Introduction to Matlab

MATLAB R2006a
The Language of Technical Computing

Simone Gasparini
gasparini@elet.polimi.it
Argomenti avanzati di analisi delle immagini
prof. Vincenzo Caglioti
a.a. 2006-2007

Wednesday, 14 March 2007

How to get Matlab

- Computer-equipped classroom
  - Complete list available at http://www.cia.polimi.it/servizi/software.html (then select wished Matlab released
- Download it from polimi-cdserver (Campus license)
  - http://www.cia.polimi.it/servizi/software.html?cdserv=1
  - It works only inside the polimi network
- Your own copy…

Matlab Documentation

- Several tutorial available on the Internet
  - E.g. http://www.serc.iisc.ernet.in/ComputingFacilities/software/getstart.pdf
- Help on line

Matlab overview

- MATrix LABoratory
- High-performance language for technical computing.
- It integrates computation, visualization, and programming in an easy-to-use environment
- Problems and solutions are expressed in familiar mathematical notation.
- Used for:
  - Math and computation
  - Algorithm development
  - Data acquisition
  - Modeling, simulation, and prototyping
  - Data analysis, exploration, and 2D-3D visualization
  - Scientific and engineering graphics
  - Application development, including GUI building

Matlab overview

- Platforms

- Matlab woks on several platforms
  - PCs powered by Windows (provided by www.cia.polimi.it)
  - Unix/Linux Systems (provided by www.cia.polimi.it)
  - Macintosh (not provided)

- Similar GUI and interface

Matlab overview

- Toolboxes

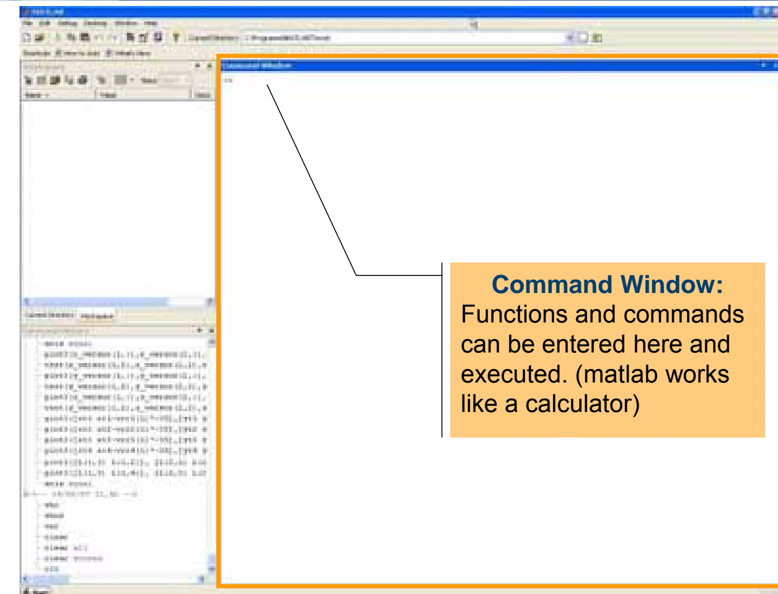
- MATLAB features a family of add-on application-specific solutions called *toolboxes*.
- Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems.
- They cover many areas:
  - signal processing,
  - control systems,
  - neural networks,
  - fuzzy logic,
  - wavelets,
  - simulation…

Matlab overview

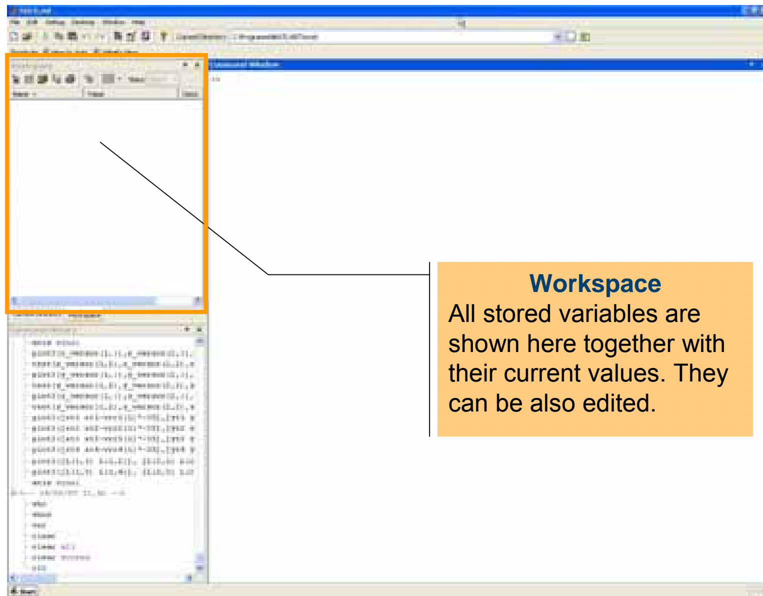
- Matrix is the basic type
  - Scalar values are 1x1 matrices
  - matrix is a rectangular array of numbers.

- Other programming languages (eg C/C++, Java) work with numbers one at a time
- MATLAB works with entire matrices quickly and easily.
- Operations in MATLAB are designed to be as natural as possible.
  - E.g.: >> A+B
  - No matters if A or B are vectors, matrices or scalar values
  - MATLAB automatically compute the sum
  - Only dimensions matter in order to have consistent operations
    - A and B can't be matrices with different dimensions

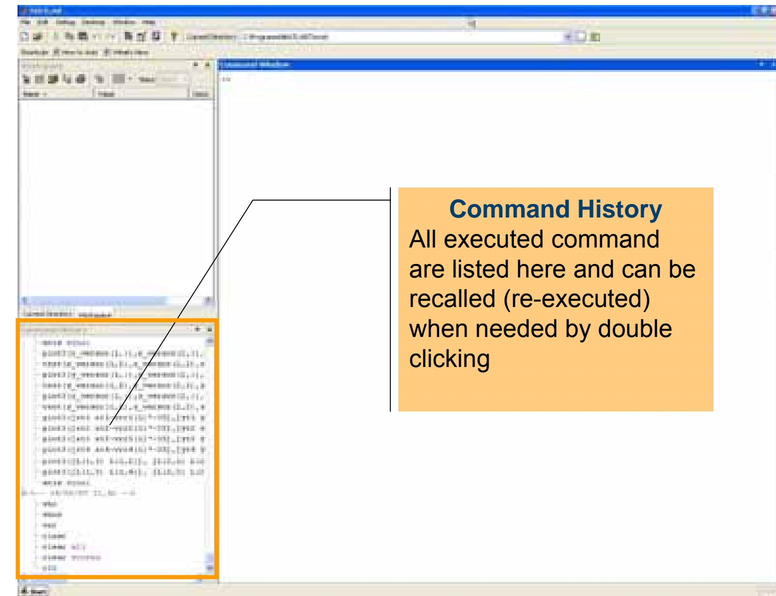
Matlab Desktop



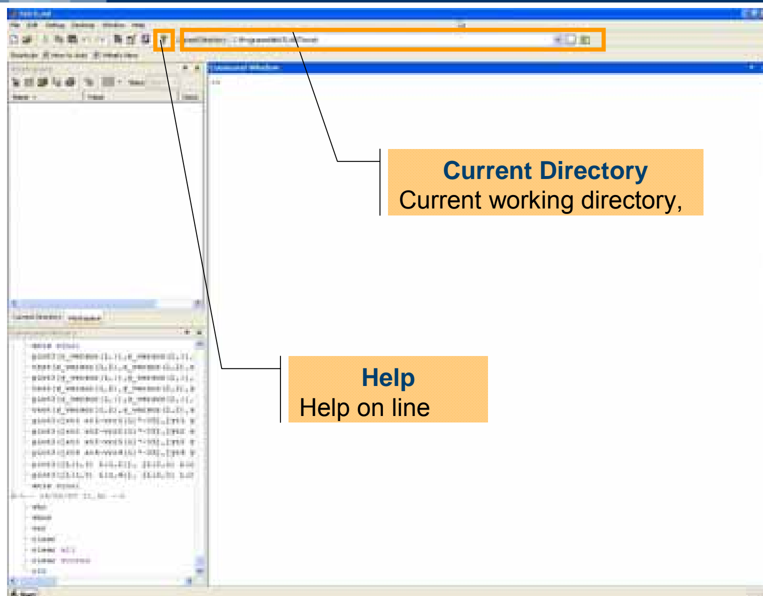
Matlab Desktop



Matlab Desktop



Matlab Desktop



Matlab helpdesk

Matlab online help


```
Command Window
>> help svd
 SVD    Singular value decomposition.
    [U,S,V] = SVD(X) produces a diagonal matrix S, of the same
    dimension as X and with nonnegative diagonal elements in
    decreasing order, and unitary matrices U and V so that
    X = U*S*V'.

    S = SVD(X) returns a vector containing the singular values.

    [U,S,V] = SVD(X,0) produces the "economy size"
    decomposition. If X is m-by-n with m > n, then only the
    first n columns of U are computed and S is n-by-n.
    For m <= n, SVD(X,0) is equivalent to SVD(X).

    [U,S,V] = SVD(X,'econ') also produces the "economy size"
    decomposition. If X is m-by-n with m >= n, then it is
    equivalent to SVD(X,0). For m < n, only the first m columns
    of V are computed and S is m-by-m.

    See also svds, gsvd.

    Overloaded functions or methods (ones with the same name in other directories)
       help sym/svd.m

    Reference page in Help browser
       doc svd

>> |
```

Variables

- MATLAB variable is a tag assigned to a value while it remains in memory (workspace).
- A “value” can be:
  - A scalar value
  - A matrix
  - A string, which is a vector of char
- Names must begin with a letter, which may be followed by any combination of letters, digits, and underscores.
- MATLAB is *case sensitive*.
- Assign a value to a scalar
  - >> Var_name = value

Variables

Workspace

Name	Value	Class
T	150	double
t	10	double

```
Command Window
>> t = 10
t =
    10
>> T = 150;
>> |
```

Variables are added to the workspace. The values are shown and can be manually edited by double-clicking

Semicolon at the end of commands suppresses output

Matrices

- There are several way to enter a matrix
  - Enter an explicit list of elements.
  - Load matrices from external data files.
  - Generate matrices using built-in functions.
  - Create matrices with your own functions in M-files.
- explicit list of elements

Workspace

Name	Value	Class
A	<4x4 double>	double

```
Command Window
>> A = [ 16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
>>
```

Matrices

```
Workspace
Name     Value          Class
A        <4x4 double>   double
```

```
Command Window
>> A = [ 16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
>>
```

- Separate the elements of a row with blanks or commas.
- Use a semicolon, ; , to indicate the end of each row.
- Surround the entire list of elements with square brackets, [ ].

Vectors

- Vectors are mono-dimensional matrices

```
Workspace
Name     Value          Class
A        <4x4 double>   double
vc       [1;2;4]        double
vr       [1 2 4]        double
```

```
Command Window
>> A = [ 16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
>> vr = [1 2 4]
vr =
     1     2     4
>> vc = [1; 2; 4]
vc =
     1
     2
     4
>>
```

Row vector

Column vector

Working with matrices: Example

```
Workspace
Name     Value          Class
A        <4x4 double>   double
ans      34             double
vc       [1;2;4]        double
vr       [1 2 4]        double

Command History
%-- 15/03/07 10.46 --%
A = [ 16 3 2 13; 5 10 11 8; 9 6 7
vr = [1 2 4]
vc = [1; 2; 4]
clc
```

```
Command Window
>> A
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
>> sum(A)
ans =
    34    34    34    34
>> A'
ans =
    16     5     9     4
     3    10     6    15
     2    11     7    14
    13     8    12     1
>> sum(A')
ans =
    34    34    34    34
>> diag(A)
ans =
    16
    10
     7
     1
>> sum(diag(A))
ans =
    34
>>
```

Sum the colums of A

Transpose of A

Return the diagonal elements of A

Working with matrices: Example

- **ans** is the default variable where the output is stored
- Results can be stored into a different variable by just assigning the result

```
Workspace
Name     Value            Class
A        <4x4 double>     double
a        [34 34 34 34]    double
ans      [34 34 34 34]    double
vc       [1;2;4]          double
vr       [1 2 4]          double
```

```
Command Window
>> sum(A)
ans =
    34    34    34    34
>> a = sum(A)
a =
    34    34    34    34
>> a
a =
    34    34    34    34
>>
```

Matrix elements

- To access to single elements of matrix use `A(r,c)` notation, where r is the index of row and c index of column
- C-like programmers, watch out!
  - The first element (row or column) has index 1

Command Window

```
>> A
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
>> A(1,1)
ans =
    16
>> r=3
r =
     3
>> c=2
c =
     2
>> A(r,c)
ans =
     6
>>
```

Command Window

```
>> A(1,1) + A(1,2) + A(1,3) + A(1,4)
ans =
    34
>> A(4,2) = 100
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4   100    14     1
>> A(5,5) = -10
A =
    16     3     2    13     0
     5    10    11     8     0
     9     6     7    12     0
     4   100    14     1     0
     0     0     0     0   -10
>>
```

When assigning a value, if index exceeded MATLAB automatically increases dimensions to place the new element, putting to 0 all the other new elements

Colon operator

- The colon ":" operator is the most important MATLAB operator
- Use `starting_value:increment:end_value`
- it returns a vector containing all values from `starting_value` to `end_value` with step `increment`

Workspace

Name	Value	Class
A	<4x4 double>	double
a	[34 34 34 34]	double
ans	[1 2 3 4 5 6 7 8 9 10]	double
c	2	double
r	3	double
v	[1 3 5 7 9]	double
vc	[1;2;4]	double
vr	[1 2 4]	double

Command Window

```
>> 1:10
ans =
     1     2     3     4     5     6     7     8     9    10
>> v = 1:10
v =
     1     2     3     4     5     6     7     8     9    10
>> v= 1:2:10
v =
     1     3     5     7     9
>>
```

Colon operator

- Very useful to access to a part of a matrix
- **`end`** refers to the last element (row or column)

Command Window

```
>> k = 3
k =
     3
>> A(1:k,3)
ans =
     2
    11
     7
>> sum(A(1:k,3))
ans =
    20
>> sum(A(:,3))
ans =
    34
>> sum(A(:,end))
ans =
    34
>>
```

- `A(1:k,3)`: Returns the first k elements of the 3d column
- `sum(A(1:k,3))`: Returns the sum of the first k elements of the 3d column
- `sum(A(:,3))`: Returns the sum of the ALL elements of the 3d column
- `sum(A(:,end))`: Returns the sum of the ALL elements of the LAST column

Operators

- Arithmetic operators
  - + addition
  - - subtraction
  - * multiplication
  - / division
  - ^ power
  - ' transpose

Addition example

```
Command Window
>> A
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4   100    14     1
>> A + A'
ans =
    32     8    11    17
     8    20    17   108
    11    17    14    26
    17   108    26     2
>> A + 2
ans =
    18     5     4    15
     7    12    13    10
    11     8     9    14
     6   102    16     3
>> |
```

Sum correspondent elements

Sum the scalar value to each element

Multiplication example

```
Command Window
>> A
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4   100    14     1
>> A * A
ans =
         341        1390         261         269
         261         981         309         285
         285        1329         301         261
         694        1196        1220        1021
>> A * 2
ans =
    32     6     4    26
    10    20    22    16
    18    12    14    24
     8   200    28     2
>> A * [1:4]'
ans =
    80
    90
    90
   250
>>
```

Classical matrix product (rows by columns)

Each element multiplied by 2

Matrix – vector product

Element-wise operators

- Operators that work on matrix elements
  - + sum
  - - subtraction
  - .* element by element multiplication
  - ./ element by element division
  - .^ element by element power

Example

```
Command Window
>> A
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4   100    14     1
>> A * A
ans =
         341        1390         261         269
         261         981         309         285
         285        1329         301         261
         694        1196        1220        1021
>> A .* A
ans =
         256           9           4         169
          25         100         121          64
          81          36          49         144
          16       10000         196           1
>> A ./ A
ans =
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
>> A ./ A'
ans =
    1.0000    0.6000    0.2222    3.2500
    1.6667    1.0000    1.8333    0.0800
    4.5000    0.5455    1.0000    0.8571
    0.3077   12.5000    1.1667    1.0000
>> |
```

Classical matrix product (rows by columns)

Elementwise division

Concatenation

- *Concatenation* is the process of joining small matrices to make bigger ones.
- The pair of square brackets, [], is the concatenation operator.

```
Command Window
>> B = [A(:,3) A(:,4) A(:,2) A(:,1)]
B =
     2    13     3    16
    11     8    10     5
     7    12     6     9
    14     1   100     4
     0     0     0     0
>> C = [B A(:,2:4) A*B]
C =
  Columns 1 through 6
           2          13           3          16           3           2
          11           8          10           5          10          11
           7          12           6           9           6           7
          14           1         100           4         100          14
           0           0           0           0           0           0
  Columns 7 through 11
          13         261         269        1390         341
           8         309         285         981         261
          12         301         261        1329         285
           1        1220        1021        1196         694
           0           0           0           0           0
>>
```

Deleting row and columns

- The pair of square brackets “[]” can be used to delete rows or columns of a matrix

```
Command Window
>> X = A
X =
    16     3     2    13     0
     5    10    11     8     0
     9     6     7    12     0
     4   100    14     1     0
     0     0     0     0   -10
>> X(:,3) = []
X =
    16     3    13     0
     5    10     8     0
     9     6    12     0
     4   100     1     0
     0     0     0   -10
>> X(2,:) = []
X =
    16     3    13     0
     9     6    12     0
     4   100     1     0
     0     0     0   -10
>> X(1,2) = []
???  Indexed empty matrix assignment is not allowed.
```

Delete 3rd column of X

Delete 2nd row of X

Single element can not be deleted

Other functions

- `max(A)` return an array with the maximum value of each column
  - Call `max(max(A))` to get the overall maximum of A
- `min(A)` like max return an array with the minimum value of each column
- `length(A)` return the value of the larger dimension of A
- `size(A)` return the values of the dimensions of A
  - `d = size(A)`
    - Put in matrix d the dimensions of A (d = [r c])
  - `[r,c] = size(A)`
    - Put in separate variables r and c the dimensions of A
  - `m = size(A,dim)`
    - Put in m the size of dimensions dim of A (dim = 1 means rows, 2 for columns)

Example

```
Command Window
>> A
A =
    16     3     2    13     0
     5    10    11     8     0
     9     6     7    12     0
     4   100    14     1     0
     0     0     0     0   -10
>> max(A)
ans =
    16   100    14    13     0
>> max(max(A))
ans =
   100
>> min(A)
ans =
     0     0     0     0   -10
>> min(min(A))
ans =
   -10
```

```
>> length(A)
ans =
     5
>> length(A(:,1:4))
ans =
     5
>> size(A)
ans =
     5     5
>> d = size(A)
d =
     5     5
>> [r c] = size(A)
r =
     5
c =
     5
>> r = size(A, 1)
r =
     5
>>
```

Special matrices

Command Window

```
>> eye(4)
ans =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
>> zeros(4,2)
ans =
     0     0
     0     0
     0     0
     0     0
>> ones(4,2)
ans =
     1     1
     1     1
     1     1
     1     1
>> rand(3,4)
ans =
    0.3784    0.5936    0.8216    0.6602
    0.8600    0.4966    0.6449    0.3420
    0.8537    0.8998    0.8180    0.2897
>>
```

eye(n) returns the n-by-n identity matrix

zeros(n,m) returns the n-by-m matrix of all zeros

ones(n,m) returns the n-by-m matrix of all one

rand(n,m) returns the n-by-m matrix of random numbers

Solving linear systems

- Linear systems in matrix form can be solved using operator "\"
- Given Ax=b with x vector of unknown, the system can be solved as x=A\b which corresponds to x = inv(A) * b

Command Window

```
>> A = [2 5 6; 6 4 -12; 1 0 4]
A =
     2     5     6
     6     4   -12
     1     0     4
>> b = [32; 6; 1]
b =
    32
     6
     1
>> x = A \ b
x =
   -1.7907
    6.2791
    0.6977
>> |
```

$$\begin{cases} 2x+5y+6z=32 \\ 6x+4y-12z=6 \\ x+4z=1 \end{cases}$$

If the equation Ax = b does not have a solution (and A is not a square matrix), x = A\b returns a *least squares solution*

Using MATLAB as a programming language

- Use M-file to create script or function that can be called from the command prompt
- Script are sequence of function/command call that can be executed in batch calling the m-file
- M-file is a normal text file that contains the sequence of commands to be execute
- To execute a M-file simply call the name of the m-file from command prompt
- M-file can be edited using the provide editor of MATLAB, but you can use any text editor you like
- MATLAB provide flow control structures (if, for, while ecc) that can be used in a m-file

MATLAB m-file editor

Editor - C:\Programmi\MATLAB7\work\prova.m

File Edit Text Cell Tools Debug Desktop Window Help

```
1  - A = [ 16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
2
3  - maximum = max(max(A))
4
5  - minimum = min(min(A))
6
7  - [r c] = size(A);
8
9  - B = A*A'
10
11 - C = A-B
12
13
```

evalline.m | findline.m | pv24param... | Fourparams... | coplanarRe... | drawCircle... | prova.m

script Ln 11 Col 7 OVR

Example – Calling a m-file

MATLAB

File Edit Debug Desktop Window Help

Current Directory: C:\Programmi\MATLAB7\work

Shortcuts How to Add What's New

Workspace

Name	Value	Class
A	<4x4 double>	double
B	<4x4 double>	double
C	<4x4 double>	double
c	4	double
maximum	16	double
minimum	1	double
r	4	double

Current Directory Workspace

Command Window

```
>> prova
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
maximum =
    16
minimum =
     1
B =
   438   236   332   150
   236   310   278   332
   332   278   310   236
   150   332   236   438
C =
  -422  -233  -330  -137
  -231  -300  -267  -324
  -323  -272  -303  -224
  -146  -317  -222  -437
>>
```

Flow control – if…else and elseif

```
if expression1
    statements1
elseif expression2
    statements2
elseif expression3
    statements3
…
else
    statements4
end
```

If `expression1` evaluates as true executes the one or more commands denoted here as `statements1`; else if `expression2` is true executes `statements2` and so on

Flow control – conditional expression

- Similar to C expressions
- Relational operators
  - <, <=, ==, >=, >, ~= (“not equal”, ALT+0126 to get ‘~’)
- Logical operators
  - & (“and”), | (“or”), ~ (“not”)

Flow control - switch

```
switch switch_expr
  case case_expr
    statement, ..., statement
  case {case_expr1, case_expr2, case_expr3, ...}
    statement, ..., statement
  otherwise
    statement, ..., statement
end
```

Flow control - for

```
for variable = startingval:step:endingval
    statements
end
```

- Example

```
[r c] = size(A)
for m = 1:r
    for n = 1:c
        A(m,n) = 1/(m+n -1);
    end
end
```

- The counter variables are managed automatically by MATLAB inside the loop!

Flow control - while

```
while expression
    statements
end
```

- Similar to C
- For and while inside the loop allow the use of:
  - `continue` to pass the control to the next iteration
  - `break` to exit early from the loop

Function

- Functions are stored in m-file named with the same name of the function
- Declaration:

```
function [y1, ... , yn] = nome_funzione (x1, ... , xm)
```

- Where
  - `x1, ... , xm` are input params (optional)
  - `[y1, ... , yn]` are output params (optional)
- N.B. All input parameters are passed by value
  - They are not affected by change inside the function

Function example

Editor - C:\Documents and Settings\gasparin\Documenti\work\torrsam20050706\ransacfithomography.m

File Edit Text Cell Tools Debug Desktop Window Help

```
34
35
36  function [H, inliers] = ransacfithomography(x1, x2, t)
37
38 -    if ~all(size(x1)==size(x2))
39 -        error('Data sets x1 and x2 must have the same dimension');
40 -    end
41
42 -    [rows,npts] = size(x1);
43 -    if rows~=2 & rows~=3
44 -        error('x1 and x2 must have 2 or 3 rows');
45 -    end
46
47 -    if npts < 4
48 -        error('Must have at least 4 points to fit homography');
49 -    end
50
51 -    if rows == 2    % Pad data with homogeneous scale factor of 1
52 -        x1 = [x1; ones(1,npts)];
53 -        x2 = [x2; ones(1,npts)];
54 -    end
55
56      % Normalise each set of points so that the origin is at centroid and
57      % mean distance from origin is sqrt(2).  normalise2dpts also ensures the
58      % scale parameter is 1.  Note that 'homography2d' will also call
59      % 'normalise2dpts' but the code in 'ransac' that calls the distance
60      % function will not - so it is best that we normalise beforehand.
```

Function example

```
52          x1 = [x1; ones(1,npts)];
53 -        x2 = [x2; ones(1,npts)];
54 -    end
55
56      % Normalise each set of points so that the origin is at centroid and
57      % mean distance from origin is sqrt(2).  normalise2dpts also ensures the
58      % scale parameter is 1.  Note that 'homography2d' will also call
59      % 'normalise2dpts' but the code in 'ransac' that calls the distance
60      % function will not - so it is best that we normalise beforehand.
61 -    [x1, T1] = normalise2dpts(x1);
62 -    [x2, T2] = normalise2dpts(x2);
63
64 -    s = 4;  % Minimum No of points needed to fit a homography.
65
66 -    fittingfn = @homography2d;
67 -    distfn    = @homogdist2d;
68 -    degenfn   = @isdegenerate;
69      % x1 and x2 are 'stacked' to create a 6xN array for ransac
70 -    [H, inliers] = ransac([x1; x2], fittingfn, distfn, degenfn, s, t);
71
72      % Now do a final least squares fit on the data points considered to
73      % be inliers.
74 -    H = homography2d(x1(:,inliers), x2(:,inliers));
75
76      % Denormalise
77 -    H = T2\H*T1;
78
79  %-----------------------------------------------------------------------
```

evalline.m | findline.m | pv24param... | Fourparams... | coplanarRe... | drawCircle... | prova.m* | funfatt.m* | ransacfitho... | prova4Rans...

ransacfithomography | Ln 1 | Col 1 | OVR

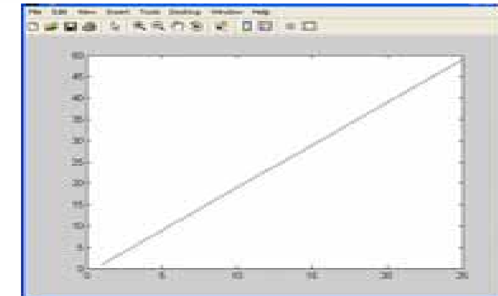
Graphics

- Function plot allow to create 2D graphics in MATLAB

`plot(Y)` plots the columns of Y versus their index

Command Window

```
>> y = 1:2:50
y =
  Columns 1 through 13
     1     3     5     7     9    11    13    15    17    19    21    23    25
  Columns 14 through 25
    27    29    31    33    35    37    39    41    43    45    47    49
>> plot(y)
>>
```



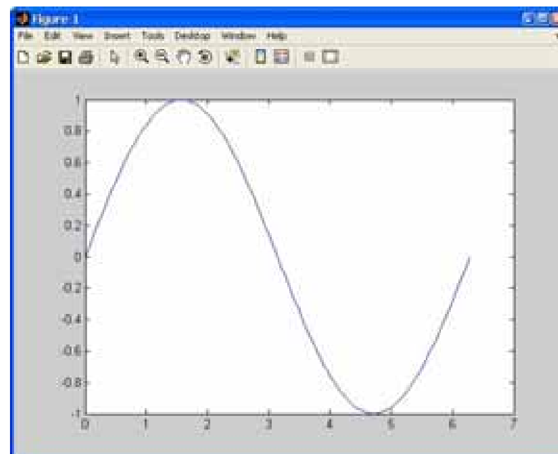
Graphics

- Function plot allow to create 2D graphics in MATLAB

`plot(X,Y)` plots the columns of X versus the column of Y

Command Window

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



Graphics

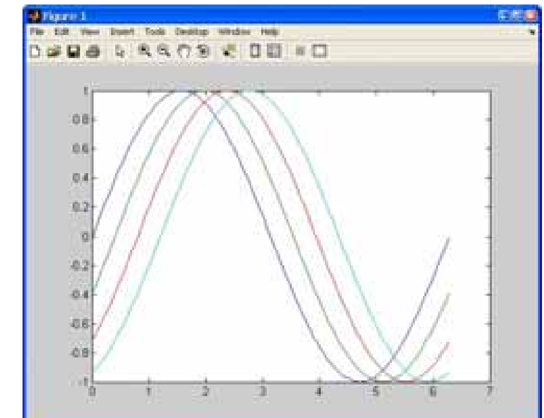
- To plot different graphics on the same reference frame

  `plot(x1,y1, x2,y2, x3,y3, x4,y4,…)`

- Or:

```
hold on
plot(x1,y1)
plot(x2,y2)
plot(x3,y3)
plot(x4,y4)
```

```
>> t = 0:pi/100:2*pi;
>> y = sin(t);
>> y2 = sin(t - .4);
>> y3 = sin(t - .8);
>> y4 = sin(t - 1.2);
>> plot(t,y,t,y2,t,y3,t,y4)
```



Graphics

- Plot style

```
plot(x1…, 'options')
```

- Option is a string delimited by ' containing the specifiers for line styles

Specifier	Line Style
-	Solid line (default)
--	Dashed line
:	Dotted line
-.	Dash-dot line

Specifier	Color
r	Red
g	Green
b	Blue
c	Cyan
m	Magenta
y	Yellow
k	Black
w	White

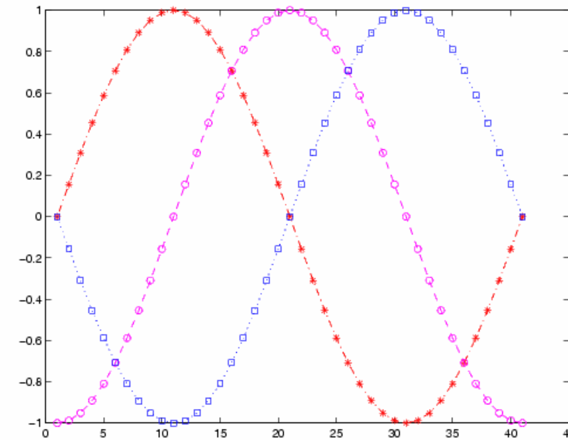
Specifier	Marker Type
+	Plus sign
o	Circle
*	Asterisk
.	Point
x	Cross
'square' or s	Square
'diamond' or d	Diamond
^	Upward-pointing triangle
v	Downward-pointing triangle
>	Right-pointing triangle
<	Left-pointing triangle
'pentagram' or p	Five-pointed star (pentagram)
'hexagram' or h	Six-pointed star (hexagram)

```
plot(x1,y1, ':ro')
```

Plot a red (r) dotted (:) line
placing a circle (o) on each point

Graphics - example

```
t = 0:pi/20:2*pi;
plot(t,sin(t),'-.r*')
hold on
plot(t,sin(t-pi/2),'--mo')
plot(t,sin(t-pi),':bs')
hold off
```



Images

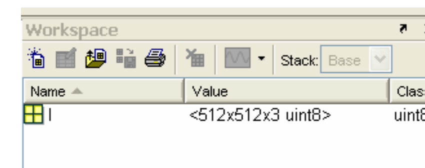
- In MATLAB images are matrices where each element store the value of the correspondent pixel
- Gray-scale images are bidimensional matrices
  - I(i,j) where i and j define the location of the pixel and k specifies the channel (R, G or B)
  - I(i,j) returns the graylevel of pixel at coordinate i,j
- Color images (RGB) are tridimensional matrices
  - I(i,j,k) where i and j define the location of the pixel and k specifies the channel (R, G or B)
  - I(i,j,k) returns the value of channel k of pixel at coordinate i,j

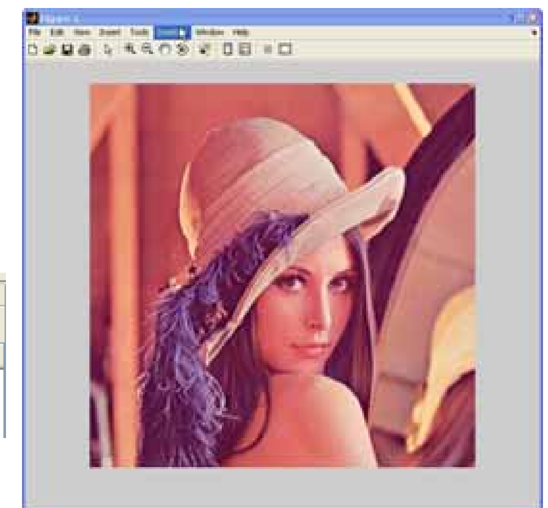
Images

- load and display an image

Command Window

```
>> I = imread('lena.jpg');
>> imshow(I);
>>
```

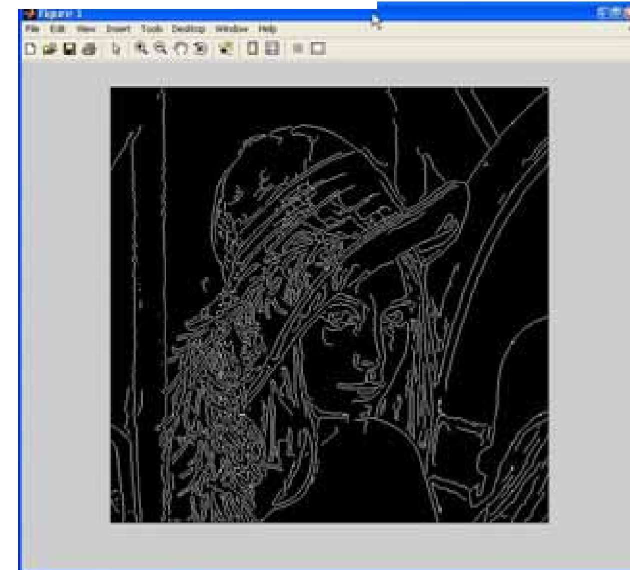




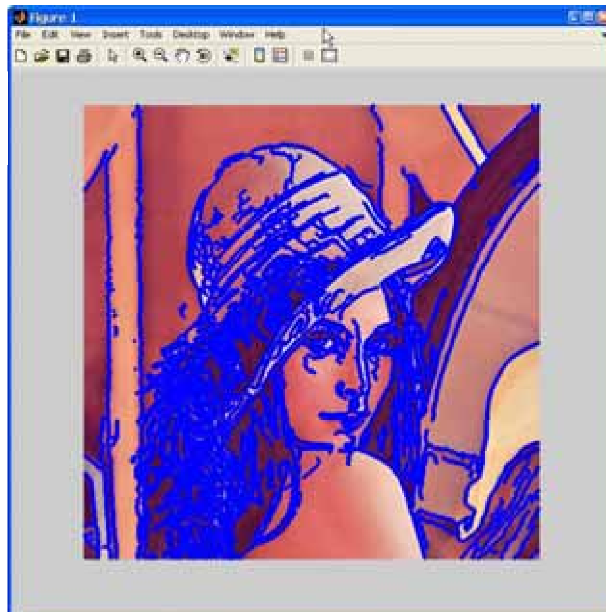
Working with images - example

```
% read input image
I = imread('lena.jpg');

% convert it to grayscale
Ig = RGB2gray(I);

%extract edge with Canny
BW = edge(Ig,'canny');

% bw2 contain black and white mask, white for edge pixel
imshow(BW);

% store in a matrix the index row-column of all edges
edges = [];

for i=1:size(BW, 1)
    for j=1:size(BW, 2)
        if BW(i,j)
            edges = [edges; i, j];
        end
    end
end

% close all active graphs
close all

%display image with edges in blue
imshow(I);
hold on;
plot(edges(:,2), edges(:,1), '.b');
```

```
% bw2 contain black and white mask, white for edge pixel
imshow(BW);
```



```
%display image with edges in blue
imshow(I);
hold on;
plot(edges(:,2), edges(:,1), '.b');
```



Fundamental matrix: example

$$\mathrm{x}'^{\mathrm{T}}\,\mathrm{F}\mathrm{x} = 0$$

$$x'xf_{11} + x'yf_{12} + x'f_{13} + y'xf_{21} + y'yf_{22} + y'f_{23} + xf_{31} + yf_{32} + f_{33} = 0$$

separate known from unknown

$$[x'x, x'y, x', y'x, y'y, y', x, y, 1][f_{11}, f_{12}, f_{13}, f_{21}, f_{22}, f_{23}, f_{31}, f_{32}, f_{33}]^{\mathrm{T}} = 0$$

(data) (unknowns)

(linear)

$$\begin{bmatrix} x'_1 x_1 & x'_1 y_1 & x'_1 & y'_1 x_1 & y'_1 y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_n x_n & x'_n y_n & x'_n & y'_n x_n & y'_n y_n & y'_n & x_n & y_n & 1 \end{bmatrix} \mathrm{f} = 0$$

$$\mathrm{Af} = 0$$

Fundamental matrix: example

$$\min_{\|f\|=1} \|Af\|^2 = 0$$

since $\|Af\|^2 = f^T A^T A f$

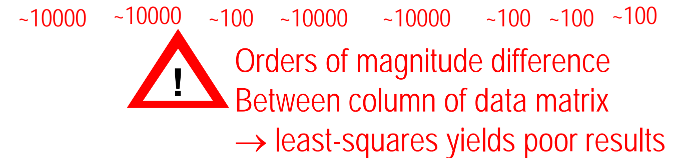
this requires to find the eigenvector associated to the smallest eigenvalue of $A^T A_{9x9}$ (e.g. svd)

Due to data noise the constraint rank(F)=2 is not enforced

Use svd on F and set the smallest singular value to 0

$svd(F) = Q\,D\,R$ (D diagonal mat, Q and R orthogonal)

Fundamental matrix: example



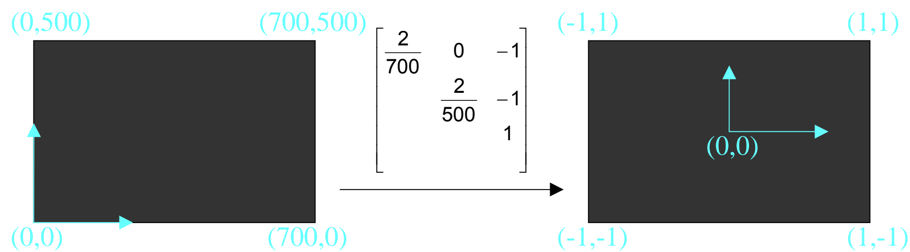
$$\begin{bmatrix} x_1x_1' & y_1x_1' & x_1' & x_1y_1' & y_1y_1' & y_1' & x_1 & y_1 & 1 \\ x_2x_2' & y_2x_2' & x_2' & x_2y_2' & y_2y_2' & y_2' & x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_nx_n' & y_nx_n' & x_n' & x_ny_n' & y_ny_n' & y_n' & x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = 0$$

~10000 ~10000 ~100 ~10000 ~10000 ~100 ~100 ~100 1

Orders of magnitude difference
Between column of data matrix
→ least-squares yields poor results

Fundamental matrix: example

Transform image to ~[-1,1]x[-1,1]



$$\begin{bmatrix} \frac{2}{700} & 0 & -1 \\ & \frac{2}{500} & -1 \\ & & 1 \end{bmatrix}$$

Least squares yields good results +Kduwh|/SDPIE<:,