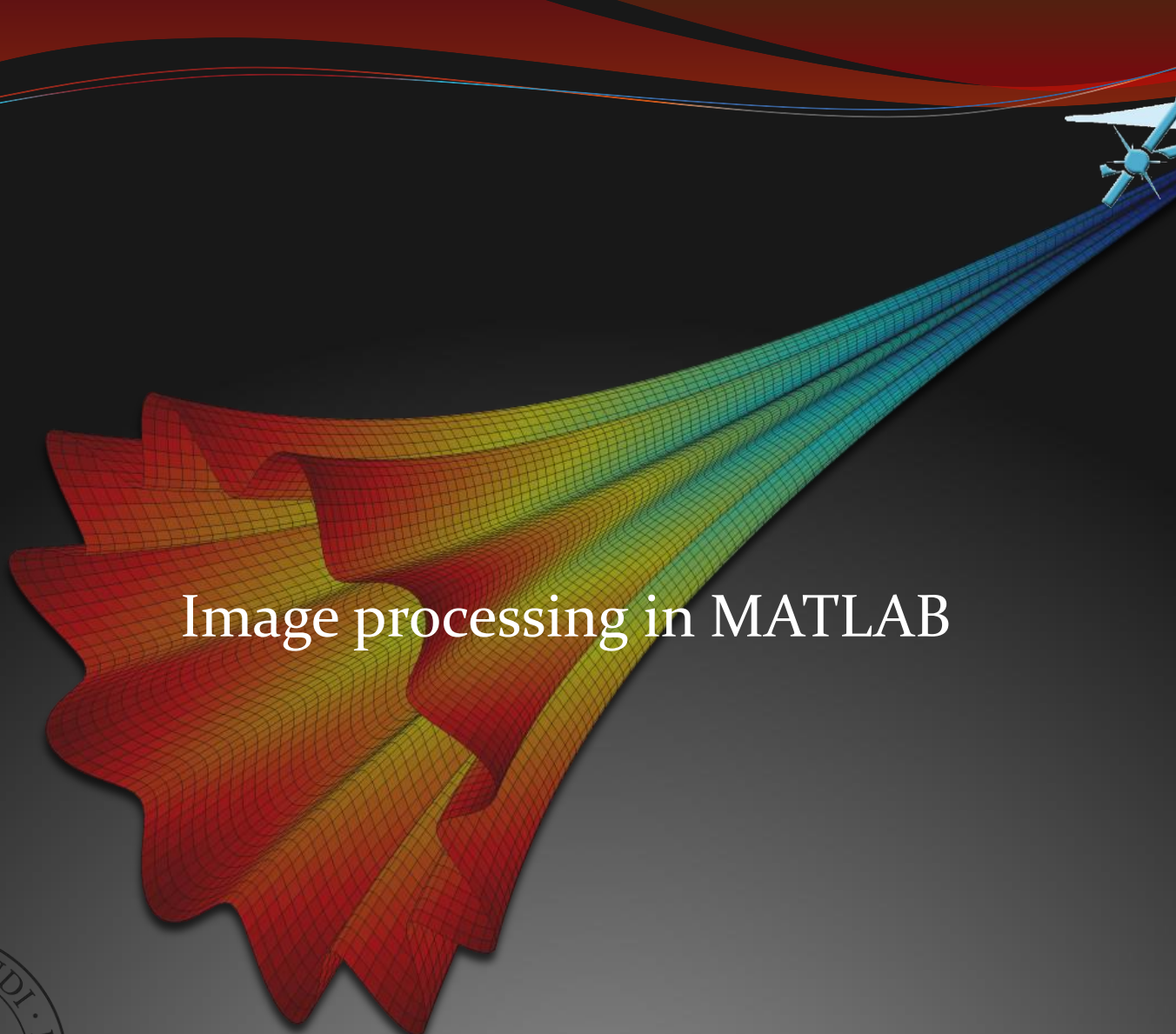


Image processing in MATLAB



Images in MATLAB

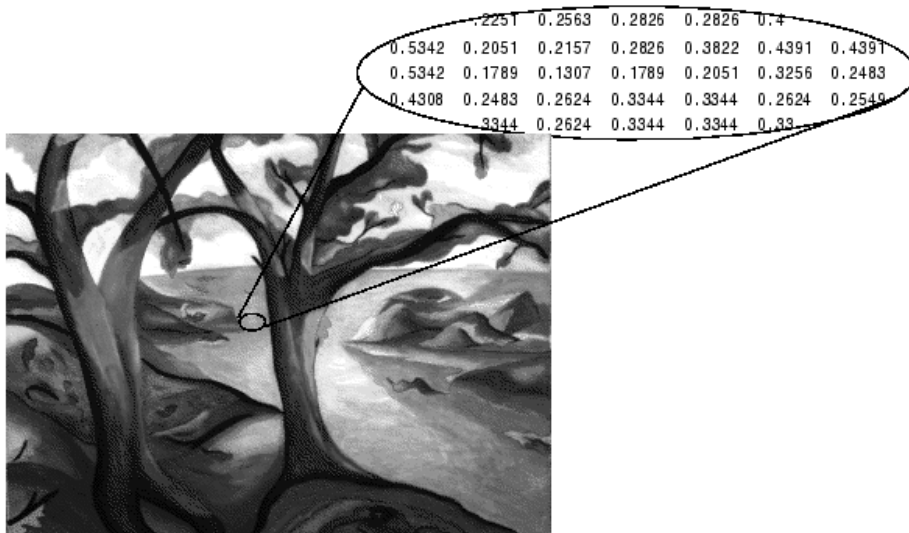
- MATLAB can import/export several image formats
 - BMP (Microsoft Windows Bitmap)
 - GIF (Graphics Interchange Files)
 - HDF (Hierarchical Data Format)
 - JPEG (Joint Photographic Experts Group)
 - PCX (Paintbrush)
 - PNG (Portable Network Graphics)
 - TIFF (Tagged Image File Format)
 - XWD (X Window Dump)
 - MATLAB can also load raw-data or other types of image data
 - DICOM data (dicomread...)
- Data types in MATLAB
 - Double (64-bit double-precision floating point)
 - Single (32-bit single-precision floating point)
 - Int32 (32-bit signed integer)
 - Int16 (16-bit signed integer)
 - Int8 (8-bit signed integer)
 - Uint32 (32-bit unsigned integer)
 - Uint16 (16-bit unsigned integer)
 - Uint8 (8-bit unsigned integer)

Images and Colors

- Images are represented as grids, or matrices, of picture elements (called “*pixels*”)
- In MATLAB an image is represented as an $m \times n$ matrix in which each element corresponds to a pixel
- All the operators in MATLAB defined on matrices can be used on images: +, -, *, /, ^, sqrt, sin, cos etc
- Each element that represents a particular pixel stores the color for that pixel

Images in MATLAB

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



Representing Color

- There are two basic ways that the color can be represented for a pixel:
 - *true color*, or *RGB*, in which the three color components are stored (red, green, and blue, in that order) → the matrix is therefore $m \times n \times 3$
 - index into a *colormap*: the value stored in each element of the $m \times n$ matrix is an integer which refers to a row in another matrix which is called a colormap
 - The colormap stores the red, green and blue components in three separate columns so its size is $p \times 3$ where p is the number of colors

Colormaps

- every element in the $m \times n$ image matrix *mat* is an integer in the range from 1 to p which refers to a row in another matrix which is called a colormap
- The colormap stores the red, green and blue components of different colors in three separate columns so its size is $p \times 3$ where p is the number of colors
- If the type of the elements is **double**, the numbers range from 0 (which represents the absence of that color) to 1 (which is the brightest possible hue)
- the **image** function displays the image matrix using the current colormap:

```
image(mat)
```
- the **colormap** function can be used two ways: if no argument is passed, it returns the current colormap; if a $p \times 3$ matrix is passed, it sets that matrix to be the current colormap

Built-in colormaps

- There are several built-in colormaps, e.g. **parula** (which is the default), **jet**, **autumn**, **pink**, etc.
- All of these all have 64 colors so the size of the colormap is 64 x 3
- For example, the first four colors from **jet** are shades of blue:

```
>> colormap(jet)
```

```
>> colormap
```

```
ans =
```

```
0    0    0.5625
```

```
0    0    0.6250
```

```
0    0    0.6875
```

```
0    0    0.7500
```

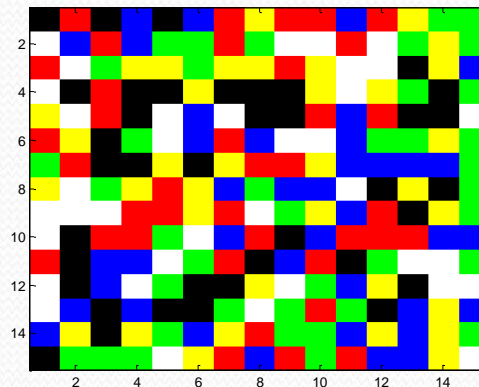
```
etc.
```

Example user-defined colormap

Create a colormap with 6 colors, and use a random matrix to display them

- $[1\ 0\ 0]$ is red
- $[0\ 1\ 0]$ is green
- $[0\ 0\ 1]$ is blue
- $[1\ 1\ 0]$ is yellow
- $[0\ 0\ 0]$ is black
- $[1\ 1\ 1]$ is white

```
>> mycolors = [1 0 0; 0 1 0; 0 0 1; ...  
1 1 0; 0 0 0; 1 1 1];  
>> colormap(mycolors)  
>> mat = randi([1 6],15,15);  
>> image(mat)
```



Example indexed image co

- `>> [X,map] = imread('trees.tif');`

```
>> size(X)
```

```
ans =
```

```
258 350
```

```
>> size(map)
```

```
ans =
```

```
256 3
```

```
>> image(X)
```

```
>> colormap summer
```

```
>> image(X)
```

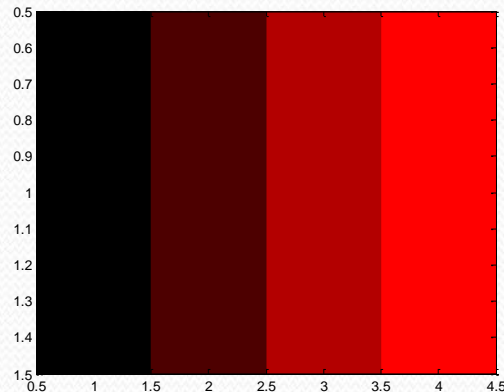
```
>> imshow(X,map)
```



colormap shades

- The numbers in the colormap do not have to be integers; real numbers represent different shades
- The following shows four shades of red, from no red (black) to the brightest possible red:

```
>> colors = [0 0 0; 0.3 0 0; 0.7 0 0; 1 0 0];  
>> colormap(colors)  
>> vec = 1:4;  
>> image(vec)
```



True Color Matrices

- True color matrices are three-dimensional matrices with size $m \times n \times 3$
 - The first two indices are the coordinates of the pixel
 - The third index is the color component; $(:,:,1)$ is the red, $(:,:,2)$ is the green, and $(:,:,3)$ is the blue component
- If every element in the matrix is of the type **uint8**, which is an unsigned integer type using 8 bits; the range of values is 0 to 255
- Therefore the minimum value, 0, represents the darkest hue available so all 0's results in a black pixel
- The maximum value, 255, represents the brightest hue
- For example, in the following the pixel in location 1,1 will be red:

```
>> mat(1,1,1) = 255;  
>> mat(1,1,2) = 0;  
>> mat(1,1,3) = 0;
```
- The image function shows the image:

```
>> mat = uint8(mat);  
>> image(mat)
```

Using imread

- The function **imread** can read in an image file, for example a JPEG (.jpg) file
- The function reads color images into a 3D matrix

```
>> myimage1 = imread('xyz.JPG');
```
- The image function displays this:

```
>> image(myimage1)
```
- Functions now in MATLAB (were in Image Processing Toolbox):
 - **imshow** displays an image
 - **rgb2gray** converts from RGB to gray scale
 - **im2double** converts an image matrix to **double**

Image import and export example

- Read and write images in Matlab

```
>> I=imread('ngc6543a.jpg');
```

```
>> imshow(I)
```

```
>> size(I)
```

```
ans = 479 600 3      (RGB image)
```

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

```
>> imshow(Igrey)
```

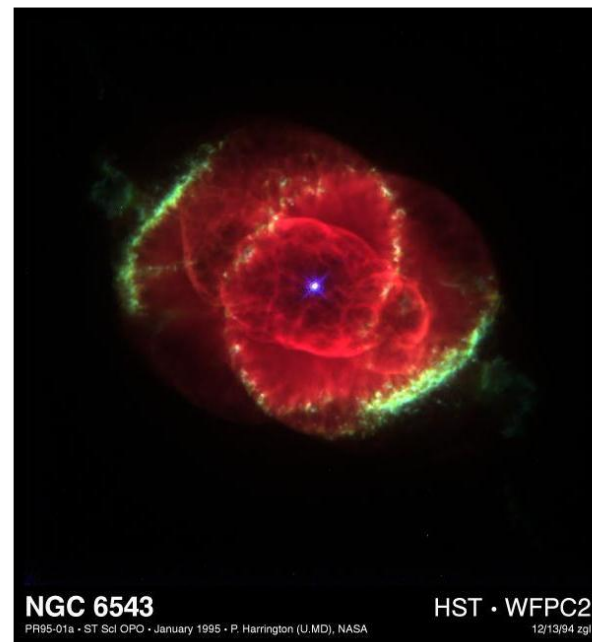
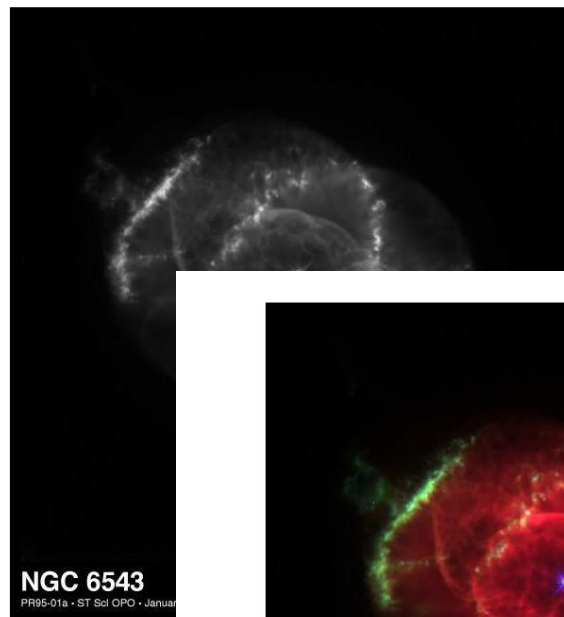
```
>> imwrite(Igrey, 'cell_gray.tif', 'tiff')
```

- Alternatives to imshow

```
>> imagesc(I)
```

```
>> imtool(I)
```

```
>> image(I)
```



Images and Matrices

- How to build a matrix (or image)?

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

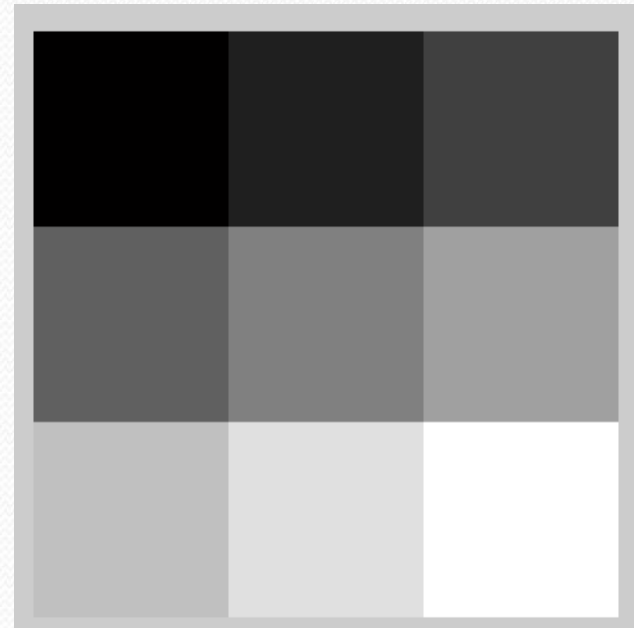
```
A =  1   2   3  
     4   5   6  
     7   8   9
```

```
>> B = zeros(3,3)
```

```
B =  0   0   0  
     0   0   0  
     0   0   0
```

```
>> C = ones(3,3)
```

```
C =  1   1   1  
     1   1   1  
     1   1   1
```



```
>>imshow(A)    (imshow(A,[]) to get automatic pixel range)
```

Images and Matrices

- Accessing image elements (row, column)

```
>> A(2,1)
```

```
ans = 4
```

- `:` can be used to extract a whole column or row

```
>> A(:,2)
```

```
ans =
```

```
2
```

```
5
```

```
8
```

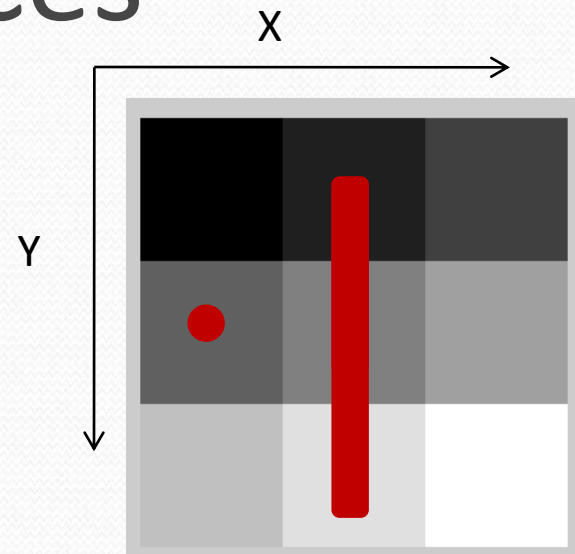
- or a part of a column or row

```
>> A(1:2,2)
```

```
ans =
```

```
2
```

```
5
```



```
A =  
 1  2  3  
 4  5  6  
 7  8  9
```

Image Arithmetic

- Arithmetic operations such as addition, subtraction, multiplication and division can be applied to images in MATLAB

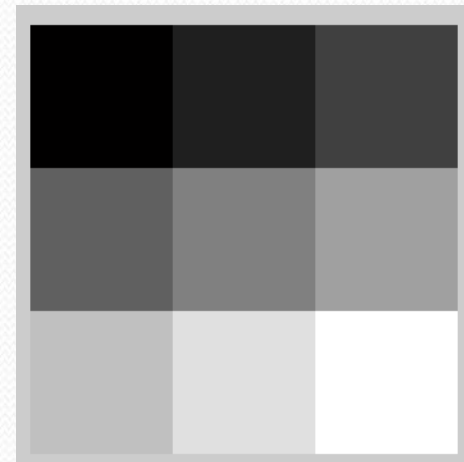
- `+`, `-`, `*`, `/` performs matrix operations

```
>> A+A
```

```
ans =  2  4  6
       8 10 12
      14 16 18
```

```
>> A*A
```

```
ans =  30  36  42
       66  81  96
      102 126 150
```



- To perform an elementwise operation use `.` (`.*`, `./`, `.*`, `.^` etc)

```
>> A.*A
```

```
ans =  1  4  9
      16 25 36
      49 64 81
```


Image Arithmetic

- equal (==) , less than and greater than (< and >), not equal (~=) and not (~)
- find('condition') - Returns indexes of A's elements that satisfies the condition.

```
>> [row col]=find(A==7)
```

```
row = 3
```

```
col = 1
```

```
>> [row col]=find(A>7)
```

```
row = 3
```

```
3
```

```
col = 2
```

```
3
```

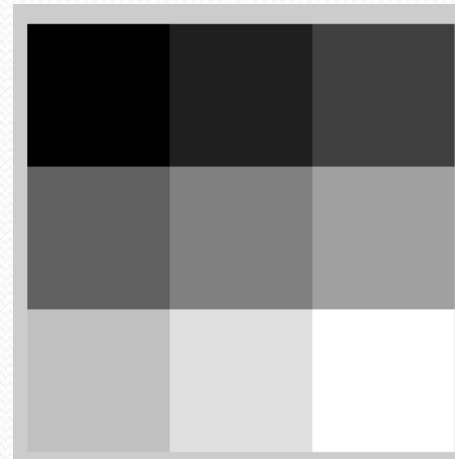
```
>> Indx=find(A<5)
```

```
Indx = 1
```

```
2
```

```
4
```

```
7
```

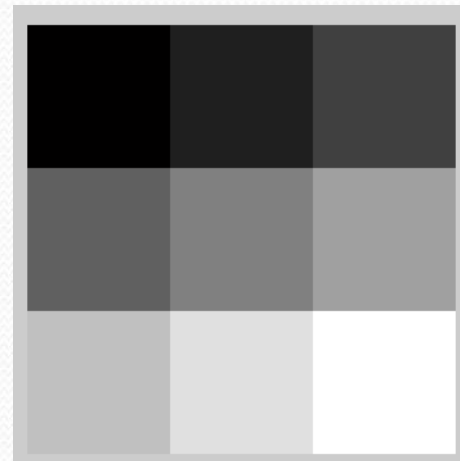


A =

1	2	3
4	5	6
7	8	9

Flow control

- Flow control in MATLAB
 - **if**, **else** and **elseif** statements
(row=1,2,3 col=1,2,3)
 - if** row==col
 A(row, col)=1;
 - elseif** abs(row-col)==1
 A(row, col)=2;
 - else**
 A(row, col)=0;
 - end**



```
A =  
 1  2  3  
 4  5  6  
 7  8  9
```

Image histogram

- The histogram plots the number of pixels in the image (vertical axis) with a particular brightness value (horizontal axis).
- Algorithms in the digital editor allow the user to visually adjust the brightness value of each pixel and to dynamically display the results as adjustments are made. Improvements in picture brightness and contrast can thus be obtained.
- In the field of [computer vision](#), image histograms can be useful tools for [thresholding](#). Because the information contained in the graph is a representation of pixel distribution as a function of tonal variation, image histograms can be analyzed for peaks and/or valleys. This threshold value can then be used for [edge detection](#), [image segmentation](#), and [co-occurrence matrices](#). (Wikipedia)

Image histogram

- `imhist(I)` calculates the histogram for the intensity image `I` and displays a plot of the histogram. The number of bins in the histogram is determined by the image type.
- `imhist(I,n)` calculates the histogram, where `I` specifies the number of bins used in the histogram. `n` also specifies the length of the colorbar displayed at the bottom of the histogram plot.
- `imhist(X,map)` displays a histogram for the indexed image `X`. This histogram shows the distribution of pixel values above a colorbar of the colormap `map`.
- Input arguments:
 - `I`-input intensity image
 - `N`- number of bins (256 for grayscale images)(default)
 - `X` input indexed image (with colormap)
 - `map` – Colormap associated with indexed image specified as a `p-by-3` array.

Image histogram example

```
>>I = imread('pout.tif');
>>imhist(I)
>> imhist(I,40)
```

```
>>[X,map] = imread('trees.tif');
>>imshow(X,map)
>>Imhist(X,map)
```

