A Quick Tutorial on MATLAB

MATLAB

- MATLAB is a software package for doing numerical computation. It was originally designed for solving linear algebra type problems using matrices. It's name is derived from MATrix LABoratory.

- MATLAB has since been expanded and now has built-in functions for solving problems requiring data analysis, signal processing, optimization, and several other types of scientific computations. It also contains functions for 2-D and 3-D graphics and animation.

MATLAB Variable names

- Variable names are case sensitive.
- Variable names can contain up to 63 characters (as of MATLAB 6.5 and newer).
- Variable names must start with a letter and can be followed by letters, digits and underscores.

  Examples :

```
>> x = 2;
>> abc_123 = 0.005;
>> 1ab = 2;
Error: Unexpected MATLAB expression
```

MATLAB Special Variables

pi	Value of π
eps	Smallest incremental number
inf	Infinity
NaN	Not a number e.g. 0/0
i and j	i = j = square root of -1 **如果写成复数形式的话，3+5i**
realmin	The smallest usable positive real number
realmax	The largest usable positive real number

MATLAB Relational operators

- MATLAB supports six relational operators.

Less Than	<	
Less Than or Equal	<=	
Greater Than	>	
Greater Than or Equal	>=	
Equal To	==	
Not Equal To	~=	(NOT != like in C)

MATLAB Logical Operators

MATLAB supports three logical operators.

```
not     ~     % highest precedence
and     &     % equal precedence with or
or      |     % equal precedence with and
```

Matrices and MATLAB

MATLAB Matrices

- MATLAB treats all variables as matrices. For our purposes a matrix can be thought of as an array, in fact, that is how it is stored.
- Vectors are special forms of matrices and contain only one row OR one column.
- Scalars are matrices with only one row AND one column

Generating Matrices

- A scalar can be created in MATLAB as follows:

```
>> x = 23;
```

- A matrix with only one row is called a row vector. A row vector can be created in MATLAB as follows (note the commas):

```
>> y = [12,10,-3]
y =
    12    10   -3
```

- A matrix with only one column is called a column vector. A column vector can be created in MATLAB as follows:

```
>> z = [12;10;-3]
z =
    12
    10
    -3
```

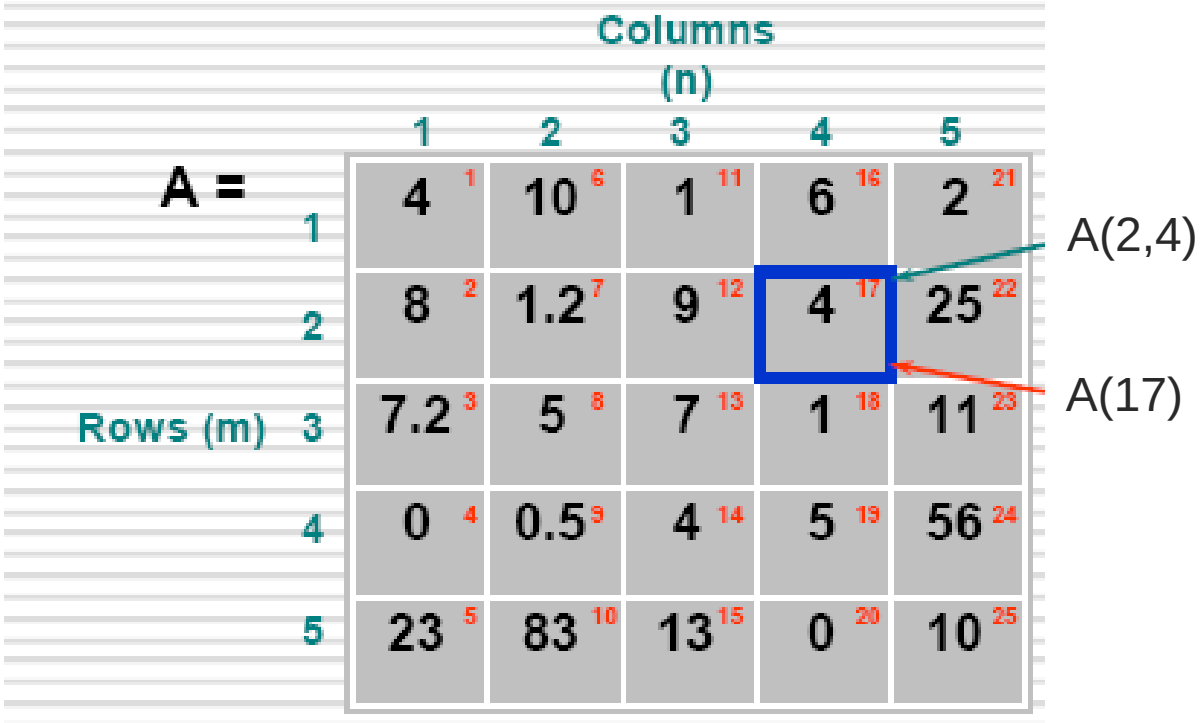
Generating Matrices

- MATLAB treats row vector and column vector very differently
- A matrix can be created in MATLAB as follows (note the commas and semicolons)

```
>> X = [1,2,3;4,5,6;7,8,9]
X =
     1     2     3
     4     5     6
     7     8     9
```

Matrices must be rectangular!

The Matrix in MATLAB



Note: Unlike C, MATLAB's indices start from 1

Extracting a Sub-matrix

- A portion of a matrix can be extracted and stored in a smaller matrix by specifying the names of both matrices and the rows and columns to extract. The syntax is:

```
sub_matrix = matrix ( r1 : r2 , c1 : c2 ) ;
```

where r1 and r2 specify the beginning and ending rows and c1 and c2 specify the beginning and ending columns to be extracted to make the new matrix.

Extracting a Sub-matrix

- Example :

```
>> X = [1,2,3;4,5,6;7,8,9]
X =
    1    2    3
    4    5    6
    7    8    9

>> X22 = X(1:2 , 2:3)
X22 =
       2    3
       5    6
```

```
>> X13 = X(3,1:3)
X13 =
       7    8    9

>> X21 = X(1:2,1)
X21 =
       1
       4
```

Matrix Extension

- ```
  >> a = [1,2i,0.56]
  a =
      1   0+2i   0.56
  >> a(2,4) = 0.1
  a =
      1   0+2i   0.56   0
      0    0      0    0.1
  ```

- repmat – replicates and tiles a matrix

  ```
  >> b = [1,2;3,4]
  b =
      1   2
      3   4
  >> b_rep = repmat(b,1,2)
  b_rep =
          1   2   1   2
          3   4   3   4
  ```

  相当于 [b b]

  1-by-2

- Concatenation

  ```
  >> a = [1,2;3,4]
  a =
      1   2
      3   4
  >> a_cat =[a,2*a;3*a,2*a]
  a_cat =
          1   2    2    4
          3   4    6    8
          3   6    2    4
          9   12   6    8
  ```

  NOTE: The resulting matrix must be rectangular
  ```

Matrix Addition

- Increment all the elements of a matrix by a single value

```
>> x = [1,2;3,4]
x =
    1    2
     3    4
>> y = x + 5
y =
    6    7
    8    9
```

- Adding two matrices

```
>> xsy = x + y
xsy =
       7     9
      11    13
>> z = [1,0.3]
z =
    1   0.3
>> xsz = x + z
??? Error using => plus
Matrix dimensions must
agree
```

Matrix Multiplication

- Matrix multiplication

```
>> a = [1,2;3,4];      (2x2)
>> b = [1,1];          (1x2)
>> c = b*a
c =
      4   6

>> c = a*b
??? Error using ==> mtimes
Inner matrix dimensions
must agree.
```

- Element wise multiplication

```
>> a = [1,2;3,4];
>> b = [1,½;1/3,¼];
>> c = a.*b
c =
    1   1
    1   1
```

Matrix Element wise operations

- ```
  >> a = [1,2;1,3];
  >> b = [2,2;2,1];
  ```

- Element wise division

  ```
  >> c = a./b
  c =
      0.5    1
      0.5    3
  ```

- Element wise multiplication

  ```
  >> c = a.*b
  c =
      2     4
      2     3
  ```

- Element wise power operation

  ```
  >> c = a.^2
  c =
      1     4
      1     9

  >> c = a.^b
  c =
      1     4
      1     3
  ```
  ```

Matrix Manipulation functions

如果一个变量，则约定为方阵

- zeros : creates an array of all zeros, Ex: x = zeros(3,2)
- ones : creates an array of all ones, Ex: x = ones(2)
- eye : creates an identity matrix, Ex: x = eye(3)
- rand : generates uniformly distributed random numbers in [0,1]
- diag : Diagonal matrices and diagonal of a matrix
- size : returns array dimensions
- length : returns length of a vector (row or column)
- det : Matrix determinant
- inv : matrix inverse
- eig : evaluates eigenvalues and eigenvectors
- rank : rank of a matrix
- find : searches for the given values in an array/matrix.

MATLAB inbuilt math functions

Elementary Math functions

- abs - finds absolute value of all elements in the matrix
- sign - signum function
- sin,cos,… - Trignometric functions
- asin,acos… - Inverse trignometric functions
- exp - Exponential
- log,log10 - natural logarithm, logarithm (base 10)
- ceil,floor - round towards +infinity, -infinity respectively
- round - round towards nearest integer
- real,imag - real and imaginary part of a complex matrix
- sort - sort elements in ascending order

Elementary Math functions

- sum,prod - summation and product of elements
- max,min - maximum and minimum of arrays
- mean,median – average and median of arrays
- std,var - Standard deviation and variance

and many more…

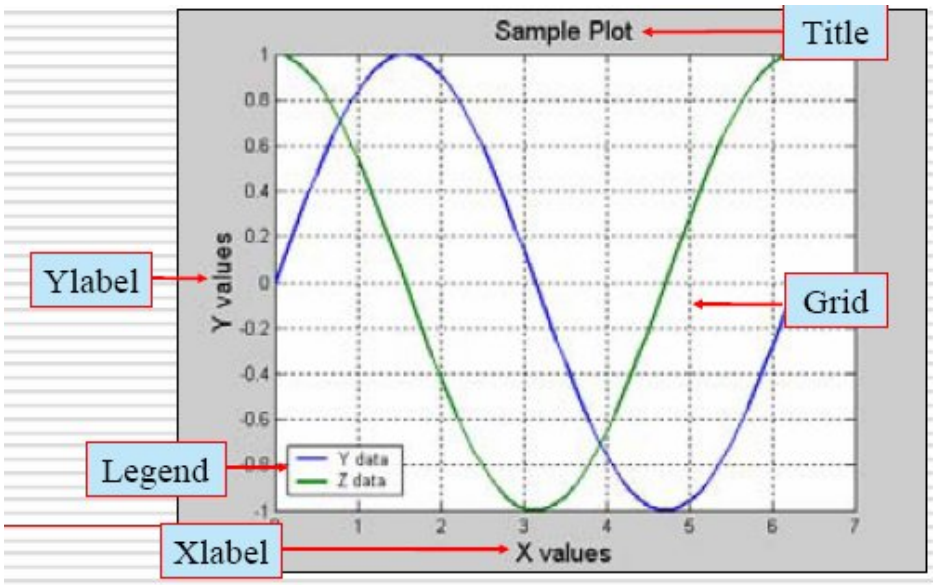
Graphics Fundamentals

2D Plotting

- Example 1: Plot sin(x) and cos(x) over [0,2π], on the same plot with different colours

Method 1:

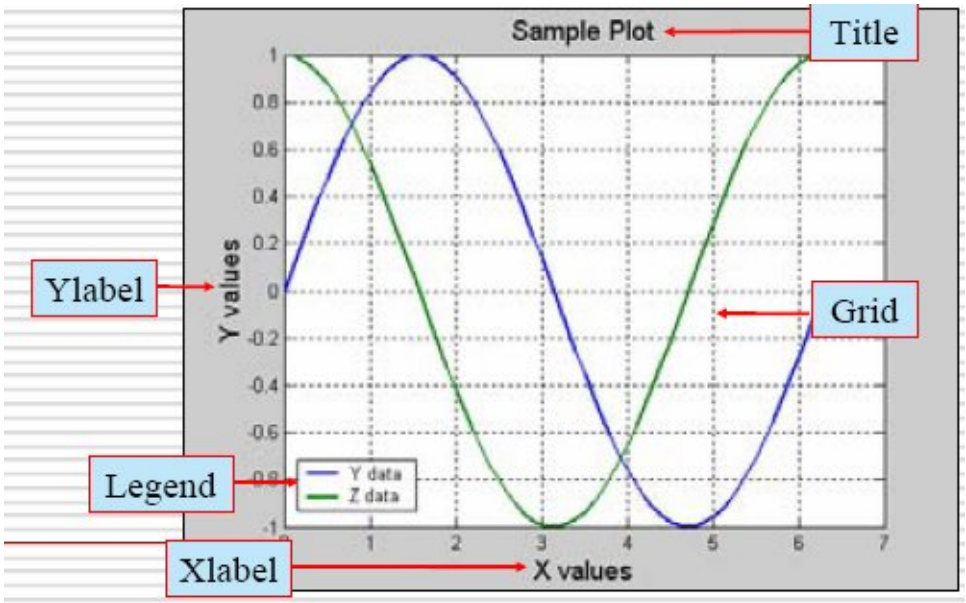
```
>> x = linspace(0,2*pi,1000);
>> y = sin(x);
>> z = cos(x);
>> hold on;
>> plot(x,y,'b');
>> plot(x,z,'g');
>> xlabel 'X values';
>> ylabel 'Y values';
>> title 'Sample Plot';
>> legend ('Y data','Z data');
>> hold off;
```



2D Plotting

Method 2:

```
>> x = 0:0.01:2*pi;
>> y = sin(x);
>> z = cos(x);
>> figure
>> plot (x,y,x,z);
>> xlabel 'X values';
>> ylabel 'Y values';
>> title 'Sample Plot';
>> legend ('Y data','Z data');
>> grid on;
```

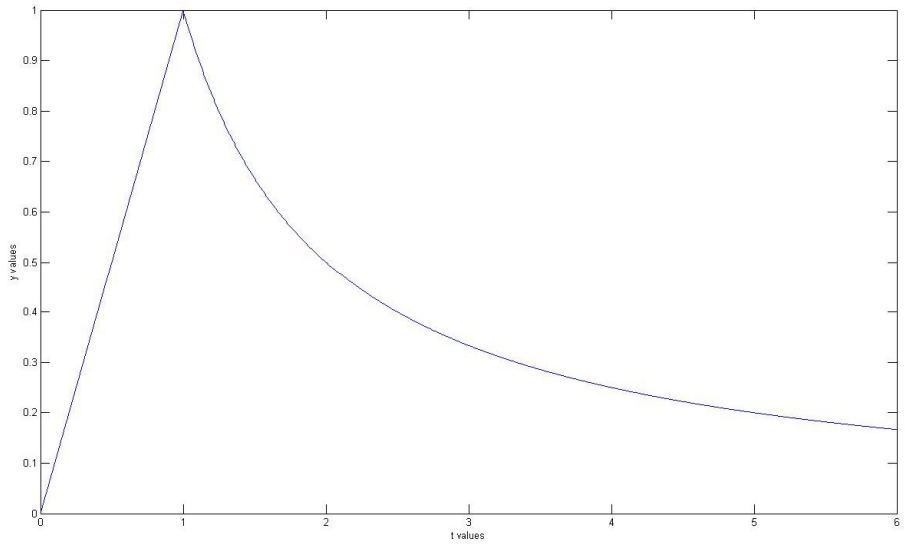


2D Plotting

- Example 2: Plot the following function $y = \begin{cases} t & 0 \le t \le 1 \\ 1/t & 1 \le t \le 6 \end{cases}$

Method 1:

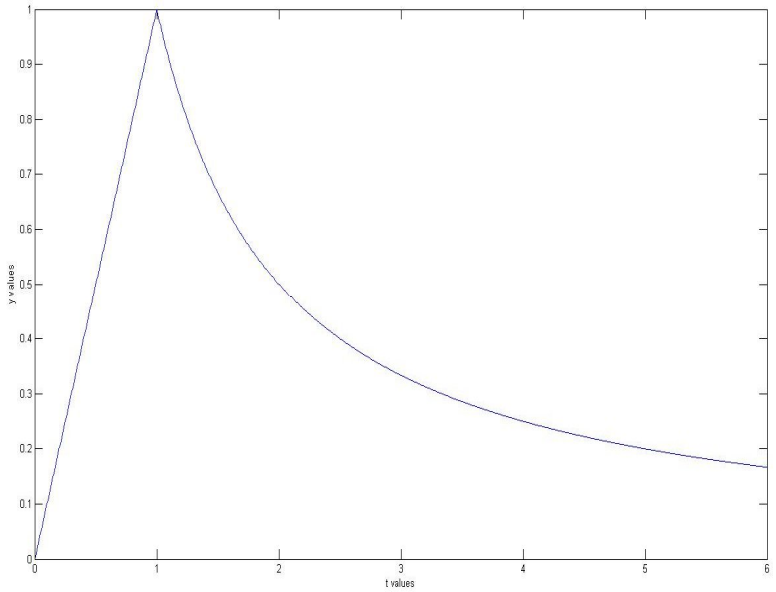
```
>> t1 = linspace(0,1,1000);
>> t2 = linspace(1,6,1000);
>> y1 = t1;
>> y2 = 1./ t2;
>> t = [t1,t2];
>> y = [y1,y2];
>> figure
>> plot(t,y);
>> xlabel ‘t values’, ylabel ‘y values’;
```



2D Plotting

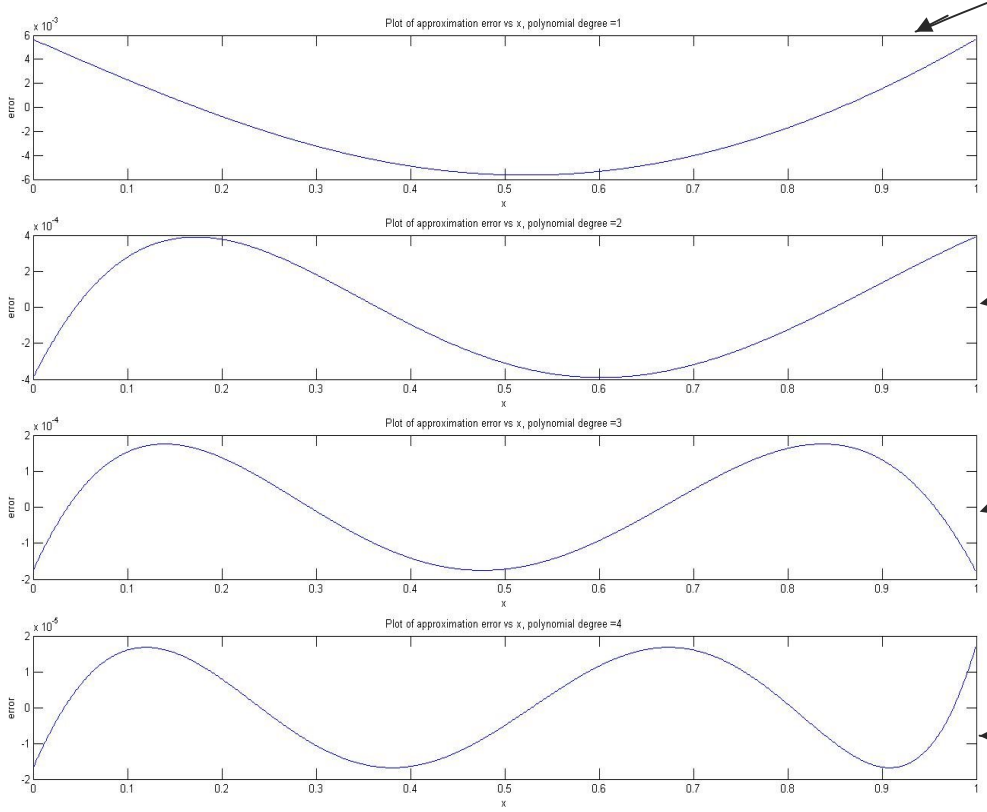
Method 2:

```
>> t = linspace(0,6,1000);
>> y = zeros(1,1000);
>> y(t()<=1) = t(t()<=1);
>> y(t()>1) = 1./ t(t()>1);
>> figure
>> plot(t,y);
>> xlabel‘t values’;
>> ylabel‘y values’;
```



Subplots

- Syntax: subplot (rows, columns, index)




```
>> subplot(4,1,1)
>> …
>> subplot(4,1,2)
>> …
>> subplot(4,1,3)
>> …
>> subplot(4,1,4)
>> …
```

Importing/Exporting Data

Load and Save

- Using load and save

load filename - loads all variables from the file “filename”
load filename x - loads only the variable x from the file
load filename a* - loads all variables starting with ‘a’
for more information, type help load at command prompt

save filename - saves all workspace variables to a binary .mat file named filename.mat
save filename x,y - saves variables x and y in filename.mat
for more information, type help save at command prompt

Import/Export from Excel sheet

- Copy data from an excel sheet

```
>> x = xlsread(filename);
```

% if the file contains numeric values, text and raw data values, then

```
>> [numeric,txt,raw] = xlsread(filename);
```

- Copy data to an excel sheet

```
>>x = xlswrite('c:\matlab\work\data.xls',A,'A2:C4')
```

% will write A to the workbook file, data.xls, and attempt to fit the elements of A into the rectangular worksheet region, A2:C4. On success, ‘x’ will contain ‘1’, while on failure, ‘x’ will contain ‘0’.

for more information, type help xlswrite at command prompt

Read/write from a text file

- Writing onto a text file

```
>> fid = fopen('filename.txt','w');
>> count = fwrite(fid,x);
>> fclose(fid);
```

% creates a file named 'filename.txt' in your workspace and stores the values of variable 'x' in the file. 'count' returns the number of values successfully stored. Do not forget to close the file at the end.

- Read from a text file

```
>> fid = fopen('filename.txt','r');
>> X = fscanf(fid,'%5d');
>> fclose(fid);
```

% opens the file 'filename.txt' which is in your workspace and loads the values in the format '%5d' into the variable x.

Other useful commands: fread, fprintf

Flow Control in MATLAB

Flow control

- MATLAB has five flow control statements

  - if statements
  - switch statements
  - for loops
  - while loops
  - break statements

‘if’ statement

- The general form of the ‘if’ statement is

```
>> if expression
>> …
>> elseif expression
>> …
>> else
>> …
>> end
```

- Example 1:

```
>> if i == j
>>    a(i,j) = 2;
>> elseif i >= j
>>    a(i,j) = 1;
>> else
>>    a(i,j) = 0;
>> end
```

- Example 2:

```
>> if (attn>0.9)&(grade>60)
>>    pass = 1;
>> end
```

‘switch’ statement

- switch Switch among several cases based on expression

- The general form of the switch statement is:

```
>> switch switch_expr
>>   case case_expr1
>>       …
>>   case case_expr2
>>       …
>>   otherwise
>>       …
>> end
```

- Example :

```
>> x = 2, y = 3;
 >> switch x
>>  case x==y
>>   disp('x and y are equal');
>>  case x>y
>>   disp('x is greater than y');
>>  otherwise
>>   disp('x is less than y');
>> end
x is less than y
```

Note: Unlike C, MATLAB doesn’t need BREAKs in each case

‘for’ loop

- for Repeat statements a specific number of times
- The general form of a for statement is

```
>> for variable=expression
>>    …
>>    …
>> end
```

- Example 1:

```
>> for x = 0:0.05:1
>>     printf(‘%d\n’,x);
>> end
```

注意： Matlab中printf 应该是fprintf

双精度

- Example 2:

```
>> a = zeros(n,m);
>> for i = 1:n
>>   for j = 1:m
>>       a(i,j) = 1/(i+j);
>>   end
>> end
```

‘while’ loop

- while Repeat statements an indefinite number of times
- The general form of a while statement is

```
>> while expression
>>    …
>>    …
>> end
```

- Example 1:

```
>> n = 1;
>> y = zeros(1,10);
>> while n <= 10
>>    y(n) = 2*n/(n+1);
>>    n = n+1;
>> end
```

- Example 2:

```
>> x = 1;
>> while x
>>   %execute statements
>> end
```

Note: In MATLAB ‘1’ is synonymous to TRUE and ‘0’ is synonymous to ‘FALSE’

‘break’ statement

- break terminates the execution of for and while loops
- In nested loops, break terminates from the innermost loop only

- Example:

```
>> y = 3;
>> for x = 1:10
>>   fprintf(‘%5d’,x);
>>    if (x>y)
>>        break;
>>    end
>> end
1     2     3     4
```

这里d代表双精度，f代表浮点

Efficient Programming

Efficient Programming in MATLAB

- Avoid using nested loops as far as possible
- In most cases, one can replace nested loops with efficient matrix manipulation.
- Preallocate your arrays when possible
- MATLAB comes with a huge library of in-built functions, use them when necessary
- Avoid using your own functions, MATLAB's functions are more likely to be efficient than yours.

Example 1

- Let x[n] be the input to a non causal FIR filter, with filter coefficients h[n]. Assume both the input values and the filter coefficients are stored in column vectors x,h and are given to you. Compute the output values y[n] for n = 1,2,3 where

$$y[n] = \sum_{k=0}^{19} h[k]x[n+k]$$

Solution

- Method 1:

```
>> y = zeros(1,3);
>> for n = 1:3
>>   for k = 0:19
>>     y(n)= y(n)+h(k)*x(n+k);
>>    end
>> end
```

- Method 2 (avoids inner loop):

```
>> y = zeros(1,3);
>> for n = 1:3
>>     y(n) = h'*x(n:(n+19));
>> end
```

- Method 3 (avoids both the loops):

```
>> X= [x(1:20),x(2:21),x(3:22)];
>> y = h'*X;
```

Example 2

- Compute the value of the following function

$$y(n) = 1^3*(1^3+2^3)*(1^3+2^3+3^3)*\ldots*(1^3+2^3+\ldots+n^3)$$

for n = 1 to 20

Solution

注：在Matlab中，变量temp自动赋初值为0

- Method 1:

```
>> y = zeros(20,1);
>> y(1) = 1;
>> for n = 2:20
>>   for m = 1:n
>>      temp = temp + m^3;
>>   end
>>   y(n) = y(n-1)*temp;
>>   temp = 0
>> end
```

- Method 2 (avoids inner loop):

```
>> y = zeros(20,1);
>> y(1) = 1;
>> for n = 2:20
>>     temp = 1:n;
>>     y(n) = y(n-1)*sum(temp.^3);
>> end
```

- Method 3 (avoids both the loops):

```
>> X = tril(ones(20)*diag(1:20));
>> x = sum(X.^3,2);
>> Y = tril(ones(20)*diag(x))+ …
       triu(ones(20)) – eye(20);
>> y = prod(Y,2);
```

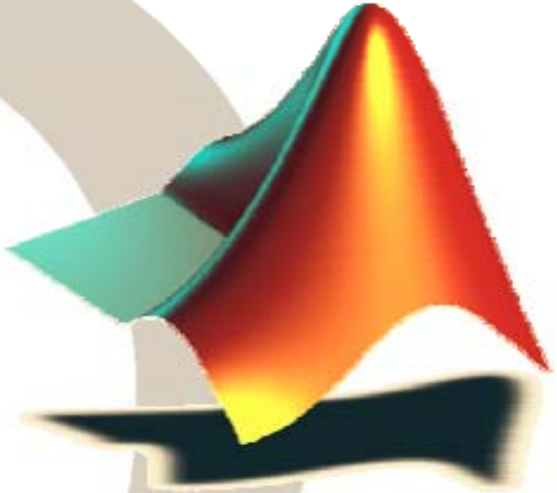
Getting more help

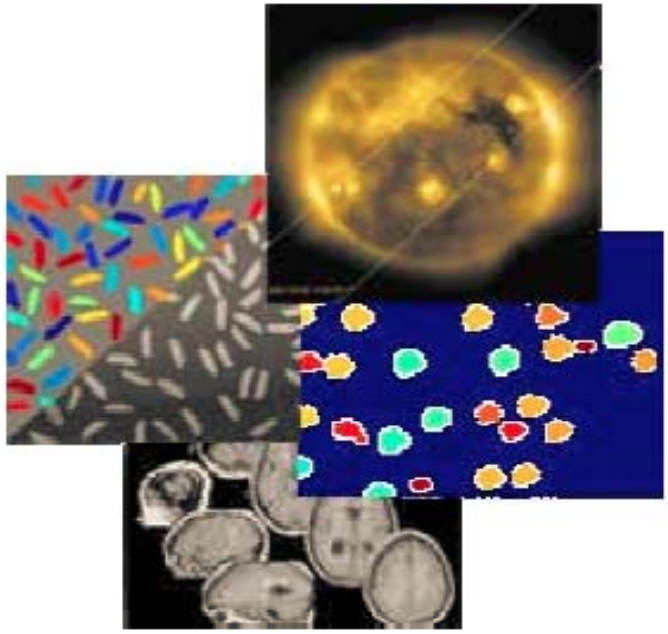
Where to get help?

- In MATLAB's prompt type :
  help, lookfor, helpwin, helpdesk, demos

- On the Web :
  http://www.mathworks.com/support
  http://www.mathworks.com/products/demos/#
  http://www.math.siu.edu/MATLAB/tutorials.html
  http://math.ucsd.edu/~driver/21d -s99/MATLAB-primer.html
  http://www.mit.edu/~pwb/cssm/
  http://www.eecs.umich.edu/~aey/eecs216/.html

Image Processing Using MATLAB®

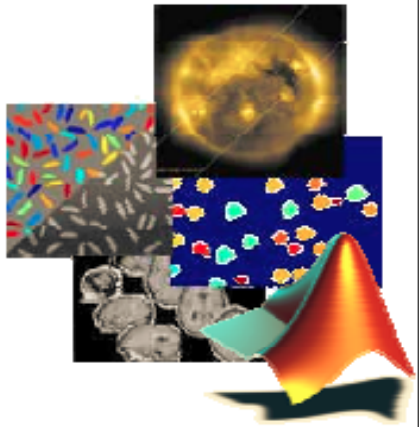




TechSource Systems Sdn. Bhd.

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**



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

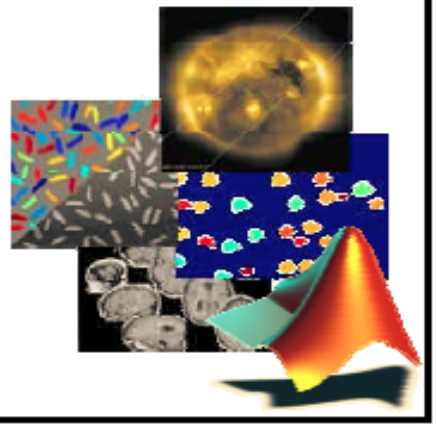


Image Types

- Four basic types of images are supported in MATLAB
  - Index images: m-by-3 colormap matrix
  - Intensity images: [0 1] or `uint8`
  - Binary images: {0, 1}
  - RGB images: m-by-n-by-3 matrix

```
>> 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.

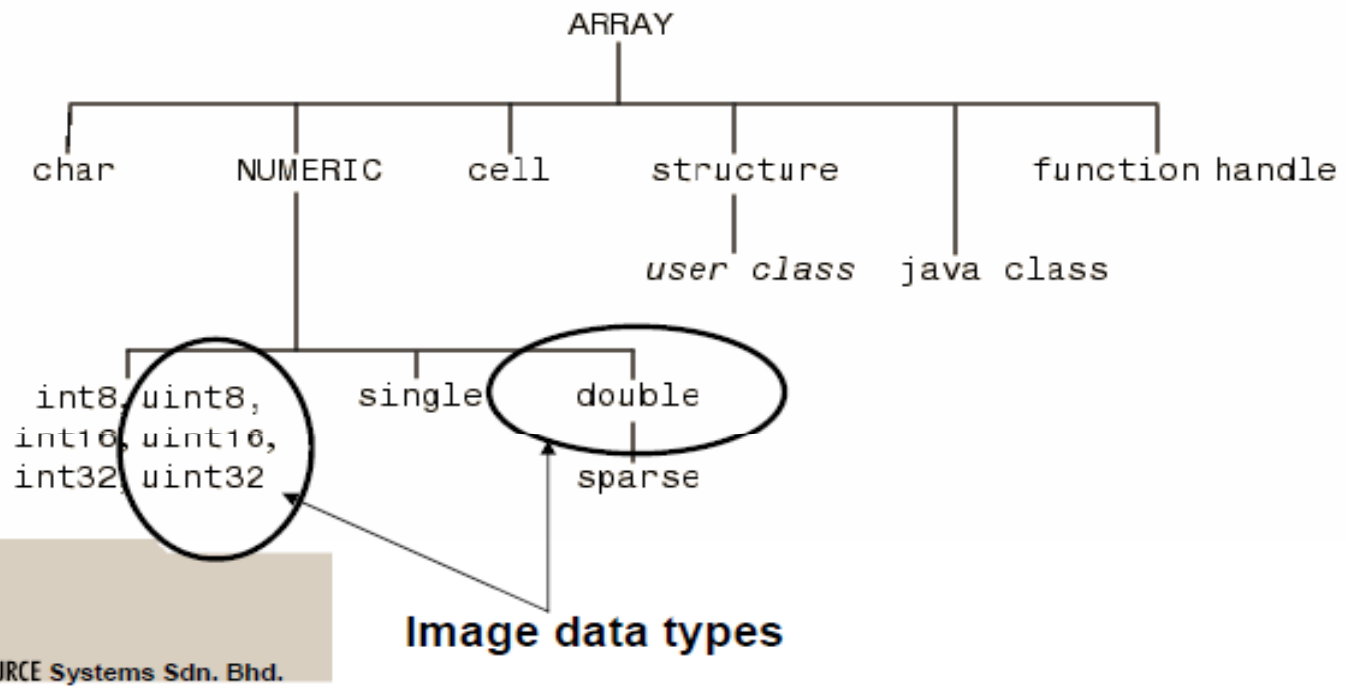
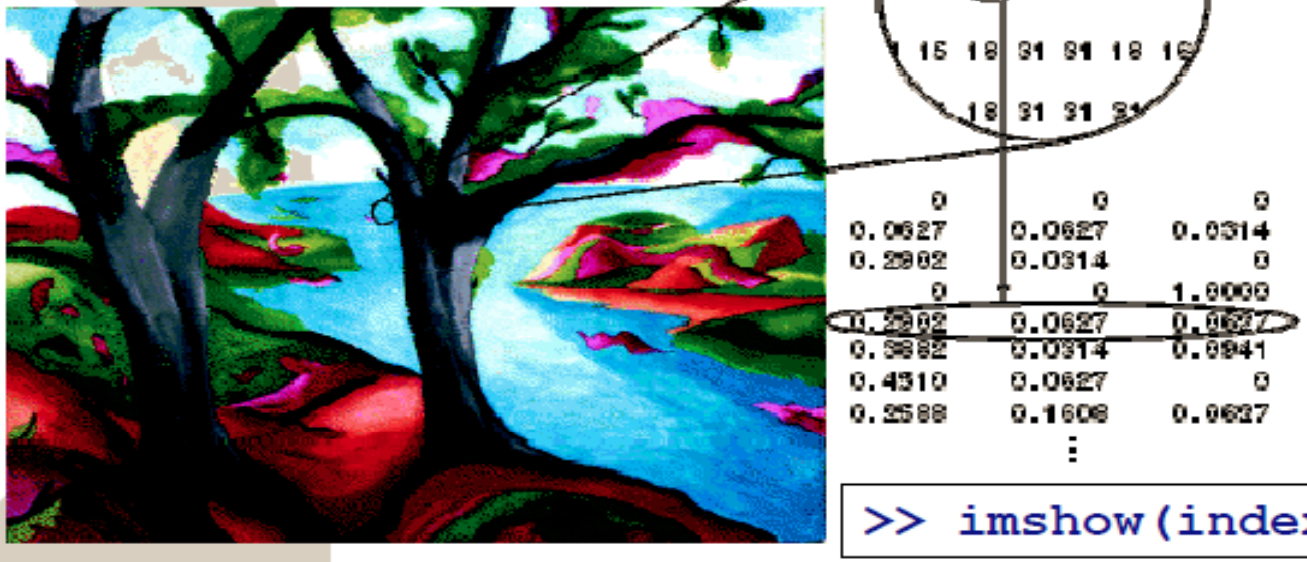


Image Types: Index Images

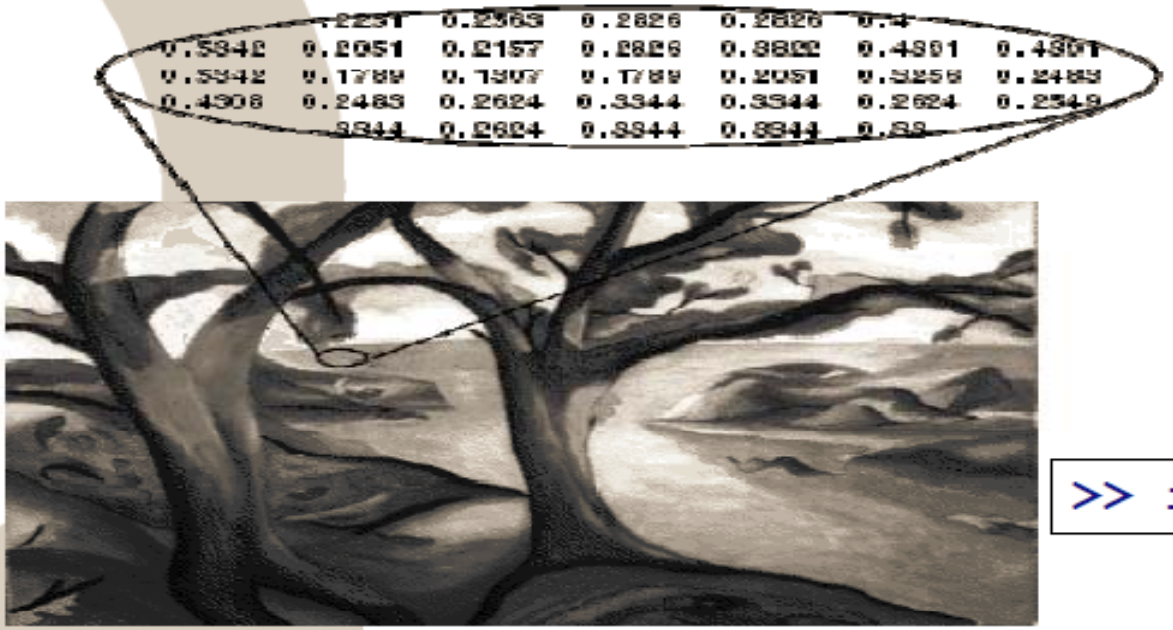
- An indexed image consists of a data matrix, X, and a colormap matrix, map.




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

Image Types: Intensity Images

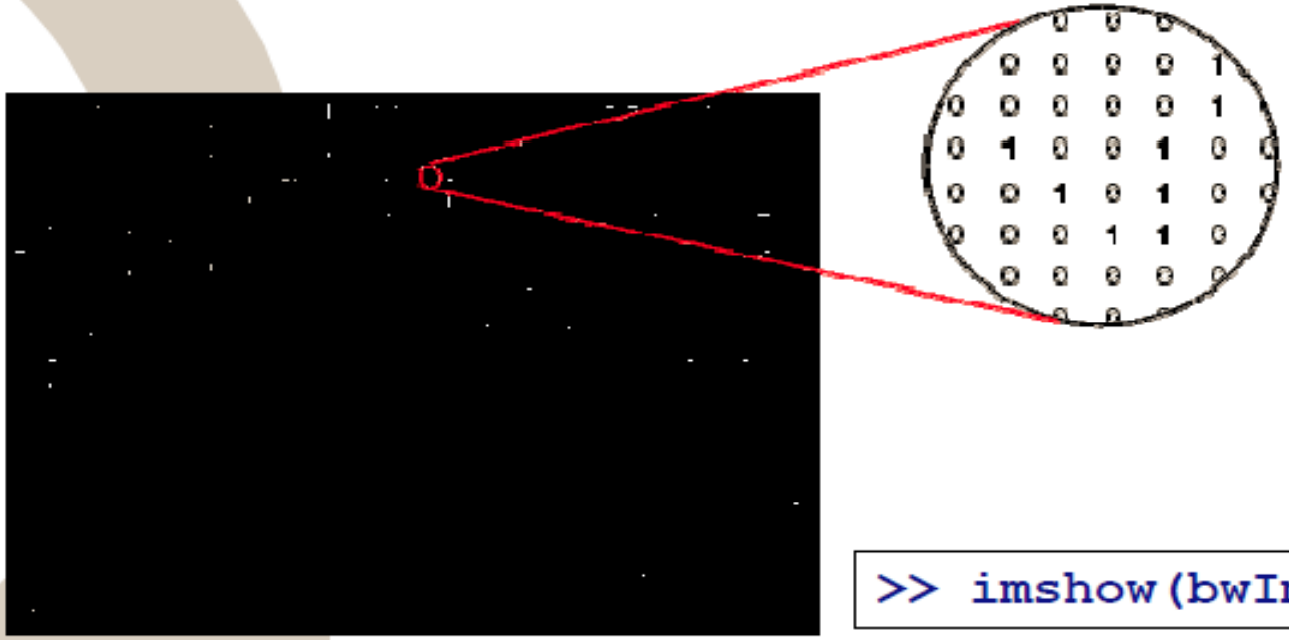
- An intensity image only consists of one matrix, I, whose values represent intensities within some range, for example [0 1] or `uint8`. （无符号8 bit 整数）




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

Image Types: Binary Images

- In a binary image, each pixel assumes one of only two discrete values: 0 (off) and 1 (on).




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

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.

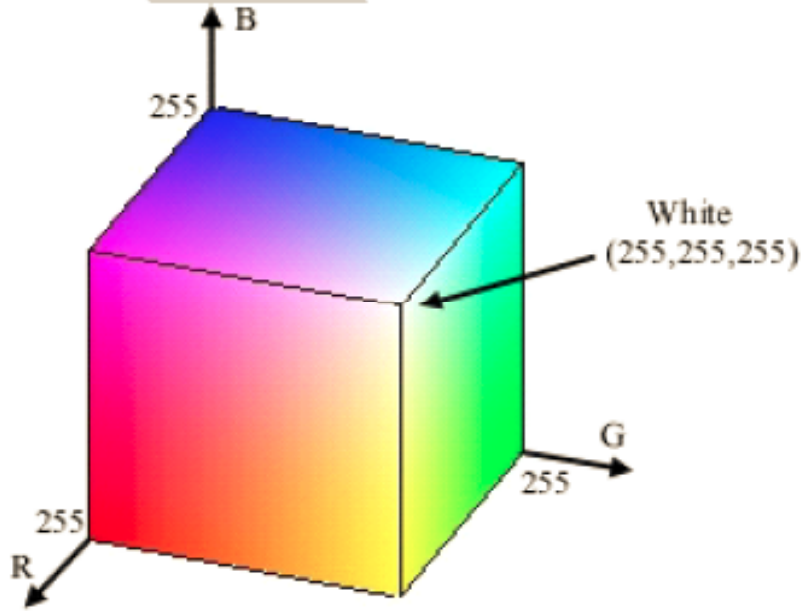
```
>> imshow(rgbImg)
```

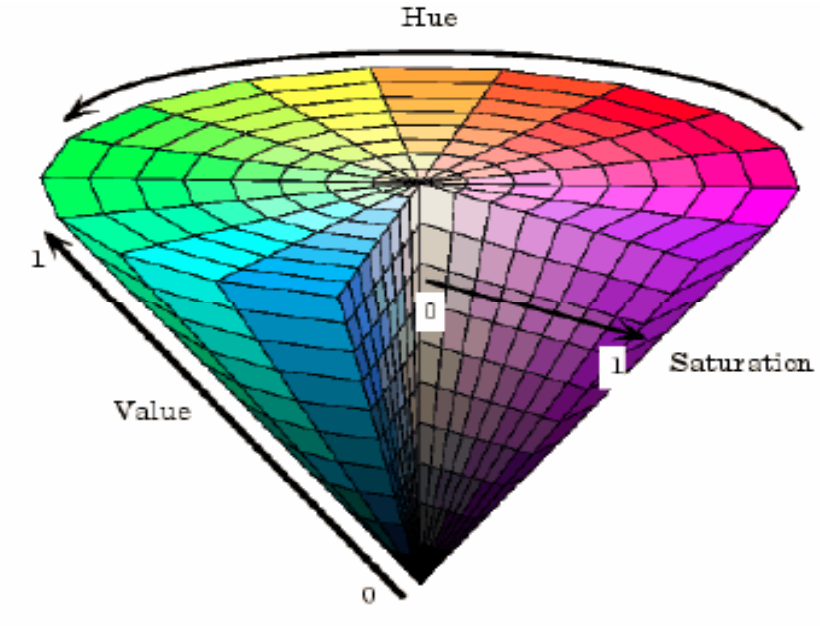
Overview: How Images are Represented?

Image Type	Indexed 	Intensity 	Binary 	RGB 
Double Data	Image is an M-by-N array of integers in the range [1,P]. Colormap is a P-by-3 array of floating-point values in the range [0,1].	Image is an M-by-N array of floating-point values. The conventional dynamic range is [0,1].	Image is an M-by-N logical array containing only 0's and 1's.	Image is an M-by-N-by-3 array of floating-point values in the range [0,1].
Uint8 Data	Image is an M-by-N array of integers in the range [0,P-1]. Colormap is a P-by-3 array of floating-point values in the range ~~[0,1].~~ **[0,255]**	Image is an M-by-N array of unsigned 8-bit integers. The conventional dynamic range is [0,255].	Image is an M-by-N logical array containing only 0's and 1's. Unlike uint8 intensity images, 1 represents white	Image is an M-by-N-by-3 array of floating-point values in the range [0,255].

Color Space of RGB and HSV

- There are two main types of color spaces that are used with images: RGB and Hue-Saturation- Value (HSV).



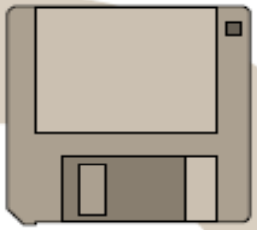


Importing and Exporting Images in MATLAB

- **Image Processing Toolbox**
  - `imfinfo` - Returns info about graphics file.
  - `imread` - Read image from graphics file.
  - `imwrite` - Write image to graphics file.
- **MATLAB**
  - `uiimport` - Starts the Import Wizard.

```
>> imfinfo('canoe.tif')
>> [X, map] = imread('canoe.tif');
>> imshow(X, map);
>> imwrite(X, map, 'canoe2.bmp');
```

Graphical Representation of Importing an Image



`imfinfo('file.fmt')`

file.fmt

`[x,map]=imread('file.fmt');`

Colormap?

`x=imread('file.fmt');`

Image Type	Indexed
Double Data	[MXN] +
Uint8 Data	[MXN] +

Image Type	Intensity	Binary	RGB
Double Data	1 0.5 0.2 0 0.1 0.9 0 0.7 0.8	1 0 1 0 1 0 0 1 1	0.9 0.4 0.1 0.7 0.3 1 0.5 0.9 0.2
Uint8 Data	1 45 220 100 78 110 200 7 98	1 0 1 0 1 0 0 1 1	90 140 30 100 70 230 10 50 109 220

Displaying Images

- `imshow` - Display image.
- `image` - Create and display image object (MATLAB).
- `imagesc` - Scale data and display as image (MATLAB).

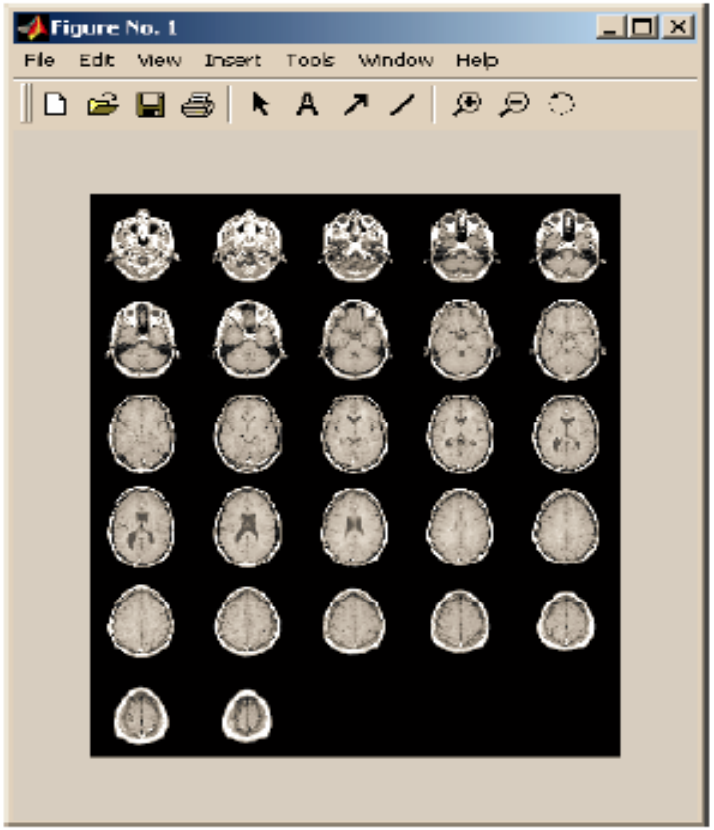
- `colorbar` - Display colorbar (MATLAB).
- `colormap` - Sets the color map of the image (MATLAB).
- `montage` - Display multiple image frames.
- `truesize` - Adjust display size of image.
- `warp` - Display image as texture-mapped surface.

Montage

- An example of displaying multiple image frames as a rectangular montage.

```
>> load mri
>> montage(D, map)
```

D是M-by-N-by-K index array，map是调色板

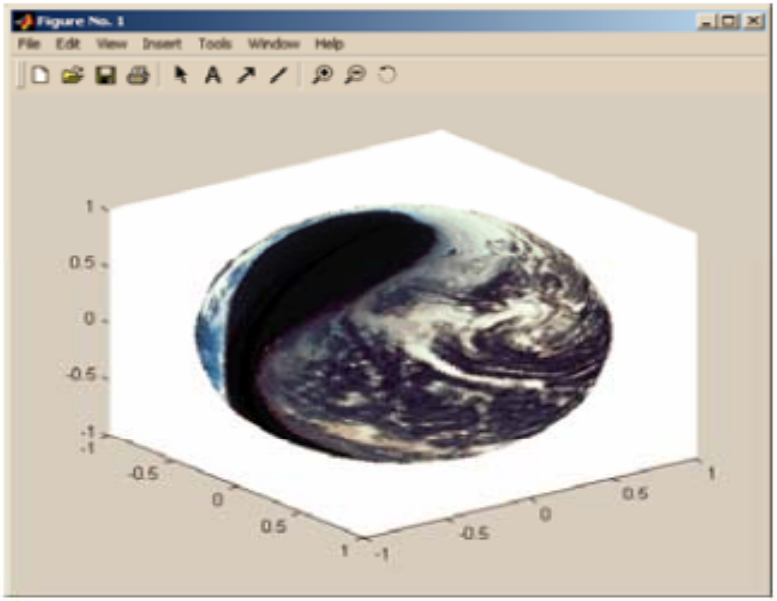


Warping

- The warp function allows you to display an image as a texture-mapped surface.

```
>> [x, y, z] = sphere;
>> load earth
>> warp(x, y, z, X, map)
```

在一个目标表面显示一个带调色板map的索引图像X



Converting Image Formats

- `ind2gray` – indexed image to intensity image.
- `ind2rgb` – indexed image to RGB image (MATLAB).
- `gray2ind` – intensity image to indexed image.
- `rgb2gray` – RGB image or colormap to grayscale.
- `rgb2ind` – RGB image to indexed image.

- `mat2gray` – matrix to intensity image.
- `im2bw` – image to binary image by thresholding.
- `im2double` – image array to double precision.
- `im2uint8` – image array to 8-bit unsigned integers.
- `im2uint16` – image array to 16-bit unsigned integers.

Exercise 1: Loading and Viewing an Image

1. Load in the `trees.tif` file into MATLAB.
   - What type of image is it? Can you tell before loading in the file?
2. Display the loaded image.
3. Convert the image to an intensity image (grayscale).
4. Now convert it to a binary (black and white) image.

Extra credit:

1. Use subplots to display all three images.
2. Did you find the "Easter egg" in the "trees"?

Solution: Loading and Viewing an Image

```
>> im_info = imfinfo('trees.tif');
>> im_info(1).ColorType

>> [I,map] = imread('trees.tif');
>> subplot(2,2,1), subimage(I,map)

>> I_gray = ind2gray(I,map);
>> subplot(2,2,2), subimage(I_gray)

>> I_bw = im2bw(I,map,0.4);
>> subplot(2,2,3), subimage(I_bw)

% Easter egg
>> figure
>> [I2,map] = imread('trees.tif',2);
>> imshow(I2,map)
```

Why use subimage?

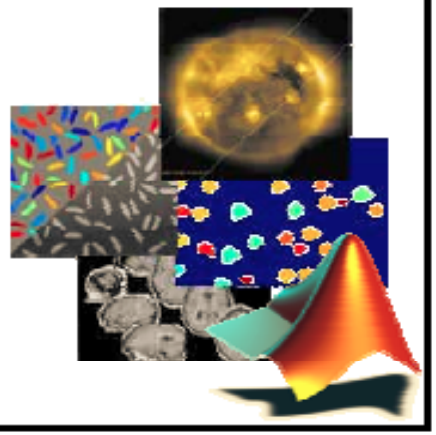
Threshold

Section Summary:

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



Section Outline:

1. Image enhancement
    - Image histogram
    - Image contrast adjustment
    - Image brightness adjustment

2. Image thresholding

3. Image arithmetic

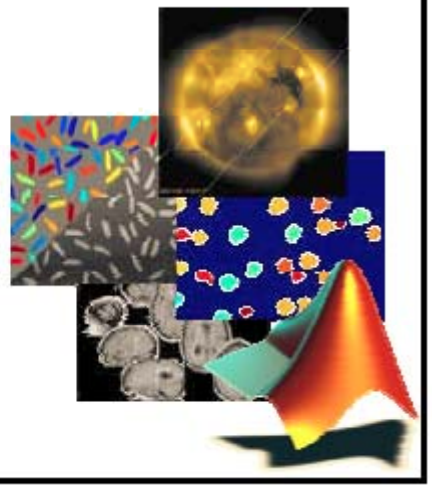


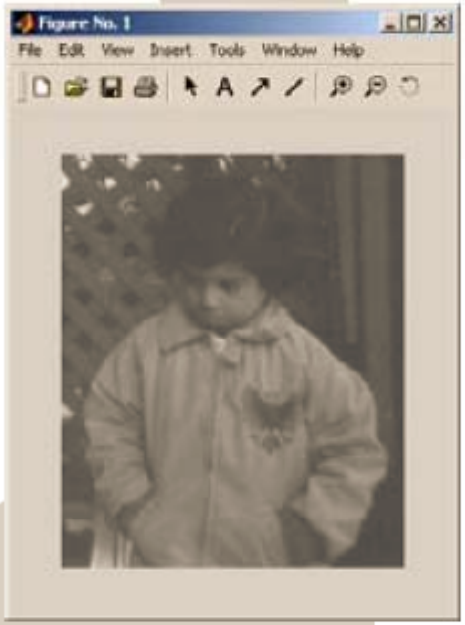
Image Enhancement

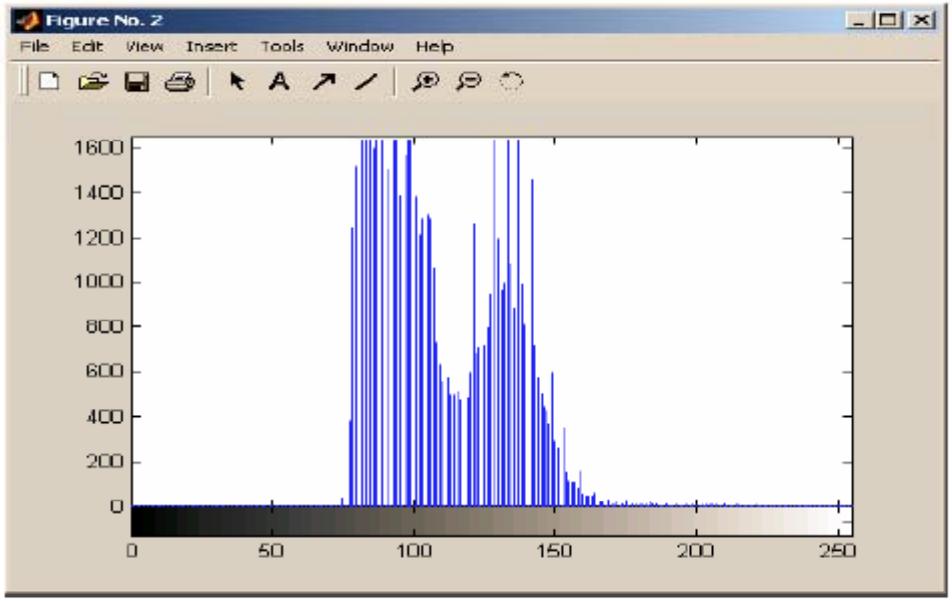
- One of the most basic ways to enhance an image is to change its *brightness* and its *contrast*, and this can be done is by working with the image's histogram.
  - By stretching the color distribution
  - By equalizing the distribution of colors to use the full range
  - By adjusting the scaling of the colors
- If you are separating an object from its background, *thresholding* is a technique that could be used as well.

Histogram

- A histogram of an image shows the current level of contrast (the distribution of gray levels).

```
>> I = imread('pout.tif');
>> imshow(I)
>> figure, imhist(I)
```

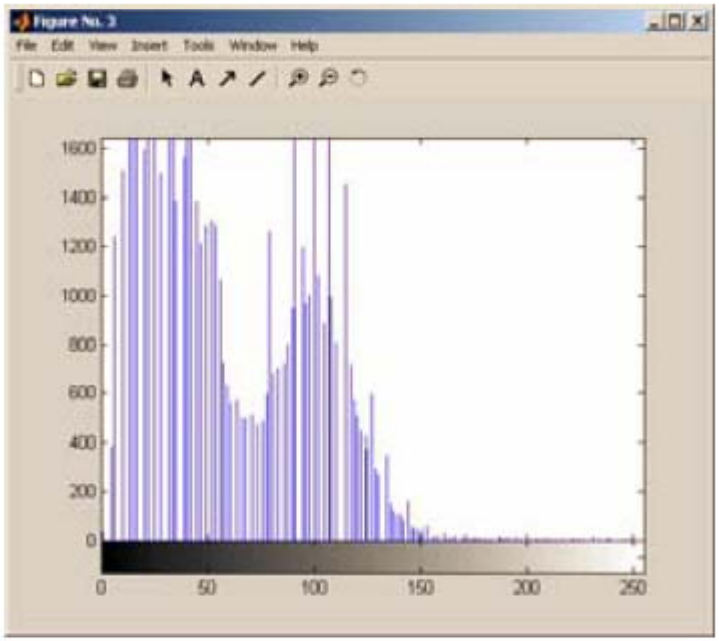




Histogram Stretching

- One way to increase the contrast of an image is to stretch the pixel values (min == 0 and max == 255).

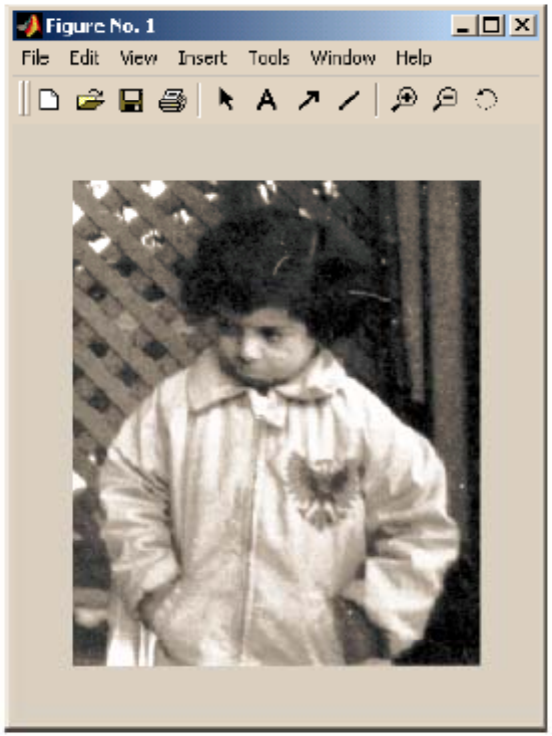
$$J = 255 \cdot \frac{I - I_{\min}}{I_{\max} - I_{\min}}$$

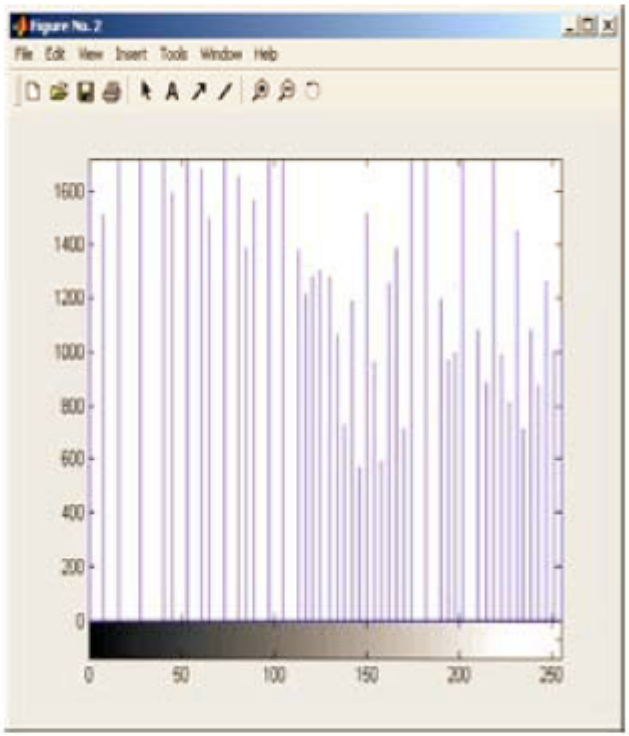


Histogram Equalization

- The `histeq` function can be used to equally distribute the histogram and enhance the contrast.

```
>> J = histeq(I);
```

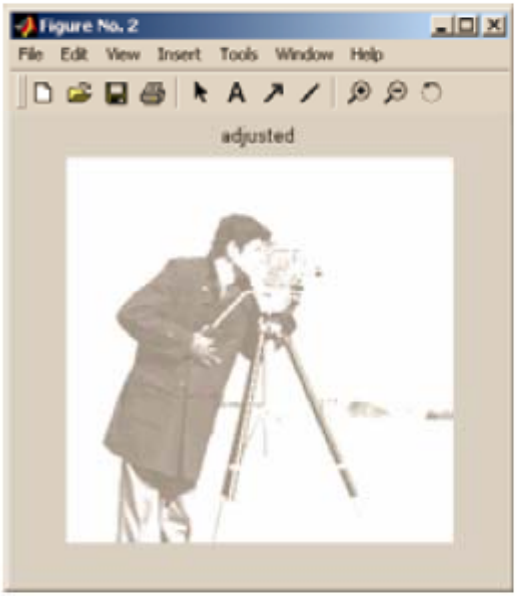




Histogram Adjustment

- Intensity adjustment is a technique for mapping an image's intensity values to a new range (`imadjust`).

```
>> I = imread('cameraman.tif');
>> J = imadjust(I,[0 0.2],[0.5 1]);
>> imshow(I)
>> figure, imshow(J)
```



Understanding Intensity Adjustment

- The following example demonstrates how an image's intensity can be changed to enhance different characteristics of an image.

```
>> imadjdemo
```

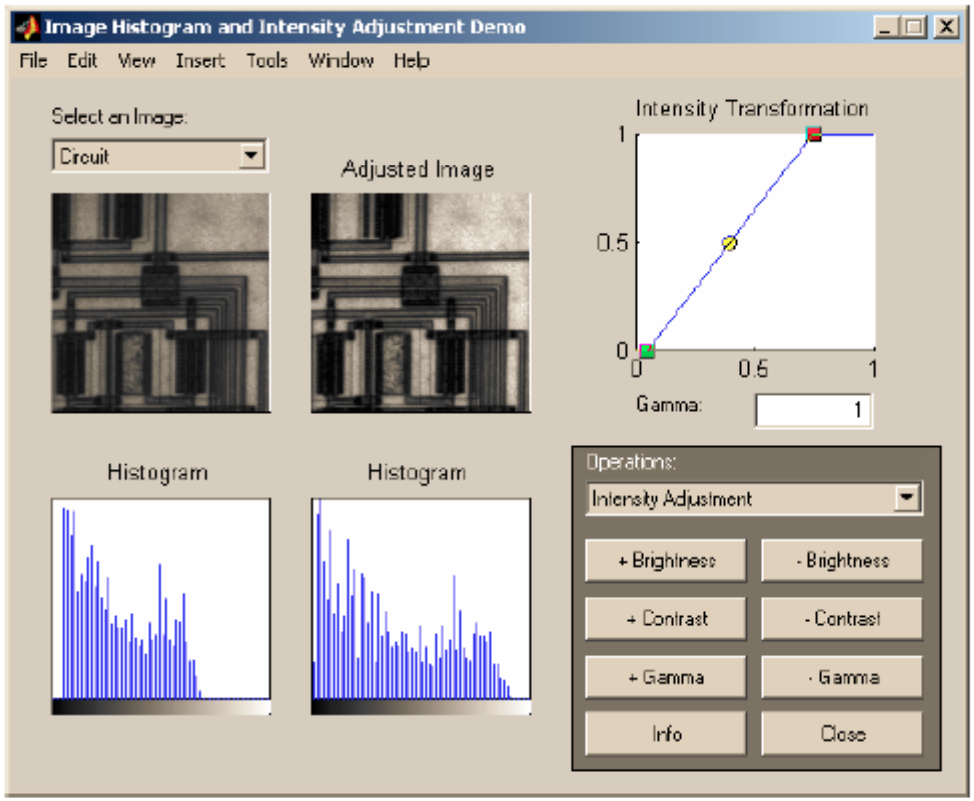
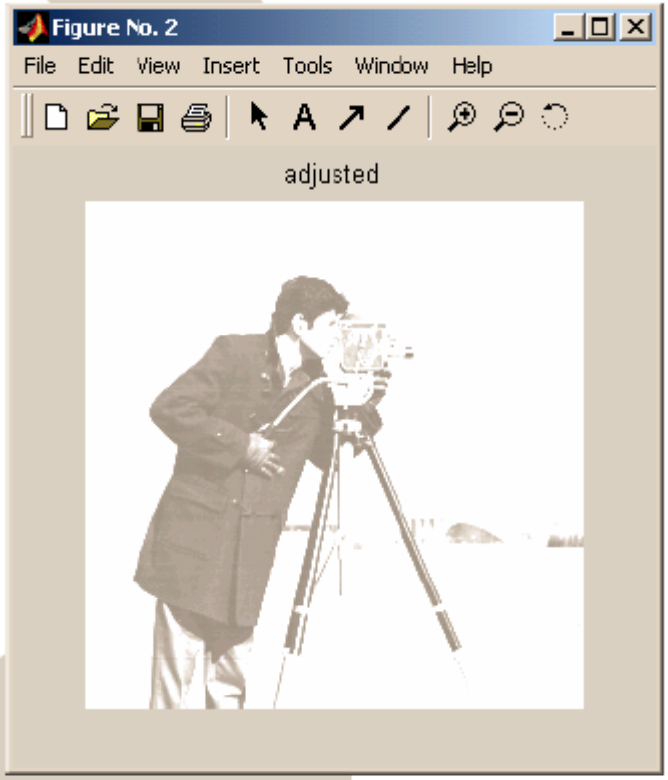
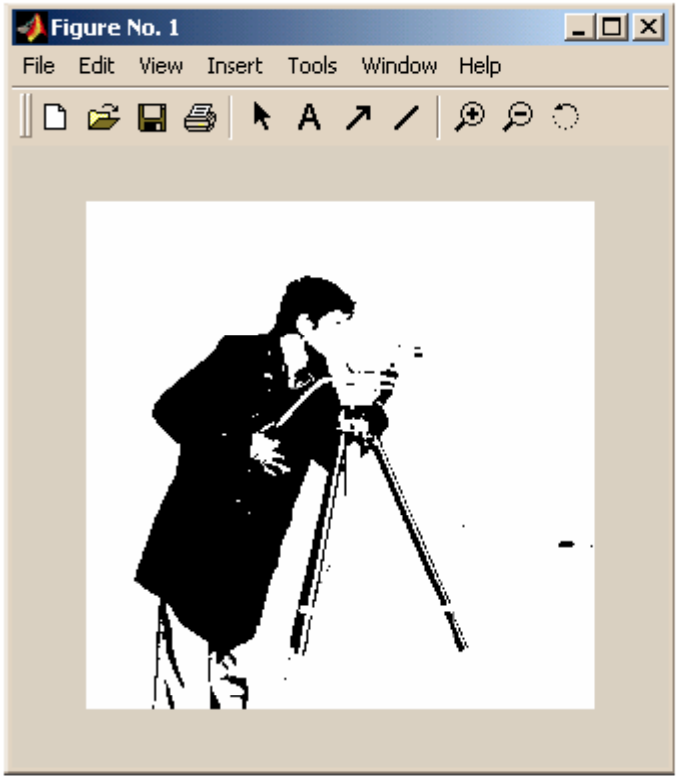


Image Thresholding

- The adjusted cameraman image can be thresholded to create a black and white image of the cameraman and the camera.





Using `imtool` GUI for Image Analysis

- The `imtool` is an image display GUI that provides access to Pixel Region tool, the Image Information tool, and the Adjust Contrast tool.

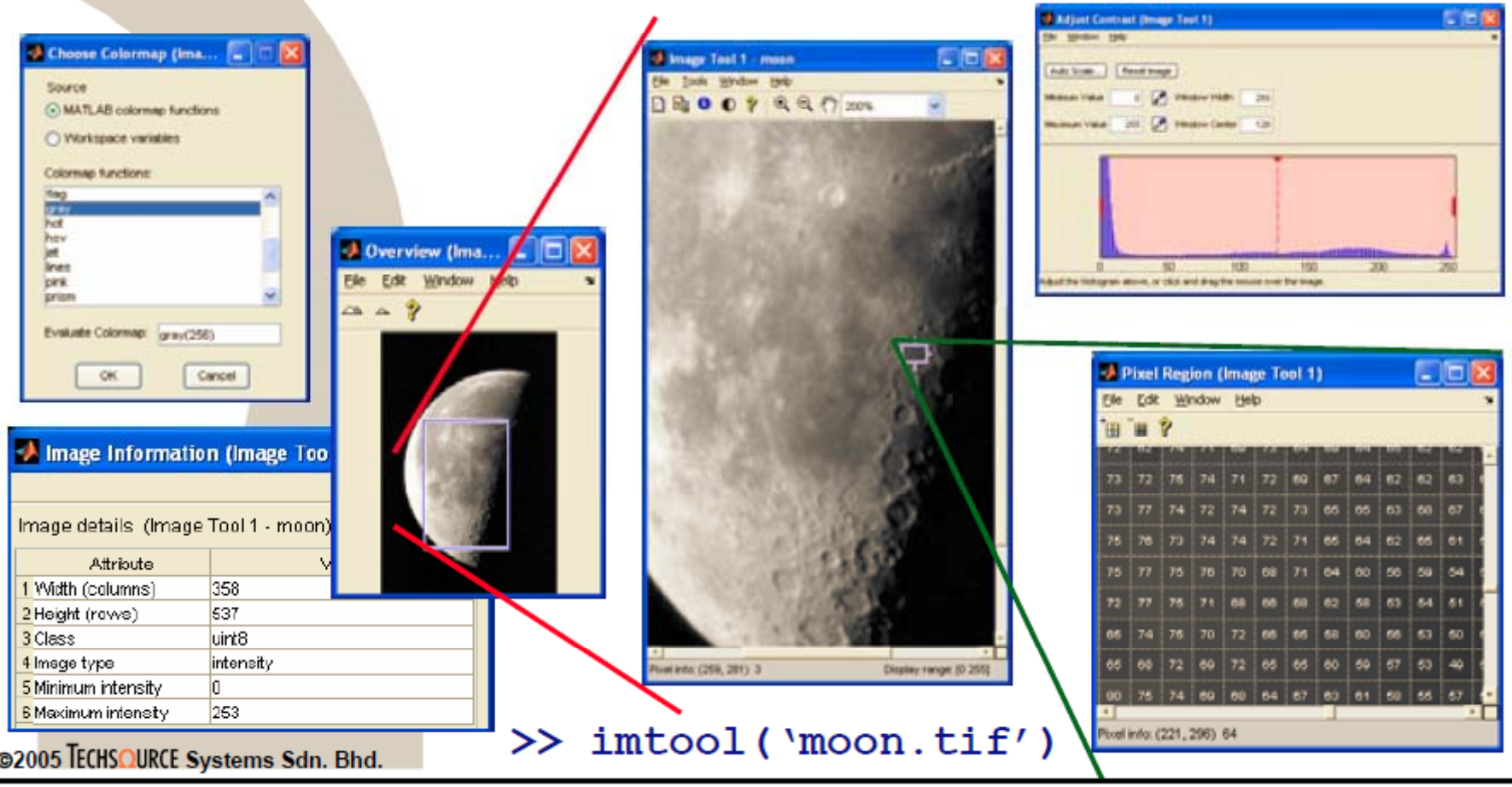


Image Arithmetic

- **With just simple addition, subtraction, multiplication and division a number of different image processing techniques can be implemented.**
  - With addition and multiplication an image contrast can be increased that facilitates edge detection process.
  - With subtraction and division changes can be detected from one image to another.

`imabsdiff` - Compute absolute difference of two images

`imadd` - Add two images or add constant to image

`imcomplement` - Complement image

`imdivide` - Divide two images or divide image by a constant

`imlincomb` - Compute linear combination of images

`immultiply` - Multiply two images or multiply image by constant

`imsubtract` - Subtract two images or subtract constant from image

Image Addition

- Image addition makes it possible to superimpose an image on top of another or control the brightness of an image.
- Each resulting pixel is the sum of the respective pixels of the two images, of the same size and of the same class.

```
>> I1 = imread('peppers.png');
>> I2 = imadd(I1, 50);
>> subplot(2,1,1), imshow(I1)
>> subplot(2,1,2), imshow(I2)
```

```
>> % MATLAB 7 New Features
>> I1 = imread('peppers.png');
>> I2 = I1 + 50 % direct addition
>> subplot(2,1,1), imshow(I1)
>> subplot(2,1,2), imshow(I2)
```

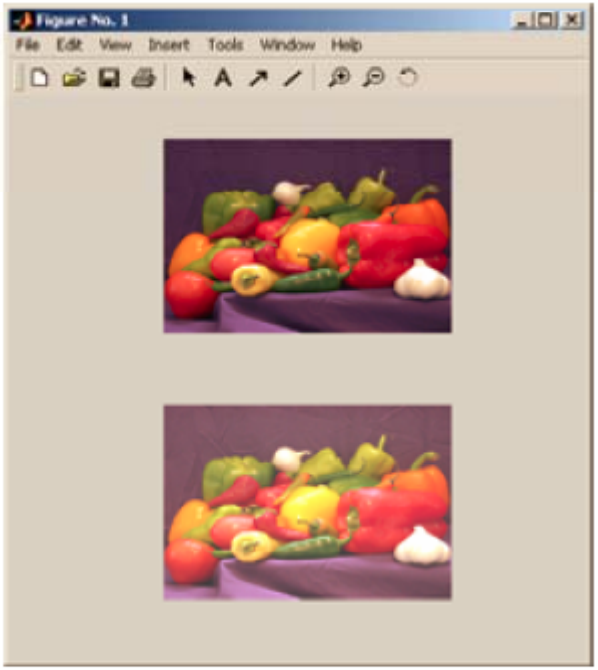


Image Addition (Continued)

```
>> I = imread('rice.png');
>> J = imread('cameraman.tif');
>> K = imadd(I, J);
>> imshow(K)
```

```
>> % MATLAB 7 New Features
>> I = imread('rice.png');
>> J = imread('cameraman.tif');
>> K = I + J; % direct addition
>> imshow(K);
```

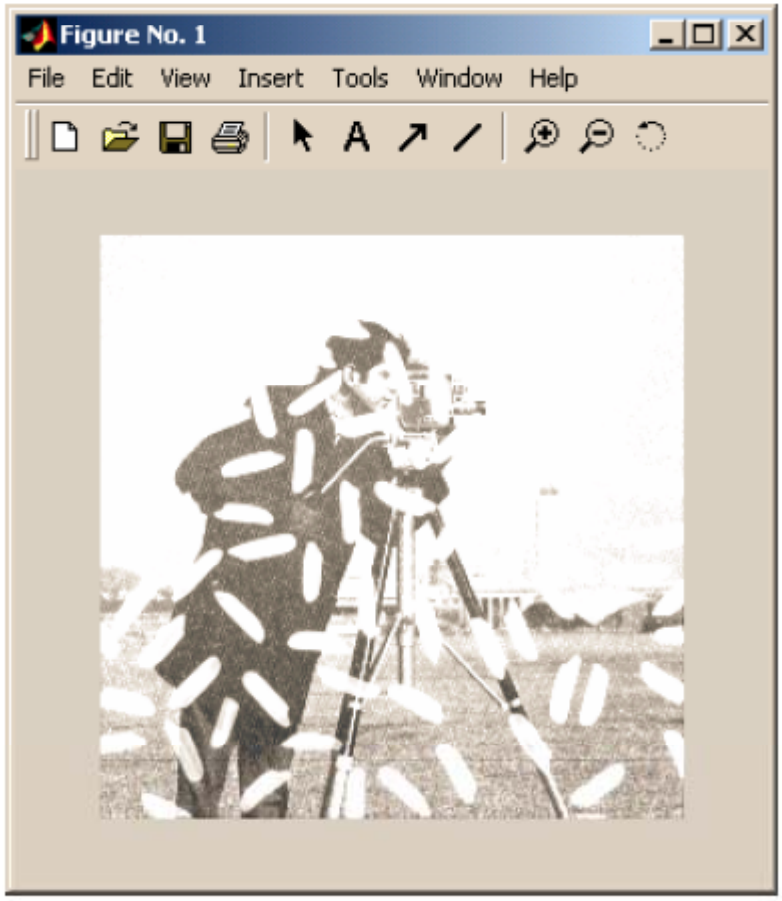


Image Multiplication

```
>> I = imread('moon.tif');
>> J = immultiply(I, 1.2);
>> subplot(1,2,1),imshow(I)
>> subplot(1,2,2),imshow(J)
```

```
>> % MATLAB 7 New Features
>> I = imread('moon.tif');
>> J = 1.2 * I; % direct multiply
>> subplot(1,2,1),imshow(I)
>> subplot(1,2,2),imshow(J)
```

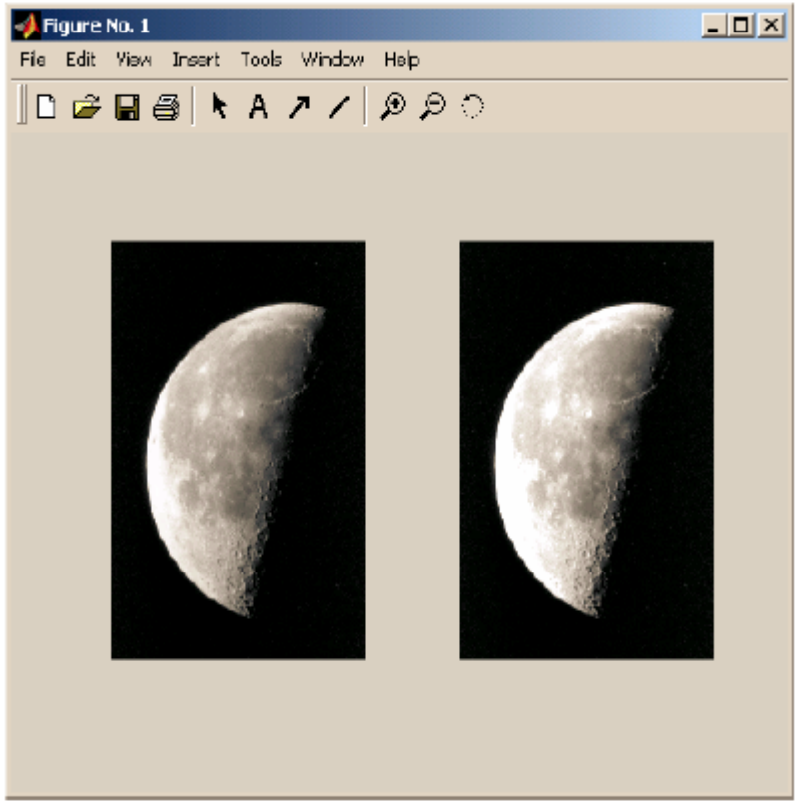
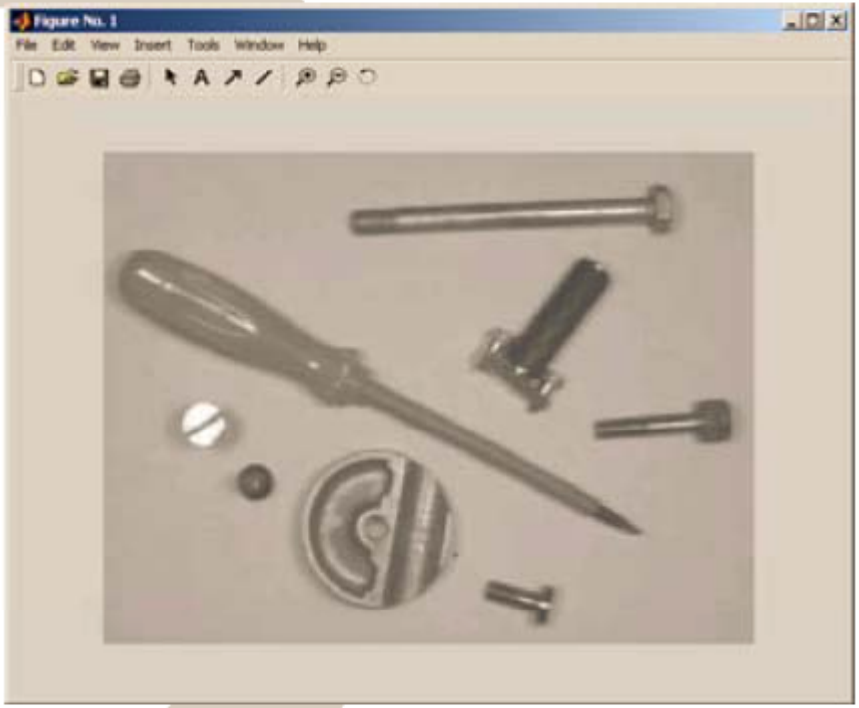
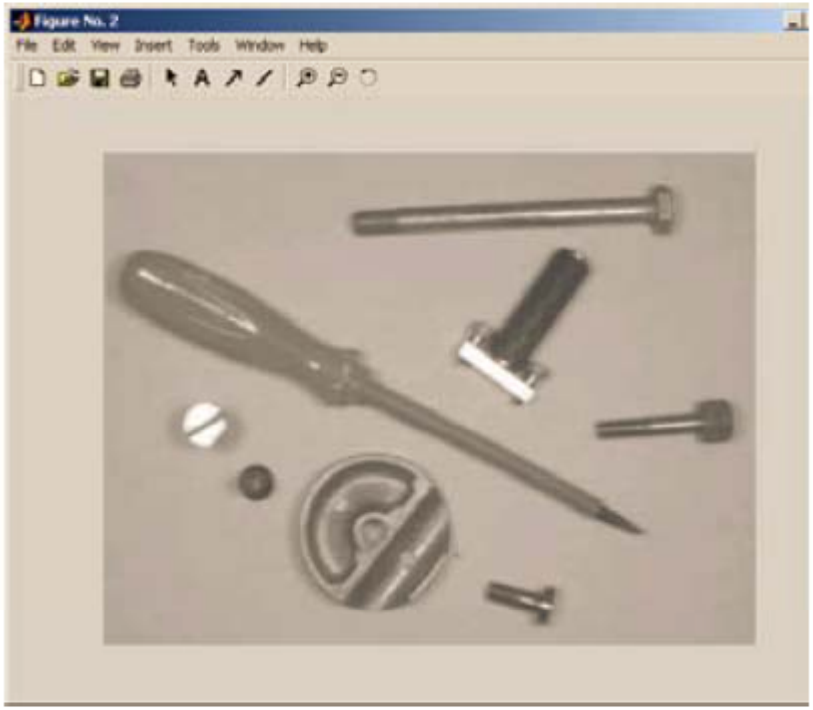


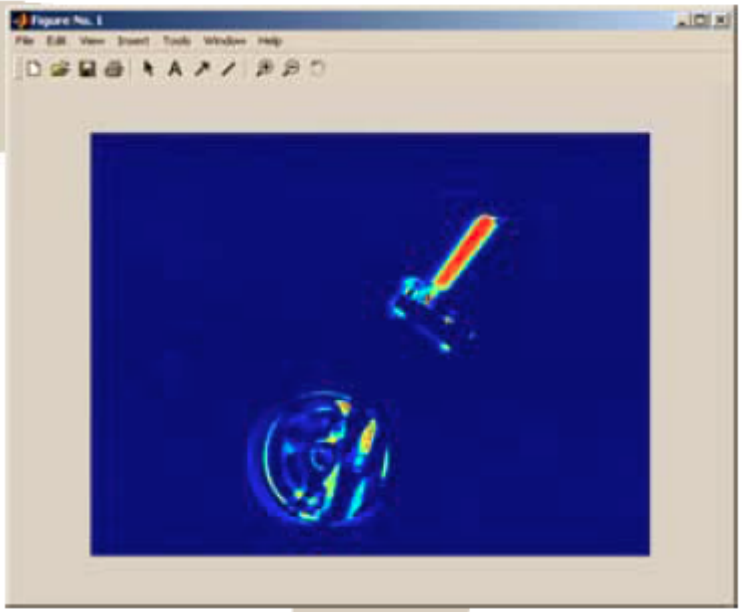
Image Subtraction

- Can you see anything different about the two images?





- Using subtraction, you can identify the following differences.

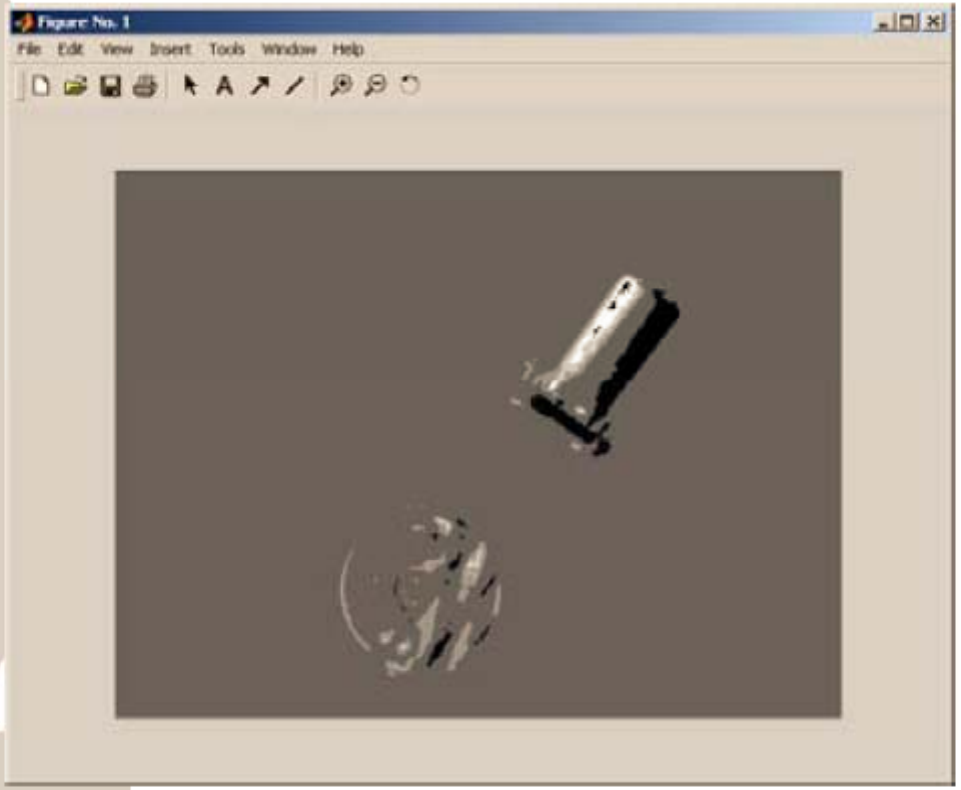


Did the image match up with what you were expecting?

```
>> [im1, map1] = imread('change1.gif');
>> [im2, map2] = imread('change2.gif');
>> im_diff = imsubtract(im1, im2);
>> % im_diff = im1 - im2; % direct subtraction
>> imshow(im_diff)
>> colormap(jet)
>> set(gca, 'clim', [0 60]); % adjust colorbar
```

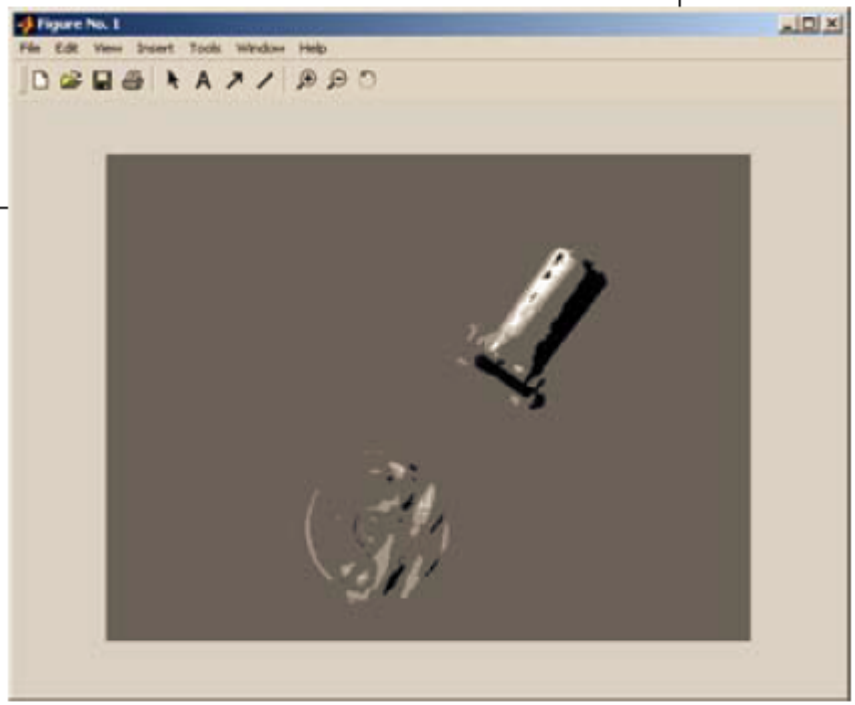
Exercise 2: Image Division

- Repeat what you just did with image subtraction, except, this time, use division instead.



Solution: Loading and Viewing an Image

```
>> [im1, map1] = imread('change1.gif');
>> [im2, map2] = imread('change2.gif');
>> im_diff = imdivide(im1, im2);
>> % im_diff = im1./im2; % direct element-wise division
>> imshow(im_diff)
>> colormap(map1)
>> min(im_diff(:)); % ans = 0
>> max(im_diff(:)); % ans = 5
>> set(gca, 'clim', [0 5])
```

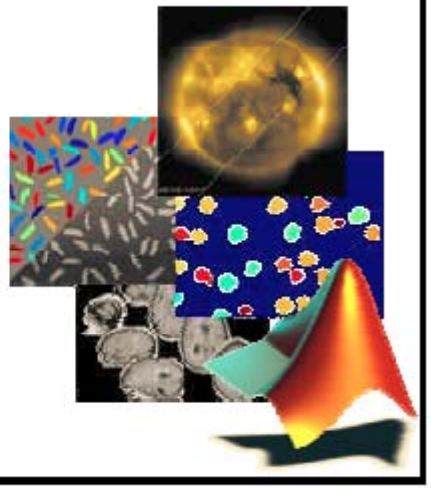


Section Summary:

1. Image enhancement
   - Image histogram
   - Image contrast adjustment
   - Image brightness adjustment

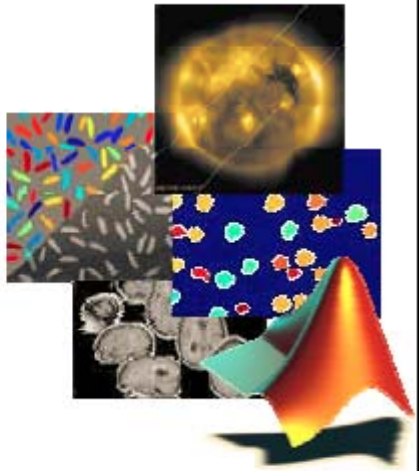
2. Image thresholding

3. Image arithmetic



Section Outline:

1. What is block processing?
2. Distinct block operations
3. Sliding neighbourhood operations
4. Example of block processing
   - Convolution
   - Correlation
5. Column processing



What is block processing?

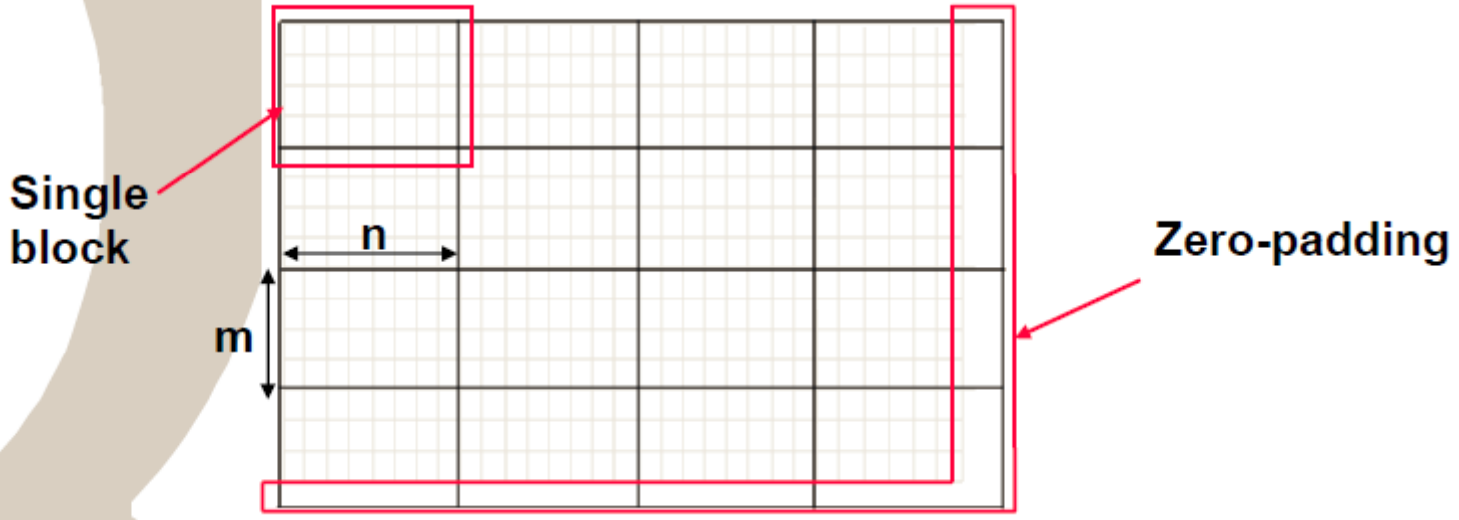
An operation in which an image is processed in blocks rather than all at once.

- The blocks have the same size across the image.
- An operation is applied to one block at a time.
- Once processed, the blocks are re-assembled to form an output image.

Distinct Block Operations

Distinct blocks are rectangular partitions that divide an image matrix into m-by-n sections.

- Blocks are overlaid by starting at the upper-left corner.
- Zeros are padded onto blocks that exceed the size of the image.



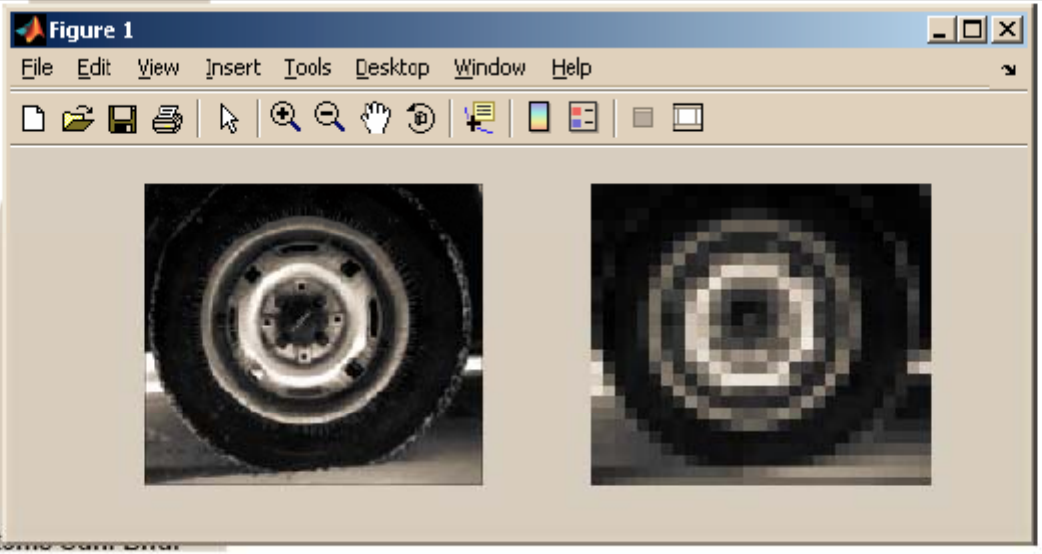

```
B = blkproc(A,[m n],fun)
```

Block Processing – Averaging Filter

Find the average value of each 8-by-8 block and replace all the pixels in the block with the average value.

```
>> I1 = imread('tire.tif');
>> f = @(x) uint8(round(mean2(x)*ones(size(x))))
>> I2 = blkproc(I1,[8 8], f);
>> subplot(1,2,1), imshow(I1)
>> subplot(1,2,2), imshow(I2)
```

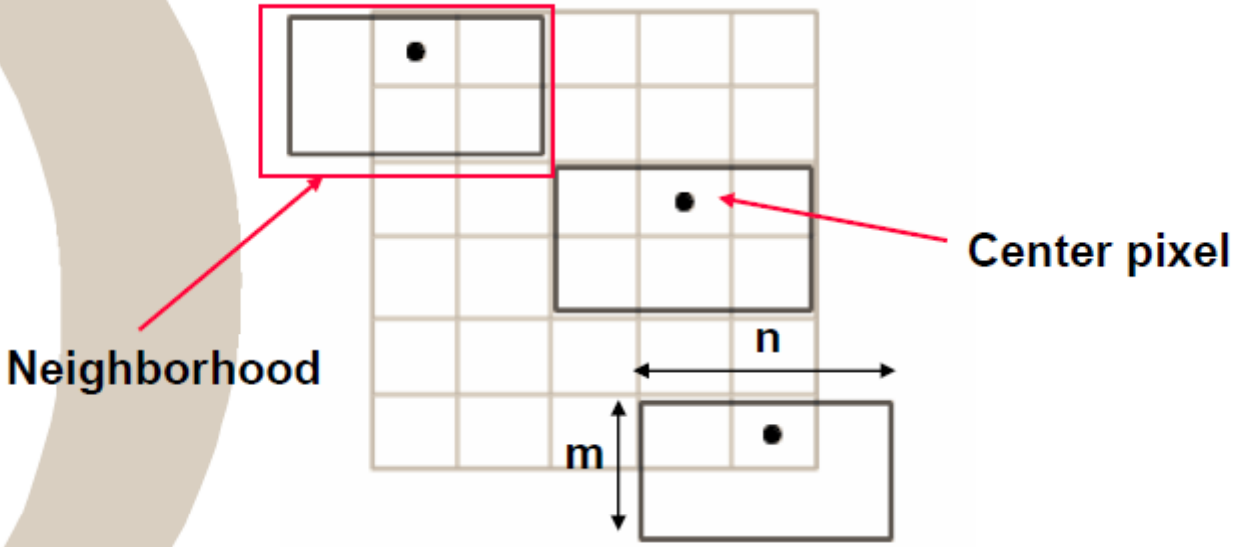
Function handle



说明：
1）@x unit8 ... 返回函数句柄
2）x：8x8 矩阵
3）mean2：求2维块均值
4）ones（size（x））：返回8x8 全1矩阵
5）f：将8x8阵中元素替换为其均值

Sliding Neighborhood Operations

Each center pixel value is determined by applying some algorithm to its neighboring pixels of a defined size – neighborhood



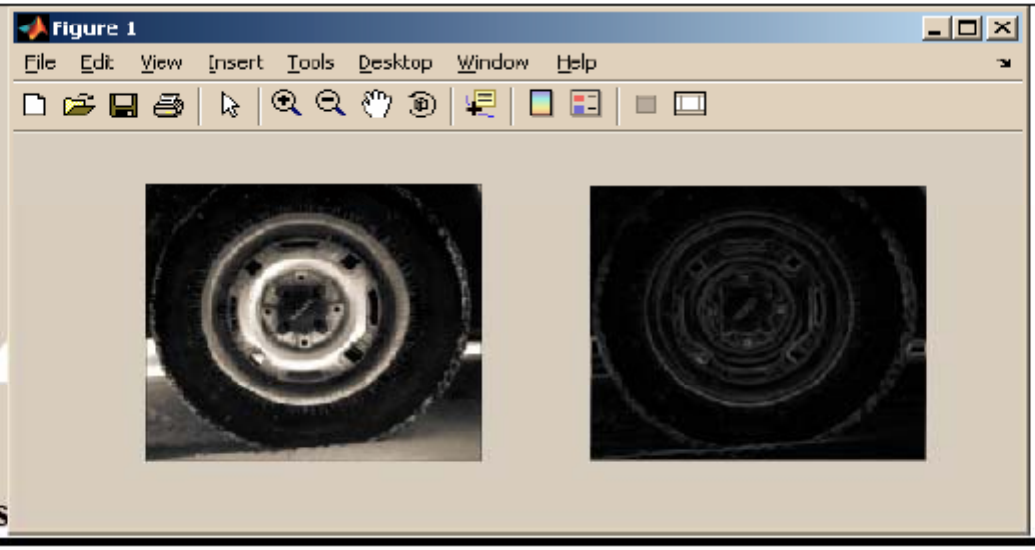

```
B = nlfilter(A,[m n],fun)
```

Note:
For an m-by-n neighborhood, the center pixel is `floor(([m n]+1)/2)`

Sliding Neighborhood Operations – Nonlinear Filter

Replace each pixel with the standard deviation of the values of the input pixel's 3-by-3 neighborhood.

```
>> I1 = imread('tire.tif');
>> f = @(x) uint8(round(std2(x))); % function handle
>> I2 = nlfilter(I1,[3 3], f);
>> subplot(1,2,1), imshow(I1)
>> subplot(1,2,2), imshow(I2)
```



说明：nlfilter( )是在灰度图像每个像素x的m-by-n邻域作函数运算，并替换像素x

Example of Block Processing - Convolution

In *convolution*, the value of an output pixel is computed as a weighted sum of neighboring pixels. The matrix of weights is called the *convolution kernel* (or *filter*).

Steps for convolving an image

- Rotate the convolution kernel 180 degrees about the center.
- Slide the rotated convolution kernel over the image.
- Multiply each weight in the rotated convolution kernel by the pixel of the image
- Sum up all individual products.

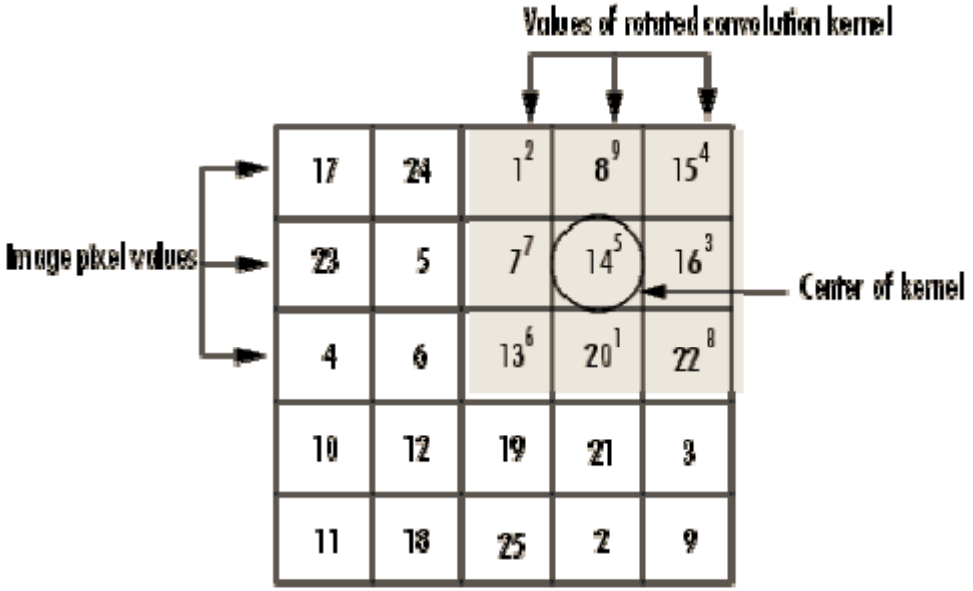
卷积核的反摺

Given an image, A, and the convolution kernel, h, the procedure for calculating the convolution for the value of 14 would be as follows.

- Notice that in the calculation the convolution kernel, h, is rotated a 180 degrees.

```
>> A = [17 24 1   8   15
        23 5  7   14  16
        4  6  13  20  22
        10 12 19  21  3
        11 18 25  2   9]
>> h = [8 1 6
        3 5 7
        4 9 2]

>> conv2(A,h, 'same')
```

$$1 \cdot 2 + 8 \cdot 9 + 15 \cdot 4 + 7 \cdot 7 + 14 \cdot 5 + 16 \cdot 3 + 13 \cdot 6 + 20 \cdot 1 + 22 \cdot 8 = 575$$



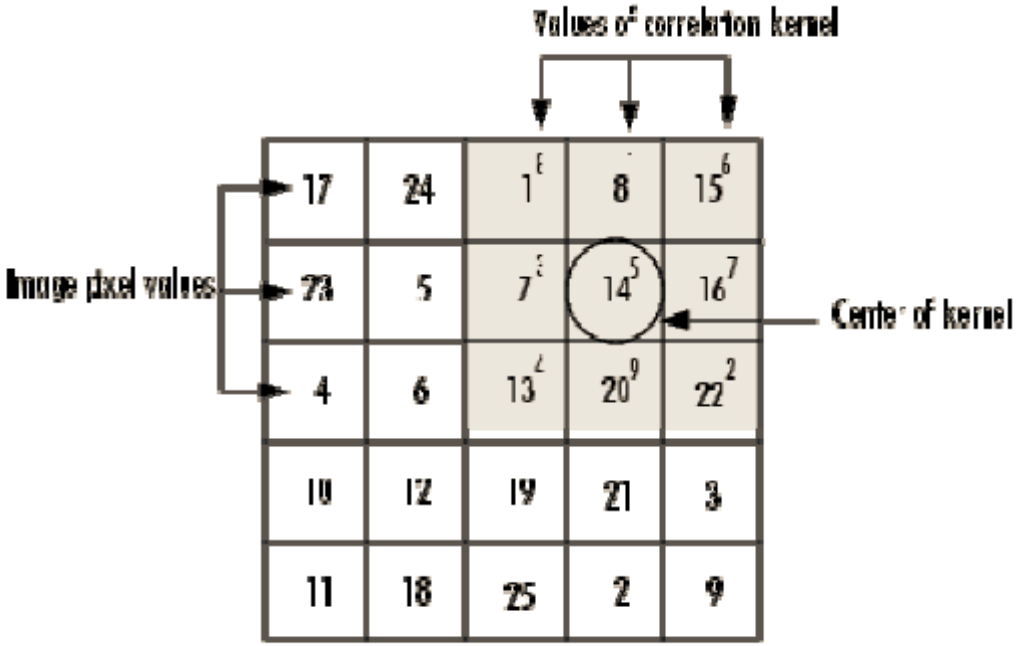
Example of Block Processing - Correlation

In *correlation*, the value of an output pixel is also computed as a weighted sum of neighboring pixels. The difference is that the matrix of weights (*correlation kernel*) is *not* rotated.

```
>> A = [17 24 1  8  15
        23 5  7  14 16
        4  6  13 20 22
        10 12 19 21 3
        11 18 25 2  9]
>> h = [8 1 6
        3 5 7
        4 9 2]

>> filter2(h,A)
```

filter2：2-D 数字滤波操作

$$1\cdot 8+8\cdot 1+15\cdot 6+7\cdot 3+14\cdot 5+16\cdot 7+13\cdot 4+20\cdot 9+22\cdot 2=583$$



Column Processing

Arranging each sliding neighborhood or distinct block as separate columns, and process each column as a block.

- Faster, since most of MATLAB is column-based.
- But uses more memory.

Syntax

```
B = colfilt(A,[m n],block_type,fun)
```

`A` – image matrix

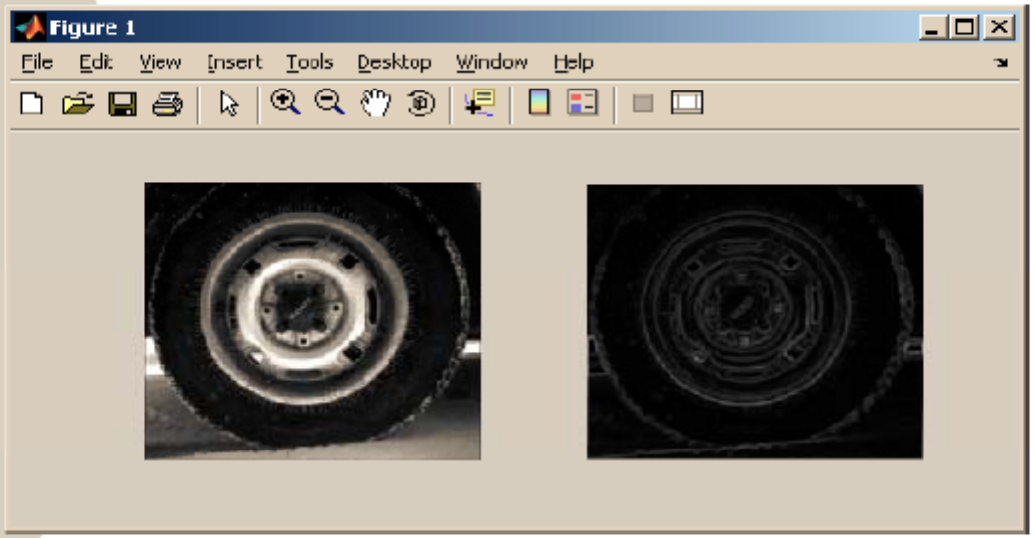
`[m n]` – size of block

`block_type` – Either `'distinct'` or `'sliding'`

`fun` – function to apply

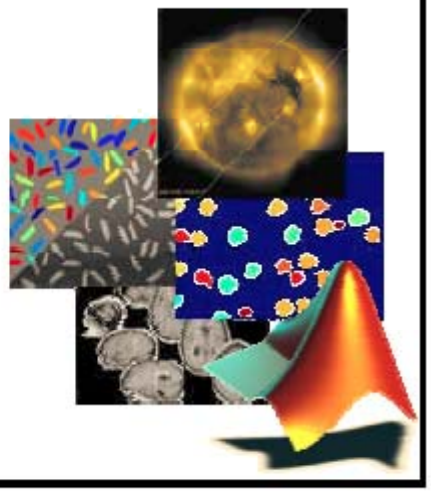
Block Processing of Images


```
>> I1 = imread('tire.tif');
>> f = @(x) uint8(round(std(double(x))));
>> I2 = colfilt(I1,[3 3],'sliding',f);
>> subplot(1,2,1), imshow(I1)
>> subplot(1,2,2), imshow(I2)
```



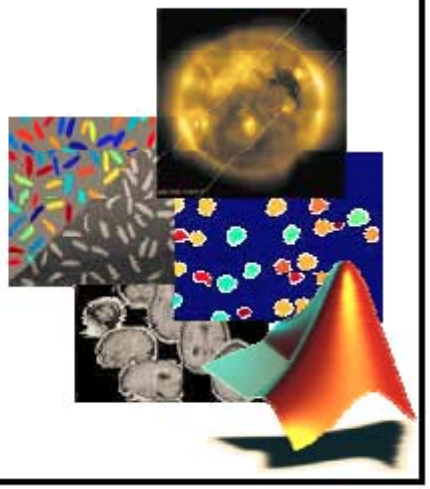
Section Summary:

1. What is block processing?
2. Distinct block operations
3. Sliding neighbourhood operations
4. Example of block processing
   - Convolution
   - Correlation
5. Column processing



Section Outline:

1. Reducing noise
    - Filters
    - Region-based processing
2. Image alignment
    - Rotation
    - Cropping
    - Resizing



Reducing Noise

Where does noise come from? 来自图像获取和操作过程

- Scanner resolution
- Film grain (granularity)
- Hardware (interference patterns)
- Other

Common types of noise

- Whitenoise (Gaussian)
- Local variance (Gaussian with intensity-dependent variance)
- Salt and pepper 椒盐噪声，椒-黑，盐-白，随机出现的黑白（脉冲）噪声
- Speckle 斑点噪声：出现在合成孔径雷达（SAR）图像上

A filter can be used to reduce the effect of noise in an image. The Image Processing Toolbox provides three main methods of filtering:

- Linear filtering
- Median filtering
- Adaptive filtering

Different methods are better for different kinds of noise.

In the following section we will investigate the types of methods, and what type of noise they are most effective in reducing.

How Do I Model Noise?

The function `imnoise` allows different types of noise to be modeled.

Syntax:

```
J = imnoise(I,type,parameters)
```

`I` – Image

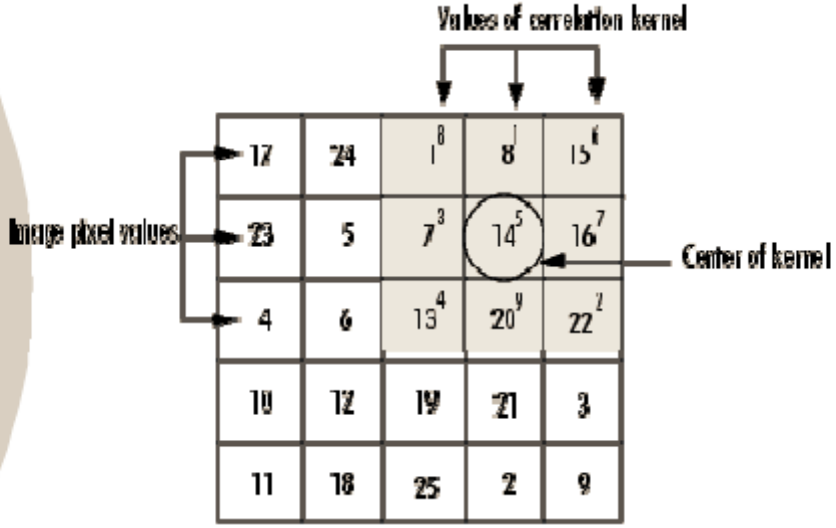
`type` – `gaussian`, `localvar`, `poisson`, `salt & pepper`, `speckle`

`parameters` – additional parameters needed given the type of noise

Linear Filtering

A linear filter computes each output pixel value according to a linear combination of the input pixel's neighborhood.

- The basics of linear filtering are done through *correlation* and *convolution*.



- In the Image Processing Toolbox both these operations are performed using the `imfilter` command.

Filtering using `imfilter`

Syntax

```
B = imfilter(A,H)
B = imfilter(A,H,option1,option2,...)
```

`A` – Input image

`H` – The filter, also known as the correlation/convolution kernel

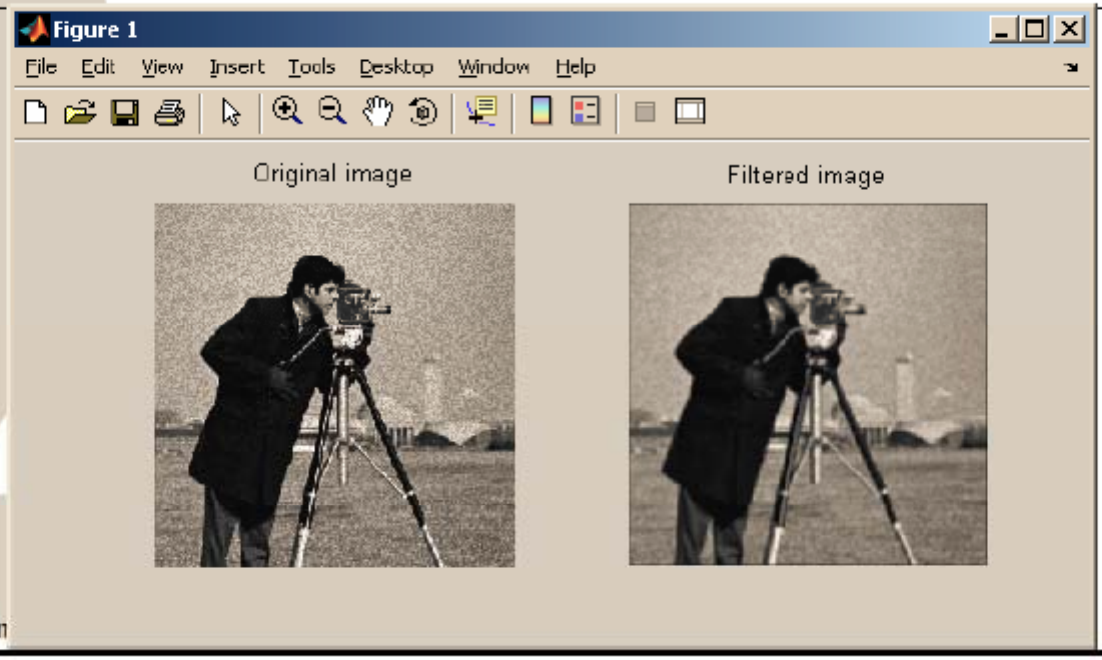
`options` – Boundary options, output size option, correlation and convolution option

Note: By default `imfilter` performs correlation.

Averaging Filter

A very basic example of a linear filter is an averaging filter.

```
>> I = imread('cameraman.tif');
>> % addition of graininess (i.e. noise)
>> I_noise = imnoise(I, 'speckle', 0.01);
>> % the average of 3^2, or 9 values
>> h = ones(3,3) / 3^2;
>> I2 = imfilter(I_noise,h);
>> subplot(1,2,1), imshow(I_noise), title('Original image')
>> subplot(1,2,2), imshow(I2), title('Filtered image')
```



Special Linear Filters

The function `fspecial` creates a variety of two-dimensional filters.

产生一个filter kernel，再使用imfilter（）

Syntax

```
h = fspecial(type, parameter)
```

`h` – two-dimensional correlation kernel

`type` – one of the specified special filter types: `gaussian`, `sobel`, `prewitt`, `laplacian`, `log`, `motion`, averaging (`average`), circular averaging (`disk`), and a contrast sharpening (`unsharp`) filter

9 种滤波器

`parameters` – particular to the type of filter chosen

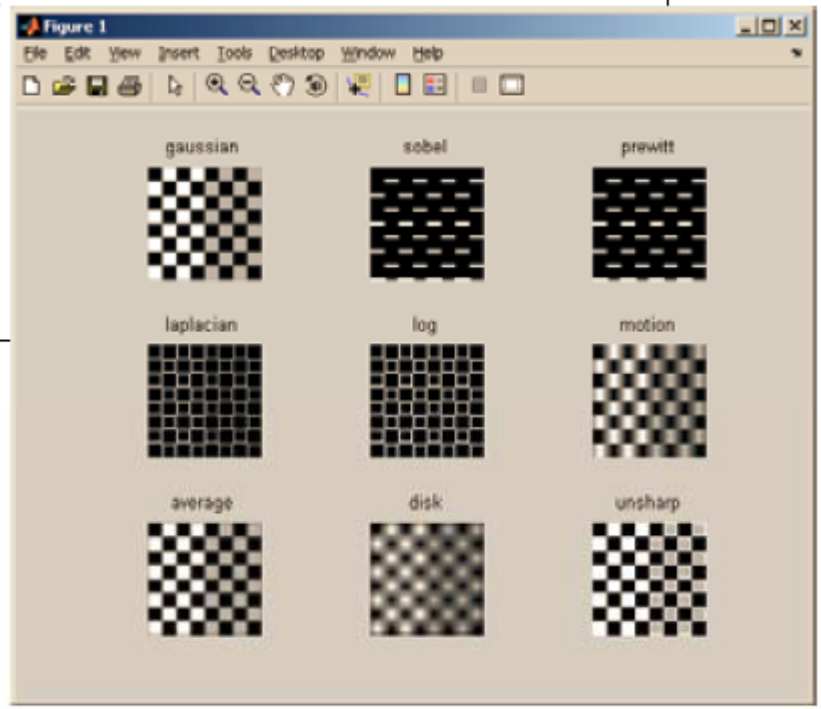
Exercise 3: Investigating Linear Filters

- Create a checkerboard image using the `checkerboard` function.

- Filter the checkerboard image with each `fspecial` filter type and display the image. For each filter type you can just use its default parameter(s).
- Extra credit: Display all nine images in one figure.

Solution: Investigating Linear Filters

```
>> I = checkerboard;
>> type = {'gaussian','sobel','prewitt',...
        'laplacian','log','motion',...
        'average','disk','unsharp'}; % cell arrays
>> for index = 1:length(type)
     h = fspecial(type{index});
     I2 = imfilter(I,h);
     subplot(3,3,index)
     imshow(I2)
     title(type{index})
   end
```



Median Filtering

When noise causes the pixels to vary greatly from the original value (salt and pepper), a median filter is more effective in reducing the noise.

Syntax

```
B = medfilt2(A,[m n])
```

说明：将以像素x为中心的m-by-n邻域内的像素排序，选出其中值替换x

`A` – Input image

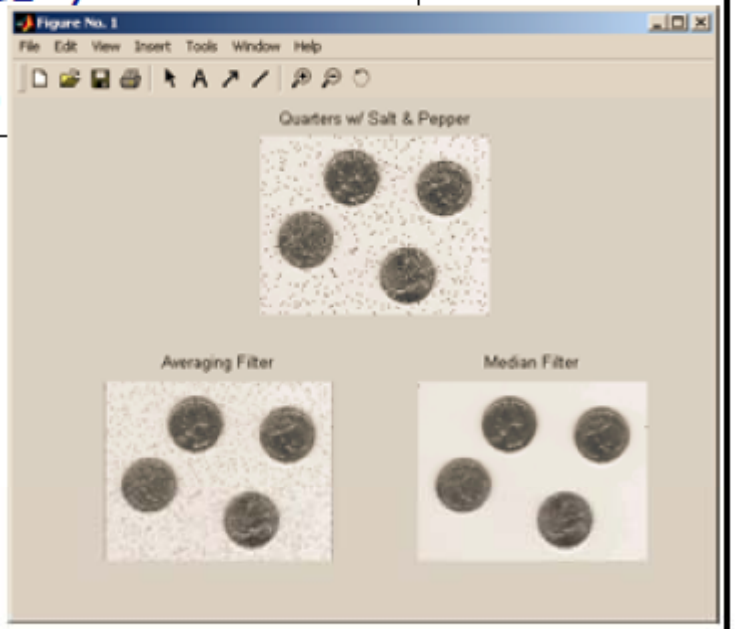
`B` – Output image

`[m n]` – Neighborhood block size to be used to calculate the median.

Example: Median Filter

```
>> I = imread('eight.tif');
>> I_noise = imnoise(I,'salt & pepper');
>> h = fspecial('average',[3 3]);
>> I_avg = imfilter(I_noise,h);
>> I_med = medfilt2(I_noise, [3 3]);
>> subplot(2,2,1.5)
>> imshow(I_noise), title('Quarters w/ Salt & Pepper')
>> subplot(2,2,3)
>> imshow(I_avg), title('Averaging Filter')
>> subplot(2,2,4)
>> imshow(I_med), title('Median Filter')
```

显示在第一行中间



Adaptive Filter

The `wiener2` adaptive filter tailors itself to the local image variance *adaptively*. If the variance is large, it minimally smoothes the image. If the variance is small, it performs more smoothing.

防止边缘信息被滤除

This type of filter is effective in reducing the effects of Gaussian whitenoise.

Syntax

```
[J,noise] = wiener2(I,[m n],noise)
```

`I` – Input image

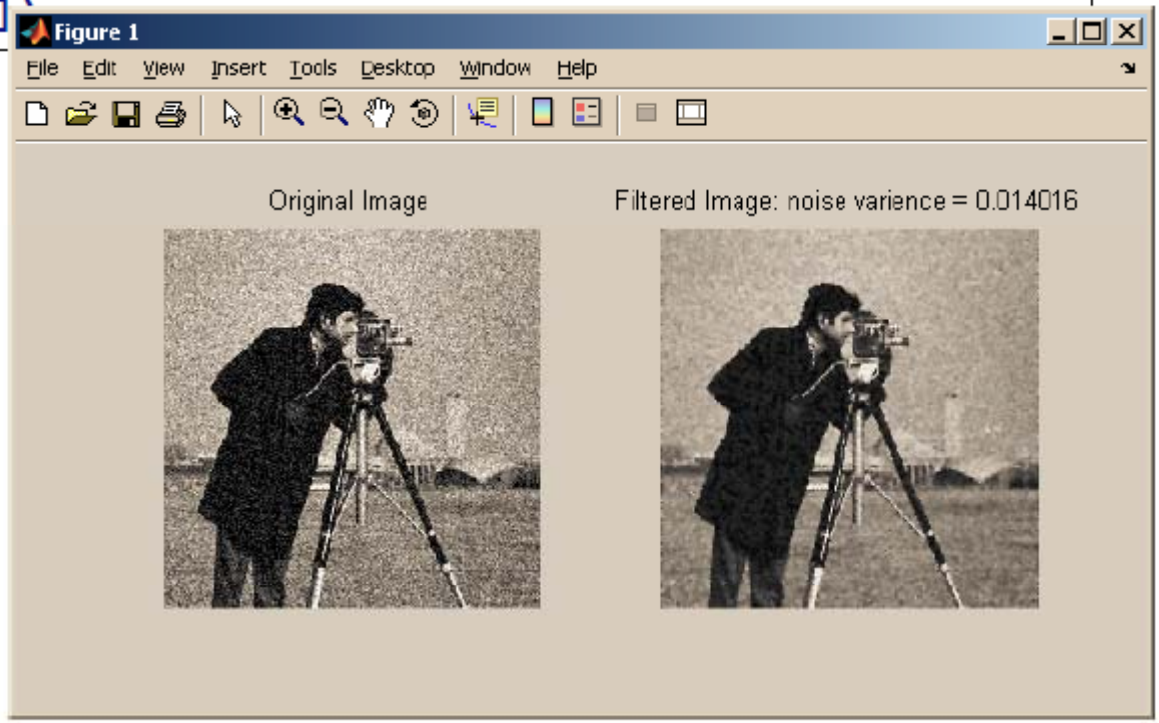
`[m n]` – Size neighborhood block used to calculate the ~~median~~. the local variance

`J` – Output image

`noise` – Noise variance

Wiener Filter Example

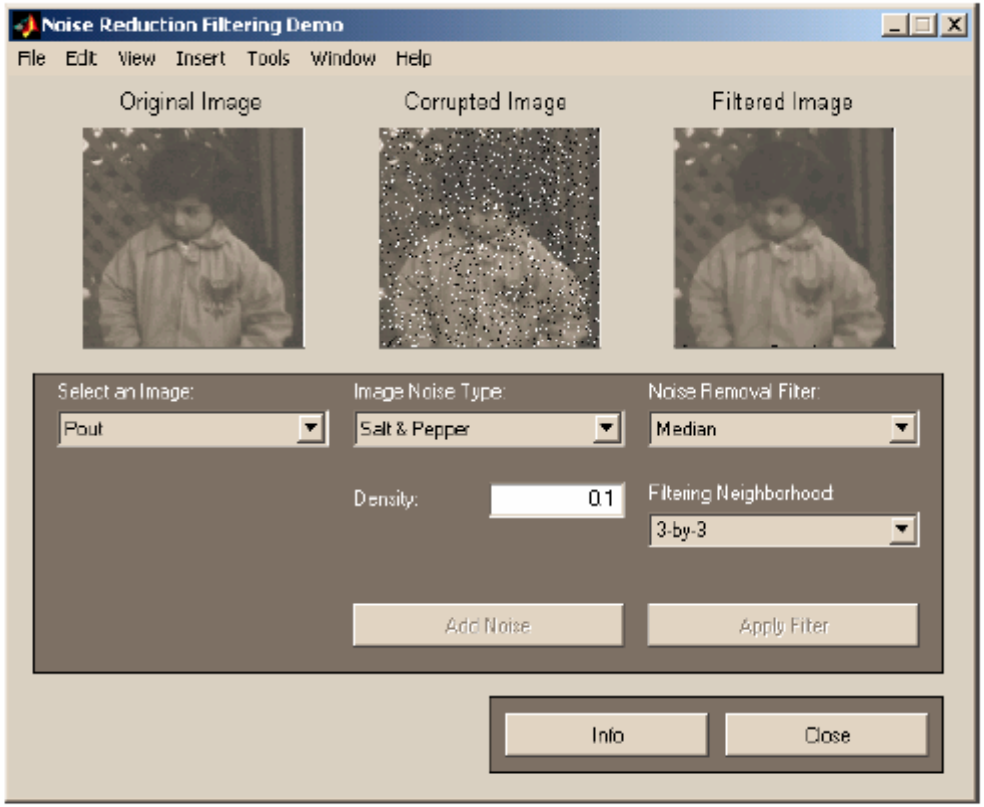
```
>> I = imread('cameraman.tif');
>> I_noise = imnoise(I, 'gaussian', 0.01);
>> [I2, noise] = wiener2(I_noise,[3 3]);
>> subplot(1,2,1), imshow(I_noise), title('Original Image')
>> subplot(1,2,2), imshow(I2)
>> title(['Filtered Image: noise variance =',...
        num2str(noise)])
```



Filter Demonstration

Explore noise reduction in images using linear and non-linear filtering techniques by running the following demo:

```
>> nrfiltdemo
```



Region-Based Processing

Region-based processing allows you to select a region of interest (ROI), and process only upon the selected area.

- A ROI is defined using a binary mask – The mask contains 1's for all pixels that are part of the region of interest and 0's everywhere else.
- Types of region-based processing that can be done:
  - Specify a region of interest (`roipoly`, `roicolor`)
  - Filter a region (`roifilt2`)
  - Fill in a region (`roifill`)

Specifying a Region of Interest

A region of interest can be specified using one of the Image Processing functions or with any user defined binary mask.

The options are:

- Using `roipoly`, allows you to specify a polygonal region of interest. When called with no inputs a cursor can be used to create a polygon.
- Using `roicolor`, where you can define the region of interest based on a color or intensity range.
- Using Boolean indexing to create a binary mask.

Filtering a Region

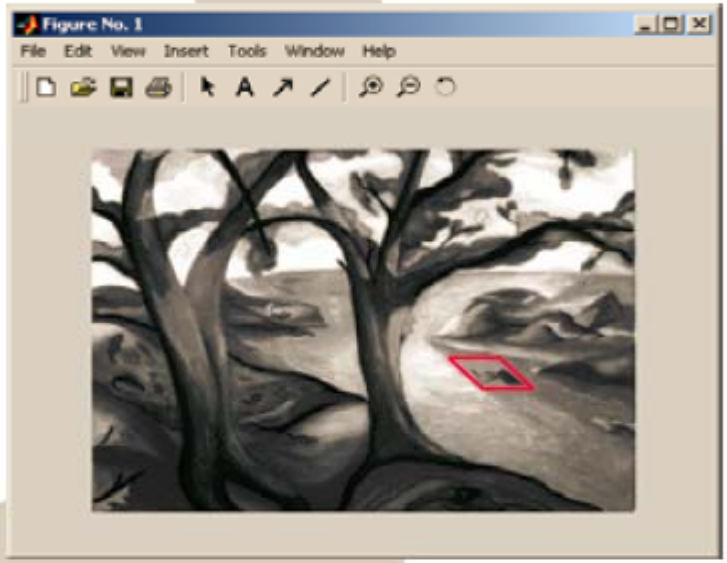
Once a region has been specified, a filter can be created and implemented on the region.

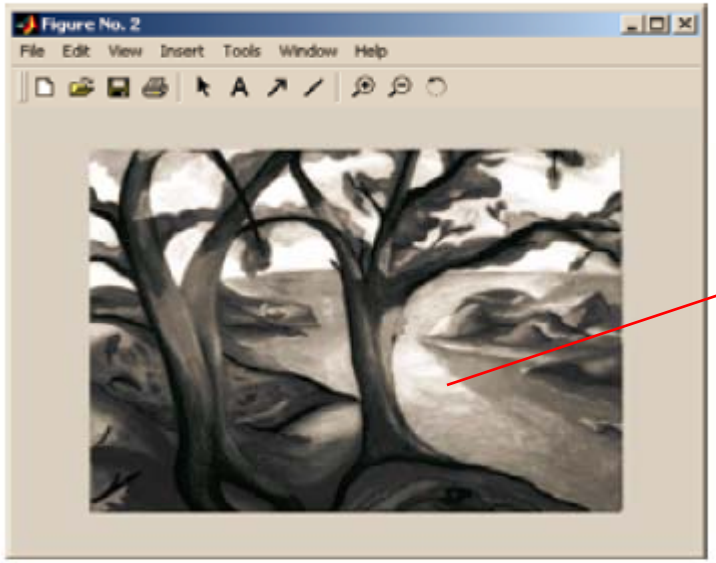
```
>> I = imread('cameraman.tif');
>> imshow(I)
>> BW = roipoly;
>> figure, imshow(BW)
>> h = fspecial('unsharp');
>> I2 = roifilt2(h,I,BW);
>> imshow(I2)
```

Filling in a Region of Interest (ROI)

```
>> [X,map] = imread('trees.tif');
>> I = ind2gray(X,map);
>> imshow(I)
>> I2 = roifill; % intensity img only
>> figure, imshow(I2)
```

产生一个交互式方框选择工具





所选择的细节被填充了

Image Alignment

When the alignment is off in an image, you can apply some basic spatial transformation techniques.

- Rotate the image (`imrotate`)
- Crop the image (`imcrop`)
- Resize the image (`imresize`)

With the `imtransform` function, you can transform your image to any geometric shape as well.

Image Rotation

注：沿图像中心，按逆时针方向旋转

Syntax

```
B = imrotate(A,angle,method)
B = imrotate(A,angle,method,'crop')
```

`B` – Output image

`A` – Input image

`angle` – Degrees of rotation in the counter-clockwise direction

`method` – Type of interpolation:
`[{nearest}, bilinear, bicubic]`

`'crop'` – Returns only central portion of `B` which is the same size as `A`.

Image Cropping

Syntax

```
I2 = imcrop(I,rect)
```

`I2` – Output image

`I` – Input image

`rect` – Spatial coordinates of
`[xmin ymin width height]`

If `rect` is omitted, you specify the crop region on the image directly using the mouse.

Image Resizing

Syntax

```
B = imresize(A,m,method)
```

`B` – Output image

`A` – Input image

`m` – Magnification factor

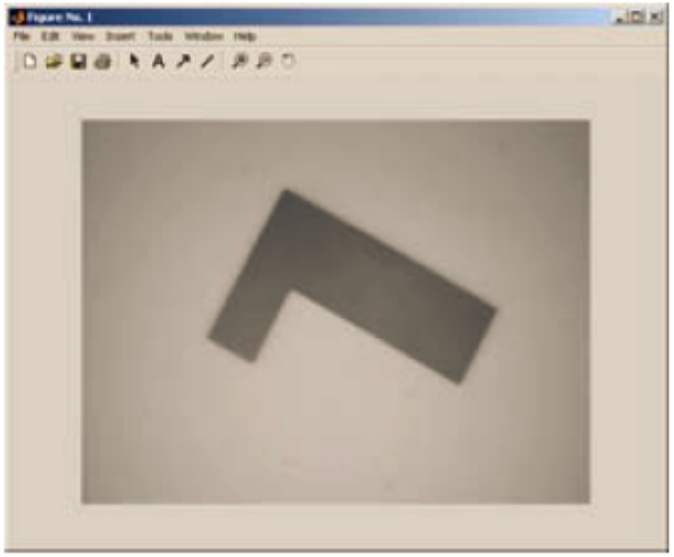
`method` – Type of interpolation:

```
[{nearest}, bilinear, bicubic]
```

Exercise 4: Aligning an Object

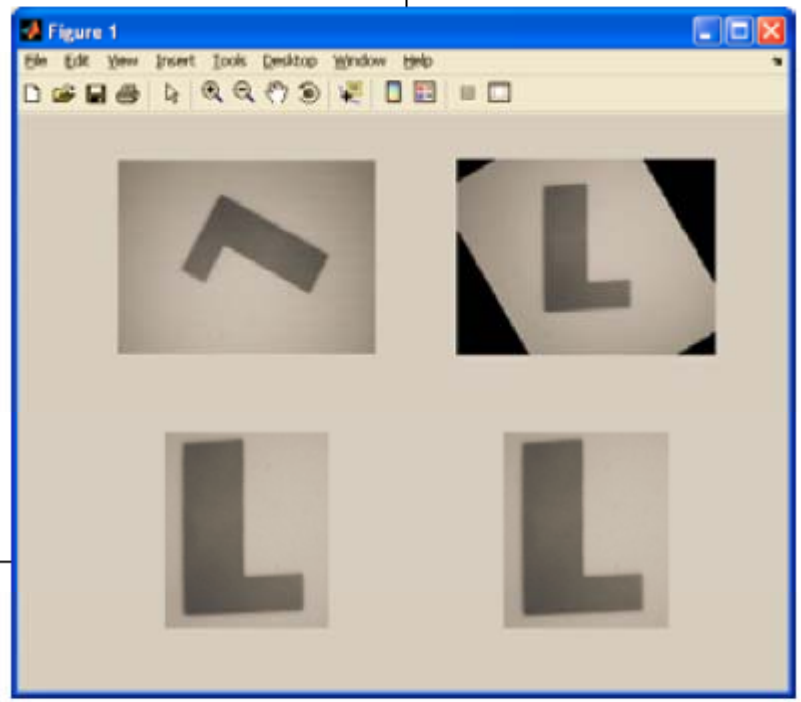
- Rotate the image so that the letter L is standing right side up.
- Crop the image so that only the letter is showing.
- Resize the image so that it is the same as the original image.

```
>> [X, map] = imread('bw_L.gif');
>> I = ind2gray(X,map);
```



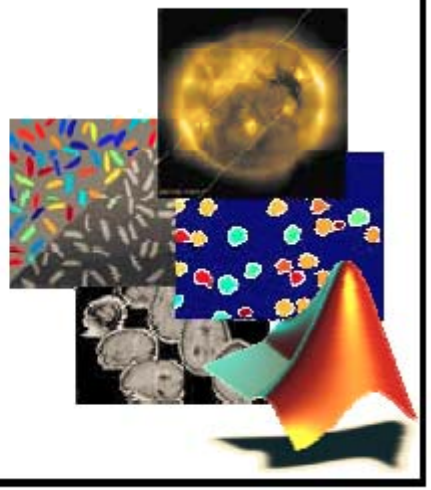
Solution: Aligning an Object

```
>> [X, map] = imread('bw_L.gif');
>> I = ind2gray(X,map);
>> size_I = size(I);
>> subplot(2,2,1), imshow(I)

>> J = imrotate(I,120,'nearest','crop');
>> subplot(2,2,2), imshow(J)

>> K = imcrop(J);
>> subplot(2,2,3), imshow(K)

>> L = imresize(K,size_I);
>> subplot(2,2,4), imshow(L)

>> whos I J K L
```



Section Summary:

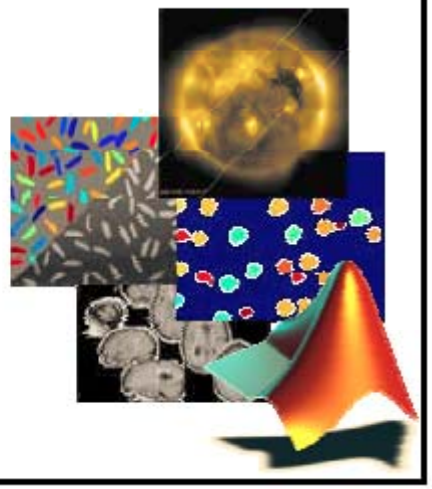
1. Reducing noise
    - Filters
    - Region-based processing
2. Image alignment
    - Rotation
    - Cropping
    - Resizing



Section Outline:

1. Morphology and segmentation
2. Structuring element
3. Dilation and erosion
4. Edge detection

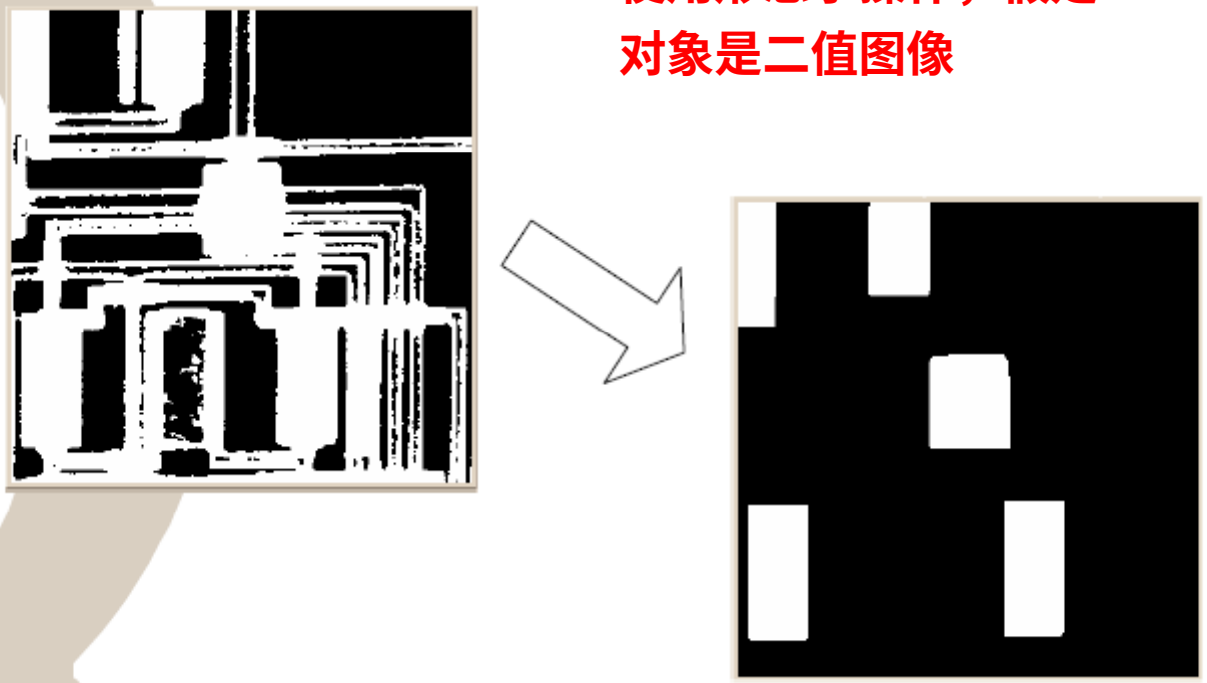
形态学操作：
膨胀和腐蚀



Example Problem

We are assigned to locate all the rectangular chips on a black and white integrated circuit image, but how?

使用形态学操作，假定
对象是二值图像



Morphology & Segmentation

One way we can solve the problem of identifying objects is using morphological techniques to segment the objects.

- Morphology – technique used for processing image based on shapes.
- Segmentation – the process used for identifying objects in an image.

Structuring Element

To process an image according to a given shape, we need to first define the shape, or *structuring element*.

The Image Processing Toolbox provides a function for creating a structuring element that can be used, `strel`.

Create morphological structuring element

Syntax

```
SE = strel(shape,parameters)
```

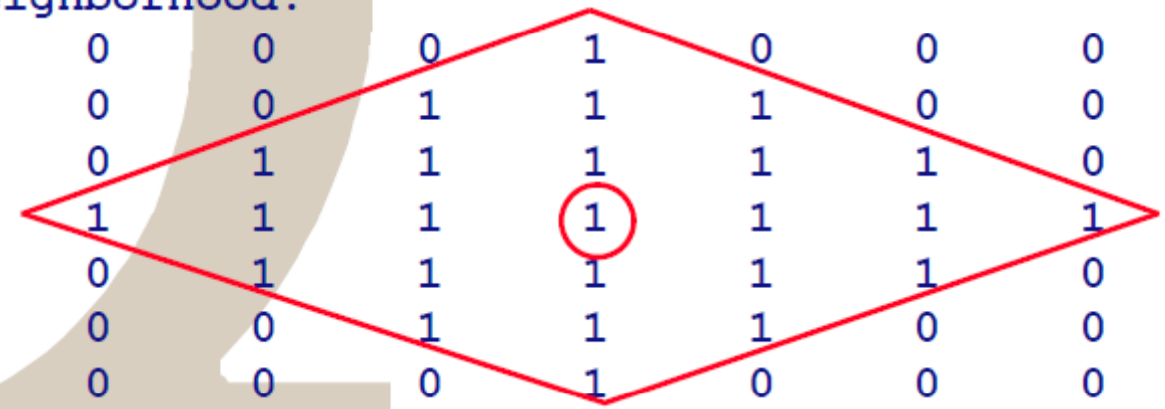
`SE` – Structuring element

`shape` – Flat ['arbitrary' 'pair' 'diamond' 'periodicline' 'disk' 'rectangle' 'line' 'square' 'octagon']
Nonflat ['arbitrary' 'ball']

`parameters` – Associated with the selected shape

Below is an example of a diamond shaped structuring element with the distance from the structuring element origin to the points of the diamond being 3.

```
>> SE = strel('diamond',3)

SE =
Flat STREL object containing 25 neighbors.
Decomposition: 3 STREL objects containing a total of 13
neighbors

Neighborhood:
     0     0     0     1     0     0     0
     0     0     1     1     1     0     0
     0     1     1     1     1     1     0
     1     1     1     1     1     1     1
     0     1     1     1     1     1     0
     0     0     1     1     1     0     0
     0     0     0     1     0     0     0
```



Dilation & Erosion

To dilate an image, use the `imdilate` function, and to erode an image, use the `imerode` function.

Syntax

```
IM2 = imdilate(IM,SE)
IM2 = imerode(IM,SE)
```

`IM2` – The dilated/eroded image

`IM` – The image to dilate/erode

`SE` – The structuring element

Below is an example of dilating a square binary image.

```
BW = zeros(9,10);
BW(4:6,4:7) = 1
SE = strel('square',3)
BW2 = imdilate(BW,SE)
```

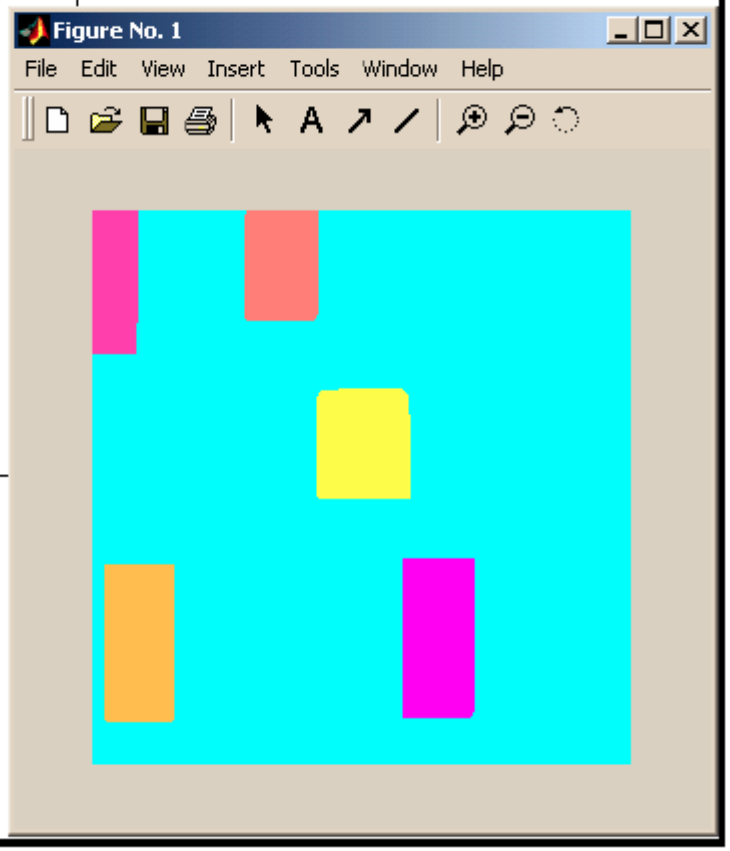
```
BW =
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     1     1     1     1     0     0     0
     0     0     0     1     1     1     1     0     0     0
     0     0     0     1     1     1     1     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
```

```
BW2 =
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     1     1     1     1     1     1     0     0
     0     0     1     1     1     1     1     1     0     0
     0     0     1     1     1     1     1     1     0     0
     0     0     1     1     1     1     1     1     0     0
     0     0     1     1     1     1     1     1     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
```

Example Solution

To locate the chips on the integrated circuit we can use the morphological techniques of erosion and dilation.

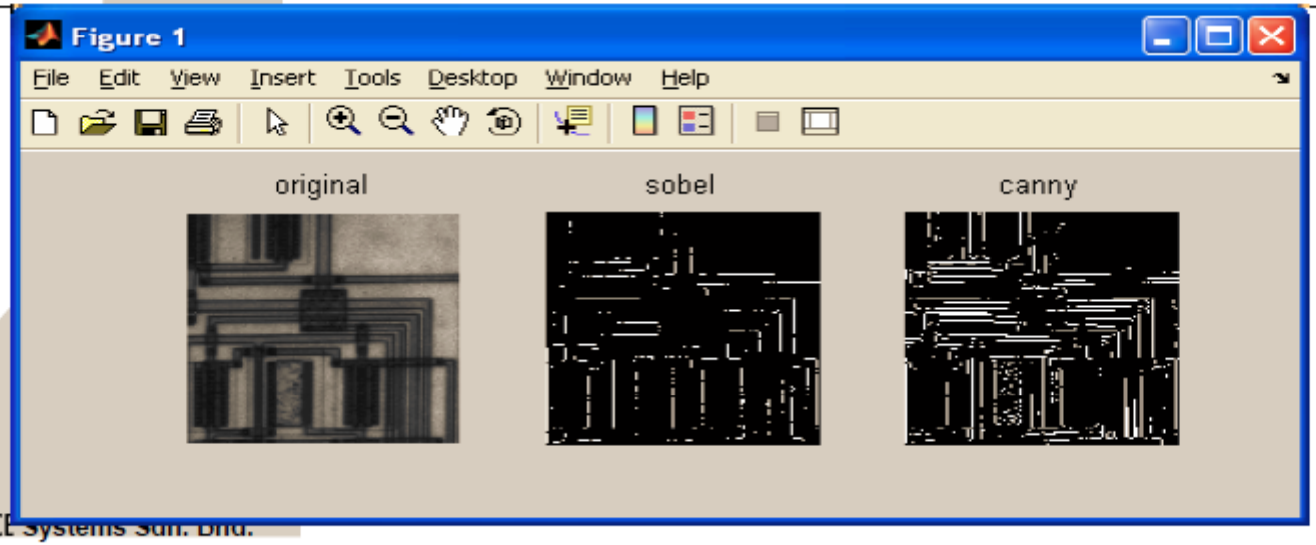
```
>> BW1 = imread('circbw.tif');
>> SE = strel('rectangle',[40 30])
>> BW2 = imerode(BW1,SE);
>> BW3 = imdilate(BW2,SE);
>> L = bwlabel(BW3);
>> RGB = label2rgb(L, 'spring',...
 'c', 'shuffle');
>> imshow(RGB)
```



Edge Detection

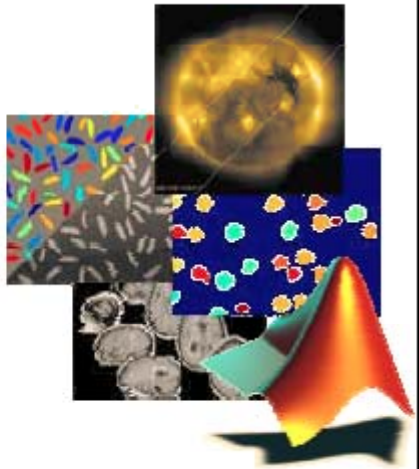
The edge function of the Image Processing Toolbox makes it easy to perform edge detection.

```
>> I = imread('circuit.tif');
>> BW1 = edge(I,'sobel');
>> BW2 = edge(I,'canny');
>> subplot(131),imshow(I),title('Original')
>> subplot(132),imshow(BW1),title('sobel')
>> subplot(133),imshow(BW2),title('canny')
>> edgedemo
```



Section Summary:

1. Morphology and segmentation
2. Structuring element
3. Dilation and erosion
4. Edge detection



The End

Kindly return your Evaluation Form

Tel: 603 – 8076 1953 Fax: 603 – 8076 1954
Email: info@techsource.com.my Web: www.techsource.com.my
Tech-Support: techsupport@techsource.com.my