POLITECNICO
MILANO 1863

INTRODUZIONE A MATLAB

LAB #1 – Segnali per le Comunicazioni

Francesco Asaro | francesco.asaro@polimi.it

6 Mar. 2018

UN PO' DI STORIA

- MATLAB (**MAT**rix **LAB**oratory e non **MAT**h **LAB**oratory) è un linguaggio di programmazione interpretato e di alto livello.

- Le radici di MATLAB affondano negli anni cinquanta e sono legate ai lavori di calcolo numerico e algebra lineare di J. Wilkinson. Il vero e proprio MATLAB nasce nel '84 con la Fondazione di The MathWorks da parte di Jack Little e soci. Al tempo MATLAB era basato su due librerie FORTRAN di algebra lineare. Con l'avvento dei Personal Computers MATLAB fu immediatamente riprogrammato in C.
(https://it.mathworks.com/company/newsletters/articles/the-origins-of-matlab.html)

- Rappresenta uno standard nel campo della prototipizzazione, sia per la sua solidità, ma soprattutto grazie alla completezza ottenuta con I' introduzione di numerosi toolboxes.

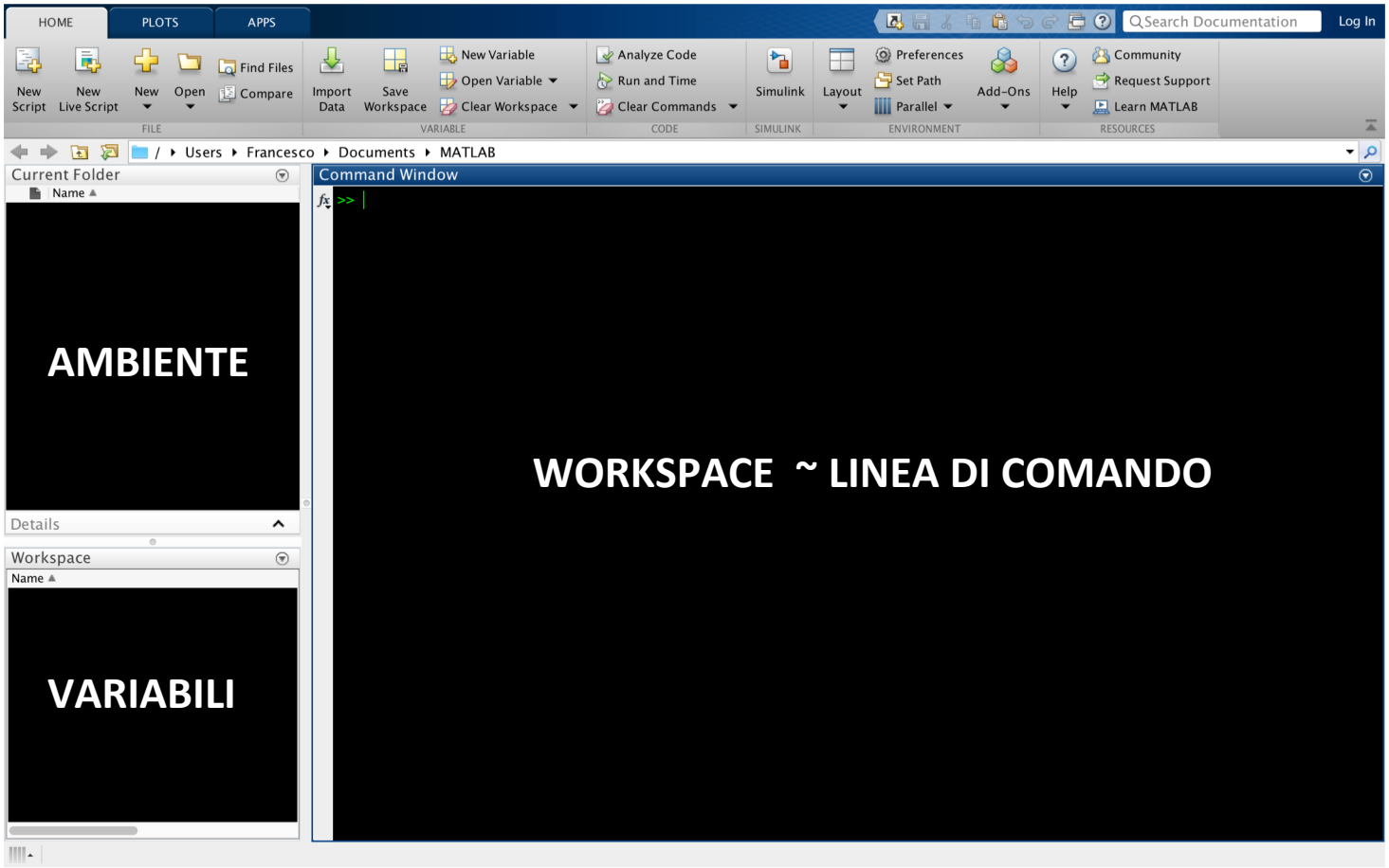
- Negli ultimi anni si sono affiancati a MATLAB altri linguaggi simili quali Python e R, che però sono particolarmente orientati su particolari campi della programmazione.

OTTENERE IL SOFTWARE

- È possibile scaricare MATLAB con licenza studenti PoliMi dalla pagina dedicata:

http://www.software.polimi.it/software-download/studenti/matlab/

INTERFACCIA



DOCUMENTAZIONE

- Uno dei punti di forza di MATLAB è sicuramente la documentazione

OFFLINE

Help di MATLAB (`>>help`)

ONLINE

MATLAB community (https://it.mathworks.com/matlabcentral/)
Stackoverflow https://stackoverflow.com

ARRAY

➔ In MATLAB le variabili possono essere organizzate secondo diverse classi

➔ NUMERIC
- *single*
- *double (default)*
- *int8/16/32*
- *uint8/16/32*

➔CHARACTER
- *string (‘ ‘)*
- *char (“ “)*

➔LOGICAL
- *true*
- *false*

➔CELL

➔STRUCTURE

➔FUNCTION HANDLE

VARIABILI NUMERICHE

- Le variabili numeriche vengono trattate come matrici

```
>>x=2;                              %scalare (1x1)
>>x=[2,2,2];                       %vettore riga (1x3)
>>x=[2 2 2];
>>x=[2; 2; 2];                     %vettore colonna (3x1)
>>x=x';                            %via trasposizione
>>A=[1,1,1; 2,2,2; 3,3,3];        %matrice 3x3
```

ACCESSO ALLE VARIABILI

➔ Le variabili numeriche vengono trattate come matrici

```
>>a=x(2,3);        %accesso all'elemento riga 2 colonna 3
>>a=x(:,3);        %estrazione della terza colonna
>>a=x(3,:);        %estrazione della terza riga
>>a=x(:);          %unpacking
>>a=x(1,end);      %primo elemento dell'ultima colonna
>>a=x(1:3,end);    %elementi dal primo al terzo dell'ultima colonna
>>a=diag(x);       %diagonale della matrice
```

N.B. il primo elemento è indicizzato con 1 e non 0 a differenza di Python
N.B. 2 è possibile assegnare valori in posizioni non esistenti della variabile, che verrà orlata altrove di zeri per essere resa rettangolare

OPERAZIONI

- Le operazioni in MATLAB sono definite con i classici operatori (somma **+**, sottrazione **-**, divisione **/**, elevamento a potenza **^**) e tra scalari risultano banali.

- Le operazioni riguardanti matrici e vettori, invece, sono definite secondo le regole dell'algebra e richiedono compatibilità delle variabili coinvolte. In questo caso gli operatori usati senza alcun accorgimento fanno riferimento all'operazione algebrica del caso e.g. :

```
>>A=rand(3,3);
>>B=rand(3,3);
>>C=A*B; %restituisce il risultato di una moltiplicazione
          %matrice per matrice (righe per colonne)
>>D=A.*B; %restituisce il risultato di una moltiplicazione
          %elemento per elemento delle due matrici
```

DEFINIZIONE DI MATRICI SPECIALI

- Sono ovviamente presenti dei comandi built-in per costruire matrici speciali

```
>>I=eye(3,3);    %matrice identità 3x3
>>A=zeros(3,3);  %matrice nulla 3x3
>>A=ones(3,3);   %matrice ”uno” 3x3
>>A=randn(3,3); %matrice random dist. normale
>>A=randn(3,3); %matrice random dist. uniforme
>>A=diag(v);     %matrice con v su diagonale
```

PARTICOLARI FUNZIONI SU MATRICI

- Ugualmente è possibile trattare in maniera molto rapida le matrici grazie a uno specifico set di funzioni *ad hoc*

```
>>C=A';          %trasposizione
>>C=inv(A);      %inversa
>>C=A\I;         %inversa (algoritmo più efficiente dal punto
                 %di vista computazionale)
>>d=det(A);      %determinante
>>r=rank(A);     %rango
>>[v,D]=eig(A);  %decomposizione autovalori – autovettori
>>c=cond(A);     %condizionamento
```

CICLI DI CONTROLLO

- In MATLAB è ovviamente possibile costruire i classici cicli di controllo ***if/else if*** , **for** e ***while***, il ciclo viene chiuso con la parola chiave ***end***.

- Gli operatori logici necessari per definire le condizioni di controllo sono definiti con la seguente simbologia:

==	UGUALE
~=	DIVERSO
&&	AND
||	OR

ALCUNI ESEMPI - FOR

➔ Somma da 1 a n (stampando i valori intermedi)

```
n=100;
somma=0;
for i=1:n
somma=somma+i
end
```

Ovviamente questo compito può essere assolto più elegantemente da un punto di vista matematicocon la formula di Gauss **(n(n+1))/2** o dal punto di vista computazionale semplicemente con:

```
>>somma=sum (1:100);
```

ALCUNI ESEMPI - IF

➔ *If* e *While* trovano larga applicazione nei criteri di arresto degli algoritmi iterativi

```
theta=1:360;
a=rand(1,360);
s=a.*cos(theta);
tol=0.3;

c=zeros(1,size(s,2)-1);
i=1;
c(i)=abs(s(2)-s(1));

for i=2:size(s,2)-1
    c(i)=    s(i+1)-s(i);
    if abs(c(i))<tol
    fprintf('convergenza ok')
    i
    return
    end
end
```

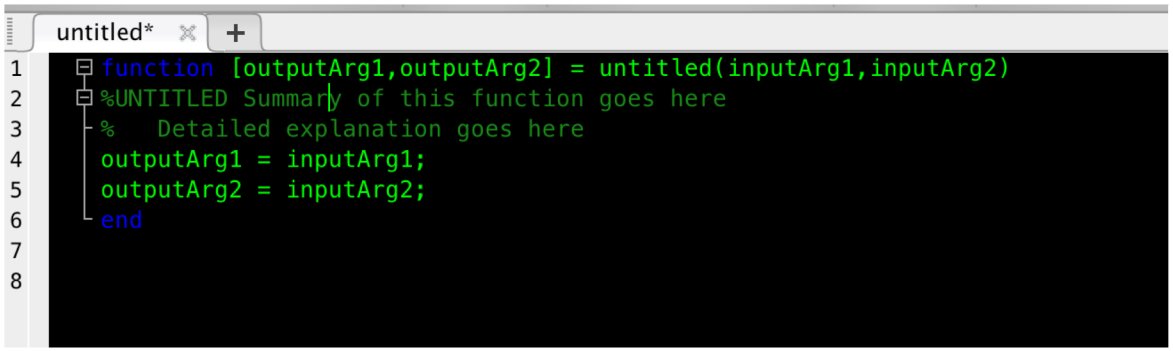
ALCUNI ESEMPI - WHILE

```
theta=1:360;
a=rand(1,360);
s=a.*cos(theta);
tol=0.01;

c=abs(s(2)-s(1));
i=1;
   while abs(c)>tol && i <length(s)
        i=i+1
        c=s(i+1)-s(i);
   end
```

FUNCTIONS E SCRIPTS

- In MATLAB è possibile costruire delle funzioni nell'editor attraverso una sintassi standard



N.B. Il nome della function deve corrispondere al nome del .m file in cui è salvata.

A differenza delle function in cui sono ben definiti un certo numero di input e output è anche possibile realizzare degli script MATLAB, praticamente degli eseguibili.

È solitamente conveniente lavorare nell'editor di testo e compilare il codice all'occorrenza, piuttosto che direttamente nel workspace

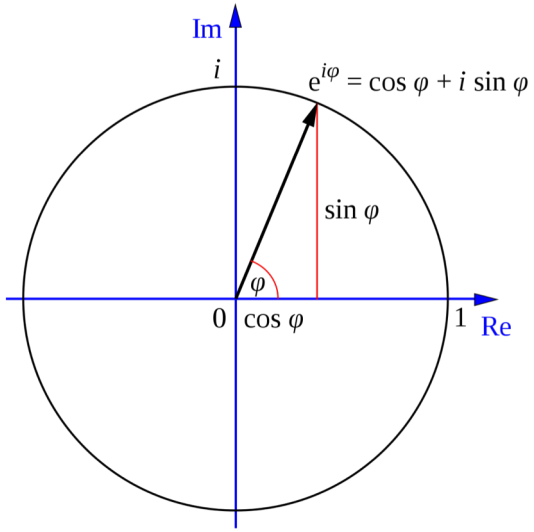
NUMERI COMPLESSI

In MATLAB è possibile gestire al meglio anche i numeri complessi, motivo che ne ha tra l'altro determinato l'affermazione anche nel trattamento dei segnali.

L'unità immaginaria è riconosciuta da MATLAB sia come *i* che *j*, a patto che non siano state assegnate ad una variabile.

N.B. É buona norma quando si lavora nei complessi escludere i e j dalle indicizzazioni (come altrimenti viene fatto molto spesso).

```
>>z=5+i*3;
>>real(z)     %parte reale di z
>>imag(z)     %parte complessa di z
>>angle(z)    %argomento di z
>>abs(z)      %modulo di z
>>conj(z)     %coniugato di z
```



NUMERI COMPLESSI - NOTAZIONE ESPONENZIALE

Grazie all'identità di Eulero è possibile scrivere un numero complesso in forma esponenziale ovvero attraverso una componente in fase (seno) ed una in quadrature (coseno).

Ad esempio creiamo un handle all'identità di Eulero e plottiamo i valori delle due componenti rispetto alla fase:

```
euler=@(A,phi) A*cos(phi)+A*i*sin(phi);
phi=deg2rad(0:360);
signal=euler(10,phi);
figure;
subplot(2,1,1); plot(phi,real(signal))
subplot(2,1,2); plot(phi,imag(signal))
```

N.B. MATLAB lavora in radianti, per utilizzare i gradi sono necessarie le rispettive funzioni di conversione

UN ALTRO ESEMPIO

Estraiamo da una distribuzione normale 100 realizzazioni di ampiezza e da una distribuzione uniforme 100 di fase e ricostruiamo il segnale

```
phi_max=deg2rad(90);
phi= -phi_max + (2*phi_max)*rand(1000,1);
[y,x]=histcounts(phi,100,'BinLimits',[-phi_max,phi_max]);
figure; plot(x(1:end-1),y)

A=randn(1000,1);
A=A+max(A);
[y,x]=histcounts(A,100);
figure; plot(x(1:end-1),y)

euler=@(A,phi) A.*cos(phi)+A.*i.*sin(phi)
signal=euler(A,phi);
Figure; plot(abs(signal))
```

POLITECNICO
MILANO 1863

francesco.asaro@polimi.it

Edificio 20, terzo piano, ufficio 056