Matlab

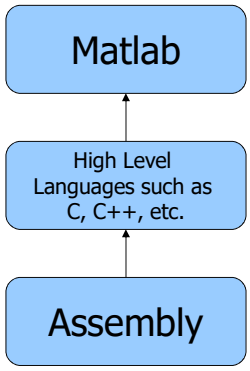
Matrices – Functions - Image and Video Proceessing – Interfaces (GUI)

Masters
MQL – M2I
Faculty of Science - Tetouan

Given by Youssef Zaz

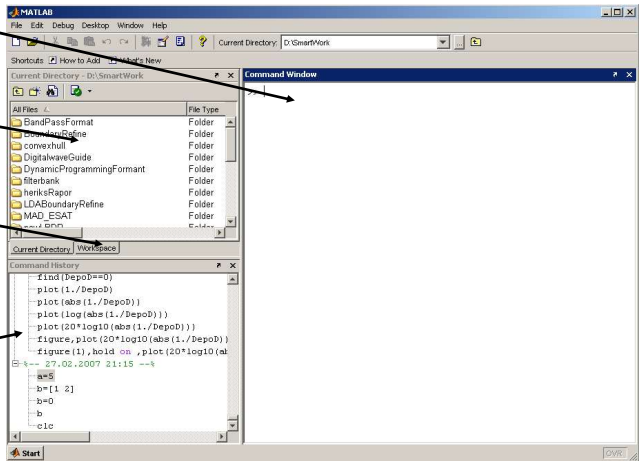
What is Matlab?

- Matlab is basically a high level language which has many specialized toolboxes for making things easier for us
- How high?



Matlab Screen

- Command Window
  - type commands
- Current Directory
  - View folders and m-files
- Workspace
  - View program variables
  - Double click on a variable to see it in the Array Editor
- Command History
  - view past commands
  - save a whole session using diary



Notes

- "%" is the neglect sign for Matlab (equaivalent of "//" in C). Anything after it on the same line is neglected by Matlab compiler.
- Sometimes slowing down the execution is done deliberately for observation purposes. You can use the command "pause" for this purpose

```
pause %wait until any key
pause(3) %wait 3 seconds
```

Useful Commands

- The two commands used most by Matlab users are

```
>>help functionname
```

```
>>lookfor keyword
```

Variables

- No need for types. i.e.,

```
int a;
double b;
float c;
```

- All variables are created with double precision unless specified and they are matrices.

```
Example:
>>x=5;
>>x1=2;
```

- After these statements, the variables are 1x1 matrices with double precision

How do we assign a value to a variable?


```
>>> v1=3
v1 =
    3
>>> i1=4
i1 =
    4
>>> R=v1/i1
R =
   0.7500
>>>
```

```
>>> whos
  Name      Size            Bytes  Class
  R         1x1                 8  double array
  i1        1x1                 8  double array
  v1        1x1                 8  double array
Grand total is 3 elements using 24 bytes
>>> who
Your variables are:
R       i1      v1
>>>
```

How do we assign values to vectors?


```
>>> A = [1 2 3 4 5]
A =
  1   2   3   4   5
>>>
```

A row vector – values are separated by spaces

```
>>> B = [10;12;14;16;18]
B =
     10
     12
     14
     16
     18
>>>
```

A column vector – values are separated by semi–colon (;)

How do we assign values to vectors?


```
>>> A = [1 2 3 4 5]
A =
  1  2  3  4  5
>>>
```

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$

```
>>> B = [10;12;14;16;18]
B =
     10
     12
     14
     16
     18
>>>
```

$$B = \begin{bmatrix} 10 \\ 12 \\ 14 \\ 16 \\ 18 \end{bmatrix}$$

How do we assign values to vectors?

- vector
```
A = [1 2 5 1]
A =
     1   2   5   1
```
- matrix
```
x = [1 2 3; 5 1 4; 3 2 -1]
x =
     1     2     3
     5     1     4
     3     2    -1
```
- transpose
```
y = A'
y =
     1
     2
     5
     1
```

Long Array, Matrix

- `t =1:10`

```
t =
     1   2   3   4   5   6   7   8   9   10
```

- `k =2:-0.5:-1`

```
k =
     2  1.5  1  0.5  0  -0.5  -1
```

- `B = [1:4; 5:8]`

```
x =
     1     2     3     4
     5     6     7     8
```

Generating Vectors from functions

- zeros(M,N) MxN matrix of zeros

```
A=zeros(3,4)

A =

     0     0     0     0
     0     0     0     0
     0     0     0     0
```

Generating Vectors from functions

- ones(M,N) MxN matrix of ones

```
B=ones(3,4)

B =

     1     1     1     1
     1     1     1     1
     1     1     1     1
```

Generating Vectors from functions


```
C=rand(3,4)

C =

    0.8147    0.9134    0.2785    0.9649
    0.9058    0.6324    0.5469    0.1576
    0.1270    0.0975    0.9575    0.9706
```

Matrix Index

- The matrix indices begin from 1 (not 0 (as in C))
- The matrix indices must be positive integer

Given:

```
A =

     3     5     3
     6     8     2
     2     7     3
```

```
>> A(6)

ans =

     7
```

```
>> A(3,2)

ans =

     7
```

```
>> A(2,:)

ans =

     6     8     2
```

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

ans =

     5
     8
```

A(-2), A(0)

Error: ??? Subscript indices must either be real positive integers or logicals.

A(4,2)
Error: ??? Index exceeds matrix dimensions.

Matrix Index

Vector and matrix indexing

```
>> C=1 2 3
     4 5 6
     7 8 9
```

```
>>b=1
     2
     3
```

```
>>a=1 2 3
```

- **To index an element from a vector**

  a(2)
  >> 2

  b(end)
  >> 3

- **To index an element from a matrix**

  C(2,3)
  >> 6

  C(1,end)
  >> 7

- **The colon ":" operator will yield a specified range**

  C(3,:)
  >> 7 8 9

  C(:,2)
  >>2
  5
  8

Concatenation of Matrices

Vector and matrix indexing

- `x = [1 2], y = [4 5], z=[ 0 0]`

```
A = [ x y]

            1   2   4   5

B = [x ; y]

      1 2
      4 5
```

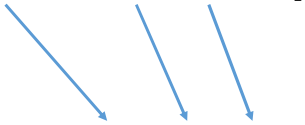
```
C = [x y ;z]
Error:
??? Error using ==> vertcat CAT arguments dimensions are not consistent.
```

Submatrices

A matrix can be indexed using another matrix, to produce a subset of its elements:

a = [100 200 300 400 500 600 700] b = [3 5 6]

c = a(b):



300 500 600

Operators (arithmetic)

+ addition
- subtraction
* multiplication
/ division
^ power
‘ complex conjugate transpose

Matrices Operations

Given A and B:

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

A =

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

```
>> B = [3 5 2; 5 2 8; 3 6 9]

B =

     3     5     2
     5     2     8
     3     6     9
```

Addition

```
>> X = A + B

X =

     4     7     5
     9     7    14
    10    14    18
```

Subtraction

```
>> Y = A - B

Y =

    -2    -3     1
    -1     3    -2
     4     2     0
```

Product

```
>> Z = A * B

Z =

    22    27    45
    55    66   102
    88   105   159
```

Transpose

```
>> T = A'

T =

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

Operators (Element by Element)

.* element-by-element multiplication

./ element-by-element division

.^ element-by-element power

Matrix functions – perform operations on matrices


```
>>> x=rand(4,4)
x =
   0.9501   0.8913   0.8214   0.9218
   0.2311   0.7621   0.4447   0.7382
   0.6068   0.4565   0.6154   0.1763
   0.4860   0.0185   0.7919   0.4057
>>> xinv=inv(x)
xinv =
   2.2631  -2.3495  -0.4696  -0.6631
  -0.7620   1.2122   1.7041  -1.2146
  -2.0408   1.4228   1.5538   1.3730
   1.3075  -0.0183  -2.5483   0.6344
```

```
>>> x*xinv

ans =

   1.0000   0.0000   0.0000   0.0000
        0   1.0000        0   0.0000
   0.0000        0   1.0000   0.0000
        0        0   0.0000   1.0000
```

Performing operations to every entry in a matrix


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

Add and subtract

```
>>> A+3
ans =
     4     5     6
     7     8     9
    10    11    12

>>> A-2
ans =
    -1     0     1
     2     3     4
     5     6     7
```

Performing operations to every entry in a matrix


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

Multiply and divide

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

>>> A/3
ans =
    0.3333    0.6667    1.0000
    1.3333    1.6667    2.0000
    2.3333    2.6667    3.0000
```

Performing operations between matrices


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

A^2 = A * A

Power

To square every element in A, use the element–wise operator **.^**

```
>>> A.^2
ans =
     1     4     9
    16    25    36
    49    64    81
```

```
>>> A^2
ans =
    30    36    42
    66    81    96
   102   126   150
```

Performing operations between matrices


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

```
>>> B=[1 1 1;2 2 2;3 3 3]
B =
     1     1     1
     2     2     2
     3     3     3
```

A*B

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\\7 & 8 & 9\end{bmatrix}\begin{bmatrix}1 & 1 & 1\\2 & 2 & 2\\3 & 3 & 3\end{bmatrix} = \begin{bmatrix}14 & 14 & 14\\32 & 32 & 32\\50 & 50 & 50\end{bmatrix}$$

A.*B

$$\begin{bmatrix}1x1 & 2x1 & 3x1\\4x2 & 5x2 & 6x2\\7x3 & 8x3 & 9x3\end{bmatrix} = \begin{bmatrix}1 & 2 & 3\\8 & 10 & 12\\21 & 24 & 27\end{bmatrix}$$

Performing operations between matrices

A/B ⇨ **? (matrices singular)**

A./B ⇨

$$\begin{bmatrix} 1/1 & 2/1 & 3/1 \\ 4/2 & 5/2 & 6/2 \\ 7/3 & 8/3 & 9/3 \end{bmatrix} = \begin{bmatrix} 1.0000 & 2.0000 & 3.0000 \\ 2.0000 & 2.5000 & 3.0000 \\ 2.3333 & 2.6667 & 3.0000 \end{bmatrix}$$

Performing operations between matrices

A^B ⇨ ??? Error using ==> ^
At least one operand must be scalar

A.^B ⇨

$$\begin{bmatrix} 1^1 & 2^1 & 3^1 \\ 4^2 & 5^2 & 6^2 \\ 7^3 & 8^3 & 9^3 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 16 & 25 & 36 \\ 343 & 512 & 729 \end{bmatrix}$$

Built in functions (commands)

Scalar functions – used for scalars and operate element-wise when applied to a matrix or vector

e.g. sin cos tan atan asin log
abs angle sqrt round floor

e.g. `>>help sin`

Built in functions (commands)


```
>> a=linspace(0,(2*pi),10)
a =
  Columns 1 through 7
         0    0.6981    1.3963    2.0944    2.7925    3.4907    4.1888
  Columns 8 through 10
    4.8869    5.5851    6.2832
>>> b=sin(a)
b =
  Columns 1 through 7
         0    0.6428    0.9848    0.8660    0.3420   -0.3420   -0.8660
  Columns 8 through 10
   -0.9848   -0.6428    0.0000
```

Built in functions (commands)

e.g. max min mean prod sum length

```
>>> a=linspace(0,(2*pi),10);
>>> b=sin(a);
```

```
>>> max(b)
ans =
    0.9848
>>> max(a)
ans =
    6.2832
>>> length(a)
ans =
    10
>>>
```

Matrix functions – perform operations on matrices


```
>> help elmat
```

```
>> help matfun
```

e.g. eye size inv det eig

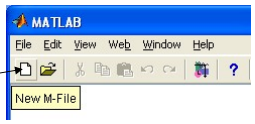
At any time you can use the command `help` to get help

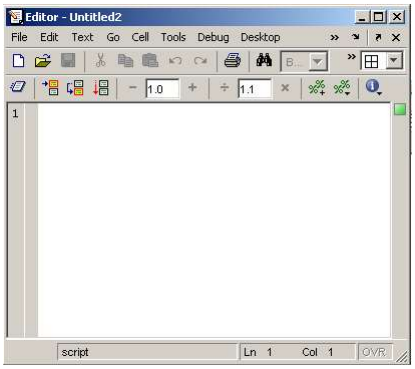
Data classes

Class	Range	# bytes/element
double	$[-10^{308},10^{308}]$	8
uint8	[0,255]	1
uint16	[0,65535]	2
Uint32	[0,4294967295]	4
Int8	[-128,127]	1
Int16	[-32768,32767]	2
int32	[-2147483648, 2147483647]	4
Single	$[-10^{38},10^{38}]$	4
char	characters	2
logical	[0,1]	1

Use of M-File

Click to create a new M-File





- Extension ".m"
- A text file containing script or function or program to run

Writing User Defined Functions

- Functions are m-files which can be executed by specifying some inputs and supply some desired outputs.
- The code telling the Matlab that an m-file is actually a function is

```
function out1=functionname(in1)
function out1=functionname(in1,in2,in3)
function [out1,out2]=functionname(in1,in2)
```

- You should write this command at the beginning of the m-file and you should save the m-file with a file name same as the function name

Writing User Defined Functions

- Another function which takes an input array and returns the sum and product of its elements as outputs

```
function [a,b]=somprod(tab)
a=sum(tab);
b=prod(tab);
```

- The function sumprod(.) can be called from command window or an m-file as

```
[x,y]=somprod(c)
```

Writing User Defined Functions

Task: *The area, A, of a triangle with sides of length a, b and c is given by*

$$A = \sqrt{s(s-a)(s-b)(s-c)},$$

where $s = (a+b+c)/2$.

File area.m:

```
function [A] = area(a,b,c)
s = (a+b+c)/2;
A = sqrt(s*(s-a)*(s-b)*(s-c));
```

Usage example: To evaluate the area of a triangle with side of length 10, 15, 20:

```
>> Area = area(10,15,20)
Area =
72.6184
```

Notes:

- "%" is the neglect sign for Matlab (equaivalent of "//" in C). Anything after it on the same line is neglected by Matlab compiler.
- Sometimes slowing down the execution is done deliberately for observation purposes. You can use the command "pause" for this purpose

```
pause %wait until any key
pause(3) %wait 3 seconds
```

Solve 2nd degree equations in R

```
% trinome.m
disp('Solve ax²+bx+c=0');
choix='Y';
while (choix~='N' & choix~='n'),
a=input('a=? ');
b=input('b=? ');
c=input('c=? ');
delta=b*b-4*a*c;
if (delta<0),
disp('No solution');
end
if (delta==0),
disp(1 solution:');
racine=-b/(2*a);
disp(racine);
end
if (delta>0),
disp('2 solutions');
racine1=(-b+sqrt(delta))/(2*a);
racine2=(-b-sqrt(delta))/(2*a);
disp(racine1);
disp(racine2);
end
choix=input('Another equation (Y/N)? ','s');
end
```

```
trinome
Solve ax²+bx+c=0
a=? 1
b=? 2.36
c=? -4.5
2 solutions :
1.2474
-3.6074
Another equation (Y/N)? n
```

solve 2nd degree equation in C

```
% trinome1.m
disp('Solve ax²+bx+c=0');
p(1)=input('a=? ');
p(2)=input('b=? ');
p(3)=input('c=? ');
disp(' Solutions :');
disp(roots(p));
```

```
>> trinome1
Solve ax²+bx+c=0
a=? 2+i
b=? -5
c=? 1.5-5i
Solutions :
0.8329 + 0.6005i
-0.2079 - 0.6005i
```

Factorial calculation : n!

```
% facto.m
disp('Factorial calculation') ;
n=input('n = ? ') ;
fact=1;
for i=1:n,
        fact=fact*i;
end
disp([num2str(n) '!=']);
format long e;
disp(fact);
```

```
>> facto
Factorial calculation
n = ? 134
134!=
1.992942746161518e+228
```

Pascal's triangle

A simple example:

```
a = 1
while length(a) < 10
a = [0 a] + [a 0]
end
```

which prints out Pascal's triangle:

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
```

(with "a=" before each line).

Integration example

Find the integral: $\int_0^{10}\left(\frac{1}{2}\sqrt{x}+x\sin(x)\right)dx$

example with *trapz* function:

```
>> x = 0:0.5:10; y = 0.5 * sqrt(x) + x .* sin(x);
>> integral1 = trapz(x,y)
integral1 =
18.1655
```

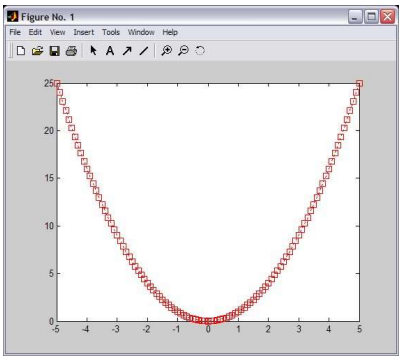
Graphics - 2D Plots

```
plot(xdata, ydata, 'marker_style');
```

For example:

```
>> x=-5:0.1:5;
>> sqr=x.^2;
>> pl1=plot(x, sqr, 'r:s');
```

Gives:



Graphics - 2D Plots

Basic Task: Plot the function sin(x) between $0 \leq x \leq 4\pi$

- Create an x-array of 100 samples between 0 and 4π.

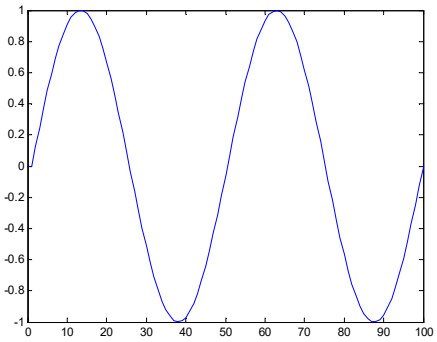
```
>>x=linspace(0,4*pi,100);
```

- Calculate sin(.) of the x-array

```
>>y=sin(x);
```

- Plot the y-array

```
>>plot(y)
```



Graphics - 2D Plots

Display Facilities

- title(.)

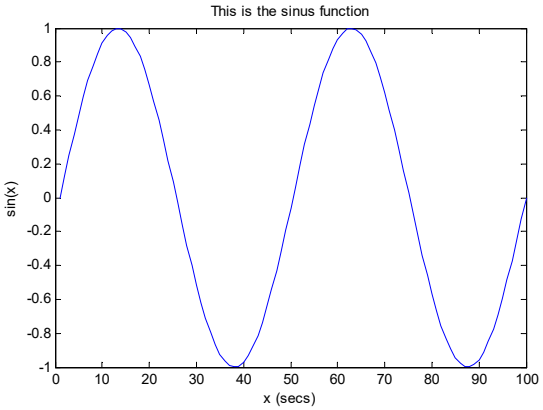
```
>>title('This is the sinus function')
```

- xlabel(.)

```
>>xlabel('x (secs)')
```

- ylabel(.)

```
>>ylabel('sin(x)')
```



Graphics - 2D Plots

Plot the function $e^{-x/3}\sin(x)$ between $0 \leq x \leq 4\pi$

- Create an x-array of 100 samples between 0 and 4π.

```
>>x=linspace(0,4*pi,100);
```

- Calculate sin(.) of the x-array

```
>>y=sin(x);
```

- Calculate $e^{-x/3}$ of the x-array

```
>>y1=exp(-x/3);
```

- Multiply the arrays y and y1

```
>>y2=y*y1;
```

Graphics - 2D Plots

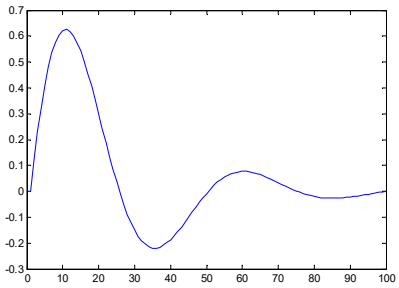
Plot the function $e^{-x/3}\sin(x)$ between $0 \leq x \leq 4\pi$

- Multiply the arrays y and y1 correctly

```
>>y2=y.*y1;
```

- Plot the y2-array

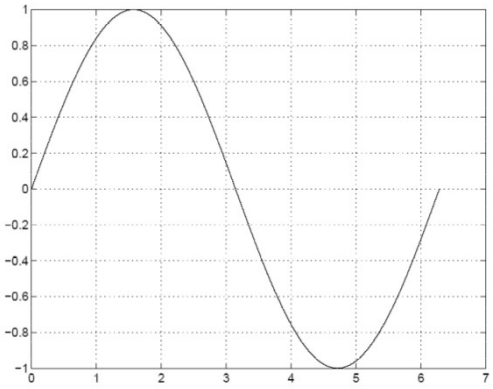
```
>>plot(y2)
```



Graphics - 2D Plots

Another simple example:

```
t = 0:pi/100:2*pi;
y = sin(t);
plot(t,y)
```



Graphics - 2D Plots

Another simple example:

```
t = 0:pi/100:2*pi;
y = sin(t);
plot(t,y)
```

Remember:

EVERYTHING IN MATLAB IS A MATRIX !

creates 1 x 200 Matrix

Argument and result: 1 x 200 Matrix

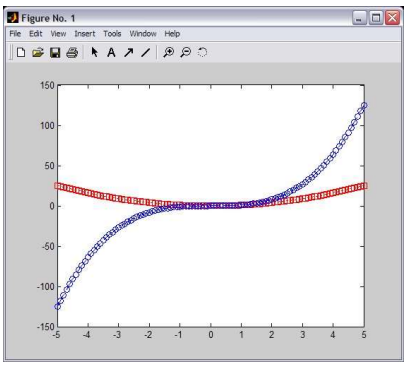
Graphics - 2D Plots - Overlay Plots

Use `hold on` for overlaying graphs

So the following:

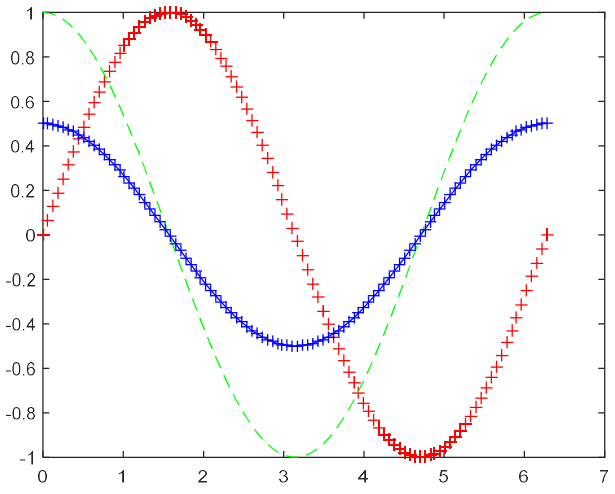
```
>> hold on;
>> cub=x.^3;
>> pl2=plot(x, cub,'b-o');
```

Gives:



Graphics - 2D Plots - Overlay Plots

```
x=linspace(0,(2*pi),100);
y1=sin(x);
y2=cos(x);
y3=cos(x)/2;
plot(x,y1,'r+')
hold
plot(x,y2,'g--')
plot(x,y3,'b+-')
```

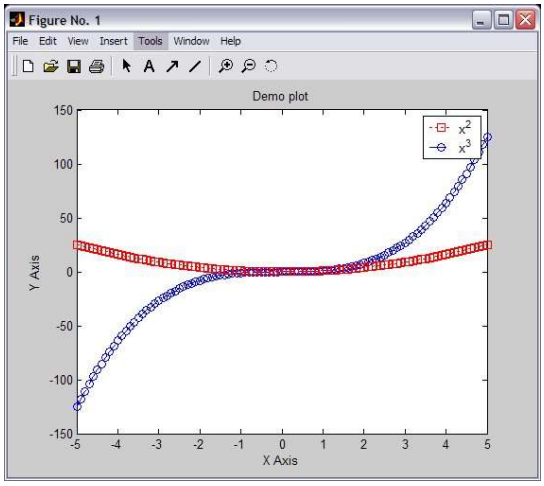


Graphics - 2D Plots - Annotation

Use `title, xlabel, ylabel` and `legend` for annotation

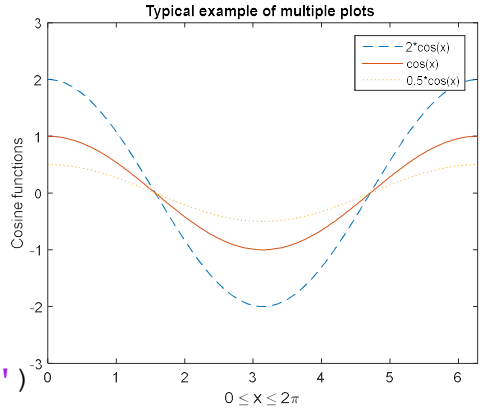
```
>> title('Demo plot');
>> xlabel('X Axis');
>> ylabel('Y Axis');
>> legend([pl1, pl2], 'x^2', 'x^3');
```

Graphics - 2D Plots - Annotation



Graphics - 2D Plots - Annotation

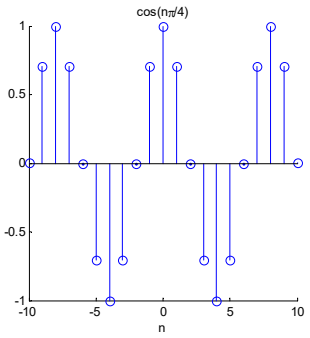
```
x = 0:pi/100:2*pi;
y1 = 2*cos(x);
y2 = cos(x);
y3 = 0.5*cos(x);
plot(x,y1,'--',x,y2,'-',x,y3,':')
xlabel('0 \leq x \leq 2\pi')
ylabel('Cosine functions')
legend('2*cos(x)','cos(x)','0.5*cos(x)')
title('Typical example of multiple
plots')
axis([0 2*pi -3 3])
```



Graphics - 2D Plots – stem()

- `stem()` is to plot discrete sequence data
- The usage of `stem()` is very similar to `plot()`

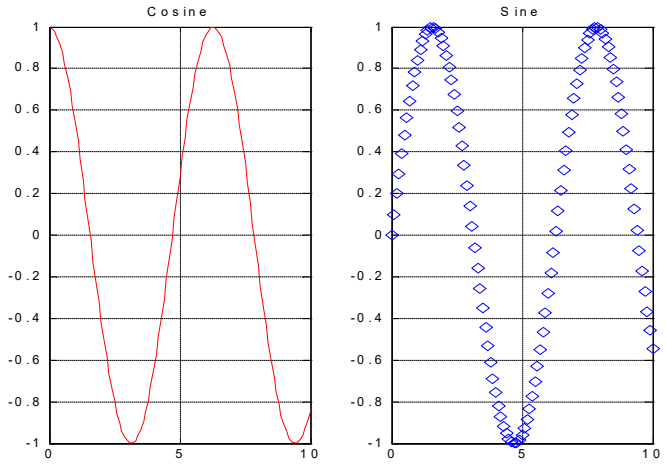
```
>> n=-10:10;
>> f=stem(n,cos(n*pi/4))
>> title('cos(n\pi/4)')
>> xlabel('n')
```



Graphics - 2D Plots - subplots

Use subplots to divide a plotting window into several panes.

```
>> x=0:0.1:10;
>> f=figure;
>> f1=subplot(1,2,1);
>> plot(x,cos(x),'r');
>> grid on;
>> title('Cosine')
>> f2=subplot(1,2,2);
>> plot(x,sin(x),'d');
>> grid on;
>> title('Sine');
```



Graphics - Save plots

- Use `saveas(h,'filename.ext')` to save a figure to a file.

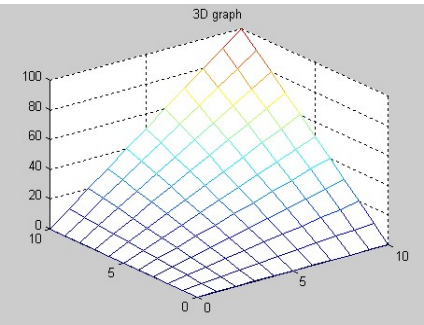
```
>> f=figure;
>> x=-5:0.1:5;
>> h=plot(x,cos(2*x+pi/3));
>> title('Figure 1');
>> xlabel('x');
>> saveas(gfc,'figure1‘,’jpg’)
```

Useful extension types:
- bmp: Windows bitmap
- emf: Enhanced metafile
- eps: EPS Level 1
- fig: MATLAB figure
- jpg: JPEG image
- m: MATLAB M-file
- tif: TIFF image, compressed

Data visualization – plotting graphs

Example on *mesh* and *surf* – 3 dimensional plot

- x=[0:10]; y=[0:10]; z=x'*y;
- mesh(x,y,z); title('3-D Graph');



Data visualization – plotting graphs

Example on *mesh* and *surf* – 3 dimensional plot

Supposed we want to visualize a function

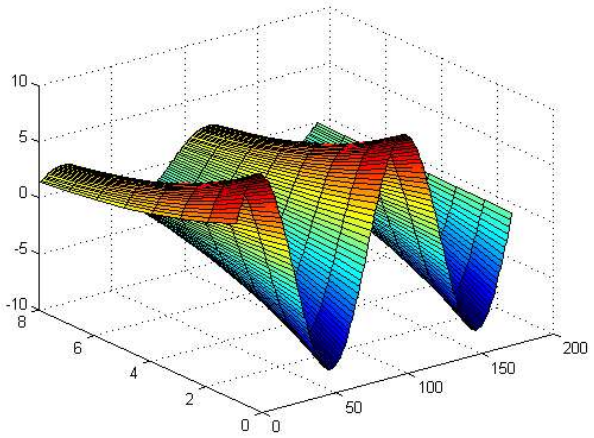
$$Z = 10e^{(-0.4a)} \sin(2\pi ft) \qquad \text{for } f = 2$$

when *a* and *t* are varied from 0.1 to 7 and 0.1 to 2, respectively

```
>>> [t,a] = meshgrid(0.1:.01:2, 0.1:0.5:7);
>>> f=2;
>>> Z = 10.*exp(-a.*0.4).*sin(2*pi.*t.*f);
>>> surf(Z);
>>> figure(2);
>>> mesh(Z);
```

Data visualization – plotting graphs

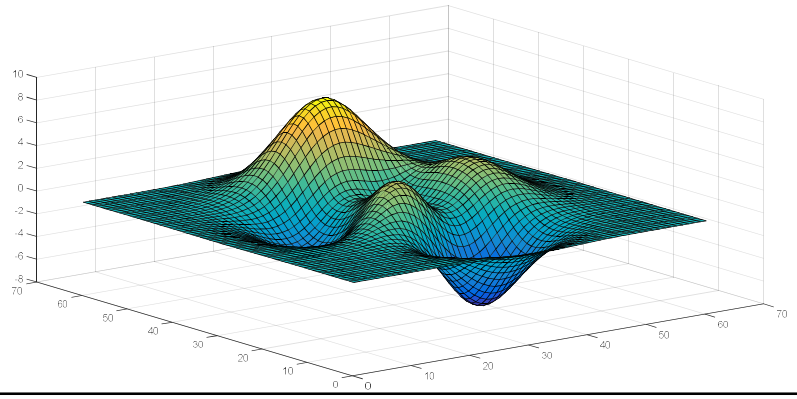
Example on *mesh* and *surf* – 3 dimensional plot



Data visualization – plotting graphs

Example on *mesh* and *surf* – 3 dimensional plot

```
[x,y] = meshgrid(-3:.1:3,-3:.1:3);
 z = 3*(1-x).^2.*exp(-(x.^2) - (y+1).^2) -
10*(x/5 - x.^3 - y.^5).*exp(-x.^2-y.^2) -
1/3*exp(-(x+1).^2 - y.^2);
surf(z);
```



Operators (relational, logical)

- == Equal to
- ~= Not equal to
- < Strictly smaller
- > Strictly greater
- <= Smaller than or equal to
- >= Greater than equal to
- & And operator
- | Or operator

Flow Control

- if
- for
- while
- break
- ….

Control Structures

- If Statement Syntax

```
if (Condition_1)
        Matlab Commands
elseif (Condition_2)
        Matlab Commands
elseif (Condition_3)
        Matlab Commands
else
        Matlab Commands
end
```

Some Dummy Examples

```
if ((a>3) & (b==5))
    Some Matlab Commands;
end
```

```
if (a<3)
    Some Matlab Commands;
elseif (b~=5)
    Some Matlab Commands;
end
```

```
if (a<3)
    Some Matlab Commands;
else
    Some Matlab Commands;
end
```

Control Structures

- For loop syntax

```
for i=Index_Array
  Matlab Commands
end
```

Some Dummy Examples

```
for i=1:100
    Some Matlab Commands;
end
```

```
for j=1:3:200
    Some Matlab Commands;
end
```

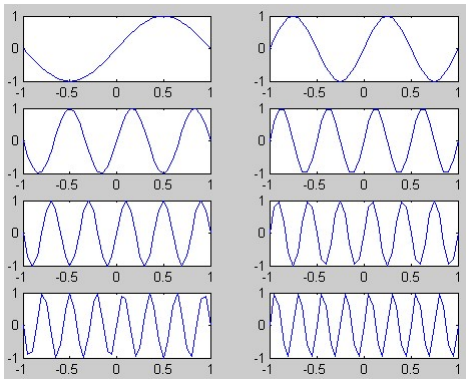
```
for m=13:-0.2:-21
    Some Matlab Commands;
end
```

```
for k=[0.1 0.3 -13 12 7 -9.3]
    Some Matlab Commands;
end
```

Control Structures

"for" loop example

```
>> x = -1:.05:1;
>> for n = 1:8
     subplot(4,2,n);
     plot(x,sin(n*pi*x));
   end
```



Control Structures

- While Loop Syntax

```
while (condition)
  Matlab Commands
end
```

Dummy Example

```
while  ((a>3) & (b==5))
    Some Matlab Commands;
end
```

switch

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

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

switch

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

- CODE

Image Processing Toolbox

collection of functions that extend the capabilities of the MATLAB's numeric computing environment. The toolbox supports a wide range of image processing operations, including:

- Geometric operations
- Neighborhood and block operations
- Linear filtering and filter design
- Transforms
- Image analysis and enhancement
- Binary image operations
- Region of interest operations

Images in MATLAB

MATLAB can import/export several image formats:

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

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 in MATLAB

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

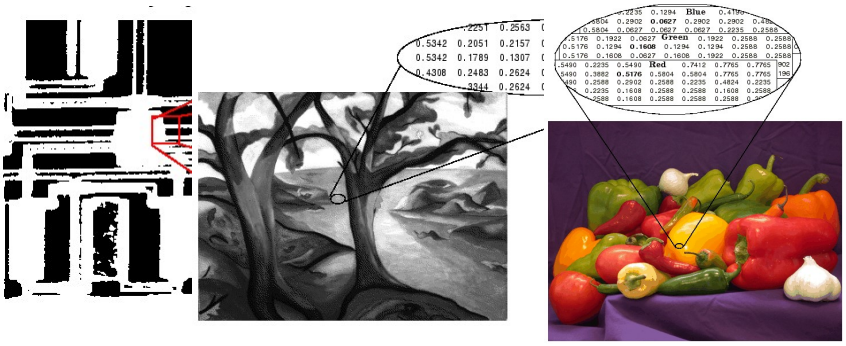


Image Import and Export

- Read and write images in Matlab

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

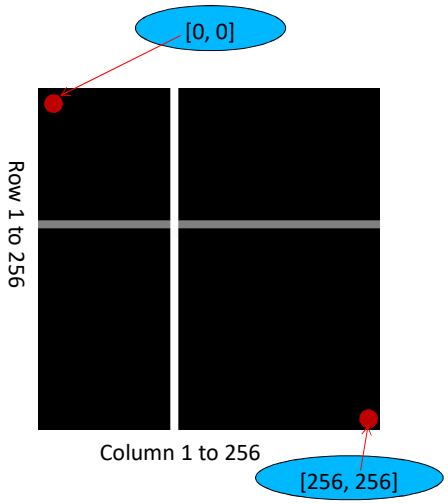
- Alternatives to `imshow`

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

Images and Matrices

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

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



Images and Matrices

Binary Image:

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

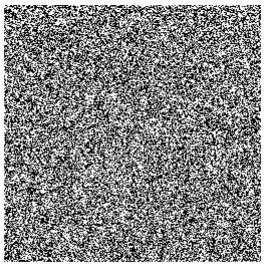


Image Display

- `image` - create and display image object
- `imagesc` - scale and display as image
- **`imshow`** - display image
- `colorbar` - display colorbar
- `getimage` - get image data from axes
- `truesize` - adjust display size of image
- `zoom` - zoom in and zoom out of 2D plot

Image Display

Loading an image:

```
a = imread('picture.jpg');
imshow(a);
```

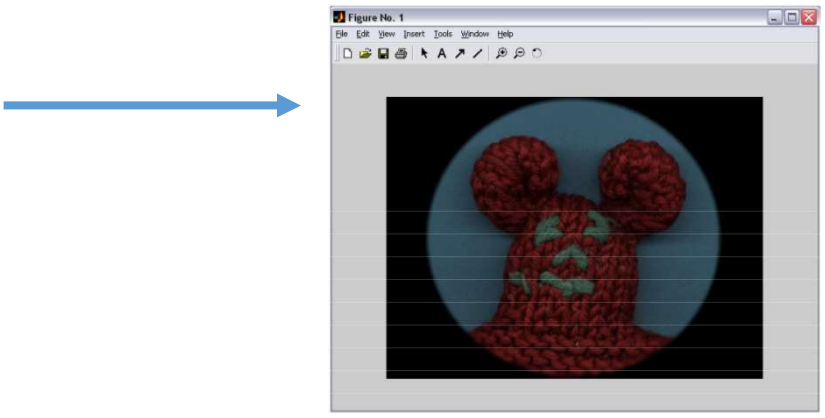


Image info

Image (=matrix) size:
size(a): 384 512 3

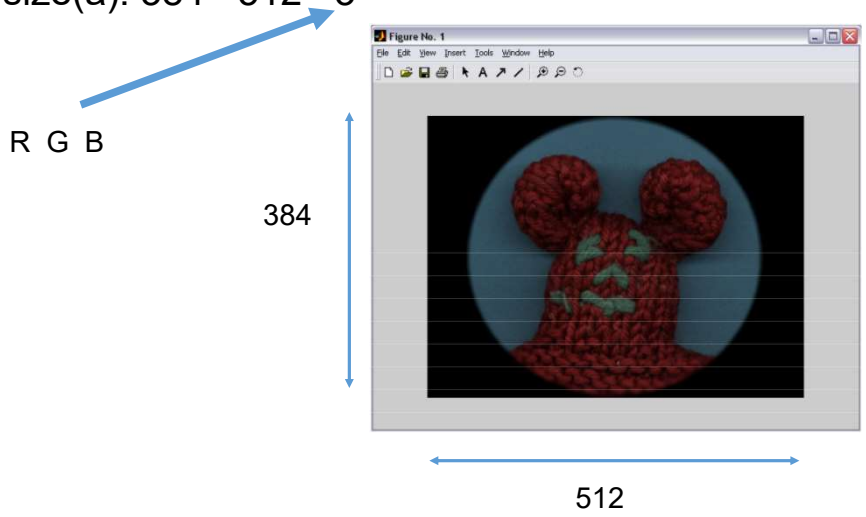
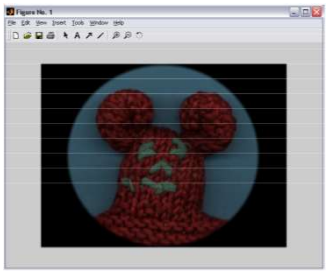


Image Display



Color image:
3D Matrix of RGB planes

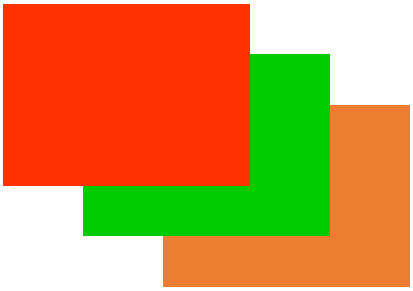
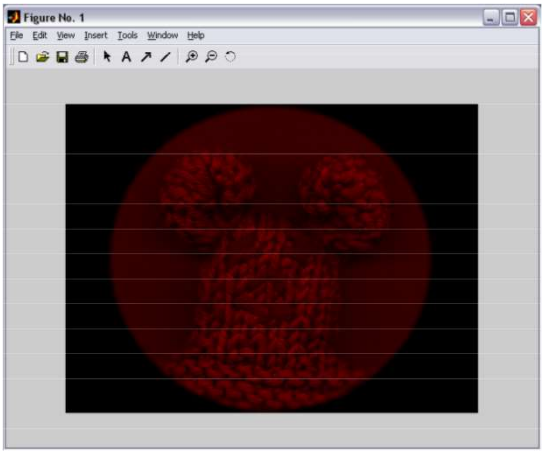


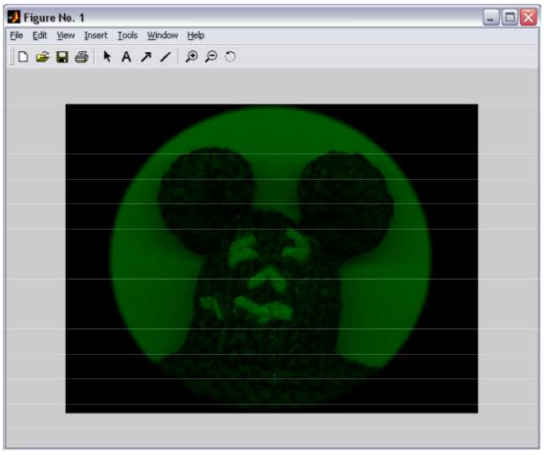
Image Display



Show RED plane:

```
a(:,:,2:3) = 0;
imshow(a);
```

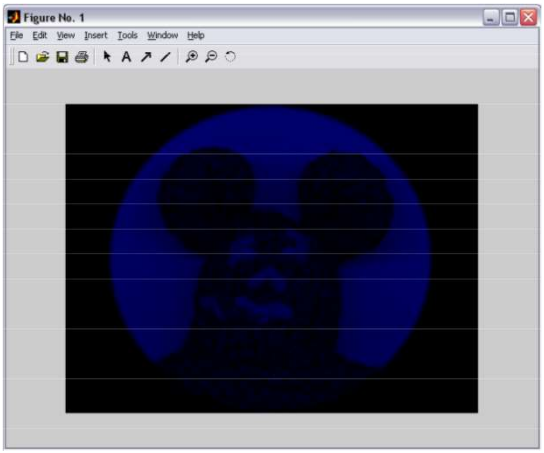
Image Display



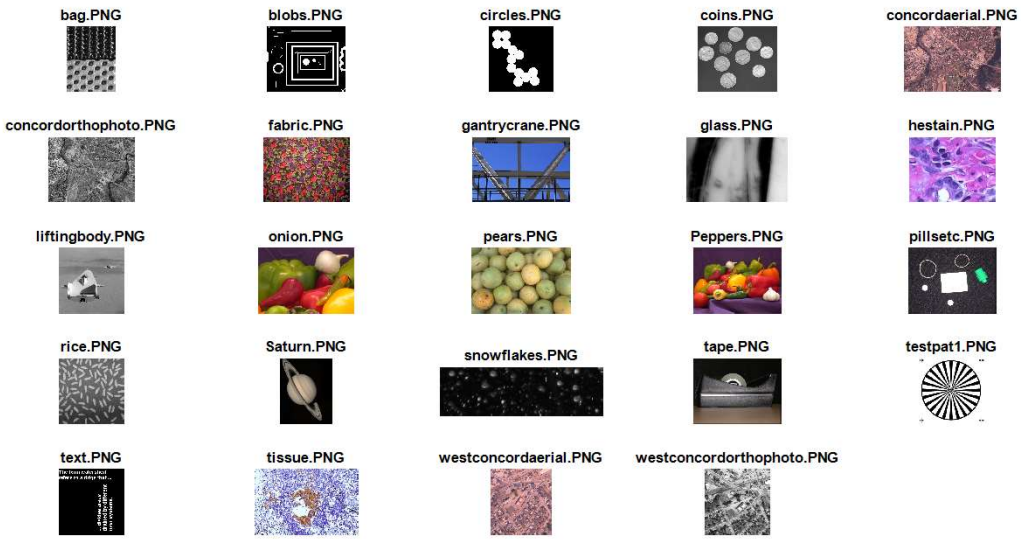
Show GREEN plane:

```
a(:,:,[1 3]) = 0;
imshow(a);
```

Image Display



Images included in MATLAB



Images included in MATLAB

autumn.tif 

Board.tif 

cameraman.tif 

Canoe.tif 

cell.tif 

circbw.tif 

circuit.tif 

eight.tif 

forest.tif 

kids.tif 

logo.tif 

M83.tif 

Mandi.tif 

Moon.tif 

MRI.tif 

Paper1.tif

pout.tif 

shadow.tif 

spine.tif 

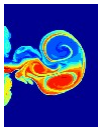
tire.tif 

trees.tif 

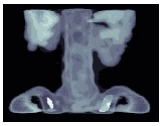
Images included in MATLAB


mandrill.MAT


clown.MAT


flujet.MAT


Earth.MAT


spine.MAT


Cape.MAT


Durer.MAT


Gatlin.MAT


gatlin2.MAT

Image Conversion

- `gray2ind` - intensity image to index image
- `im2bw` - image to binary
- `im2double` - image to double precision
- `im2uint8` - image to 8-bit unsigned integers
- `im2uint16` - image to 16-bit unsigned integers
- `ind2gray` - indexed image to intensity image
- `mat2gray` - matrix to intensity image
- `rgb2gray` - RGB image to grayscale
- `rgb2ind` - RGB image to indexed image

Image Operations

- image conversion type
- Image resize, crop, mirror, rotate
- Image histogram, histogram equalization
- Transformations: operation, LUT,…
- Convolution,
- Filters: mean, Gaussian, median,…
- Contours detection
- Erosion and dilation
- Image FFT, DCT, DWT,…
- …