Laboratorio per l'Elaborazione MultiMediale
Lezione 1 -
Introduzione a MATLAB

Prof. Michele Scarpiniti

Dipartimento di Ingegneria dell'Informazione, Elettronica e Telecomunicazioni
"Sapienza" Università di Roma

http://ispac.diet.uniroma1.it/scarpiniti/index.htm
michele.scarpiniti@uniroma1.it

Matlab®

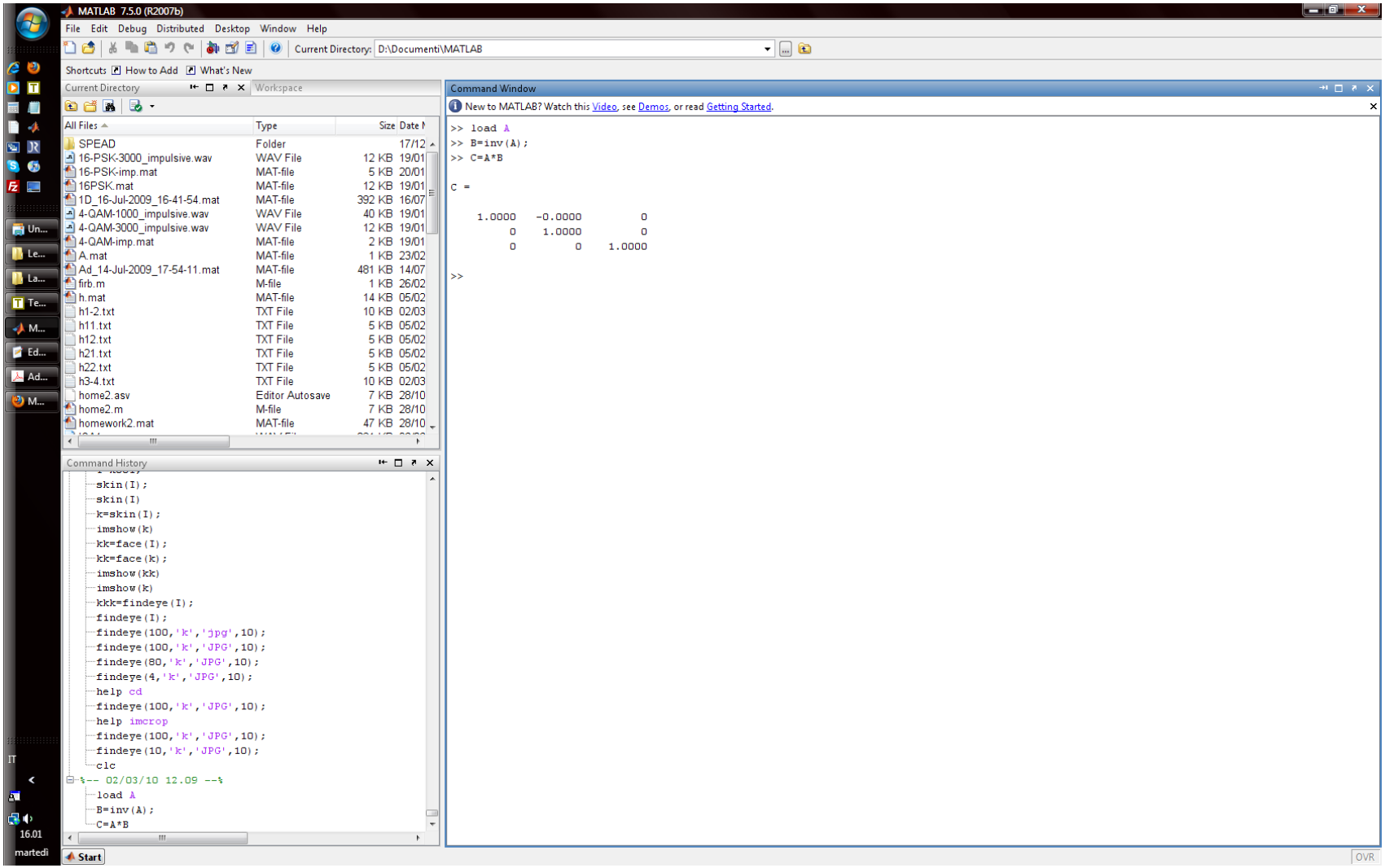
MATLAB®

L'ambiente Matlab®

Il **Matlab®** è un ambiente a linea di comando per il calcolo numerico e vettoriale. L'interfaccia principale di MATLAB è composta da diverse finestre che è possibile affiancare, spostare, ridurre a icona, ridimensionare e così via. Le finestre principali, più usate, sono quattro:

1. **Command Window**: è una finestra dell'interfaccia principale di MATLAB, nella quale è possibile digitare comandi supportati, e visualizzare a video in tempo reale i risultati.
2. **Workspace**: è lo spazio di memoria contente le variabili dichiarate.
3. **Current directory**: permette di esplorare il contenuto delle cartelle sul prorpio hard disk. Da questa finestra è possibile aprire direttamente file compatibili con MATLAB con un semplice doppio click.
4. **Command history**: sono elencati tutti i comandi digitati di recente, divisi per ora e data. È possibile rilanciare direttamente da Command History, un comando digitato in Command Windows in precedenza, semplicemente con un doppio click.

Matlab: l'interfaccia principale



Matlab: la linea di comando

La linea di comando di MATLAB è indicata da un **prompt** come in **DOS**:

```
>>
```

Accetta dichiarazioni di variabili, espressioni e chiamate a tutte le funzioni disponibili nel programma.

```
>> t=linspace(-10,10,1000);
>> y=3*sin(2*pi*t);
>> plot(t,y);
```

Tutte le funzioni di **MATLAB®** non sono altro che file di testo, simili a quelli che l'utente può generare con un editor di testo, e vengono eseguite semplicemente digitandone il nome sulla linea di comando.

MATLAB® permette inoltre di richiamare le ultime righe di comandi inseriti usando le frecce in alto e in basso.

Matlab: la linea di comando

MATLAB® presenta un **help** in linea con informazioni sulla sintassi di tutte le funzioni disponibili, basta digitare:

```
>> help nome_funzione
```

E' anche possibile consultare un help molto dettagliato, accessibile dal **menù Help** oppure digitando:

```
>> doc nome_funzione
```

Per cancellare la **Command Window** è possibile utilizzare l'istruzione:

```
>> clc
```

Matlab: le istruzioni

Le istruzioni (siano esse contenute in un file *.m lanciato da **MATLAB®**, oppure digitate direttamente dalla linea di comando) vanno sempre terminate con un punto e virgola (**;**), altrimenti è visualizzato il risultato dell'applicazione dell'istruzione.

```
>> A=rand(3);
>> B=rand(3)
B =

    0.8147    0.9134    0.2785
    0.9058    0.6324    0.5469
    0.1270    0.0975    0.9575

>>
```

Le variabili seguono le regole dei linguaggi di programmazione come il Pascal o il C. **MATLAB®** è **case-sensitive** e accetta nomi di variabili lunghi fino ad un massimo di 19 caratteri alfanumerici, con il primo obbligatoriamente alfabetico.

Matlab: le istruzioni

Per visualizzare il contenuto di una variabile è sufficiente digitarne il nome senza punto e virgola sulla linea di comando.

```
>> B

B =

    0.8147    0.9134    0.2785
    0.9058    0.6324    0.5469
    0.1270    0.0975    0.9575
```

Il risultato dell'ultima operazione, senza assegnazione, è memorizzato nella variabile `ans`.

Matlab: le istruzioni

Tutti i calcoli effettuati in **MATLAB®** sono eseguiti in **doppia precisione**, ma si possono visualizzare in un formato diverso usando i comandi:

- Virgola fissa con 4 decimali:

```
format short
```

- Virgola fissa con 15 decimali:

```
format long
```

- Notazione scientifica 4 decimali:

```
format short e
```

- Notazione scientifica 15 decimali:

```
format long e
```

Matlab: i comandi

I **comandi** di uso generale sono:

- `who`: elenco delle variabili definite in memoria;
- `whos`: informazioni su tutte le variabili in memoria;
- `clear`: cancella tutte le variabili in memoria o una in particolare se specificata;
- `save`: salva tutte le variabili in memoria sul file specificato, in vari formati;
- `load`: richiama in memoria le variabili salvate sul file specificato;
- `diary`: salva nel file di testo ASCII denominato diary, quanto da quel momento appare sulla linea di comando;
- `what`: elenco di tutte le funzioni MATLAB nell'area di lavoro (estensione *.m) e dei file di dati che sono stati salvati (estensione *.mat).

Matlab: gli operatori scalari

Gli **operatori** disponibili sono:

- $+$, $-$, $*$, $/$, $\hat{}$;
- *sin*, *cos*, *tan*;
- *asin*, *acos*, *atan*;
- *exp*, *log* (*naturale*), *log*10 (*in base 10*);
- *abs*, *sqrt*, *sign*.

Matlab: i numeri complessi

L'*unità complessa* è i o j (oppure $1j$ o $1i$) ed è predefinita, quindi **NON** usare i o j come variabili o indici nei cicli.

Un numero complesso si scrive nella forma $a + j * b$:

```
>> 3+j*2
```

Gli **operatori** applicabili a *numeri complessi* sono:

- *abs*: modulo, es. $abs(z)$;
- *angle*: fase, es. $angle(z)$;
- *real*: parte reale, es. $real(z)$;
- *imag*: parte immaginaria, es. $imag(z)$.

Matlab: matrici e vettori

L'inserimento di un **vettore** o di una **matrice** in generale viene effettuato tra parentesi quadre, separando gli elementi delle righe con spazi o virgole, e le diverse righe con punti e virgola (oppure andando a capo ad ogni nuova riga).

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

L'operazione di **trasposta** è eseguita con il comando .', mentre l'**hermitiano** con '

```
>> y.'
ans =
     1   2   3
```

Matlab: matrici e vettori

Per far riferimento agli elementi di una matrice **A**:

- l'**elemento** a_{mn} è indirizzato come $A(m, n)$:

```
>> A(2,3)
```

- la **riga** m-esima è indirizzata come $A(m, :)$:

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

- la **colonna** n-esima è indirizzata come $A(:, n)$:

```
>> A(:,3)
```

- la **sotto-matrice** avente elementi a_{mn}, con $m_1 \leq m \leq m_2$ e $n_1 \leq n \leq n_2$, è indirizzata come $A(m1 : m2, n1 : n2)$:

```
>> A(1:2,2:3)
```

Matlab: matrici e vettori

Gli **operatori** applicabili a matrici sono:

- $+$ **somma** tra matrici:

```
>> A+B
```

- $-$ **differenza** tra matrici:

```
>> A-B
```

- $*$ **prodotto** tra matrici:

```
>> A*B
```

- ^ **potenza** di una matrice:

```
>> A^2
```

- $.'$ **trasposta** di una matrice:

```
>> A.'
```

- $'$ **hermitiano** di una matrice:

```
>> A'
```

- / **divisione a destra** di due matrici:

```
>> A/B = A*inv(B)
```

- \ **divisione a sinistra** di due matrici:

```
>> A\B = inv(A)*B
```

Matlab: matrici e vettori

Altre **funzioni** operanti essenzialmente su vettori (*riga* o *colonna*) sono:

- `max`, restituisce il **massimo** (e la sua posizione) di un vettore;
- `min`, restituisce il **minimo** (e la sua posizione) di un vettore;
- `sort`, **ordina** gli elementi di un vettore (crescente o decrescente);
- `sum`, restituisce la **somma** di tutti gli elementi di un vettore;
- `prod`, restituisce il **prodotto** di tutti gli elementi di un vettore;
- `mean`, restituisce la **media** degli elementi di un vettore;
- `median`, restituisce la **mediana** degli elementi di un vettore;
- `var`, restituisce la **varianza** degli elementi di un vettore;
- `std`, restituisce la **deviazione standard** degli elementi di un vettore.

Matlab: matrici e vettori

Esistono poi **particolari operatori** che permettono di effettuare operazioni su vettori elemento per elemento, senza ricorrere a cicli:

- .∗ moltiplicazione elemento per elemento;
- ./ divisione elemento per elemento;
- .^ elevamento a potenza elemento per elemento.

Ad esempio

```
>> x=[1 2 3];
>> y=[4 5 6];
>> x.*y
ans =
     4    10    18
>> x*y.'
ans =
    32
>> x.'*y
ans =
     4     5     6
     8    10    12
    12    15    18
```

Matlab: matrici e vettori

Altre **funzioni** che operano invece essenzialmente su matrici sono:

- `inv`: **inversa** di una matrice;
- `det`: **determinante** di una matrice;
- `size`: **dimensioni** di una matrice;
- `rank`: **rango** di una matrice:
- `eig`: decomposizione **autovalori-autovettori**;
- `svd`: decomposizioni in **valori singolari-vettori singolari**.

Attenzione

tutte le funzioni che operano su matrici hanno dei vincoli sugli operandi introdotti. Ad esempio non si può invertire una matrice non quadrata. Per ulteriori spiegazioni sulla sintassi della funzione utilizzare il comando help.

Matlab: matrici e vettori

La funzione `eig` opera su matrici quadrate nel modo seguente:

- `y=eig(A);` produce un vettore *y* contenente gli autovalori della matrice **A**.
- `[U,D]=eig(A);` produce una matrice **U** avente per colonne gli autovettori della matrice **A**, ed una matrice **D** diagonale avente sulla stessa gli autovalori della matrice **A**.

```
>> A
A =

    5.0000    3.0000    1.0000
    2.0000    4.0000    0.5000
    3.0000    3.0000    1.0000

>> y=eig(A)
y =
    7.6742
    2.0000
    0.3258
```

```
>> [U,D]=eig(A)
U =
   -0.7138   -0.7071   -0.1909
   -0.4604    0.7071   -0.0296
   -0.5278   -0.0000    0.9812
D =
    7.6742         0         0
         0    2.0000         0
         0         0    0.3258
```

Matlab: matrici e vettori

Esistono poi varie **funzioni** predefinite per la creazione di matrici:

- `eye(n)`: matrice **identità** n righe n colonne;
- `zeros(m,n)`: matrice di 0 con m righe e n colonne;
- `ones(m,n)`: matrice di 1 con m righe e n colonne;
- `rand(m,n)`: matrice di **elementi random** con valori compresi tra 0 e 1;
- `zeros(n)`, `ones(n)`, `rand(n)`: sono equivalenti a `zeros(n,n)`, `ones(n,n)`, `rand(n,n)`;
- `diag(X)`: se `X` è un vettore con n elementi, produce una matrice quadrata diagonale di dimensione $n \times n$ con gli elementi di `X` sulla diagonale. Se invece `X` è una matrice quadrata di dimensione $n \times n$, produce un vettore di n elementi pari a quelli sulla diagonale di X.

Matlab: matrici e vettori

Il **comando** : (*due punti*) può essere usato per generare vettori:

- *senza* specificare *incremento*

```
>> t=1:5
t =
     1     2     3     4     5
```

- con *incremento positivo* specificato

```
>> t=0:0.2:1
t =
         0    0.2    0.4    0.6    0.8    1
```

- con *incremento negativo* specificato

```
>> t=2:-0.2:1
t =
    2    1.8    1.6    1.4    1.2    1
```

Matlab: polinomi

MATLAB®tratta i **polinomi** come particolari vettori riga, i cui elementi sono i coefficienti dei monomi del polinomio in ordine di potenza decrescente.
Es. il polinomio

$$s^4 + 2s^3 + 3s^2 + 5s + 4$$

viene rappresentato come:

```
>> num=[1   2   3   5   4];
```

La funzione `conv(x,y)` moltiplica due vettori x e y, e quindi due polinomi.
Es. il prodotto tra polinomi:

$$(s^2 + s + 1) * (s^2 + 3s + 2) = s^4 + 4s^3 + 6s^2 + 5s + 2$$

viene effettuato con:

```
prod = conv([1   1   1],[1   3   2]);
```

che dà infatti, come risultato, il vettore

```
prod =
     1     4     6     5     2
```

Matlab: polinomi

- La **funzione** `roots` calcola le **radici** del polinomio.

```
>> p=[1 5 6];
>> r=roots(p)
r =
    -3.0000
    -2.0000
```

  In *r* sono memorizzate le radici del polinomio *p*.

- La *funzione inversa* è la funzione `poly`:

```
>> pp = poly(r);
```

  In *pp* viene ripristinato il polinomio originale *p*.

Matlab: controllo del flusso

Matlab mette a disposizione, come il **C**, alcune strutture di controllo che servono a specificare se, quando, in quale ordine e quante volte devono essere eseguite le istruzioni che lo compongono.

- Istruzione `IF-THEN-ELSE`: la forma generale di tale costrutto è la stessa di un qualsiasi linguaggio di programmazione:

```
if condizione1
    operazioni1;
elseif condizione2,
    operazioni2;
else
    operazioni3;
end;
```

Matlab: controllo del flusso

Le `condizione1` e `condizione2` devono essere condizioni che restituiscono come risultato **VERO** o **FALSO**. Gli operatori disponibili per tali confronti sono:

- $<$, $>$ minore o maggiore;
- $<=$, $>=$ minore uguale o maggiore uguale;
- $==$ uguale;
- $\sim=$ diverso;
- & **and** logico;
- | **or** logico;
- $\sim$ **not** logico.

Matlab: controllo del flusso

- Cicli `FOR`: il ciclo esegue le operazioni incrementando la variabile k da 1 a n con il passo indicato in `step`:

```
for k = 1:step:n
    operazioni;
end;
```

  se `step` è pari a 1, può essere omessa.

- Cicli `WHILE`: il ciclo esegue le operazioni fino a che la condizione è verificata. La condizione viene costruita con le stesse regole (vincoli ed operatori) di quella dell'`IF-THEN-ELSE`:

```
while condizione
      operazioni;
end;
```

Matlab: controllo del flusso

- Istruzione SWITCH-CASE: può essere necessario, nel corso di un programma, variare l'elaborazione in seguito a più condizioni:

```
SWITCH espressione
   CASE valore1,
            operazioni;
   CASE valore2,
            operazioni;
         ...
   OTHERWISE,
            operazioni;
END
```

Matlab: stringhe

Il **testo** in **MATLAB** viene inserito sempre tra apici:

```
>> string = 'Ciao';
```

- Per visualizzare stringhe o messaggi si adopera la funzione `disp`:

```
>> disp('Premere un tasto');
```

- La funzione `error` mostra un messaggio di errore ed interrompe l'esecuzione di un file *.m:

```
>> error('A deve essere simmetrica');
```

- La funzione `input` mostra un messaggio e permette l'inserimento di dati.

```
>> num = input('Inserire il numero di iterazioni: ');
```

- Posso **concatenare** due *stringhe*, trattandole come elementi di un vettore. Ad esempio, date `s1 = 'Hello '` e `s2 = 'world!'`:

```
>> s = [s1 s2]
>> s = Hello world!
```

Matlab: i grafici

La funzione `plot` crea **grafici bidimensionali**: riceve in ingresso due vettori della stessa lunghezza e stampa i punti corrispondenti alle coordinate fornite dai due vettori. Ad esempio se si hanno due vettori x e y, il grafico corrispondente si ottiene come:

```
>> plot(x, y);
```

Per tracciare il grafico di una qualsiasi funzione, è perciò necessario crearsi un opportuno vettore da usare come ascisse, passarlo alla funzione per ricavare un vettore contenente le ordinate, ed usare la funzione `plot` sui due vettori così ottenuti. Ad esempio per tracciare la funzione $\sin(x)$ tra -4 e 4 si può usare la serie di comandi:

```
>> x = -4:0.01:4;
>> y = sin(x);
>> plot(x, y);
```

Matlab: i grafici

Se utilizzo la funzione `plot` con un solo parametro, la scala delle ascisse è creata da Matlab definendola da 1 a N, dove N è il numero di elementi del vettore da disegnare.

Se si usa la funzione `plot` con un solo parametro **complesso**, il grafico rappresenterà la parte reale e la parte immaginaria degli elementi del vettore: ad esempio

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

con y complesso, equivale a:

```
>> plot(real(y), imag(y));
```

Per creare grafici di **colori diversi** o usando **caratteri diversi** dal punto si può specificare dopo le coordinate una stringa di 2 elementi. Il primo è il colore del grafico, il secondo il simbolo usato per contrassegnare i punti. Ad esempio

```
>> plot(x, y, 'g+');
```

crea un grafico in verde usando dei + al posto dei punti.

Matlab: i grafici

L'insieme delle scelte possibili per la stringa di 2 elementi è il seguente:

- **r** red;
- **g** green;
- **b** blue;
- **w** white;
- **m** magenta;
- **c** cyan;
- **y** yellow;
- **k** black.

- **.** point;
- **o** circle;
- **x** x-mark;
- **+** plus;
- ***** star;
- **-** solid;
- **:** dotted;
- **–** dashed;
- **-.** dash-dot.

Matlab: i grafici

Altri **comandi** utili sono:

- `grid` : sovrappone al grafico un grigliato;
- `title` : aggiunge un titolo del disegno;
- `xlabel` : aggiunge una legenda per l'asse x;
- `ylabel` : aggiunge una legenda per l'asse y;
- `axis` : scala gli assi del grafico;
- `clf` : cancella il grafico corrente;
- `zoom` : è possibile effettuare lo zoom con il mouse.

Il comando `figure` crea una nuova finestra grafica in cui far comparire il disegno; per spostarsi sulla n-esima finestra grafica, basta digitare

```
>> figure(n);
```

Matlab: i grafici

Il comando `hold on` permette di **sovrapporre** due o più grafici, mentre il comando `hold off` elimina tale possibilità di sovrapposizione di grafici.

Per visualizzare **più grafici** sulla stessa schermata si può usare la funzione `subplot`. La funzione accetta 3 parametri: il primo indica in quante parti verticali dividere lo schermo, il secondo in quante parti orizzontali, e il terzo in quale parte eseguire il plot successivo. Ad esempio, il codice seguente:

```
>> subplot(211), plot(funz1);
>> subplot(212), plot(funz2);
```

crea due finestre divise da una linea orizzontale, e visualizza in quella alta il grafico di `funz1`, e in quella bassa quello di `funz2`.

Matlab: i grafici

Per creare i *vettori delle ascisse* posso utilizzare due **funzioni**:

- `linspace(a,b,n)`: crea un vettore di n elementi equi-spaziati tra a e b;

```
>> x=linspace(1,100,1000);
```

- `logspace(a,b,n)`: crea il vettore di n elementi separati logaritmicamente tra 10^a e 10^b:

```
>> x=logspace(0,2,1000);
```

Matlab: le funzioni

In **MATLAB** è possibile creare **nuove funzioni**. Basta creare un *file* con estensione *.m e nome del file uguale a quella della funzione desiderata.

La **prima riga** del file deve contenere il nome della funzione e gli argomenti di ingresso e di uscita. Ad esempio, in

```
function z = fun1(a,b)
```

oppure in

```
function [x,y] = fun2(a,b)
```

risulta che `fun1` e `fun2` sono nomi di funzioni; *a* e *b* sono argomenti d'ingresso; *x*, *y* e *z* sono argomenti d'uscita.

Matlab: le funzioni

- Il blocco di **linee di commento** (cioè quelle precedute dal carattere %) consecutive che eventualmente segue la prima linea del file viene visualizzato digitando il comando `help` seguito dal nome della funzione creata.
- Le variabili utilizzate in una funzione sono **locali** e quindi *indipendenti* da quelle dell'ambiente chiamante.
- È possibile utilizzare anche **variabili globali**, a patto che vengano definite come tali sia nell'ambiente chiamante sia nella funzione, utilizzando il comando `global` seguito dai nomi delle variabili, separati da spazi:

```
>> global a b y G
```

Bibliografia


Matlab
MATLAB: Getting Started Guide.
Available on-line:
http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/getstart.pdf


T.A. Davis.
MATLAB Primer.
8-th Edition, CRC Press, 2010.


D.M. Smith.
Engineering Computation with MATLAB.
2-nd Edition, Addison-Wesley, 2010.


A. Gilat.
MATLAB: An Introduction with Applications.
Wiley, 2008.


A. Knight.
Basics of MATLAB and Beyond.
CRC Press, 1999.


A.D. Poularikas.
Signals and Systems Primer with MATLAB.
CRC Press, 2006.