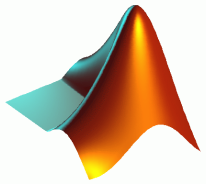


Laboratory of Image Processing

Image Processing with MatLab

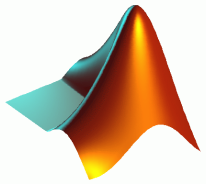
Pier Luigi Mazzeo

pierluigi.mazzeo@cnr.it

Institute of Applied Sciences and Intelligent Systems

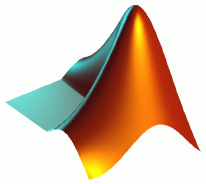
ScienceApp

National Research Council of Italy



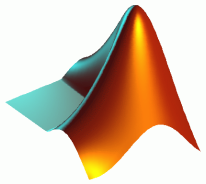
Introduction

- An image is digitized to convert it to a form which can be stored in a computer's memory or on some form of storage media such as a hard disk or CD-ROM.
- Once the image has been digitized, it can be operated upon by various image processing operations.



Introduction

- Image processing operations can be roughly divided into three major categories,
  - **Image Compression**,
  - **Image Enhancement**
  - **Restoration**, and **Measurement Extraction**.



Introduction

- Image defects which could be caused by the digitization process or by faults in the imaging set-up (for example, bad lighting) can be corrected using **Image Enhancement** techniques.
- Once the image is in good condition, the **Measurement Extraction** operations can be used to obtain useful information from the image.

Image Enhancement and Restoration

The image at the left has been corrupted by noise during the digitization process. The 'clean' image at the right was obtained by applying a median filter to the image.

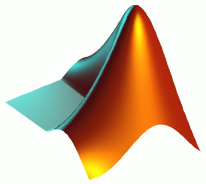
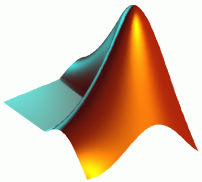


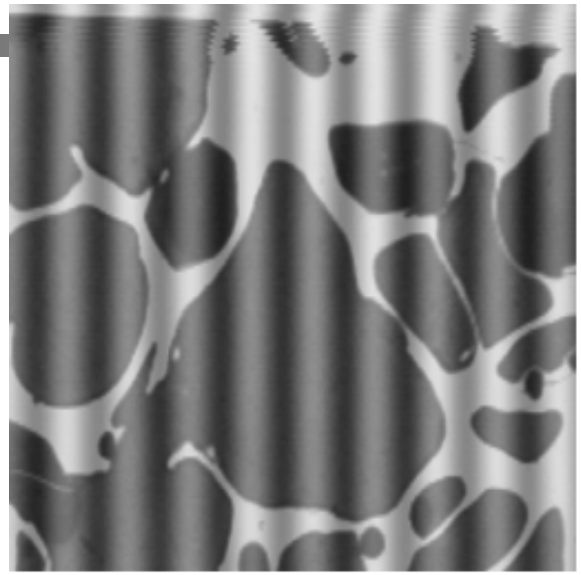
Image Enhancement and Restoration

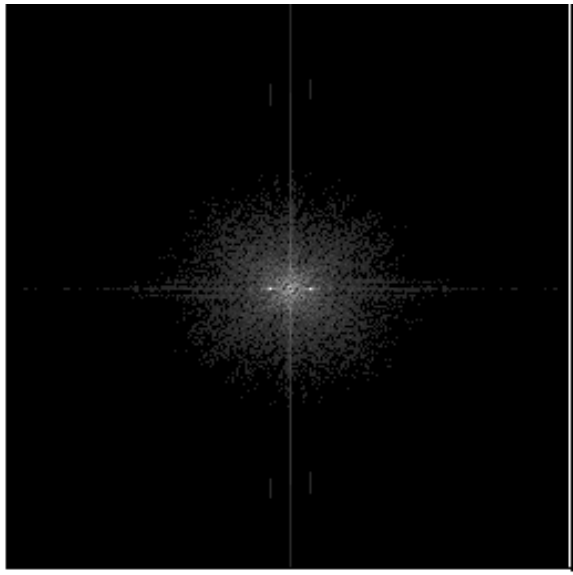
An image with poor contrast, such as the one at the left, can be improved by adjusting the image histogram to produce the image shown at the right.

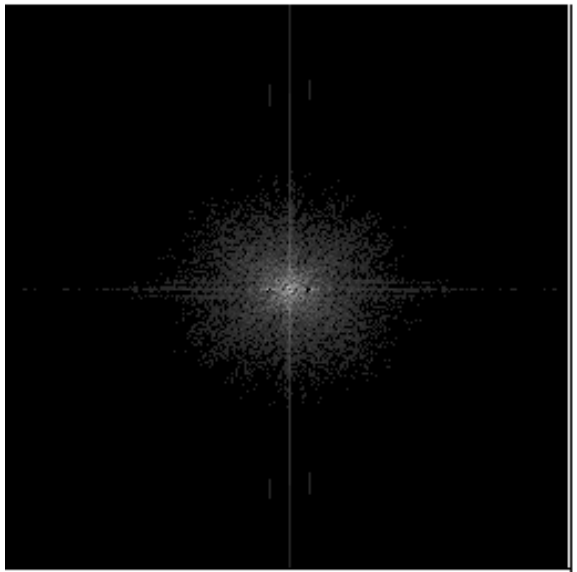
Image Enhancement and Restoration



The image at the top left has a corrugated effect due to a fault in the acquisition process. This can be removed by doing a 2-dimensional Fast-Fourier Transform on the image (top right), removing the bright spots (bottom left), and finally doing an inverse Fast Fourier Transform to return to the original image without the corrugated background (bottom right).







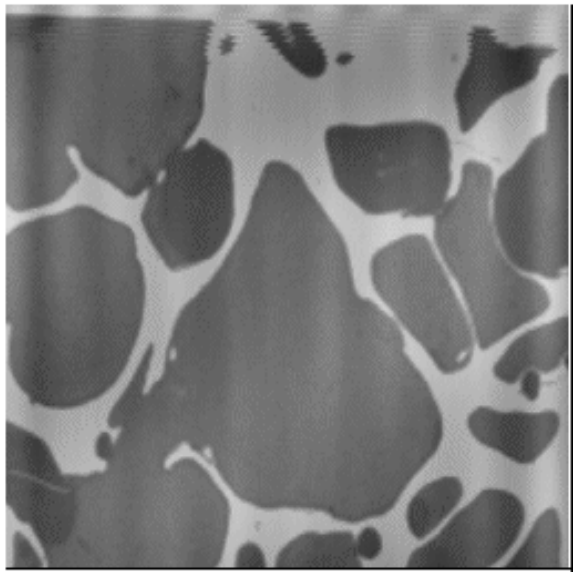
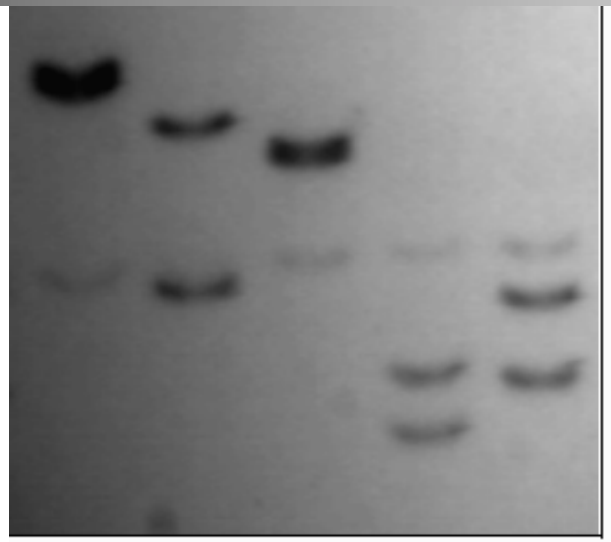
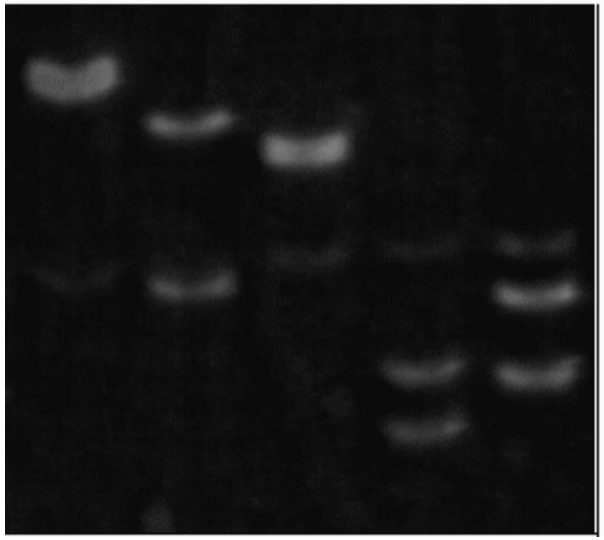


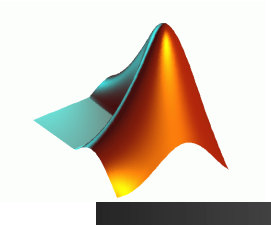
Image Enhancement and Restoration

An image which has been captured in poor lighting conditions, and shows a continuous change in the background brightness across the image (top left) can be corrected:

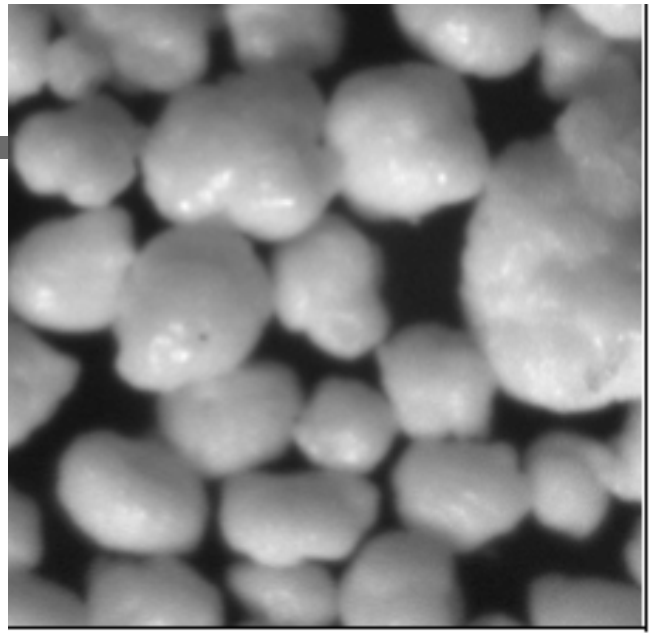
- First remove the foreground objects by applying a 25 by 25 grey-scale dilation operation (top right).
- Then subtract the original image from the background image (bottom left).
- Finally invert the colors and improve the contrast by adjusting the image histogram (bottom right)

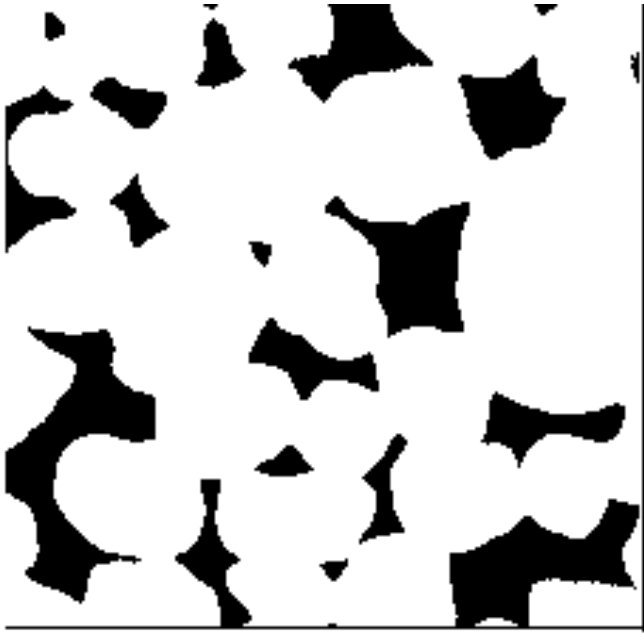


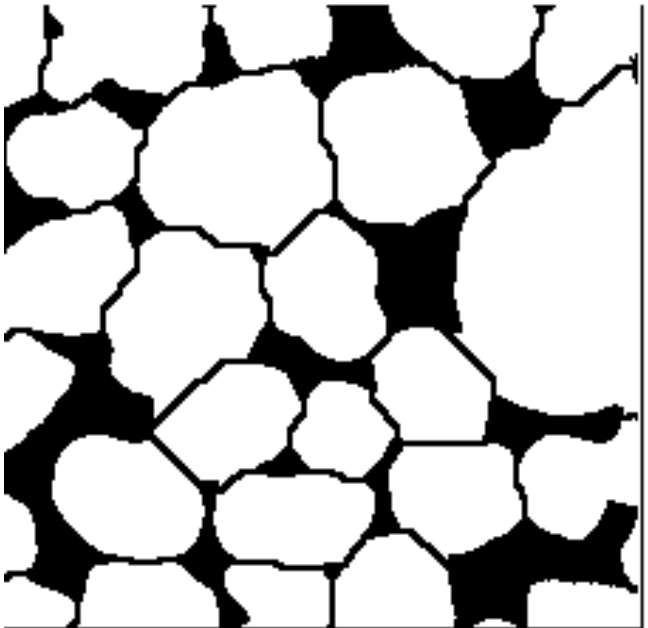


Top-left:
Original image
Top-right:
Separating image from the background
Bottom left:
Water shade seperation
Bottom right:
Separated image







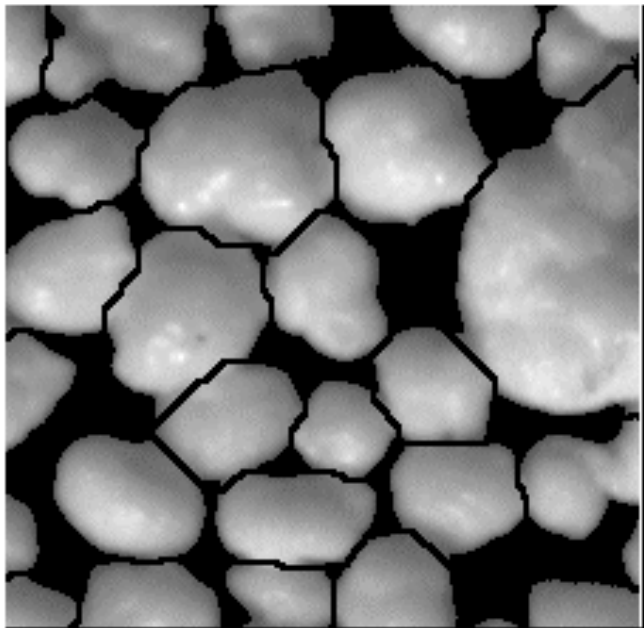
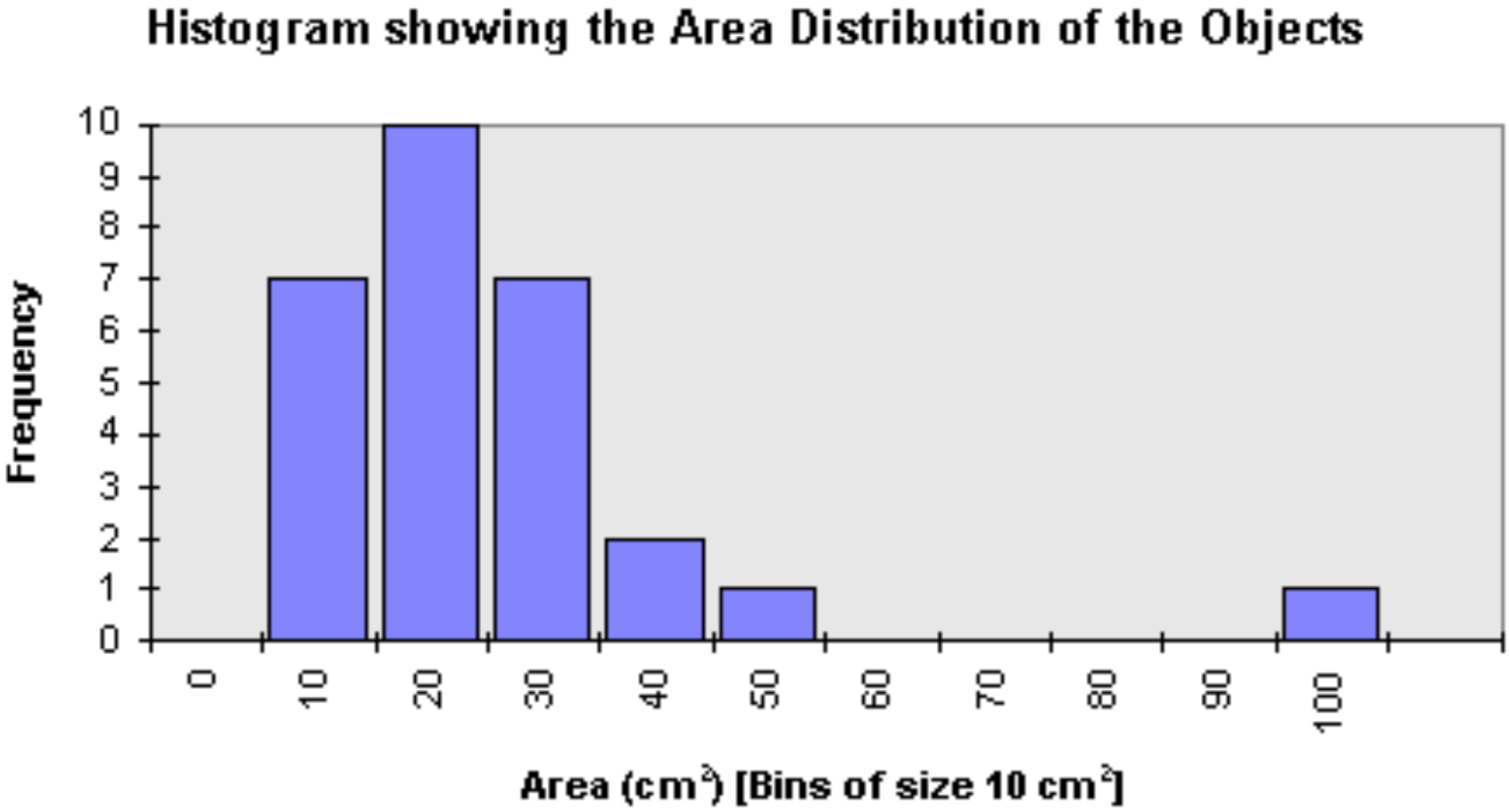
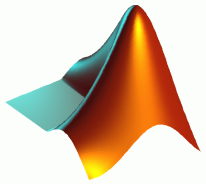


Image Measurement Extraction

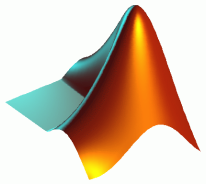


•The areas were calculated based on the assumption that the width of the image is 28 cm.



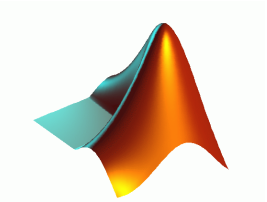
1. Digital Image Representation

- An image is defined by a 2D function $f(x, y)$
  - spatial (plane) coordinates (x, y)
  - intensity of the image $f(x, y)$
- An image may be continuous wrt x and y coordinates. Converting such an image to digital form is needed.
  - Digitizing the x and y coordinates values is called *sampling.*
  - Digitizing the ***amplitude*** (f) values is called *quantization.*

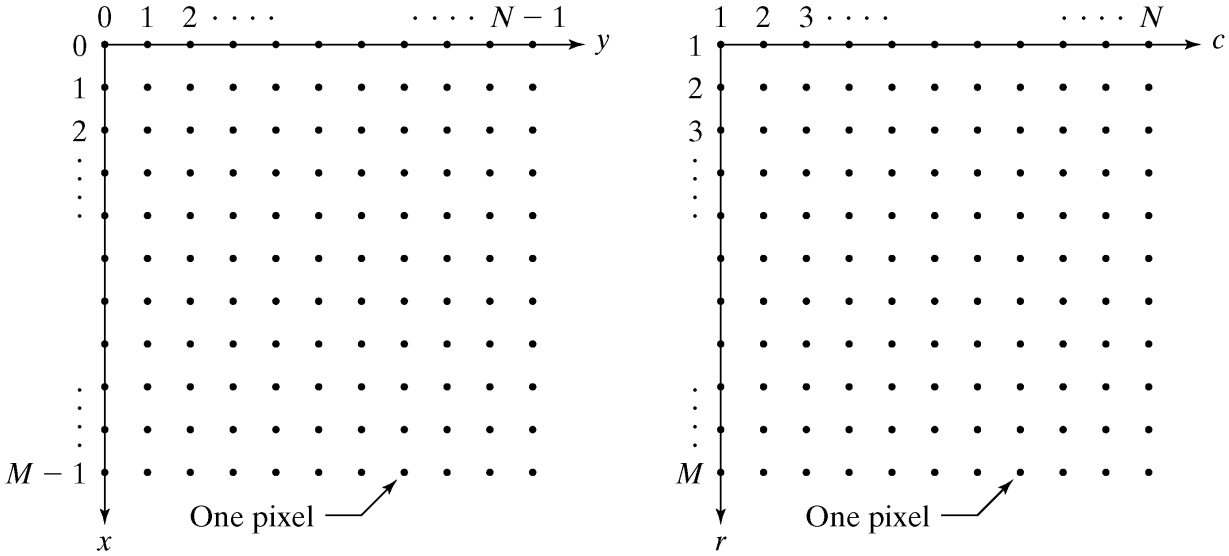


1.1 Coordinate Convention

- Two principal way to represent digital images: We assume that an image $f(x,y)$ is sampled so that the resulting image has M rows and N columns.
  - The image origin is defined to be at $(x,y) = (0,0)$
  - The image origin is defined to be at $(x,y) = (1,1)$
- The toolbox uses the notation (r,c). The origin of the coordinate system is at $(r,c)=(1,1)$.

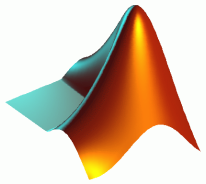


1.1 Coordinate Convention (cont.)



a b

FIGURE 2.1 Coordinate conventions used (a) in many image processing books, and (b) in the Image Processing Toolbox.

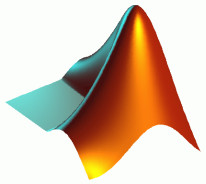


1.2 Images as Matrices

- MATLAB representation of an image

$$\mathrm{f} = \begin{bmatrix} f(1,1) & f(1,2) & \cdots & f(1,N) \\ f(2,1) & f(2,2) & \cdots & f(2,N) \\ \vdots & \vdots & \cdots & \vdots \\ f(M,1) & f(M,2) & \cdots & f(M,N) \end{bmatrix}$$

- Matrices in MATLAB are stored in variables with names such as **A**, **a** ...etc.



1.2 Reading Images

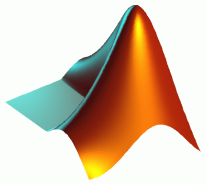
- Images are red into the MATLAB environment as

```
imread (' file name ')
```

Example:

```
f = imread ('chest-xray.tif')
```

reads the tif image chest-xray.tif into image array f.

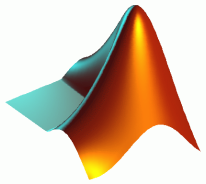


1.2 Reading Images

- Function size gives the row and column dimensions

  Example:

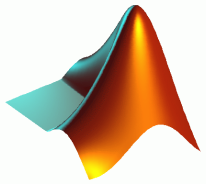
```
>> size(f)
ans =
     494   600
```



1.2 Reading Images (cont.)

- To determine the size of an image

```
>> [M,N]=size (f)
   M =
      494
   N =
      600
```



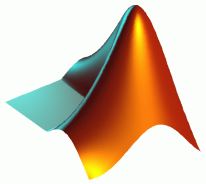
1.2 Reading Images (cont.)

- To display information about an array

- ```
  >> whos f
    Name        Size                     Bytes
  Class

    f         494x600                   296400
  uint8 array

  Grand total is 296400 elements using 296400 bytes
  ```
  ```

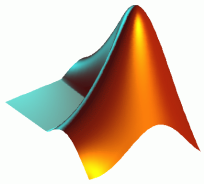


1.3 Displaying Images)

- Images are displayed using function imshow

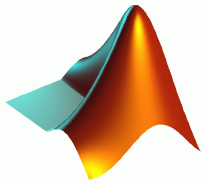
  imshow(f,G)

  where f is an image array, and G is the number of intensity levels used to display it.



1.3 Displaying Images

FIGURE 2.2
Screen capture showing how an image appears on the MATLAB desktop. However, in most of the examples throughout this book, only the images themselves are shown. Note the figure number on the top, left part of the window.



1.3 Displaying Images(cont.)

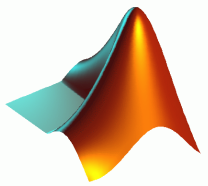
- Images are displayed using function imshow

  `imshow(f,G)`

  where f is an image array, and G is the number of intensity levels used to display it.

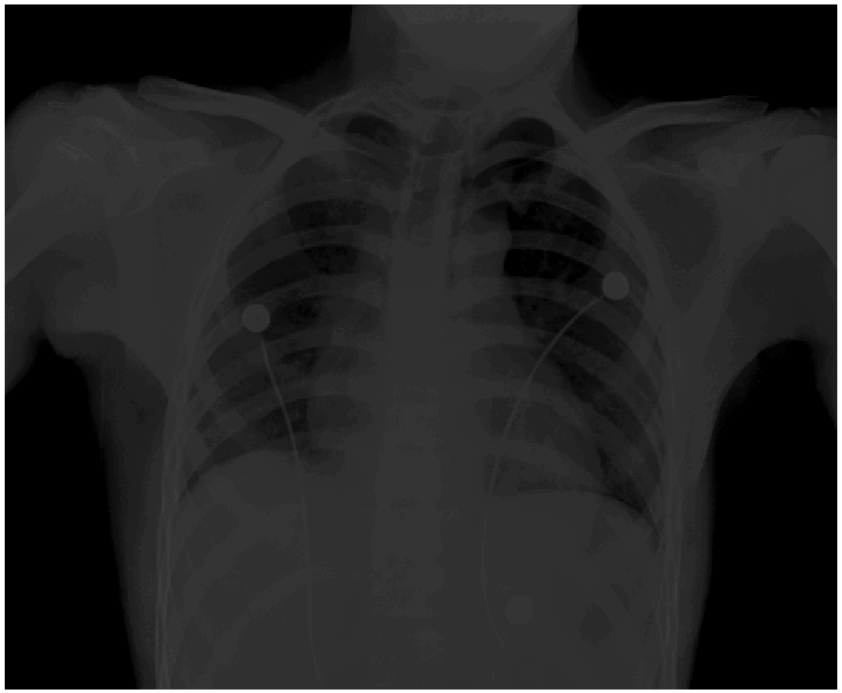
  `imshow(f,[low high])`

  displays as black all values less than or equal to low, and as white all values greater than or equal to high.

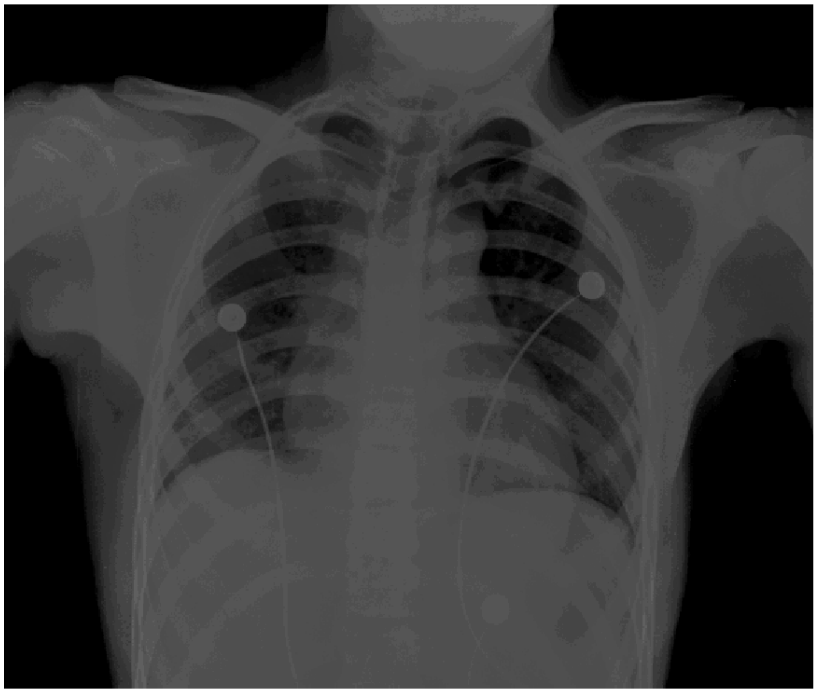


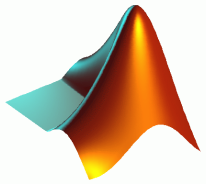
1.3 Displaying Images(cont.)

```
>> imshow(f,[0 255])
```



```
>> imshow(f,[0 155])
```





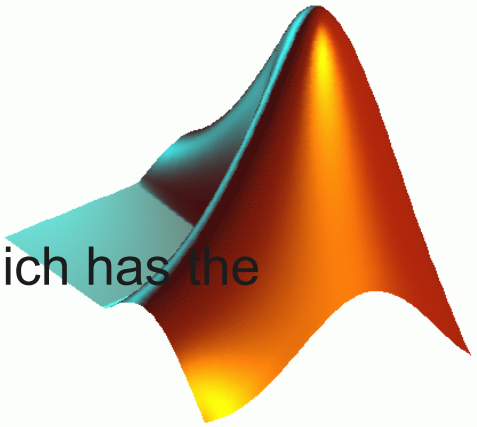
1.3 Displaying Images(cont.)

- Function pixval is used to display the intensity values of individual pixels interactively.

- Two images can be displayed simultaneously by

```
>> imshow(f,[0 255]),
figure,imshow(f,[])
```

1.4 Writing Images



Images are written to disk using function imwrite, which has the following basic syntax:

```
>> imwrite(f,'filename')
```

The desired format can be specified explicitly with a third input argument.

```
>> imwrite(f ,'patient10_run1','tif')
```
or alternatively
```
>> imwrite(f ,'patient10_run1.tif')
```

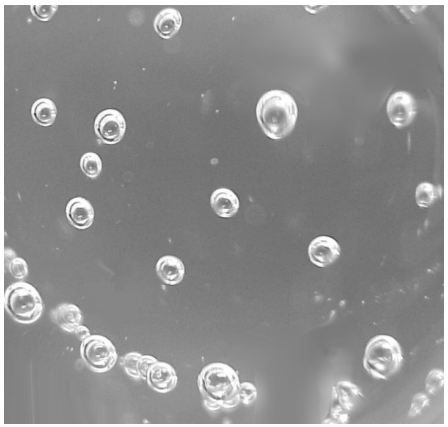
A more general imwrite syntax applicable only to JPEG images is

```
>>imwrite(f, 'filename.jpg', 'quality', 'q')
```

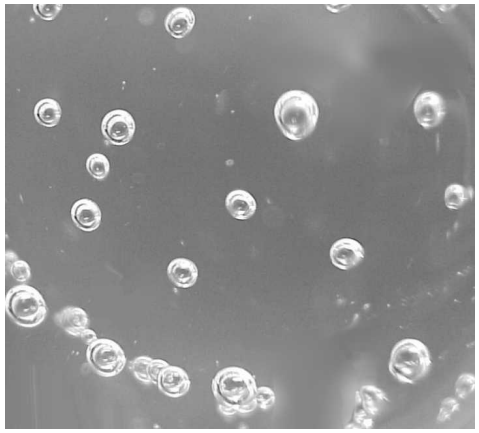
where q is an integer between 0 and 100 (the lower the number the higher the degradation due to JPEG compression)

- Writing image f to disk (in JPEG format), with q = 50, 25, 15, 5 and 0, respectively by using syntax

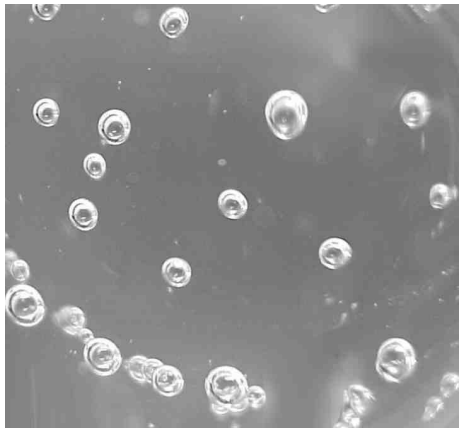
```
>>imwrite(f, 'bubbless.jpg', 'quality',q)
```



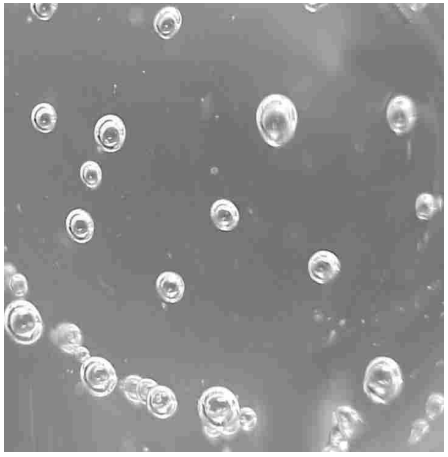
Original image



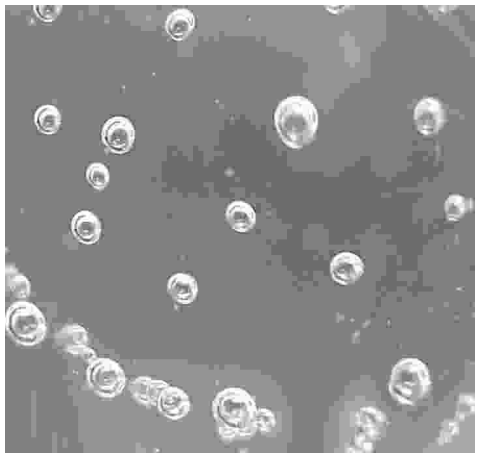
q = 50



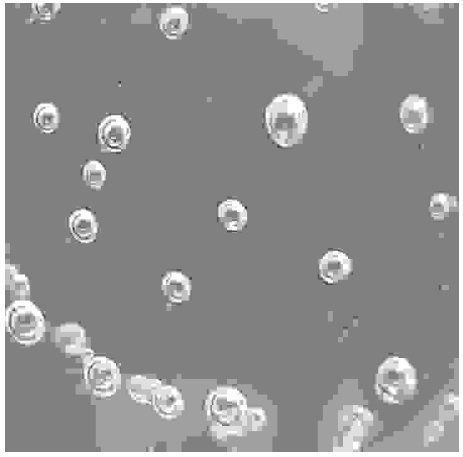
q = 25



q = 15

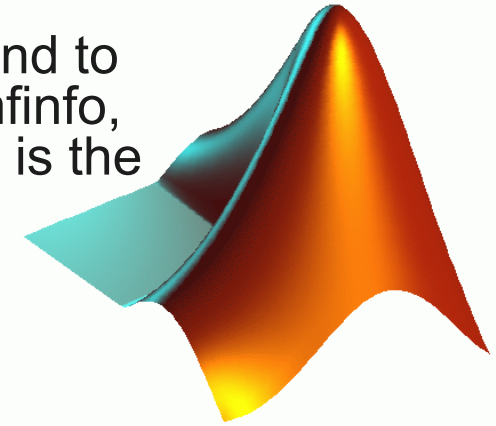


q = 5



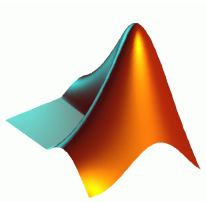
q = 0

In order to get an idea of the compression achieved and to obtain other image file details, we can use function imfinfo, which has the syntax iminfo filename where file name is the *complete* file name of the image stored in disk.



```
>> imfinfo bubbles.jpg

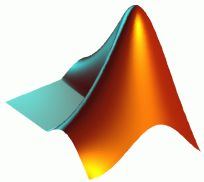
            Filename: 'bubbles.jpg'
         FileModDate: '13-Feb-2006 10:53:12'
            FileSize: 13354
              Format: 'jpg'
       FormatVersion: ' ‘
               Width: 720
              Height: 688
            BitDepth: 8
           ColorType: 'grayscale'
     FormatSignature: ''
             Comment: { }
```



• The information fields displayed by imfinfo can be captured into a so called *structure variable* that can be used for subsequent computations.

• Assigning the name K to the structure variable, we use the syntax to store into variable K all the information generated by command imfinfo

```
>> K = imfinfo ('bubbles.jpg');
```

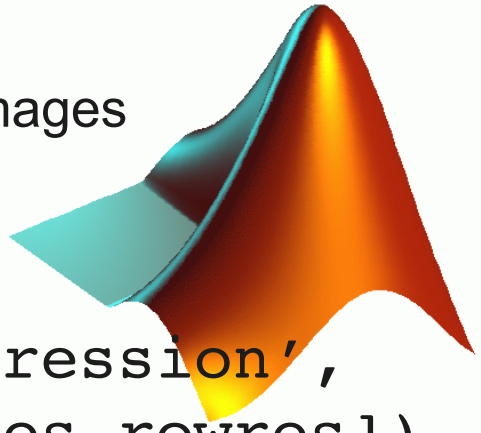


- The information generated by iminfo is appended to the structure variable by means of fields, separated from K by a dot.

```
>> K = imfinfo('bubbles.jpg');
>> image_bytes = K.Width * K.Height *
 K.BitDepth / 8;
>> compressed_bytes = K.FileSize;
>> compressed_ratio= image_bytes /
 compressed_bytes;
>> compressed_ratio

 compressed_ratio =
    0.9988
```

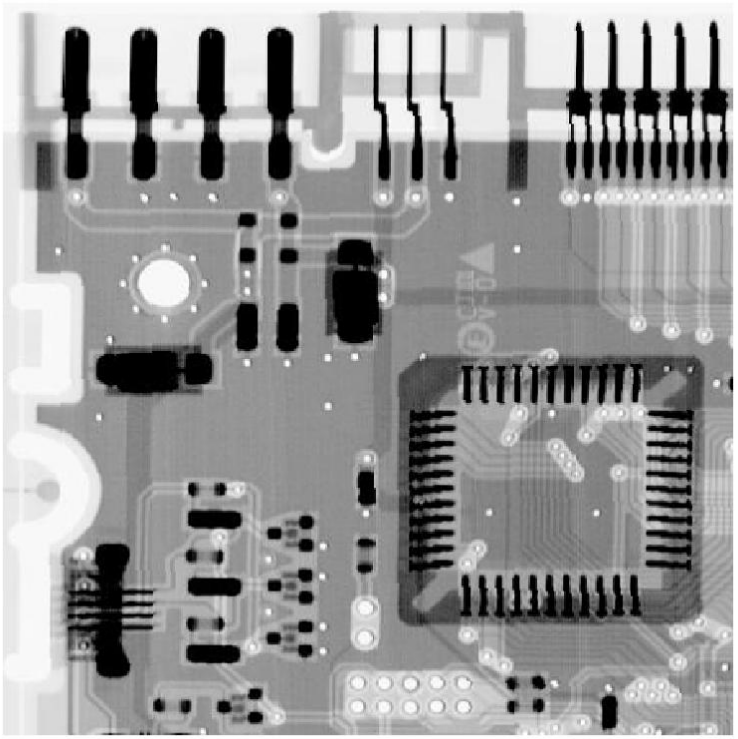
A more general imwrite syntax application only to tif images has the form

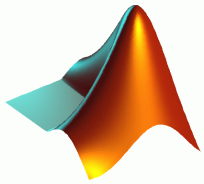


```
>> imwrite(g, 'filename.tif','compression',
'parameter',   …'resolution',[colres rowres])
```

- where 'parameter' can have one of the following principal values:
- 'none' indicates no compression;
- 'packbits' indicates packbits compression (the default for nonbinary images);
- and 'ccitt' compression (the default for binary images).
- The 1 x 2 array [colres rowres] contains two integers that give the column resolution and row resolution in dots-per-unit (the default values are [72 72]).

An 8-bit X-ray image of a circuit board generated during quality inspection, which is in jpg format at 200 dpi. It is of size 450 x 450 pixels, so its dimensions are 2.25 x 2.25 inches.





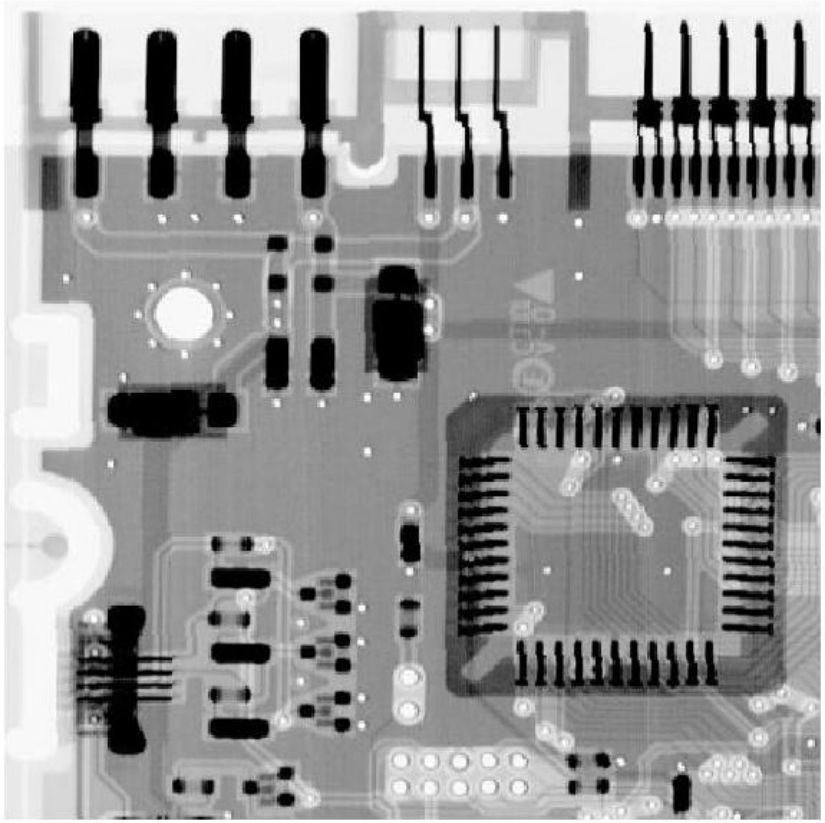
The statement to reduce the size of the image (2.5x2.5 inches) at 200 dpi to 1.5 x 1.5 inches while keeping the pixel count at 450 x 450 and to store it in tif format is:

```
>> imwrite(f, 'sf.tif', 'compression',
   'none', resolution',[300 300])
```

[colres rowres] were determined by multiplying 200 dpi by the ratio 2.25 / 1.5, which gives 300 dpi.

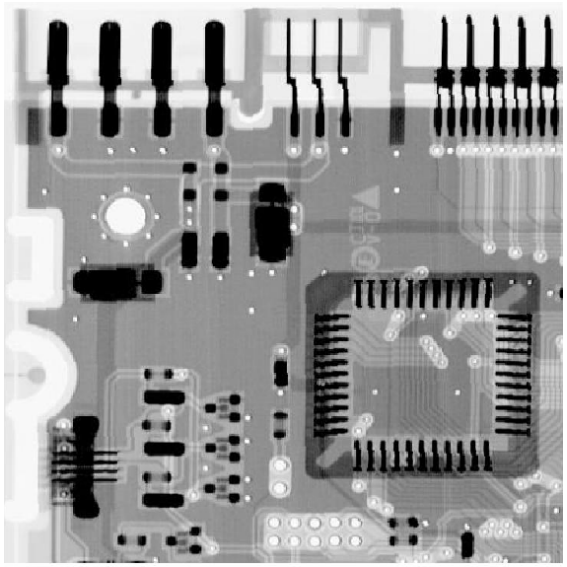
```
>> res = round(200*2.25/1.5); % rounds its argument to
the nearest integer
>> imwrite(f, 'sf.tif', 'compression', 'none',
'resolution', res)
```

The number of pixels was not changed by these commands. Only the scale of the image changed.

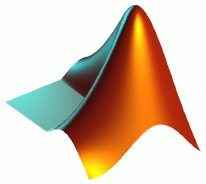


- The original 450 x 450 image at 200 dpi is of size 2.25 x 2.25 inches

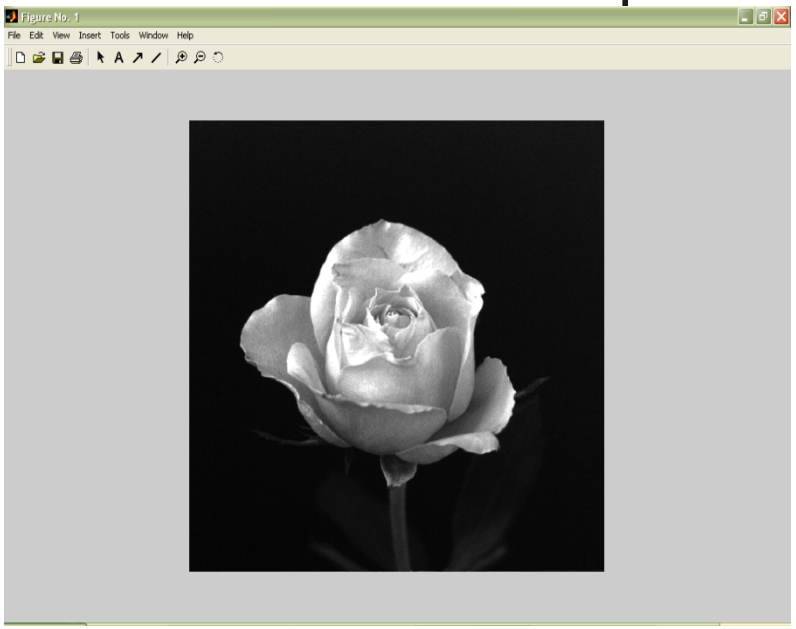
sf.tif



- The new 300-dpi image is identical, except that is 450 x 450 pixels are distributed over a 1.5 x 1.5-inch area



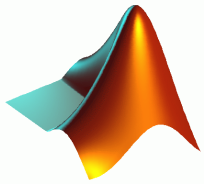
- The first is to use the File pull-down menu in the figure window and then choose Export.



The contents of a figure window can be exported to disk in two ways

- The second is using print command:

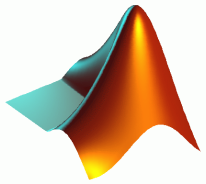
>> print –fno –d*fileformat* –r*resno* filename



- >> print –f1 –dtiff –r300 hi_res_rose sends the file hi_res_rose.tif to the current directory

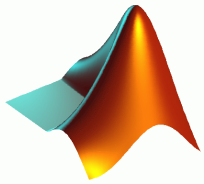
- If we simply type print at the prompt, MATLAB prints (to the default printer) the contents of the last figure window displayed.

- It is possible also to specify other options with print, such as a specific printing device



1.5 Data Classes

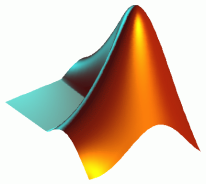
- In ımage processing usually we work with integer coordinates.
- However the pixel values are not restricted to be integers.
  - All numeric computations in MATLAB are done using double precision
  - Class unit8 also is used especially when reading data from storage devices...
- See Table 2.2 for varios data types.



1.5 Data Classes

Name	Description
`double`	Double-precision, floating-point numbers in the approximate range -10^{308} to 10^{308} (8 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).
`single`	Single-precision floating-point numbers with values in the approximate range -10^{38} to 10^{38} (4 bytes per element).
`char`	Characters (2 bytes per element).
`logical`	Values are 0 or 1 (1 byte per element).

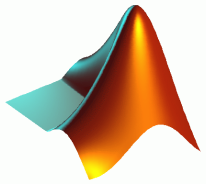
TABLE 2.2 Data classes. The first eight entries are referred to as *numeric* classes; the ninth entry is the *character* class, and the last entry is of class *logical*.



1.6 Image Types

The toolbox supports four types of images

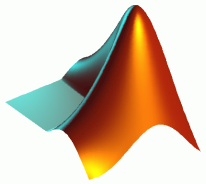
- Intensity images
- Binary images
- Indexed images
- RGB images



1.6.1 Intensity Images

An *intensity image* (*im*) is a data matrix whose values have been scaled to represent

- im's of classes unit8, or unit16 have integer values in the range [0,255] and [0,65535] respectively
- im of class double have values which are floating-point numbers in the range [0,1].



1.6.2 Binary Images

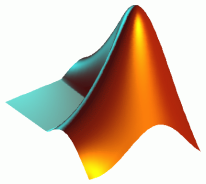
An *binary image* (*bm*) is a logical array of 0s and 1s. whose values have been scaled to represent

- A numeric array is converted to binary using funcyion *logical*.

  B=logical (A)

- An array is tested if it is logical we use the function *islogical*

  islogical (C)



1.6.3 Binary Images

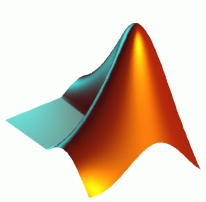
An *binary image* (*bm*) is a logical array of 0s and 1s. whose values have been scaled to represent

- A numeric array is converted to binary using funcyion *logical*.

  B=logical (A)

- An array is tested if it is logical we use the function *islogical*

  islogical (C)

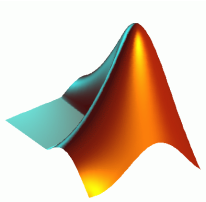


1.7. Coverting between data classes

The general syntax is

- B=data_class_name(A)

  where *data_class_name* represents one of the data classes such as *double, unit8, unit16, ...*etc.

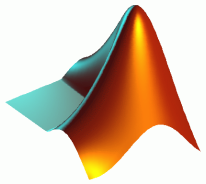


1.7.2 Coverting between image classes and Types

The toolbox provides spesific functions that perform the scaling necessary to convert between image classes and types. (See Table 2.3 for the list of the functions)

- Example: Function *im2unit8* detects the data class of the imput and performs all the necessary scaling for the toolbox to recognize the data as valid image data.

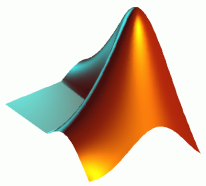
$$f = \begin{bmatrix} -0.5 & 0.5 \\ 0.75 & 1.5 \end{bmatrix}, \quad \texttt{>> g = im2unit8(f)} \Rightarrow g = \begin{bmatrix} 0 & 128 \\ 191 & 255 \end{bmatrix}$$



1.7.2 Coverting between image classes and Types

Name	Converts Input to:	Valid Input Image Data Classes
`im2uint8`	`uint8`	`logical`, `uint8`, `uint16`, and `double`
`im2uint16`	`uint16`	`logical`, `uint8`, `uint16`, and `double`
`mat2gray`	`double` (in range $[0, 1]$)	`double`
`im2double`	`double`	`logical`, `uint8`, `uint16`, and `double`
`im2bw`	`logical`	`uint8`, `uint16`, and `double`

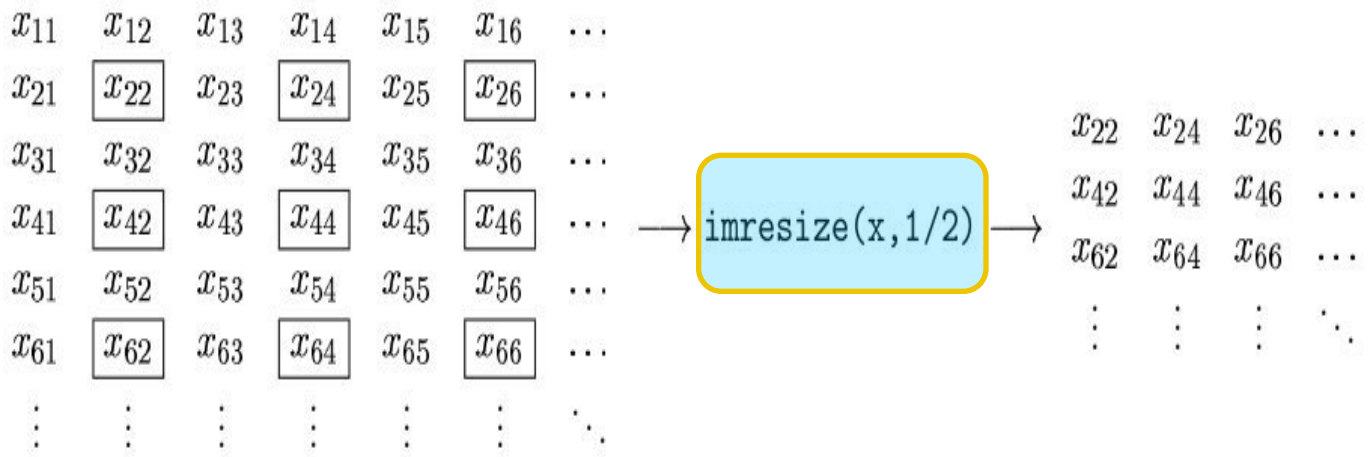
TABLE 2.3
Functions in IPT for converting between image classes and types. See Table 6.3 for conversions that apply specifically to color images.



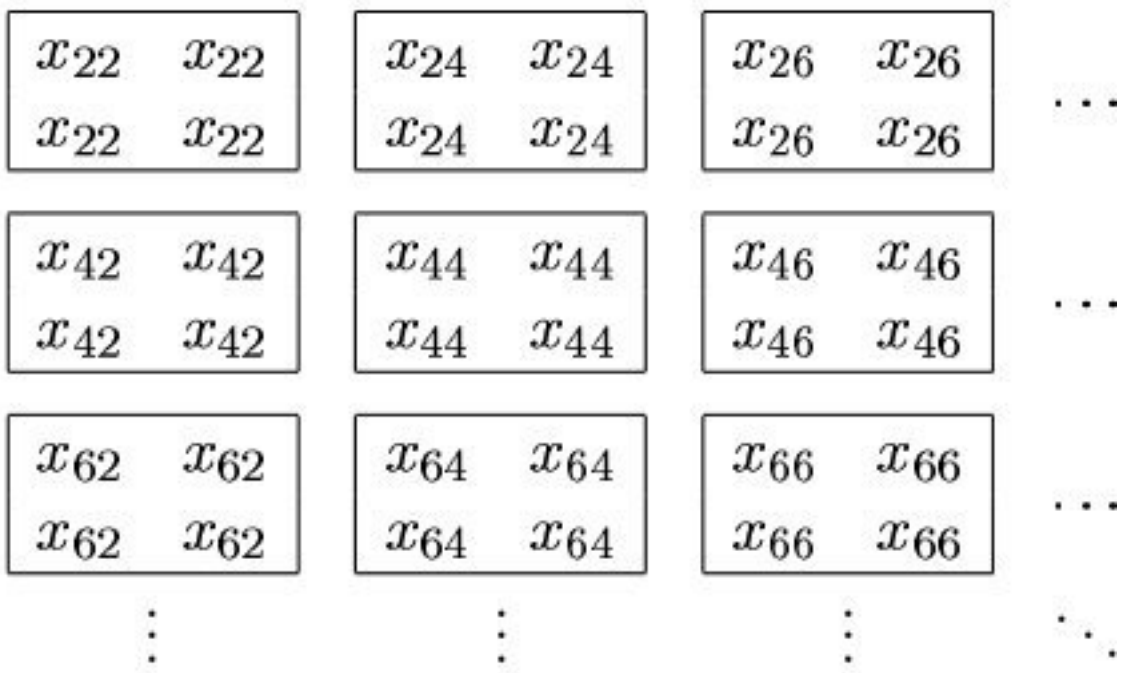
Spatial Resolution

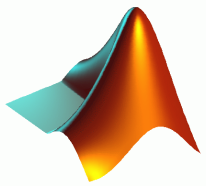
- **Spatial resolution** is the density of pixels over the image: the greater the spatial resolution, the more pixels are used to display the image.
- **Halve** the size of the image: It does this by taking out every other row and every other column, thus leaving only those matrix elements whose row and column indices are even.
- **Double** the size of the image: all the pixels are repeated to produce an image with the same size as the original, but with half the resolution in each direction.

Interpolation



Extrapolation

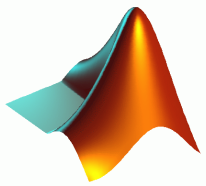




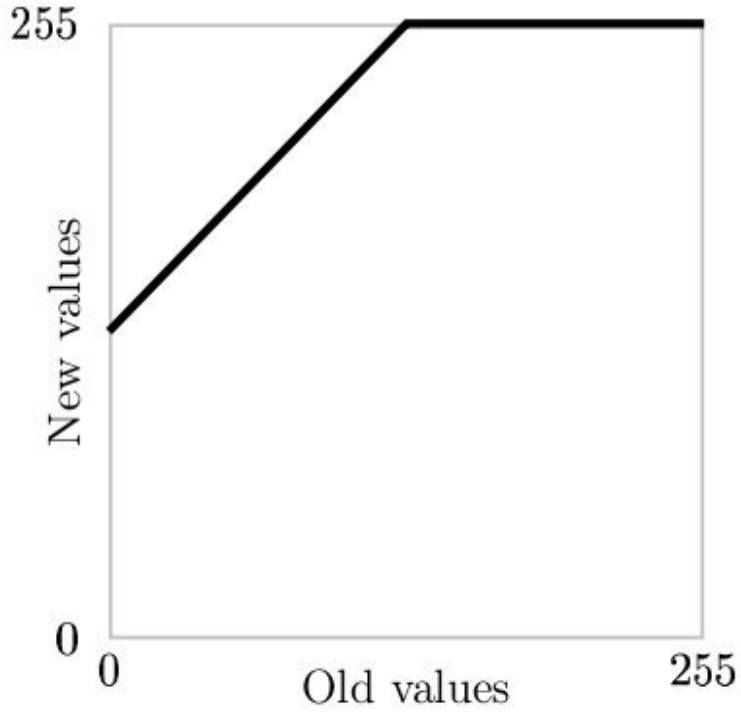
Point Processing: Arithmetic operations

These operations act by applying a simple function y=f(x) to each gray value in the image.

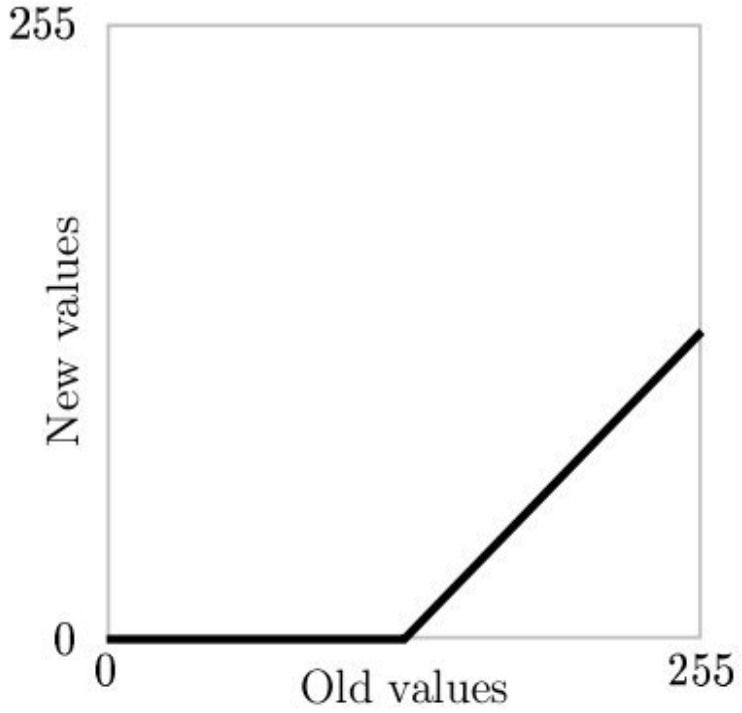
- Simple functions include **adding** or **subtract** a constant value to each pixel: y = x±C (`imadd`, `imsubtract`)
- **Multiplying** each pixel by a constant: y = C·x (`immultiply`, `imdivide`)
- **Complement**: For a grayscale image is its photographic negative.



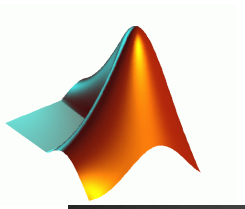
Arithmetic operations: Addition, subtraction



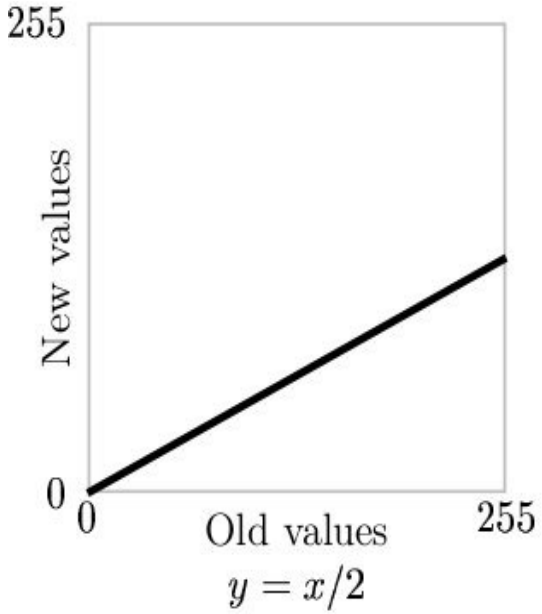
Adding 128 to each pixel

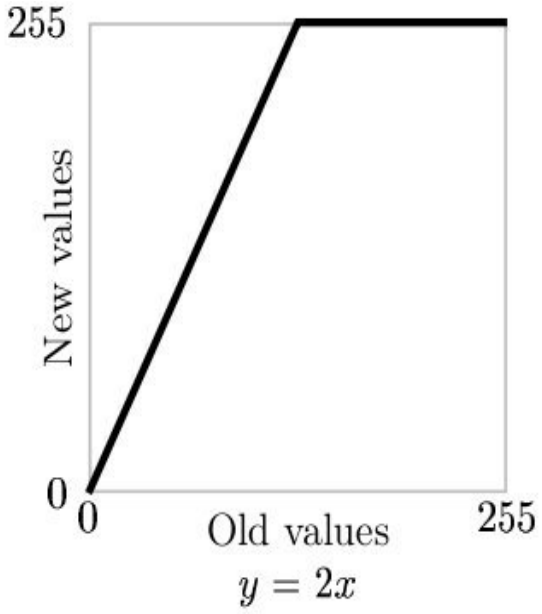


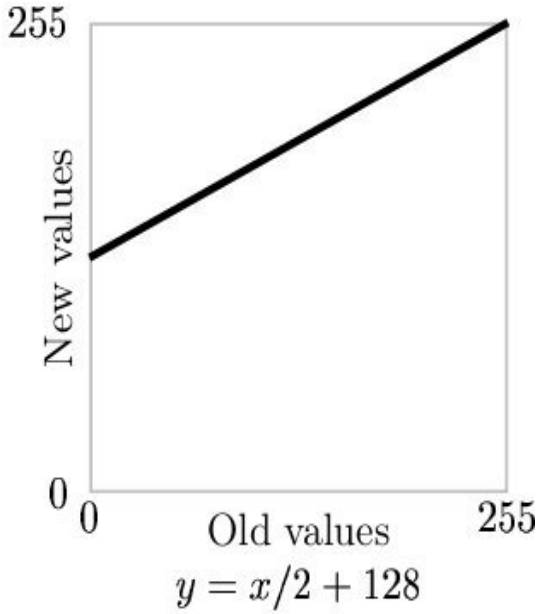
Subtracting 128 from each pixel



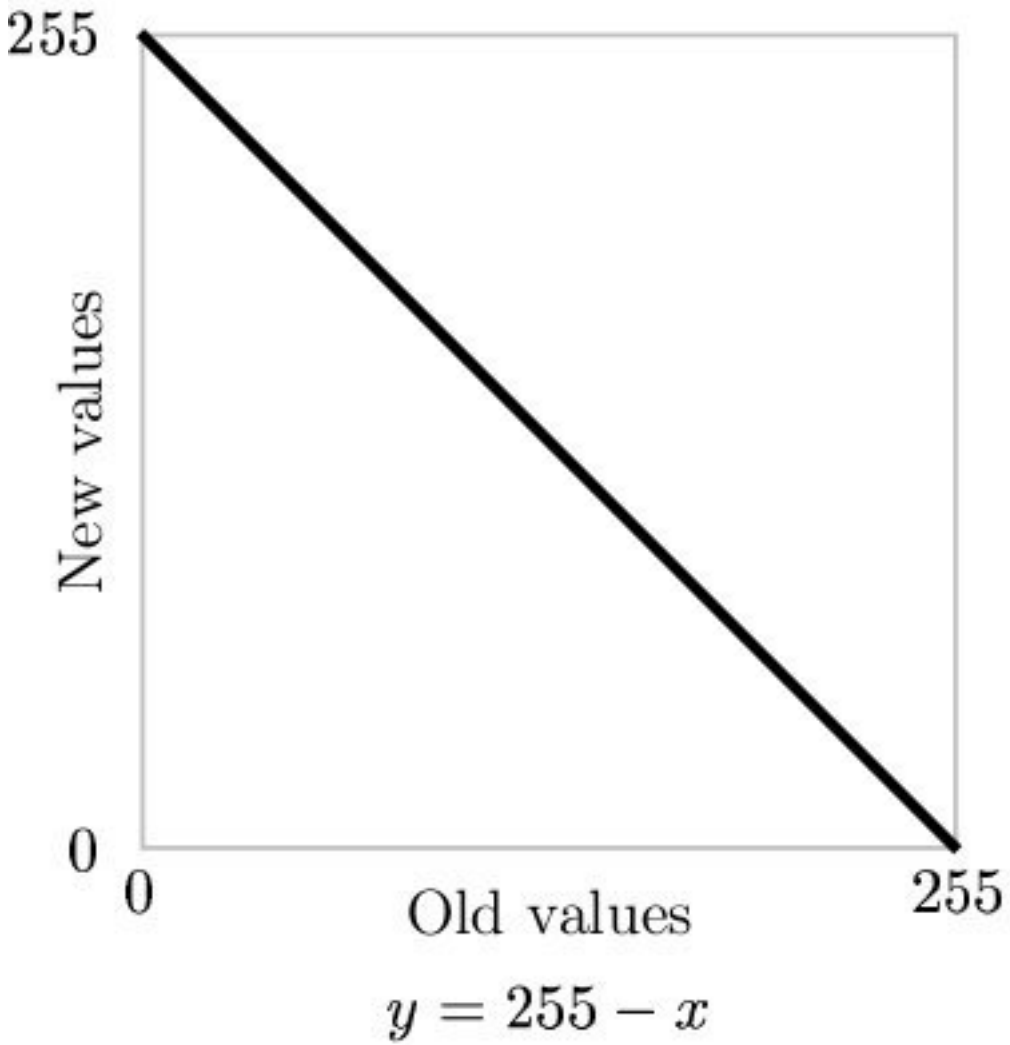
Arithmetic operations: Multiplication, division







Complement



Addition

Image: I

Image: I+50

Subtraction

Image: I

Image: I-80

Multiplication

Image: I

Image: I*3

Division

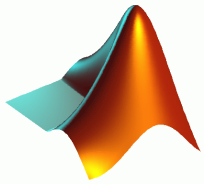
Image: I

Image: I/2

Complement

Image: I

Image: 255-I



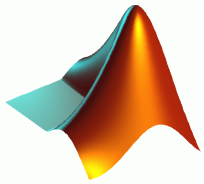
Exercises

imread, verify with *whos* and *size* the file that you have read. (Matlab folder has a lot of images, or you can pick them up at the VC folder)

- *mat2gray*, what is it used for?
- *imshow*, what does it do?, which is the difference with *imshow(h, [ ]);*, does it improve the result?
- *figure*, what does it do?, when must it be called?
- *imwrite*, what does it do?, what are its parameters?, what do they mean?
- *imfinfo*, what is it used for?, what does it mean "ColorType"?
- Look at the following code:

```
F = [ -0.5 0.5 ; 0.75 0.75]
G = im2uint8(F);
```

What is the contents of G?, why?



Exercises

- Look at the following code:

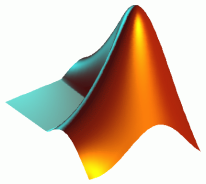
```
f = [ 1 2 ; 3 4]
g = mat2gray(f)
gb = im2bw(g,0.6)
Gbv = islogical(gb)
Gbd = im2double(gb)
```

What are the contents of each variable? What would have been the result in Gbd if gb type were uint8?

- Look at the following code:

```
>> q=imread('liftingbody.png');
>> imshow(q)
>> fp=q(end:-1:1,:); figure
>> imshow(fp)
>> fi=q(end:-1:1,end:-1:1); figure
>> imshow(fi)
>> fs =q(1:2:end; 1:2:end); figure
>> imshow(fs)
```

What do the variables fp, fi and fs contain?

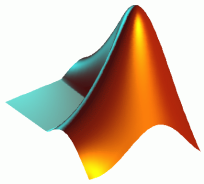


Exercises

Exercise 1: Write a function that calculates the compression ratio of any image in jpg format, that is, the quotient between the size of the compressed image divided by the natural size without compression.

Exercise 2: Using the image 'moon.tif', calculate its dimensions (in cmts) and create a new file 'm.tif' that maintains the aspect ratio x/y with a new width of 5 cmts. What will be the new resolution of the image?

Exercise 3: Write a function that rotates 90º to the right or to the left depending on a parameter, without using the function imrotate in Matlab.



Exercises

Exercise 4: Create a function that extracts a rectangular section of an image (crop) and leaves it into a new image, creating a new image getting only 1 out of 3 pixels. Realize this exercise in two ways:
1. In an iterative way using loops (for).
2. In a vectorial way without loops.

Input: an image of any type, the size of the new image (m: number of rows and n: number of columns), the upper left coordinate of the image (rx,cy)
Ouput: the new image of size mxn.
Proof: compare what is the function (iterative or vectorial) that takes longer in execution; use the functions tic and toc.