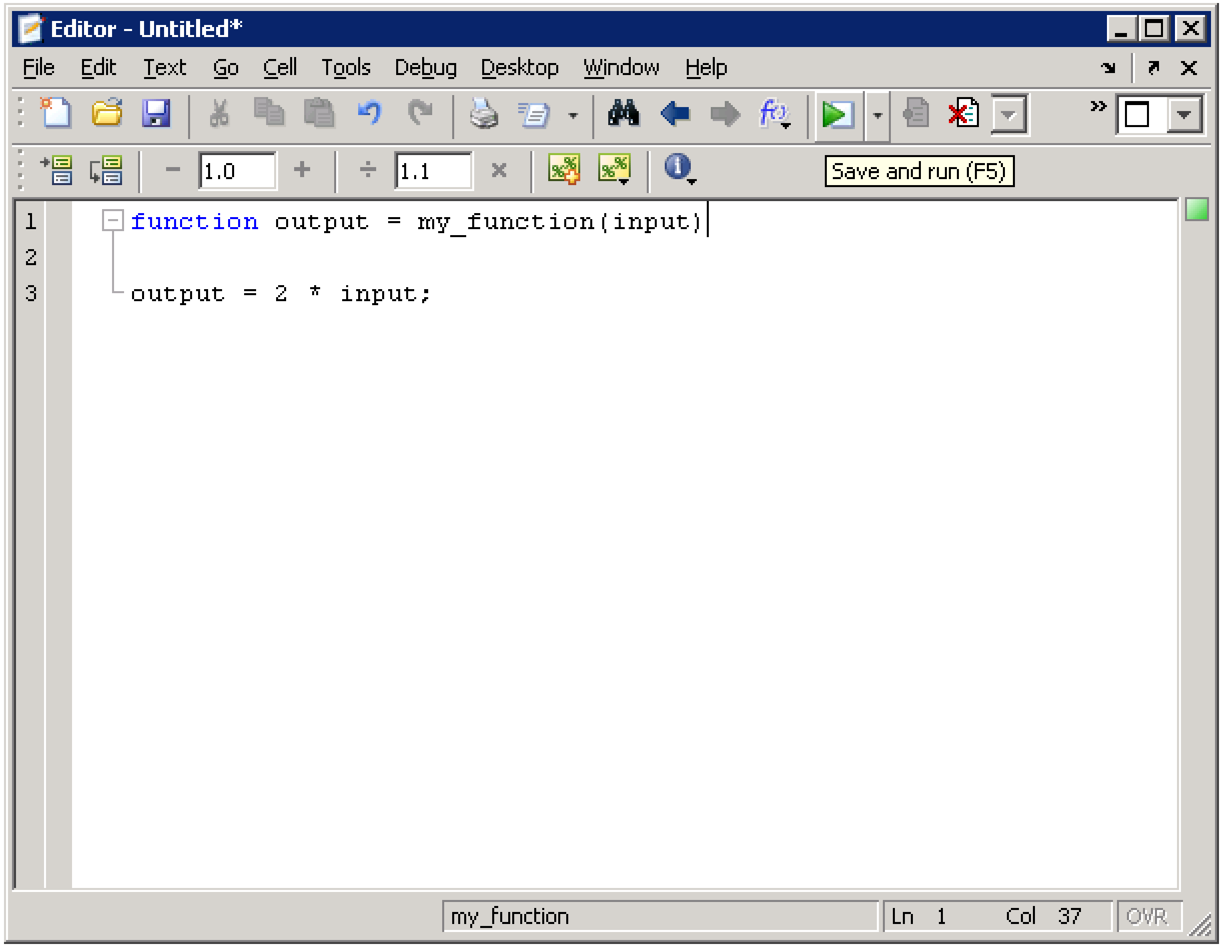
```
fx >> |
```

```
%-- 30/07/2013 13:20 --%
```

Select a file to view details

Command History: log of command window, search for previously executed statements, copy and re-execute

The MATLAB Environment



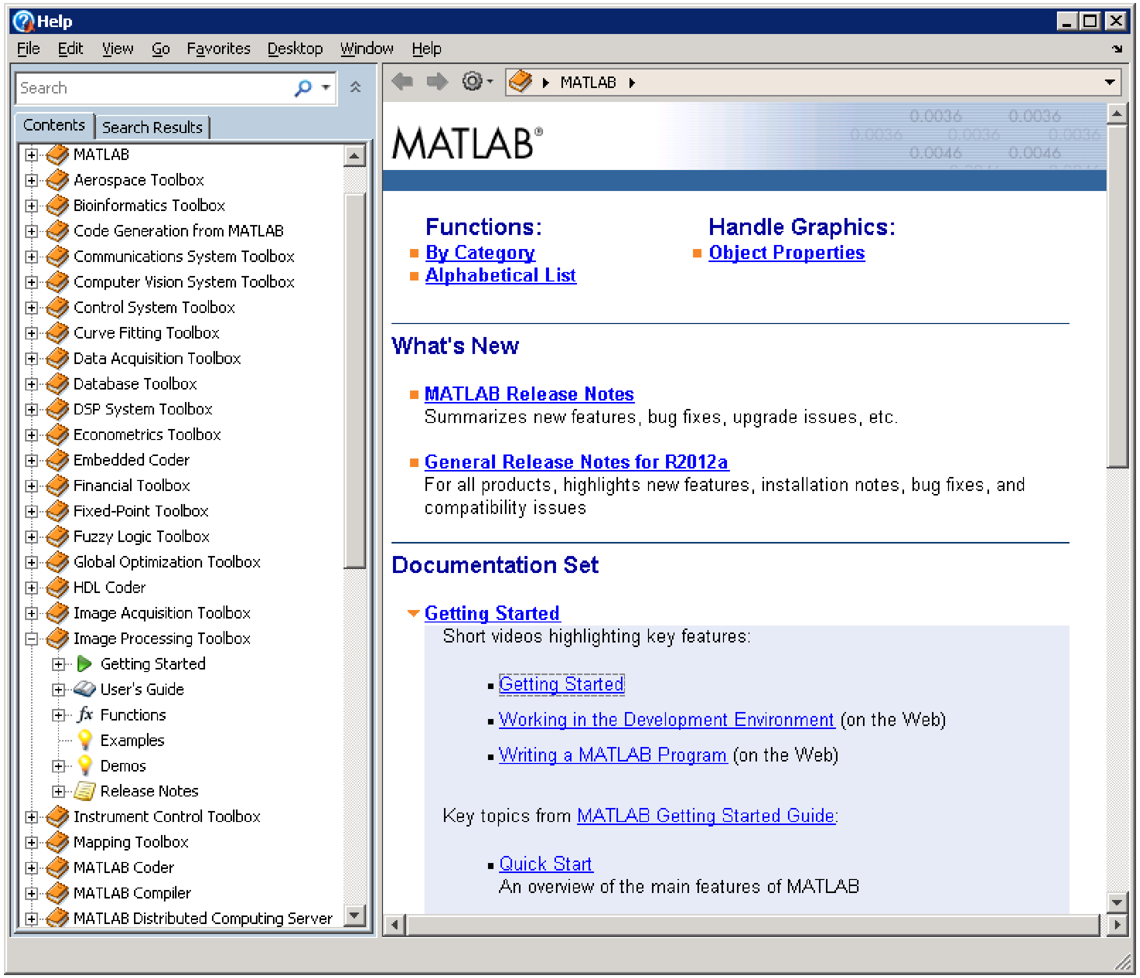
```
function output = my_function(input)

output = 2 * input;
```

File Editor Window: Define functions and scripts

MATLAB Help

- MATLAB Help is a very useful tool in learning MATLAB

- Contains help on built-in functions, theoretical background and also demos to demonstrate implementation



- MATLAB
- Aerospace Toolbox
- Bioinformatics Toolbox
- Code Generation from MATLAB
- Communications System Toolbox
- Computer Vision System Toolbox
- Control System Toolbox
- Curve Fitting Toolbox
- Data Acquisition Toolbox
- Database Toolbox
- DSP System Toolbox
- Econometrics Toolbox
- Embedded Coder
- Financial Toolbox
- Fixed-Point Toolbox
- Fuzzy Logic Toolbox
- Global Optimization Toolbox
- HDL Coder
- Image Acquisition Toolbox
- Image Processing Toolbox
  - Getting Started
  - User's Guide
  - Functions
  - Examples
  - Demos
  - Release Notes
- Instrument Control Toolbox
- Mapping Toolbox
- MATLAB Coder
- MATLAB Compiler
- MATLAB Distributed Computing Server

MATLAB®

Functions:
- By Category
- Alphabetical List

Handle Graphics:
- Object Properties

What's New

- **MATLAB Release Notes**
  Summarizes new features, bug fixes, upgrade issues, etc.

- **General Release Notes for R2012a**
  For all products, highlights new features, installation notes, bug fixes, and compatibility issues

Documentation Set

Getting Started
Short videos highlighting key features:

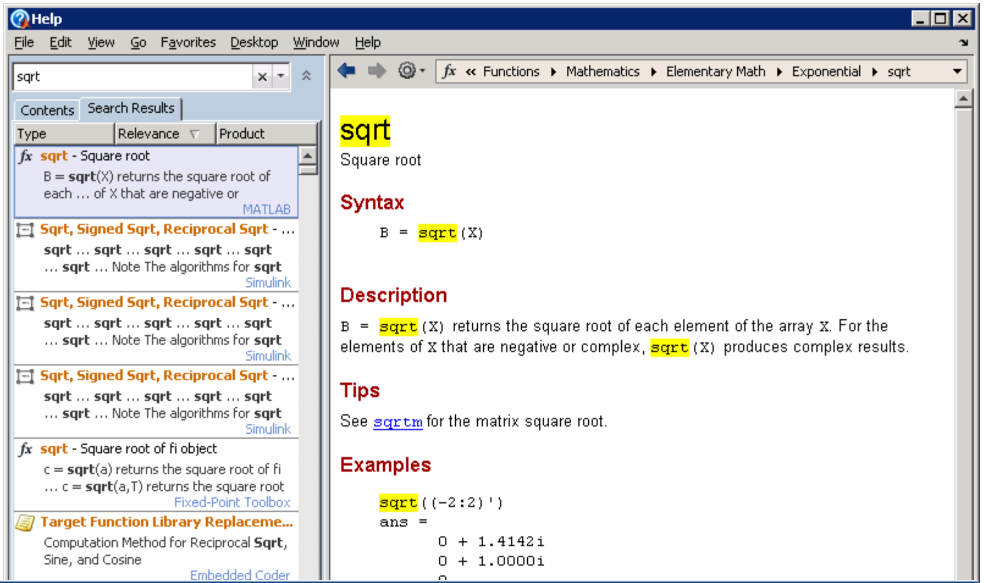
- Getting Started
- Working in the Development Environment (on the Web)
- Writing a MATLAB Program (on the Web)

Key topics from MATLAB Getting Started Guide:

- Quick Start
  An overview of the main features of MATLAB

MATLAB Help (cont.)

- Find functions using search using search in the Help window


Help
File Edit View Go Favorites Desktop Window Help
sqrt
Contents
Search Results
Type
Relevance
Product
« Functions ▸ Mathematics ▸ Elementary Math ▸ Exponential ▸ sqrt

sqrt - Square root
B = **sqrt**(X) returns the square root of each … of X that are negative or
MATLAB

Sqrt, Signed Sqrt, Reciprocal Sqrt - …
sqrt … **sqrt** … **sqrt** … **sqrt** … **sqrt** … **sqrt** … Note The algorithms for **sqrt**
Simulink

Sqrt, Signed Sqrt, Reciprocal Sqrt - …
sqrt … **sqrt** … **sqrt** … **sqrt** … **sqrt** … **sqrt** … Note The algorithms for **sqrt**
Simulink

Sqrt, Signed Sqrt, Reciprocal Sqrt - …
sqrt … **sqrt** … **sqrt** … **sqrt** … **sqrt** … **sqrt** … Note The algorithms for **sqrt**
Simulink

sqrt - Square root of fi object
c = **sqrt**(a) returns the square root of fi … c = **sqrt**(a,T) returns the square root
Fixed-Point Toolbox

Target Function Library Replaceme…
Computation Method for Reciprocal **Sqrt**, Sine, and Cosine
Embedded Coder

sqrt

Square root

Syntax

```
B = sqrt(X)
```

Description

`B = sqrt(X)` returns the square root of each element of the array `X`. For the elements of `X` that are negative or complex, `sqrt(X)` produces complex results.

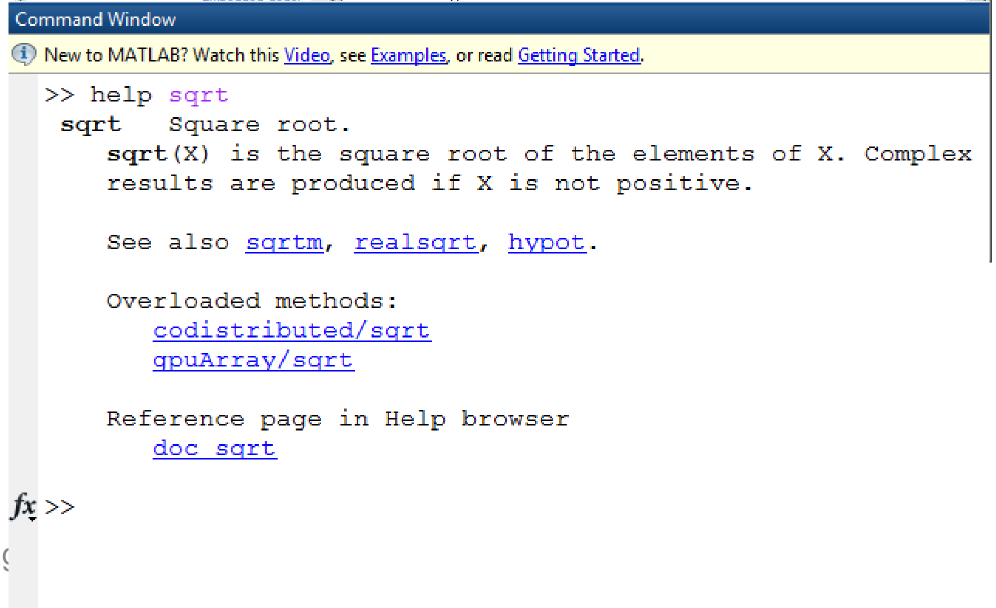
Tips

See `sqrtm` for the matrix square root.

Examples

```
sqrt((-2:2)')
ans =
        0 + 1.4142i
        0 + 1.0000i
        0
```

- Help can also be called from the command window



New to MATLAB? Watch this Video, see Examples, or read Getting Started.

```
>> help sqrt
 sqrt    Square root.
    sqrt(X) is the square root of the elements of X. Complex
    results are produced if X is not positive.

    See also sqrtm, realsqrt, hypot.

    Overloaded methods:
       codistributed/sqrt
       gpuArray/sqrt

    Reference page in Help browser
       doc sqrt

fx >>
```

Matrices in MATLAB

- MATLAB is designed to operate primarily on whole matrices and arrays.
- All MATLAB variables are multidimensional arrays.

- Array creation:

  `a = [1 2 3 4]` returns

  ```
  a =
       1     2     3     4
  ```

  which is a row vector.

Matrices in MATLAB

- Creating a matrix that has multiple rows:

```
a = [1 2 3; 4 5 6; 7 8 10]
```

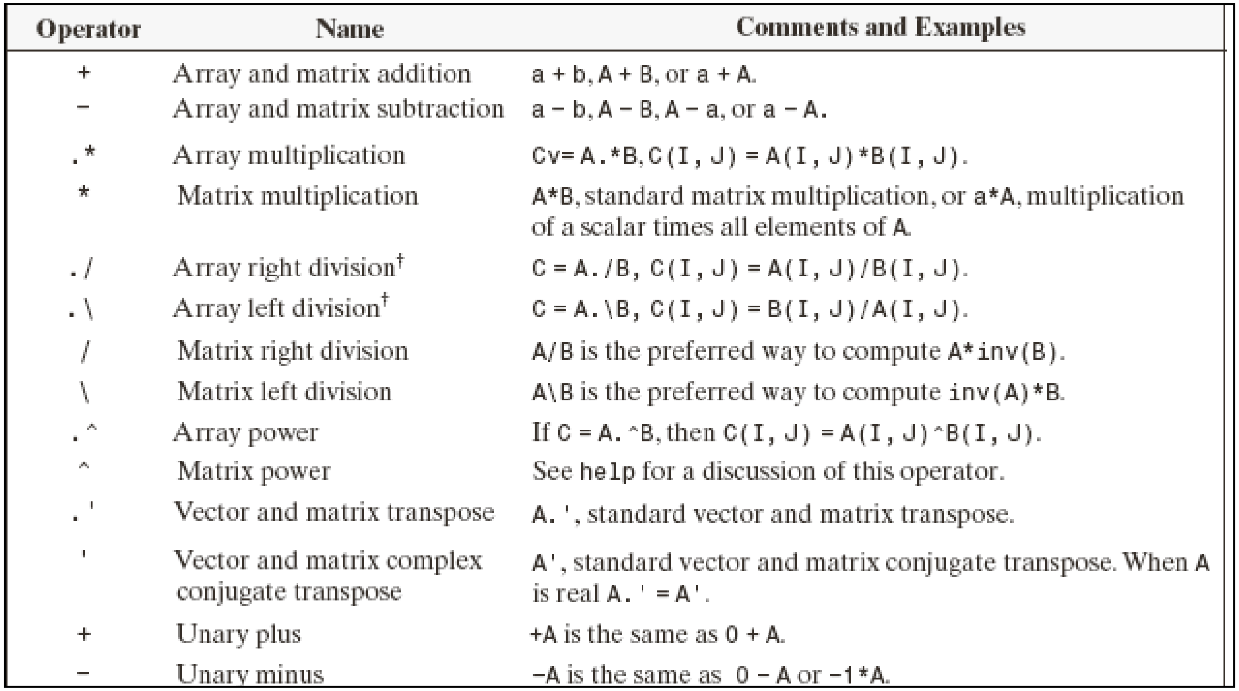
returns

```
a =
   1   2   3
   4   5   6
   7   8  10
```

- Special matrices:
  - `zeros(n,m), ones(n,m), eye(n,m), rand(), randn()`

Basic Operations on Matrices

- Operators are defined on matrices.
- Element-wise operators are defined with a preceding dot.



Operator	Name	Comments and Examples
+	Array and matrix addition	a + b, A + B, or a + A.
−	Array and matrix subtraction	a − b, A − B, A − a, or a − A.
.*	Array multiplication	Cv= A.*B, C(I, J) = A(I, J)*B(I, J).
*	Matrix multiplication	A*B, standard matrix multiplication, or a*A, multiplication of a scalar times all elements of A.
./	Array right division†	C = A./B, C(I, J) = A(I, J)/B(I, J).
.\	Array left division†	C = A.\B, C(I, J) = B(I, J)/A(I, J).
/	Matrix right division	A/B is the preferred way to compute A*inv(B).
\	Matrix left division	A\B is the preferred way to compute inv(A)*B.
.^	Array power	If C = A.^B, then C(I, J) = A(I, J)^B(I, J).
^	Matrix power	See help for a discussion of this operator.
.'	Vector and matrix transpose	A.', standard vector and matrix transpose.
'	Vector and matrix complex conjugate transpose	A', standard vector and matrix conjugate transpose. When A is real A.' = A'.
+	Unary plus	+A is the same as 0 + A.
−	Unary minus	−A is the same as 0 − A or −1*A.

Digital Image Processing Using MATLAB, R. C. Gonzalez, R. E. Woods, S. L. Eddins

More Matrix Operations

- Eigenvectors and eigenvalues of matrix A:

```
[v,d] = eig(A)
```

Where `v`’s columns are the eigenvectors, and the diagonal elements of `d` are the eigenvalues.

- Accessing matrix elements:

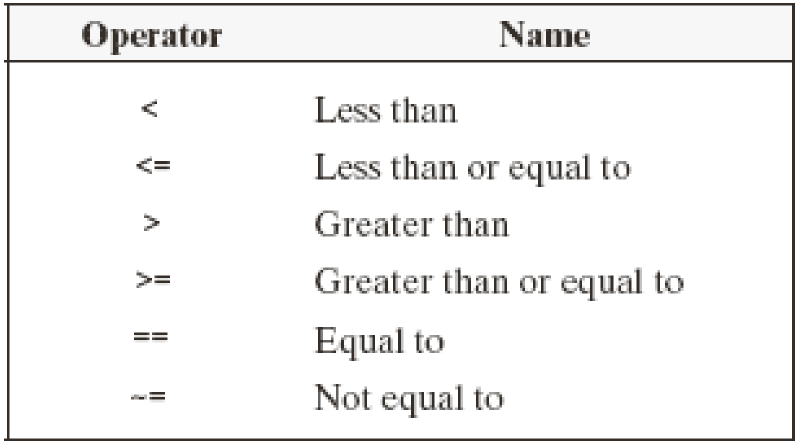
`A`’s first column: `A(:,1)`

`A`’s second row: `A(2,:)`

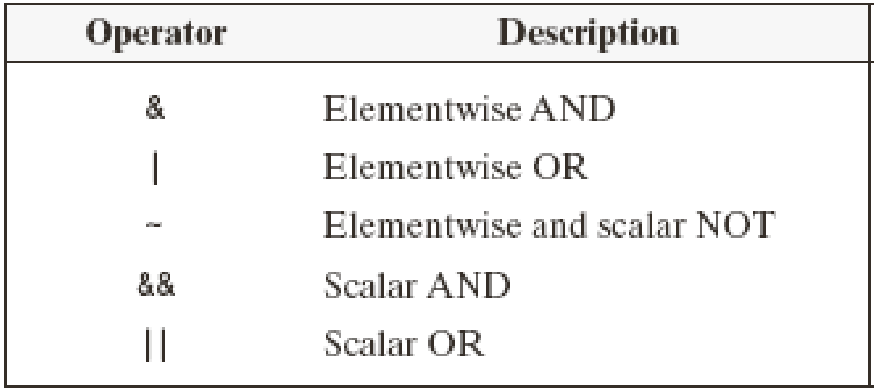
`A`’s last element: `A(end,end)`

Digital Image Processing Using MATLAB, R. C. Gonzalez, R. E. Woods, S. L. Eddins

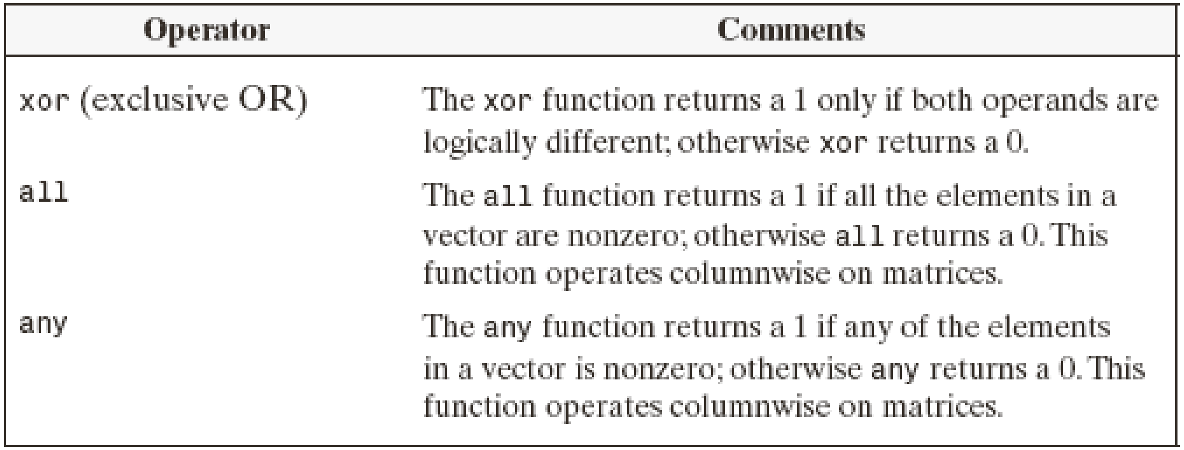
Relational and Logical Operators



Operator	Name
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
~=	Not equal to



Operator	Description
&	Elementwise AND
|	Elementwise OR
~	Elementwise and scalar NOT
&&	Scalar AND
||	Scalar OR



Operator	Comments
xor (exclusive OR)	The xor function returns a 1 only if both operands are logically different; otherwise xor returns a 0.
all	The all function returns a 1 if all the elements in a vector are nonzero; otherwise all returns a 0. This function operates columnwise on matrices.
any	The any function returns a 1 if any of the elements in a vector is nonzero; otherwise any returns a 0. This function operates columnwise on matrices.

Digital Image Processing Using MATLAB, R. C. Gonzalez, R. E. Woods, S. L. Eddins

Basic Plots

- Line plot of `y` vs. `x` (of the same length):

```
plot(x,y,’b.’)
```

- Two plots on the same axes:

```
plot(x,y1,’b.’), hold on
plot(x,y2,’r.’)
```

- Sub-plots:

```
subplot(211), plot(x,y1,’b.’)
subplot(212), plot(x,y2,’b.’)
```

Digital Image Processing Using MATLAB, R. C. Gonzalez, R. E. Woods, S. L. Eddins

Using ‘find’

- Example:

```
>>A=[8 0 7; 9 2 1], idx=find(A<4)
   A=
      8 0 7
      9 2 1
   idx=
      3
      4
      6
```

‘if’ and ‘for’ Commands

```
if <condition>
      <statement>;
elseif <condition>
      <statement>;
else <statement>;
end

for <var> = <interval>
      <statement>;
end
```

Image Processing Toolbox

“Image Processing Toolbox™ provides a comprehensive set of reference-standard algorithms and graphical tools for image processing, analysis, visualization, and algorithm development. You can perform image enhancement, image deblurring, feature detection, noise reduction, image segmentation, spatial transformations, and image registration…

Image Processing Toolbox supports a diverse set of image types...

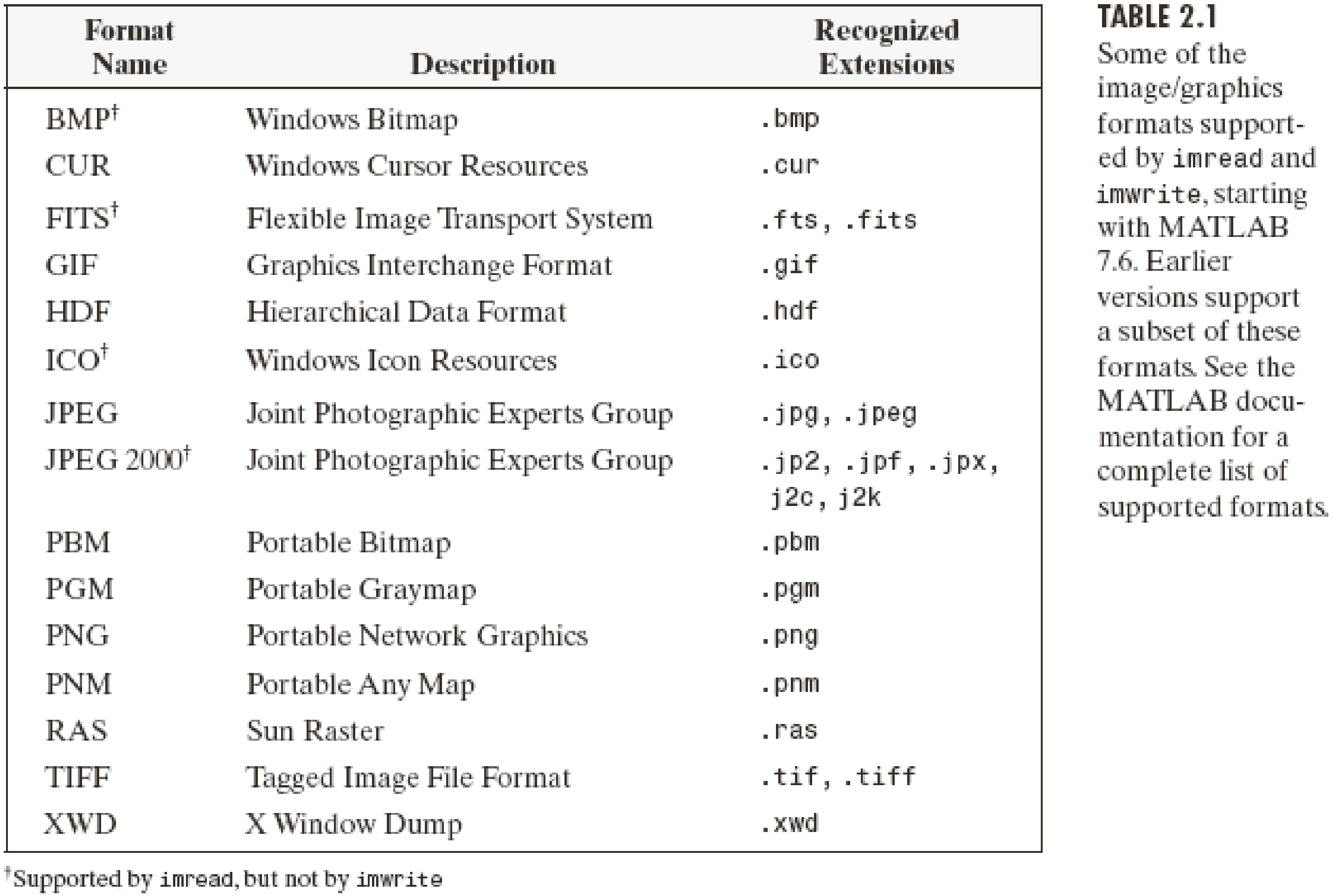
With toolbox algorithms you can restore degraded images, detect and measure features, analyze shapes and textures, and adjust color balance.”

Image Processing Toolbox

Key Features

- Image enhancement, filtering, and deblurring
- Image analysis, including segmentation, morphology, feature extraction, and measurement
- Spatial transformations and intensity-based image registration methods
- Image transforms, including FFT, DCT, Radon, and fan-beam projection
- Interactive tools, including ROI selections, histograms, and distance measurements

Image formats in MATLAB



Format Name	Description	Recognized Extensions
BMP[†]	Windows Bitmap	`.bmp`
CUR	Windows Cursor Resources	`.cur`
FITS[†]	Flexible Image Transport System	`.fts, .fits`
GIF	Graphics Interchange Format	`.gif`
HDF	Hierarchical Data Format	`.hdf`
ICO[†]	Windows Icon Resources	`.ico`
JPEG	Joint Photographic Experts Group	`.jpg, .jpeg`
JPEG 2000[†]	Joint Photographic Experts Group	`.jp2, .jpf, .jpx, j2c, j2k`
PBM	Portable Bitmap	`.pbm`
PGM	Portable Graymap	`.pgm`
PNG	Portable Network Graphics	`.png`
PNM	Portable Any Map	`.pnm`
RAS	Sun Raster	`.ras`
TIFF	Tagged Image File Format	`.tif, .tiff`
XWD	X Window Dump	`.xwd`

[†] Supported by `imread`, but not by `imwrite`

TABLE 2.1 Some of the image/graphics formats supported by `imread` and `imwrite`, starting with MATLAB 7.6. Earlier versions support a subset of these formats. See the MATLAB documentation for a complete list of supported formats.

Digital Image Processing Using MATLAB, R. C. Gonzalez, R. E. Woods, S. L. Eddins

Classes

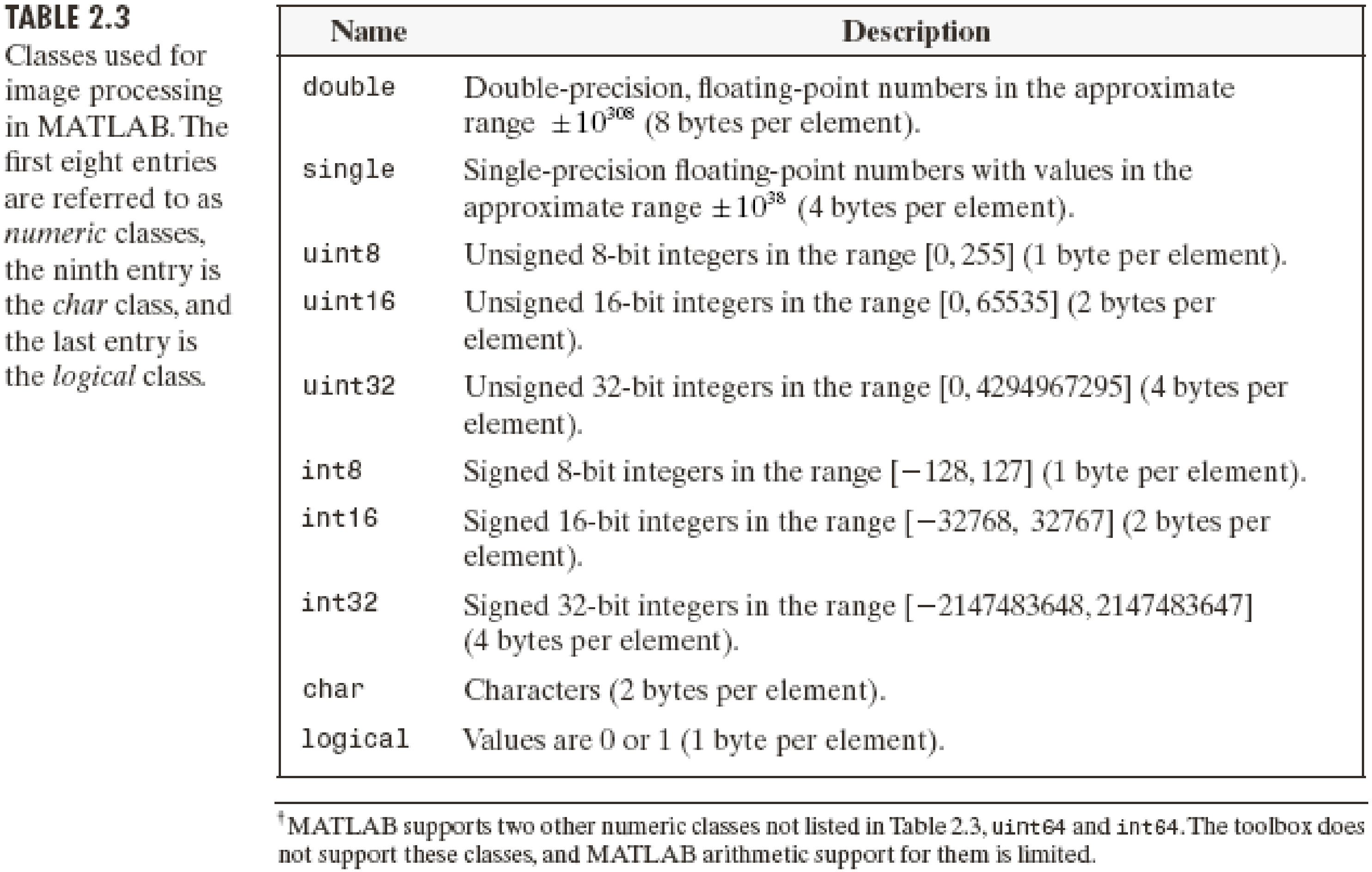


TABLE 2.3 Classes used for image processing in MATLAB. The first eight entries are referred to as *numeric* classes, the ninth entry is the *char* class, and the last entry is the *logical* class.

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

[†] MATLAB supports two other numeric classes not listed in Table 2.3, `uint64` and `int64`. The toolbox does not support these classes, and MATLAB arithmetic support for them is limited.

Digital Image Processing Using MATLAB, R. C. Gonzalez, R. E. Woods, S. L. Eddins

Images in MATLAB

- Binary images: {0,1}
- Intensity images : [0,1] or `uint8`, `double` etc.
- RGB images : m × n × 3

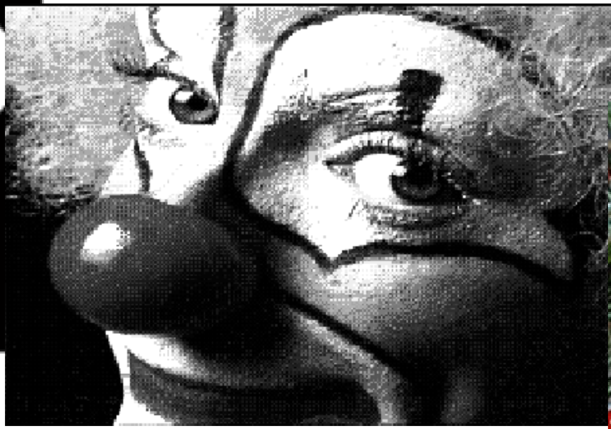


Image Import and Export

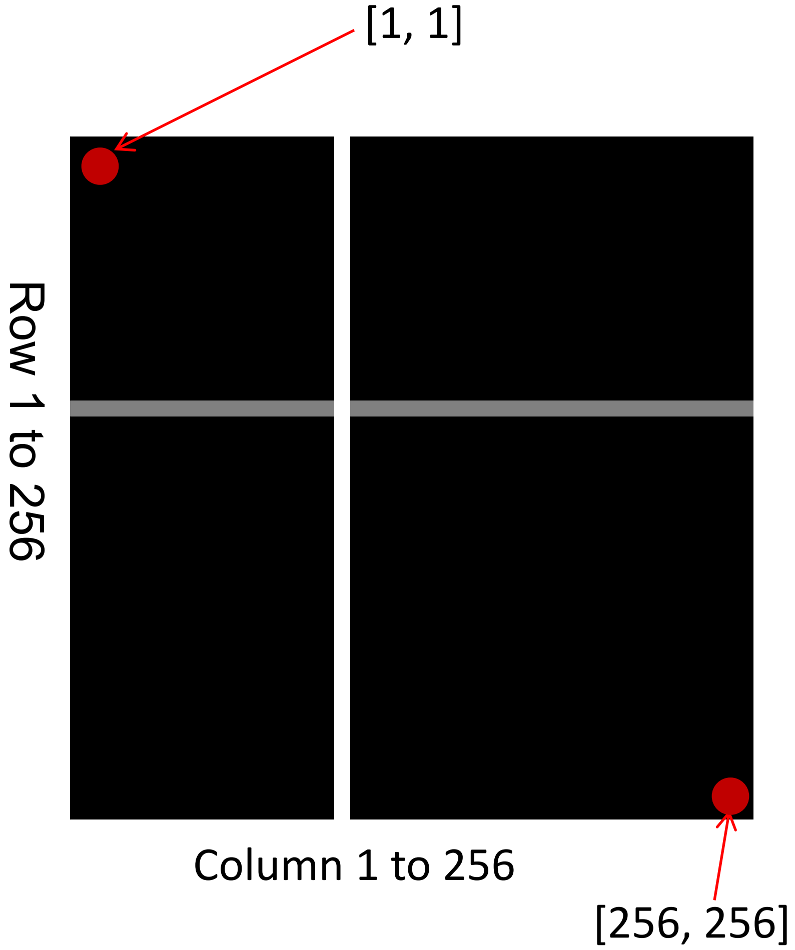
- Read and write images in Matlab

```
im = imread('pout.tif');
imshow(im)
```

```
imwrite(im, 'output.bmp', 'bmp');
```

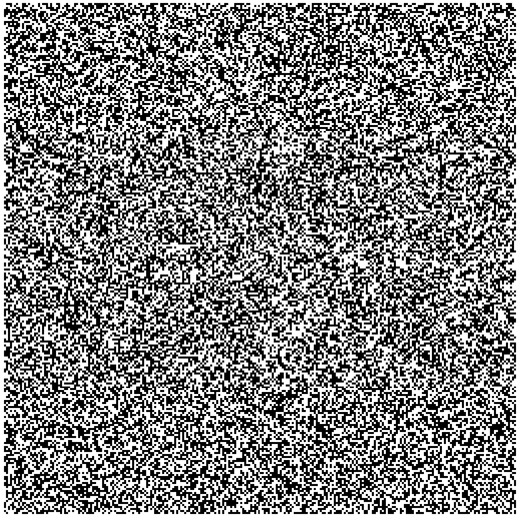
Grayscale Image

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



Binary Image

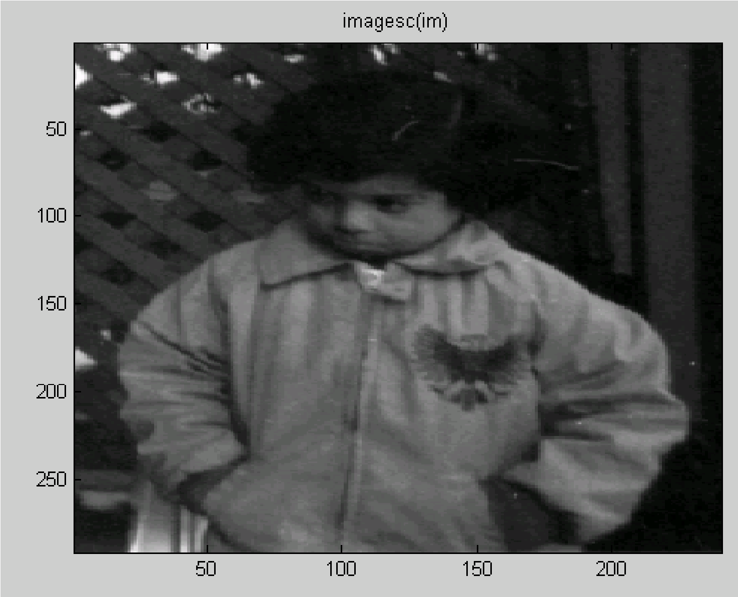
```
im = rand(row, col);
im = im<0.5;
figure;
imshow(im);
```

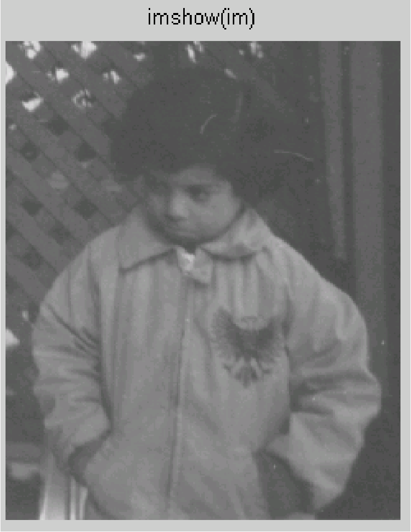


```
> whos im
 Name      Size             Bytes Class    Attributes
 im      256x256            65536 logical
```

Image Display

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





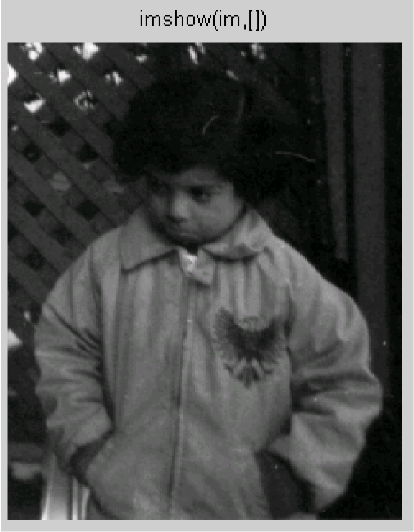


Image Display

```
figure; imagesc(im)
title('imagesc(im)')
colorbar
colormap(gray)
```

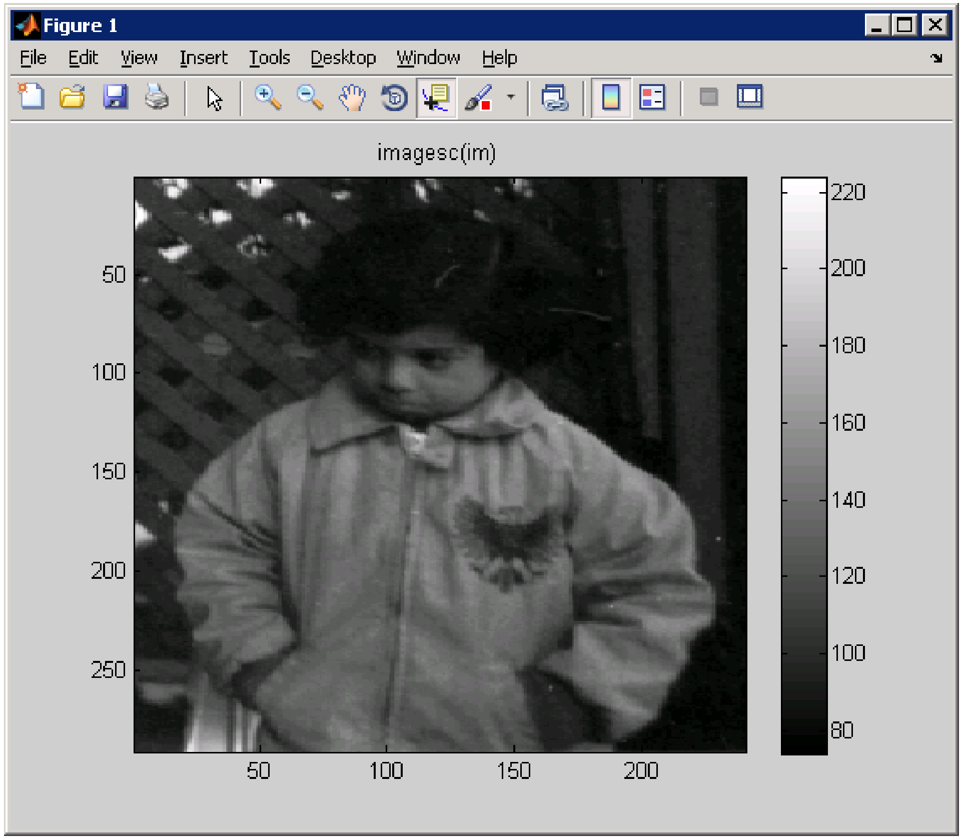
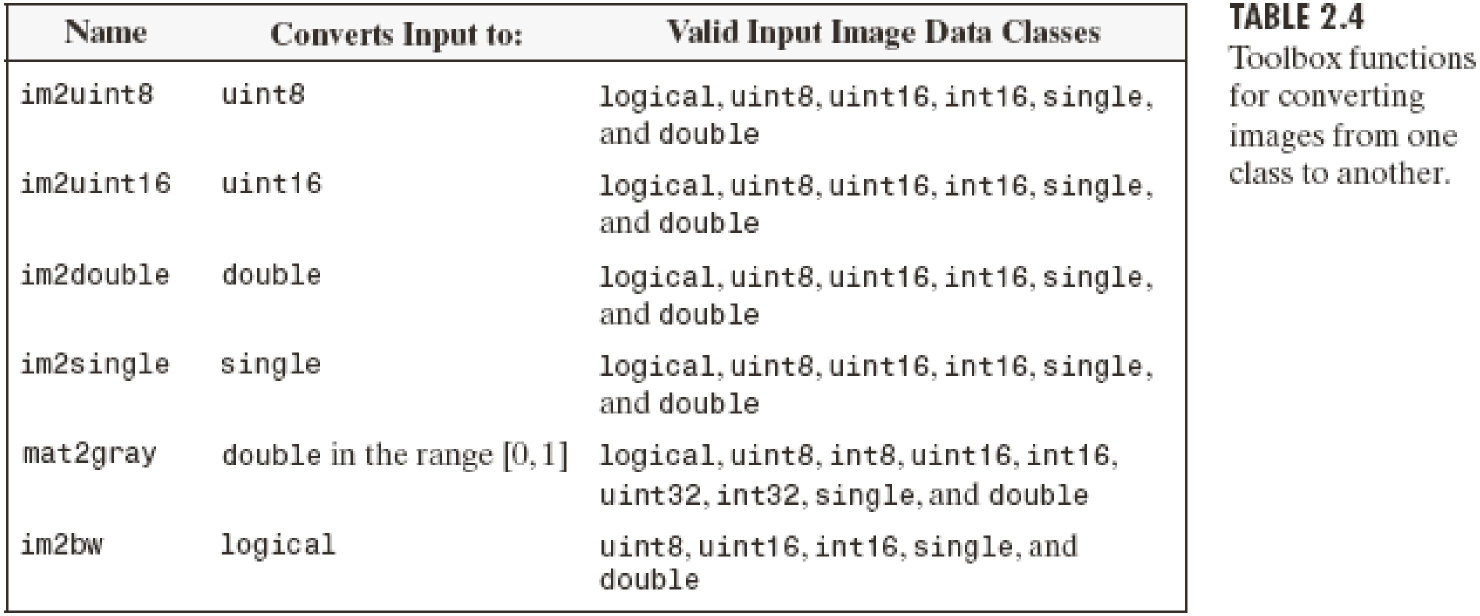


Image Conversion



Name	Converts Input to:	Valid Input Image Data Classes
`im2uint8`	`uint8`	`logical`, `uint8`, `uint16`, `int16`, `single`, and `double`
`im2uint16`	`uint16`	`logical`, `uint8`, `uint16`, `int16`, `single`, and `double`
`im2double`	`double`	`logical`, `uint8`, `uint16`, `int16`, `single`, and `double`
`im2single`	`single`	`logical`, `uint8`, `uint16`, `int16`, `single`, and `double`
`mat2gray`	`double` in the range [0, 1]	`logical`, `uint8`, `int8`, `uint16`, `int16`, `uint32`, `int32`, `single`, and `double`
`im2bw`	`logical`	`uint8`, `uint16`, `int16`, `single`, and `double`

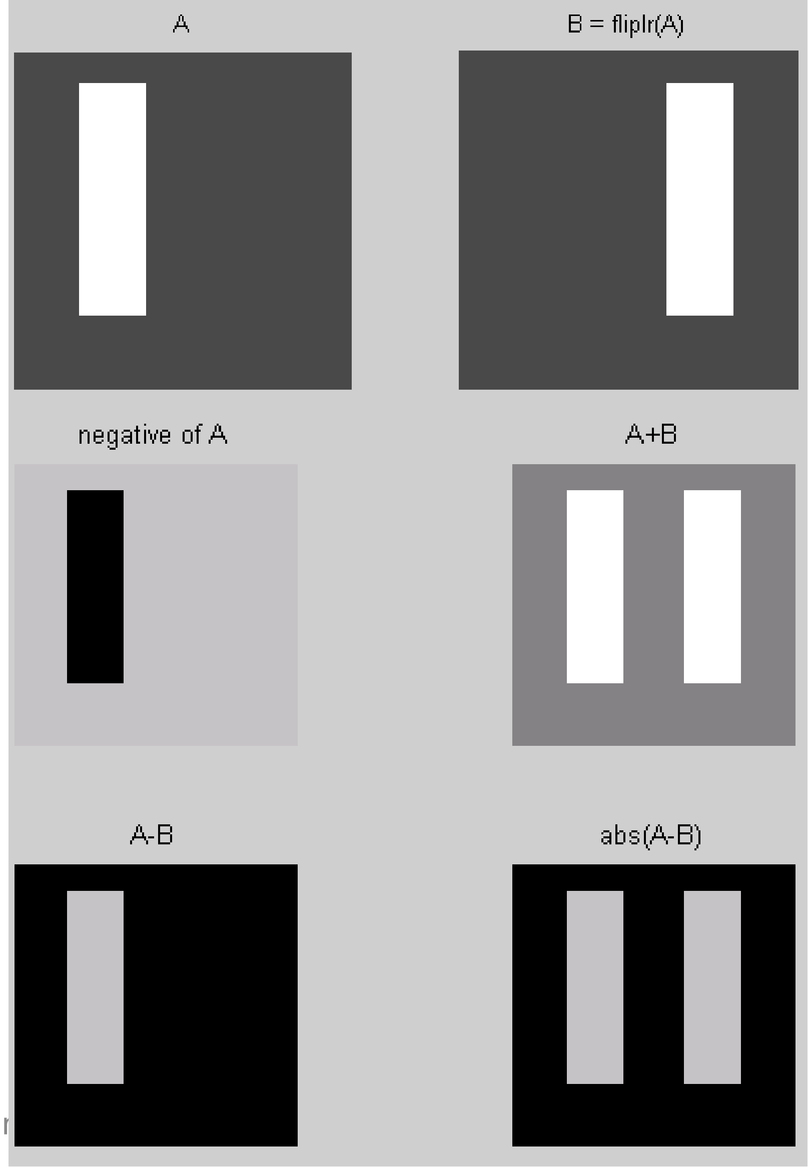
TABLE 2.4 Toolbox functions for converting images from one class to another.

- Also `rgb2gray` - RGB image to grayscale

Digital Image Processing Using MATLAB, R. C. Gonzalez, R. E. Woods, S. L. Eddins

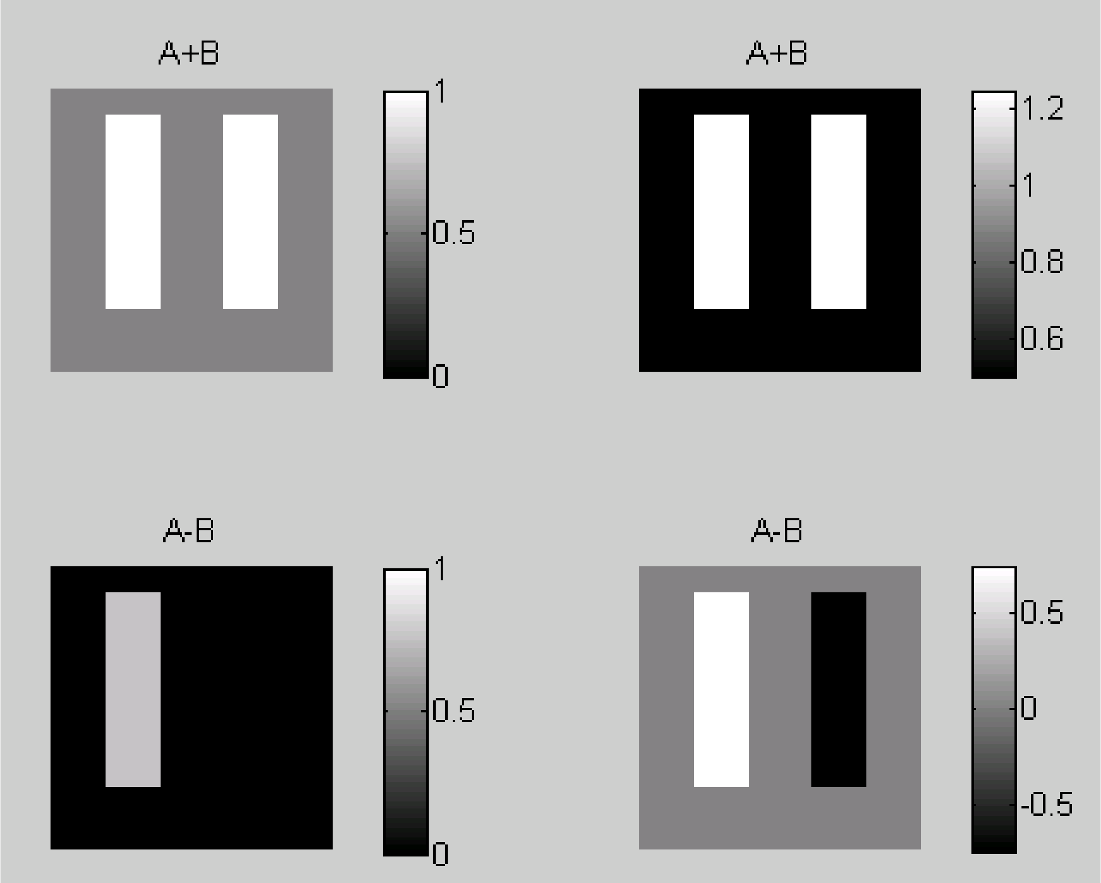
Example

```
A = 0.25 * ones(256);
A(25:200, 50:100)=1;
B = fliplr(A);

figure;
subplot(221)
imshow(1 - A);
subplot(222)
imshow(A + B)
subplot(223)
imshow(A - B)
subplot(224)
imshow(abs(A - B))
```



imshow and Clipping

```
figure;
subplot(221)
imshow(A + B);
title('A+B')
subplot(222)
imshow(A + B,[])
title('A+B')
subplot(223)
imshow(A - B)
title('A-B')
subplot(224)
imshow(A - B,[])
title('A-B')
```



More Useful Functions

- `imcrop`: Crop
- `imresize`: Resize image
- `imrotate`: Rotate image
- `imhist`: Display histogram of image data

Image enhancement

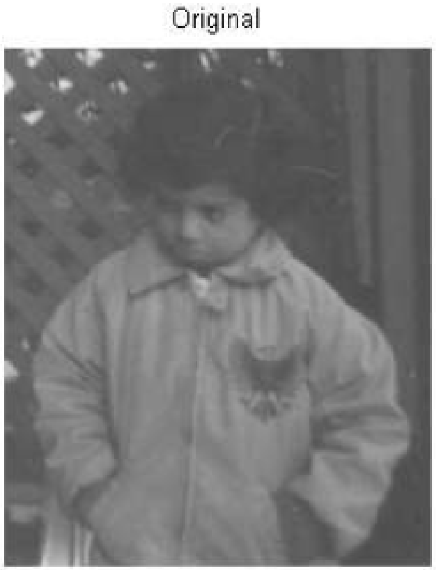
- **Step 1: Load Images**
  Read in a grayscale image

```
pout = imread('pout.tif');
```

- **Step 2: Display image and histogram**

```
imshow(pout);
title('Original');
```

```
figure, imhist(pout)
```



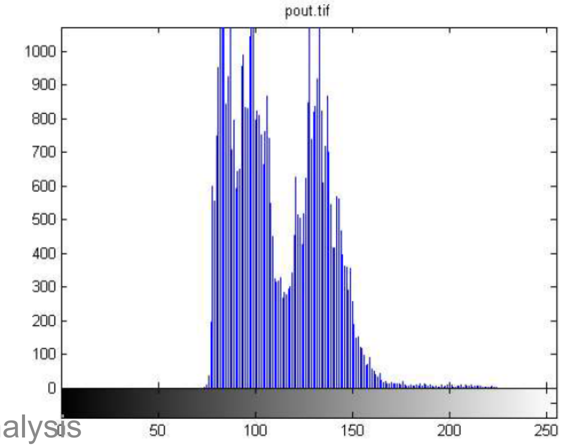
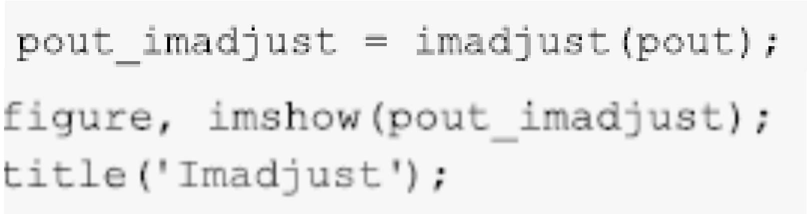
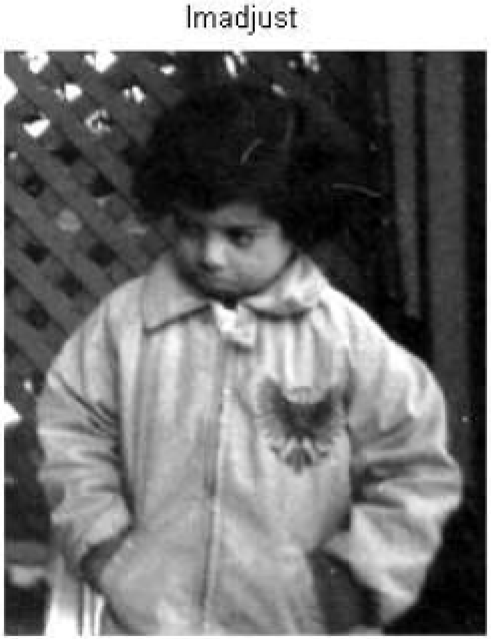


Image enhancement

- **Step 3:** `imadjust`: increases the contrast of the image by mapping the values of the input intensity image to new values such that, by default, 1% of the data is saturated at low and high intensities of the input data.



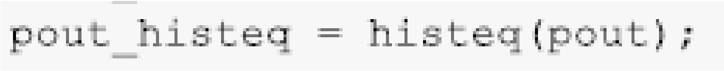
```
pout_imadjust = imadjust(pout);
figure, imshow(pout_imadjust);
title('Imadjust');
```



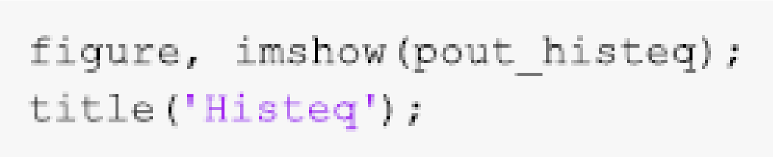
Imadjust

Image enhancement

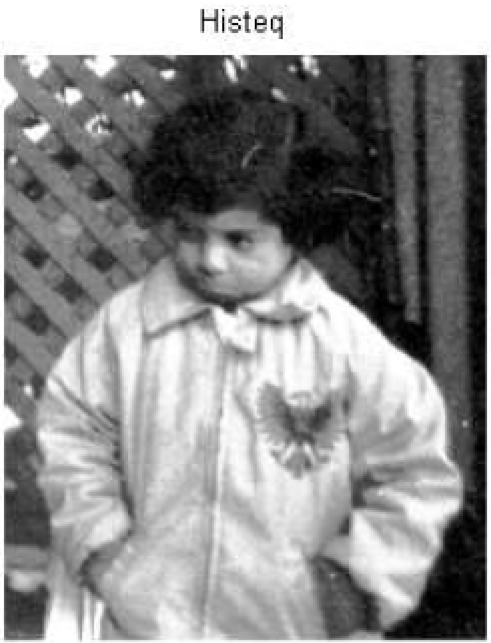
- **Step 4:** `histeq`: performs histogram equalization. It enhances the contrast of images by transforming the values in an intensity image so that the histogram of the output image approximately matches a specified histogram (uniform distribution by default).



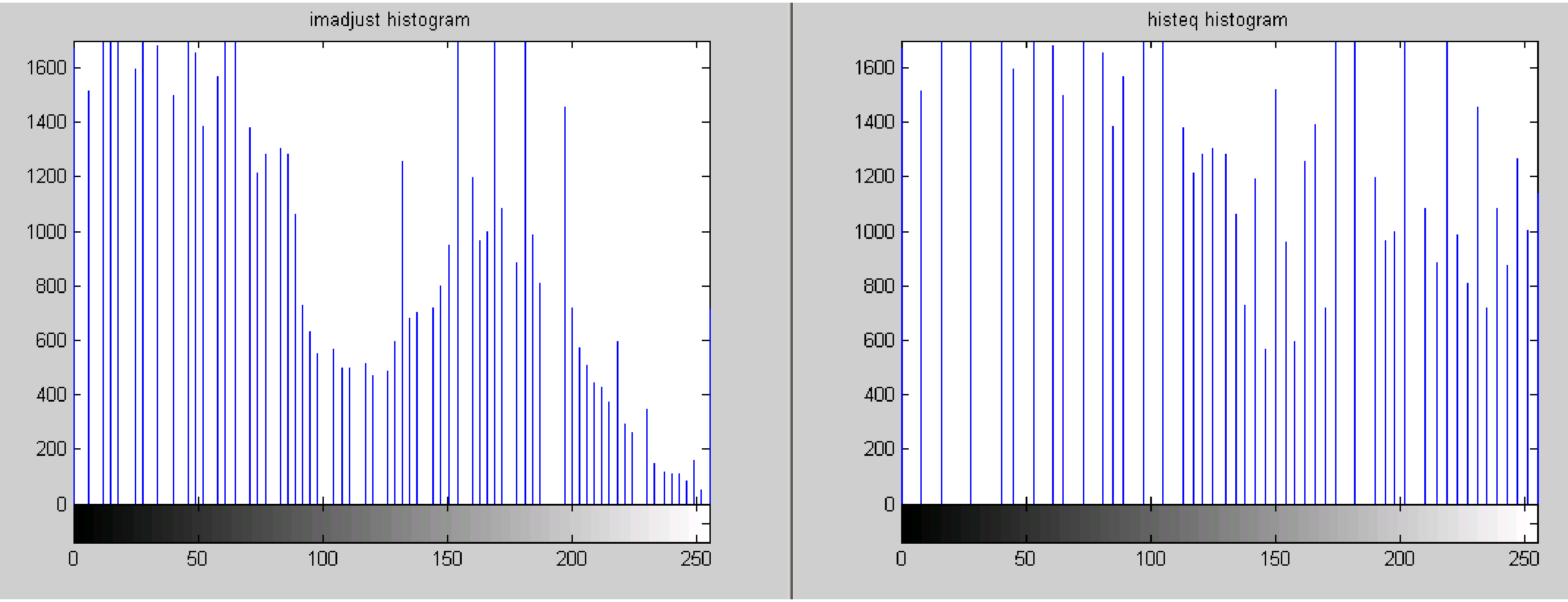
```
pout_histeq = histeq(pout);
```



```
figure, imshow(pout_histeq);
title('Histeq');
```



Histogram Comparison



Performance Issues: Vectorization

MATLAB® is optimized for operations involving matrices and vectors. The process of revising loop-based, scalar-oriented code to use MATLAB matrix and vector operations is called vectorization. Vectorizing your code is worthwhile for several reasons:

1. **Appearance**: Vectorized mathematical code appears more like the mathematical expressions found in textbooks, making the code easier to understand.
2. **Less Error Prone**: Without loops, vectorized code is often shorter. Fewer lines of code mean fewer opportunities to introduce programming errors.
3. **Performance**: Vectorized code often runs much faster than the corresponding code containing loops.

http://www.mathworks.com/help/matlab/matlab_prog/vectorization.html

Vectorization

Example:

Given two images of same size, im1 and im2, output the mean of the images

Using Loops:

```
tic
for i = 1 : size(im1, 1)
  for j = 1 : size(im1, 2)
    for k = 1 : size(im1, 3)
      output(i, j, k) = (im1(i, j, k) + im2(i, j, k))/2;
    end
  end
end
toc
```

Elapsed time is 0.100722 seconds

Vectorization and **Pre-allocation**

Example:

Given two images of same size, im1 and im2, output the mean of the images

Using Loops:

```
tic
output = zeros(size(im1));
for i = 1 : size(im1, 1)
  for j = 1 : size(im1, 2)
    for k = 1 : size(im1, 3)
      output(i, j, k) = (im1(i, j, k) + im2(i, j, k))/2;
    end
  end
end
toc
```

Elapsed time is 0.074812 seconds.

Pre-allocation improves the run-time.

Vectorization (cont.)

Vectorized

```
tic
output = (im1 + im2)/2;
toc
```

Elapsed time is 0.000503 seconds

Computation is much faster!

File Handling

Saving your environment can be very important. The following commands will help you save variables to MAT files:

```
save(filename, variables);
load(filename, variables);
```

For example: save(‘myfile.mat’,‘var1’,‘var2’);

clear – clear variables from your workspace: clear var1

Close all – close all open figures.

clc – clear the screen.