

Introduction to Matlab

Multimedial Signal Processing 1st Module

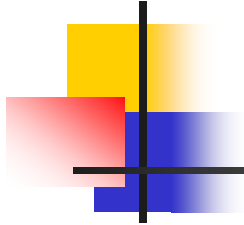
Politecnico di Milano –
Polo regionale di Como

Particulars



- Eliana Frigerio
 - efrigerio@elet.polimi.it
 - Phone: 02 2399 9653
 - Send to me an email in order to organize conferences and for any question

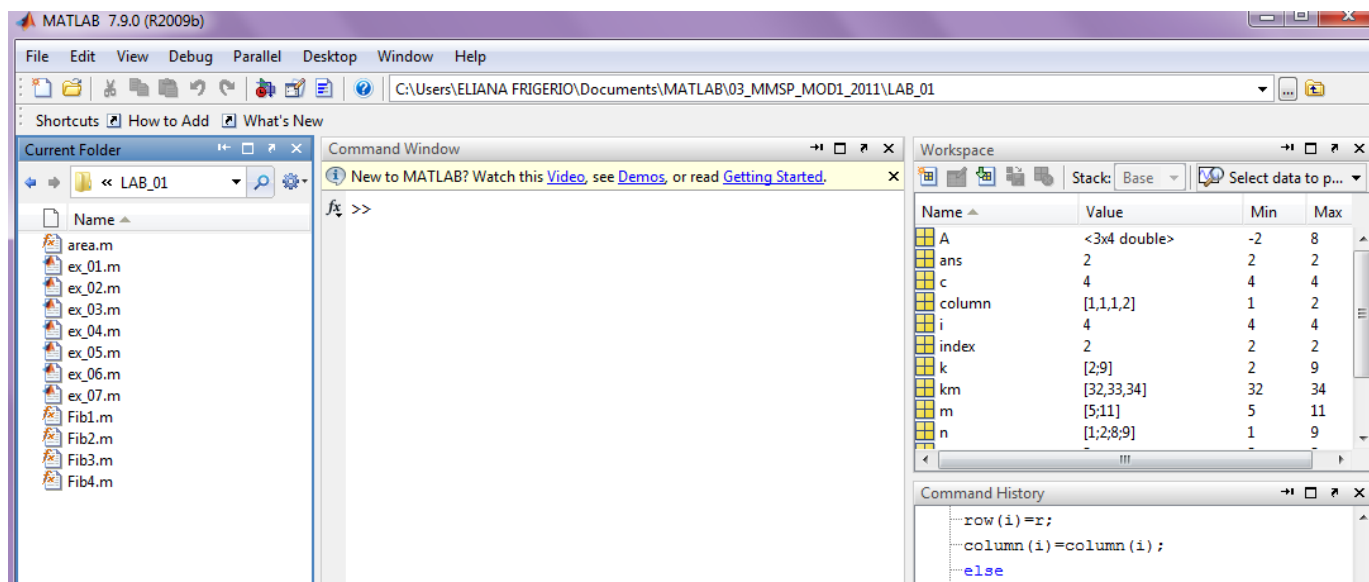
Summary:



- Introduction
- Variables
- Vectors and operations on vectors
- Matrices
- System of linear equations
- Functions
- Plotting
- Loops and logical tests

Introduction

- MATrix LABoratory is a simulation tool for generating and processing signals.
- MATLAB includes functions for compute, store and visualize signals, using a programming language oriented to mathematic objects manipulation.





Introduction

- Digit 'help' on the command prompt for a list of topics
 - Then to obtain help on "Elementary math functions ", for instance, digit: 'help elfun'
 - You can also build your own set of functions for a particular application
- A comprehensive set of demos is available by typing the command 'demo'.
- In addition to the on-line help facility, there is a hypertext browsing system giving details of (most) commands and some examples. This is accessed by "doc".



Variables

■ Ex:

■ `>> 3-2^4` `ans = -13`

■ `>> ans*5` `ans = -65`

- The result of the first calculation is labeled 'ans' by Matlab and is used in the second calculation where its value is changed.

- We can use our own names to store numbers:

■ `>> x = 3-2^4` `x = -13`

■ `>> y = x*5` `y = -65`

- In MATLAB is not required to declare the variables and their size before they are used. The variables are identified by alphanumeric names of any length, and are case-sensitive.



Whos

- Matlab's function "whos" gives which variables are stored in memory and their dimensions:

```
>> whos
```

| Name | Size | Bytes | Class | Attributes |
|------|------|-------|--------|------------|
| D | 3x3 | 72 | double | |
| F | 3x4 | 96 | double | |
| I | 3x3 | 72 | double | |
| P | 2x3 | 48 | double | |
| S | 3x3 | 72 | double | |
| St | 3x3 | 72 | double | |
| Z | 2x3 | 48 | double | |
| ans | 3x1 | 24 | double | |
| d | 1x3 | 24 | double | |
| x | 3x1 | 24 | double | |



Rounding values

- There are a variety of ways of rounding and chopping real numbers to give integers:

| command | values |
|----------------------------|---|
| <code>x = pi*(-1:3)</code> | <code>x =</code> -3.1416 0 3.1416 6.2832 9.4248 |
| <code>round(x)</code> | <code>ans =</code> -3 0 3 6 9 |
| <code>fix(x)</code> | <code>ans =</code> -3 0 3 6 9 |
| <code>floor(x)</code> | <code>ans =</code> -4 0 3 6 9 |
| <code>ceil(x)</code> | <code>ans =</code> -3 0 4 7 10 |
| <code>sign(x)</code> | <code>ans =</code> -1 0 1 1 1 |
| <code>rem(x,3)</code> | <code>ans =</code> -0.1416 0 0.1416 0.2832 0.4248 |



Vectors

- Row vectors: list of numbers separated by either spaces or commas.

- `>> v = [1 3, sqrt(5)]` \rightarrow `v = 1.0000 3.0000 2.2361`

- The number of entries is known as the “length” of the vector.

- `>> length(v)` \rightarrow `ans = 3`

- We can build row vectors from existing ones

- `>> a = [1 2 3], b = [8 9]`

- `>> c = [2*a,-b]` \rightarrow `c = 2 4 6 -8 -9`

- We can change the look at the value of particular entries

- `>> a(2)=-2` \rightarrow `a = 1 -2 3`



Vectors

- `a : b : c` produces a vector of entries starting with the value `a`, incrementing by the value `b` until it gets to `c` (it will not produce a value beyond `c`).

■ `>> -10:3:10` \rightarrow -10 -7 -4 -1 2 5 8

- Column vectors: entries are separated by semicolon `;` or “new lines”

■ `>> v = [1; 3
 sqrt(5)]` \rightarrow `v =`

| | |
|--|--------|
| | 1.0000 |
| | 3.0000 |
| | 2.2361 |



Transpose

- Transposing a vector changes it from a row to a column vector and viceversa.

- `>> v'` \rightarrow ans = 1.0000 3.0000 2.2361

- If x is a complex vector, then x' gives the complex conjugate transpose of x:

- `>> x = [1+3i, 2-2i]`

- `>> x'` \rightarrow ans = 1.0000 - 3.0000i
2.0000 + 2.0000i

- To obtain the plain transpose of a complex number use `.'`

- `>> x.'` \rightarrow ans = 1.0000 + 3.0000i
2.0000 - 2.0000i



Operations on vectors

- Scalar product: is defined by multiplying the corresponding elements together and adding the results to give a single number (scalar).
 - `>> u = [10, -11, 12],` % row vector
 - `>> v = [20; -21; -22]` % column vector
 - `>> prod = u*v` % row times column vector
 - `>> [sqrt(u*u'), norm(u)]` → `ans = 19.1050 19.1050`
- Dot product: product component by component between vectors of the same type
 - `>> u.*v'` → `ans = 200 231 -264`
- Dot power:
 - `>> u.^2` → `ans = 100 121 144`



Operations on vectors

- Dot division: ./ is defined to give element by element division

- `>> a = 1:5, b = 6:10, a./b`

- `ans = 0.1667 0.2857 0.3750 0.4444 0.5000`

- Estimate the limit: $\lim_{x \rightarrow 0} \frac{\sin \pi x}{x}$

- `>> x = [0.1; 0.01; 0.001; 0.0001]`

- `>> sin(pi*x)./x`

HINT.: The idea is to observe the behavior of the ratio for a sequence of values of x that approach zero.



Matrices

- Row and Column vectors are special cases of matrices.
- An $m \times n$ matrix is a rectangular array of numbers having m rows and n columns.

- ```
>> A = [5 7 9
 1 -3 -7]
```

- We can get the size (dimensions) of a matrix with the command `size`:

- ```
>> size(A)
```

 →

```
ans = 2 3
```

- Transposing a matrix interchanges rows with the corresponding columns: the 1st row becomes the 1st column, and so on.

- ```
>> size(A')
```

 → 

```
ans = 3 2
```



# Special Matrices (Ex2)

---

- Matlab provides a number of useful built-in matrices:
- “ones(m,n)” gives a  $m \times n$  matrix of 1's.
- “zeros(m,n)” gives a  $m \times n$  matrix of 0's.
- “eye(n)” gives a  $n \times n$  identity matrix.
- “diag(d,k)” function:
  - If  $d$  is a vector with  $N$  components is a square matrix with the elements of  $V$  on the  $K$ -th diagonal.  $K = 0$  is the main diagonal,  $K > 0$  is above the main diagonal and  $K < 0$  is below the main diagonal.
  - If when  $d$  is a matrix is a column vector formed from the elements of the  $K$ -th diagonal of  $d$ .



# System of linear equations

---

- A general system of linear equations can be expressed in terms of a coefficient matrix  $A$ , a right-hand-side (column) vector  $b$  and an unknown (column) vector  $x$  as:

$$Ax = b$$

- When  $A$  is non-singular and square, the system has a unique solution given by

$$x = A^{-1}b$$

- The standard Matlab routine for solving systems of linear equations is invoked by calling the matrix left division routine,

- `>> x = A \ b`





# mldivide

---

- `mldivide(A,b)` and the equivalent `A\b` perform matrix left division. `A` and `b` must be matrices that have the same number of rows.
- If `A` is a square matrix, `A\b` is roughly the same as `inv(A)*b`, except it is computed in a different way.
- If `A` is an  $m$ -by- $n$  matrix with  $m \approx n$  and `b` is a column vector with  $m$  components, then `x = A\b` is the solution in the least squares sense to the under- or over-determined system of equations  $Ax = b$ . In other words, `x` minimizes  $\text{norm}(A*x - b)$ .



# Functions

---

- Matlab has some functions defined that can recall just using their name and specifying the input and output variables' names.
  - `>> x = 9;`
  - `>> y = sqrt(x)`                       $\rightarrow$                       `y = 3`
- HINT: If you do not want to see the result of intermediate calculations, terminate the assignment statement or expression with semi colon.
- New functions can be defined by the user (see next).



## Find (Ex.7)

---

- The function “find” returns a list of the positions (indices) of the elements of a matrix satisfying a given condition.
- “find” first reshapes the matrix into a column vector; this is equivalent to numbering the elements by columns.
- HINT.: Given a 3x2 matrix it is equivalent to:

1 4 7 10

2 5 8 11

3 6 9 12



# Functions m-files (Ex6)

---

- The main steps to follow when defining a Matlab function are:
  - Decide on a name for the function, making sure that it does not conflict with a name that is already used by Matlab.
  - The first line of the file must have the format:  
`function [list of outputs] = function name(list of inputs)`
  - Document the function: describe briefly the purpose of the function and how it can be used. These lines should be preceded by % which signify that they are comment lines.
  - Finally include the code that defines the function.
- **WARNING:** user-defined functions to be performed must be in the current directory.



# Plotting (Ex1)

---

- Suppose we wish to plot a graph of:  
 $y = \sin(3x)$  for  $0 \leq x \leq 1$ .
- In Matlab can be represented only sequences (discrete signals), as vectors.
- Continuous signals cannot be represented in the strict sense (infinite values).
- This class of signals will then be "approximated" discretizing the independent variable, with a pitch small enough (with a sufficient number of points).

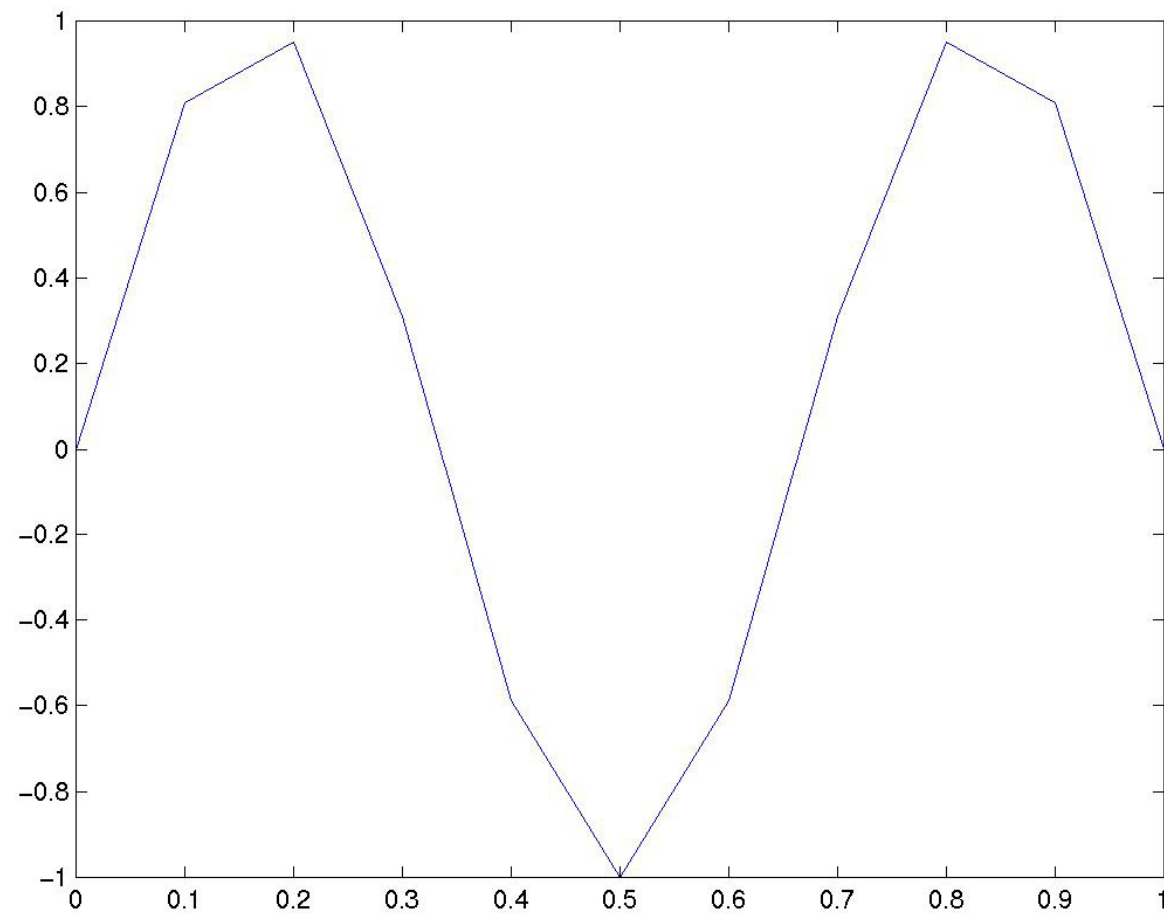


# Plotting

---

- Suppose we take  $N+1$  points equally spaced an arbitrary distance  $h$ :
  - `>> N = 10; h = 1/N; x = 0:h:1;`
  - defines the set of points  $x = 0, h, 2h, \dots, 1-h, 1$ .
- Alternatively, we may use the command “linspace”:
  - `>> x = linspace (0,1,11);`
- The corresponding  $y$  values are computed by
  - `>> y = sin(3*pi*x);`
- and finally, we can plot the points with
  - `>> plot(x,y)`

# Plotting





# Plotting

---

- The value of  $N$  is too small.
  - `>> N = 100; h = 1/N; x = 0:h:1;`
  - `>> y = sin(3*pi*x); plot(x,y)`
- To put a title and label the axes, we use
  - `>> title('Graph of y = sin(3pi x)')`
  - `>> xlabel('x axis'),                      >> ylabel('y axis')`
- A dotted grid may be added by: `>> grid`

This can be removed using either `grid` again, or `grid off`.

- Once a plot has been created in the graphics window you may wish to change the range of  $x$  and  $y$  values shown on the picture.
  - `>> axis([-0.5 1.5 -1.2 1.2])`





# Plotting

- The third argument is a string whose first character specifies the colour (optional) and the second the line style. The options for colours and styles are:

- `>> plot(x,y,'w-')`

- Use “help plot” to obtain a full list.

| Colours |         | Line Styles |         |
|---------|---------|-------------|---------|
| y       | yellow  | .           | point   |
| m       | magenta | o           | circle  |
| c       | cyan    | x           | x-mark  |
| r       | red     | +           | plus    |
| g       | green   | -           | solid   |
| b       | blue    | *           | star    |
| w       | white   | :           | dotted  |
| k       | black   | -.          | dashdot |
|         |         | --          | dashed  |



# Plotting

---

- The command `clf` clears the current figure.
- `Close 1` will close the window labelled "Figure 1".
- To open a new figure window type `figure` or, to get a window labelled "Figure 9", for instance, type `figure (9)`.

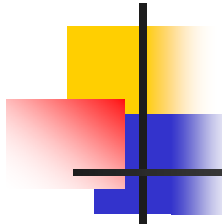
If "Figure 9" already exists, this command will bring this window to the foreground and the result subsequent plotting commands will be drawn on it.



# Multi - plotting

---

- Several graphs may be drawn on the same figure as in
  - `>> plot(x,y,'w-',x,cos(3*pi*x),'g--')`
- A descriptive legend may be included with
  - `>> legend('Sin curve','Cos curve')`
- A call to plot clears the graphics window before plotting the current graph. This is not convenient if we wish to add further graphics to the figure at some later stage. To stop the window being cleared:
  - `>> plot(x,y,'w-'), hold on`
  - `>> plot(x,y,'gx'), hold off`



## Sub - plot

---

- The graphics window may be split into an  $m \times n$  array of smaller windows into which we may plot one or more graphs.
- The windows are counted 1 to  $mn$  row-wise, starting from the top left.
- Both hold and grid work on the current subplot.
  - `>> subplot(221), plot(x,y), xlabel('x'), ylabel('sin 3 pi x')`
  - `>> subplot(222), plot(x,cos(3*pi*x)), xlabel('x'),ylabel('cos 3 pi x')`
  - `>> subplot(223), plot(x,sin(6*pi*x)), xlabel('x'),ylabel('sin 6 pi x')`
  - `>> subplot(224), plot(x,cos(6*pi*x)), xlabel('x'),ylabel('cos 6 pi x')`



# Plotting\_complex sequences

---

- Considering the signal:
- `>> t=0:.001:1;` % temporal axis [sec]
- `>> f=4;` % frequency [Hz]
- `>> s=exp(i*2*pi*f*t);` % complex exponential
- `>> plot(t,real(s),t,imag(s))`
- `>> comet(real(s),imag(s))`
- `>> plot3(t,real(s),imag(s))`
- `>> comet3(t,real(s),imag(s))`
- `>> subplot(2,1,1), plot(t, abs(s));`
- `>> subplot(2,1,2), plot(t, angle(s));`



# Generalized sinusoids (Ex8)

---

- Considering the signal:
- `>> t=0:.001:1;` `% temporal axis [sec]`
- `>> fi=pi/10;` `% fi = 2*pi*f`
- `>> s=ro^t*exp(i*fi*t);` `% generalized sinusoid`
- `>> comet3(t,real(s),imag(s))`
- `>> subplot(2,1,1), plot(t, abs(s));`
- `>> subplot(2,1,2), plot(t, angle(s));`
- See how it changes varying fi or ro



## Loops – for (Ex3)

---

- There are occasions that we want to repeat a segment of code a number of different times
  - `>> x = -1:.05:1;`
  - `>> for n = 1:8`
  - `subplot(4,2,n), plot(x,sin(n*pi*x))`
  - `end`
- All the commands between the lines starting “for” and “end” are repeated with n being given the value 1 the first time through, 2 the second time, and so forth, until n = 8. The subplot constructs a 4 x 2 array of subwindows and, on the nth time through the loop, a picture is drawn in the nth subwindow.



## Logicals (Ex4)

---

- Matlab represents true and false by means of the integers 0 and 1.
  - true = 1, false = 0
- If at some point in a calculation a scalar x has been assigned a value, we may make certain logical tests on it:
  - $x == (\sim =) 2$  is x (not) equal to 2?
  - $x > (>=) 2$  is x greater than (or equal to) 2?
  - $x < (<=) 2$  is x less than (or equal to) 2?
- We may combine logical tests using
  - & represents and
  - the vertical bar | means or
  - ~ means not





## Loops – while (Ex5)

---

- There are some occasions when we want to repeat a section of code until some logical condition is satisfied, but we cannot tell in advance how many times we have to go around the loop.
- Ex.5 What is the greatest value of  $n$  that can be used in the sum:  $1^2 + 2^2 + \dots + n^2$  and get a value of less than 100?

```
while a logical test
```

*Commands to be executed  
when the condition is true*

```
end
```



## If – then – else - end (Ex7)

---

- This allows us to execute different commands depending on the truth or falsity of some logical tests.
- The general form of if statement is

```
if logical test 1
 Commands to be executed if test 1 is
 true
elseif logical test 2
 Commands to be executed if test 2 is
 true but test 1 is false
 :
end
```