Laboratorio di Fondamenti di Segnali e Trasmissione

Marco Ferrari

Introduzione al MATLAB

Per il laboratorio del corso di Fondamenti di Segnali e Trasmissione lavoreremo in MATLAB. Questa introduzione é pensata per chi é completamente "digiuno" di cosa sia MATLAB e come si usi.
MATLAB è un programma, un'interprete di comandi adatto per effettuare operazioni matematiche. Esso integra calcolo, visualizzazione, e una programmazione abbastanza semplice.
Si lancia Start => ... MATLAB
In MATLAB sono predefinite un'infinità di operazioni matematiche che si possono applicare direttamente a numeri, oppure a variabili.

p.e.

```
» 1+1
ans =
        2

» cos(0),
ans =
        1

» sin(0)
ans =
        0
```

Il risultato di un'operazione si può assegnare ad una variabile,
p.e.

```
» a=2*3

a =

    6
```

Le variabili in MATLAB sono matrici di numeri. Anche lo scalare e i vettori si possono vedere come casi particolari di matrici.
Il tipo degli elementi può essere reale (double a 8 byte) o complesso. L'unita' immaginaria si indica con j o i,

p.e.

```
» z=2+2*i

z =

  2.0000 + 2.0000i
```

comando: per vedere l'elenco delle variabili attualmente in memoria **whos**

```
» whos
a    1x1     8  double array
z    1x1    16  double array (complex)
```

Come avrete notato, cosa comoda (anche se per certi aspetti, pericolosa) è il fatto che non è necessario dichiarare le variabili e le loro dimensioni prima di adoperarle. L'assegnazione comporta allocazione (eventualmente anche sovrascrittura), definizione del tipo (reale se gli si assegna un reale, complesso se gli si assegna un complesso), e definizione delle dimensioni della matrice.

p.e.

```
» z=[1, 2]

z =

    1   2
```

soprascrive tipo e dimensioni per la variabile z:

```
» whos
a    1x1     8  double array
z    1x2    16  double array
```

comando: per cancellare tutte le variabili attualmente in memoria **clear**

Costruzione di variabili

Come si fa a costruire una matrice? Il primo modo è quello diretto: tra parentesi quadre, con spazi o virgole che separano elementi di una riga, e ";" che separano le righe

p.e.

```
» A = [11 3 2 13; 5 1 11 8; 9 6 9 12; 2 15 14 1]

A =

   11    3    2   13
    5    1   11    8
    9    6    9   12
    2   15   14    1

» whos
  Name      Size            Bytes  Class

  A         4x4               128  double array

Grand total is 16 elements using 128 bytes
```

La stessa sintassi si può utilizzare per costruire matrici partendo da altre matrici:

p.e.

```
» B=[A A]

B =

   11    3    2   13   11    3    2   13
    5    1   11    8    5    1   11    8
    9    6    9   12    9    6    9   12
    2   15   14    1    2   15   14    1

» C=[A; A]
```

```
C =

   11    3    2   13
    5    1   11    8
    9    6    9   12
    2   15   14    1
   11    3    2   13
    5    1   11    8
    9    6    9   12
    2   15   14    1
```

Naturalmente questo è possibile finchè si cerca di accostare blocchi coerenti: per accosatre due matrici tramite spazio occorre che abbiano lo stesso numero di righe, per cui se ora provate a fare:

```
» D=[B C]
??? All matrices on a row in the bracketed expression must have the
 same number of rows.
```

Il sistema vi segnala appunto questo errore. Analogamente, si possono incolonnare solo matrici con lo stesso numero di colonne:

```
» D=[B; C]
??? All rows in the bracketed expression must have the same
number of columns.
```

Oltre a questi più diretti, ci sono metodi più spicci per costruire le matrici più "comuni". Tra questi vi segnalo le funzioni zeros, ones e eye

```
» z=zeros(4,4)

z =

    0    0    0    0
    0    0    0    0
    0    0    0    0
    0    0    0    0

» o=ones(4,4)

o =

    1    1    1    1
    1    1    1    1
    1    1    1    1
    1    1    1    1

» e=eye(4)

e =

    1    0    0    0
    0    1    0    0
    0    0    1    0
    0    0    0    1
```

Piccola anticipazione: naturalmente è definito il prodotto tra una costante ed una matrice (come prodotto di ogni elemento della matrice per quella costante), per cui se fate

```
» e=2*eye(4)

e =

     2     0     0     0
     0     2     0     0
     0     0     2     0
     0     0     0     2

» o=2.5*ones(4,4)

o =

    2.5000    2.5000    2.5000    2.5000
    2.5000    2.5000    2.5000    2.5000
    2.5000    2.5000    2.5000    2.5000
    2.5000    2.5000    2.5000    2.5000
```

Infine si possono definire vettori a passo costante tramite la sintassi partenza:passo:arrivo, p.e.

```
» v=0:1:10

v =

     0     1     2     3     4     5     6     7     8     9    10

» u=10:-2:0

u =

    10     8     6     4     2     0

» whos

  Name      Size            Bytes  Class

  A         4x4               128  double array
  e         4x4               128  double array
  o         4x4               128  double array
  u         1x6                48  double array
  v         1x11               88  double array
  z         4x4               128  double array
```

Si può sottintendere il passo se il passo è uguale a 1

```
» v=0:10

v =

     0     1     2     3     4     5     6     7     8     9    10
```

comando: ‘ traspone e coniuga

```
» w=v'
```

```
w =

     0
     1
     2
     3
     4
     5
     6
     7
     8
     9
    10
```

Nota: se **v** è complesso **v'** è il trasposto coniugato, **v.'** è il trasposto non coniugato.

Riferimento a elementi e sottoinsiemi di matrici

Ci si riferisce all'elemento di riga r e colonna c della matrice con la sintassi A(r,c), p.e.
ATTENZIONE: a differenza del C, gli indici partono da 1

```
» A

A =

    11     3     2    13
     5     1    11     8
     9     6     9    12
     2    15    14     1

» a=A(1,2)

a =

     3

» b=A(3,4)

b =

    12
```

Errori possibili: usate degli indici che vanno oltre i limiti della matrice. p.e.

```
» c=A(3,5)
??? Index exceeds matrix dimensions.

» c=A(3,0)
??? Index exceeds matrix dimensions.
```

Con la stessa sintassi si possono modificare singoli elementi , p.e.

```
» A(3,4)=0
```

```
A =
   11     3     2    13
    5     1    11     8
    9     6     9     0
    2    15    14     1
```

Nota: se gli indici eccedono le dimensioni della matrice, in questo caso la matrice viene ampliata!

Ci si può riferire a sottoinsemi della matrice A tramite sottoinsiemi di indici costruiti con la stessa sintassi che abbiamo visto prima; p.e. so che 1:3 definisce il vettore 1 2 3; per cui se voglio riferirmi alla sottomatrice di righe da 1 a 3 e colonne da 1 a 2 posso fare:

```
» d=A(1:3,1:2)

d =
   11     3
    5     1
    9     6
```

Infine, tramite la funzione find(*condizione*) si possono rintracciare gli elementi di una matrice che rispettano la condizione. La funzione find() ne restituisce gli indici, che si possono poi usare per svolgere delle operazioni. p.e.

```
» set=find(A>10)

set =

    1
    8
   10
   12
   13
   15

» A(set)=-A(set)

A =

  -11     3     2   -13
    5     1   -11     8
    9     6     9   -12
    2   -15   -14     1

» A(set)=0

A =

    0     3     2     0
    5     1     0     8
    9     6     9     0
    2     0     0     1
```

osservazione: esiste un metodo di indicizzazione alternativo a quello che vi ho dato: con un solo indice, si contano gli elementi progressivamente lungo le colonne a partire dalla prima, per cui p.e. con A che ha 4 righe, è la stessa cosa dire A(2,2) oppure A(6).

Operazioni sulle variabili

Ci sono un'infinità di operazioni e funzioni predefinite in MATLAB, a cui se ne possono aggiungere di nuove nel modo che vedremo. Ci sono le operazioni matematiche elementari definite dai simboli: + - * / ^ (elevamento a potenza)

ATTENZIONE: tutte queste operazioni si riferiscono a operazioni matriciali. Se le applicate a scalari e costanti ottenete:

```
» x=3

x =

     3

» y=x+1

y =

     4

» z=x*y

z =

    12

» z=x/y

z =

    0.7500

» z=x^y

z =

    81

» z=x^4

z =

    81
```

Quando però le applicate a matrici si intendono in modo matriciale. Se definiamo le tre matrici:

```
» B=[1 2; 0 -1]

B =

     1     2
     0    -1

» e=eye(2)

e =
```

```
    1     0
    0     1

» C=[2 2; 1 1]

C =

    2     2
    1     1
```

Il prodotto tra queste è matriciale:

```
» D=B*e

D =

    1     2
    0    -1

» D=B*C

D =

    4     4
   -1    -1
```

La somma e la divisione pure:

```
» D=B+C

D =

    3     4
    1     0

» D=B/C

Warning: Matrix is singular to working precision.
```

(C ha determinante nullo)

```
D =

  Inf   Inf
  Inf   Inf

» D=C/B

D =

    2     2
    1     1
```

Cioè la divisione genera la matrice che moltiplicata per il divisore dà il dividendo.
E infine anche la potenza...

```
» D=C^3

D =
```

```
   18    18
    9     9
```

che ha il significato di

```
» D=C*C*C

D =

   18    18
    9     9
```

Anche le dimensioni delle matrici che coinvolgete nell'operazione devono essere coerenti, p.e. se cerco di sommare B che è una 2x2 ad A che è una 4x4...

```
» D=A+B
??? Error using ==> +
Matrix dimensions must agree.
```

Per svolgere le operazioni sui singoli elementi di una matrice, occorre premettere il . all'operatore:

```
» D=C.^3

D =

    8     8
    1     1

» D=B.*C

D =

    2     4
    0    -1
```

Oltre alle operazioni elementari ci sono le funzioni exp(), log(), log10(), sin(), cos(), sinh(), abs(), sqrt(), ecc.. Queste applicate a matrici si intendono applicate elemento per elemento, e quindi restituiscono una matrice delle stesse dimensioni dell'argomento ma con operazioni svolte su ogni elemento, p.e.

```
» B

B =

    1     2
    0    -1

» D=exp(B)

D =

   2.7183    7.3891
   1.0000    0.3679
```

Esistono poi funzioni più particolari, che vogliono un argomento matriciale perché svolgono operazioni su matrici. Tra queste vi segnalo sum(), prod(), size(), length():

```
» D=sum(B)
```

```
D =

     1     1

» D=prod(B)

D =

     0    -2

» I=size(B)

I =

     2     2

» I=length(v)

I =

    11

» I=length(w)

I =

    11
```

comando: help seguito dal nome della funzione: descrive il comportamento della funzione

```
» help sum

 SUM Sum of elements.
    For vectors, SUM(X) is the sum of the elements of X. For
    matrices, SUM(X) is a row vector with the sum over each
    column. For N-D arrays, SUM(X) operates along the first
    non-singleton dimension.

    SUM(X,DIM) sums along the dimension DIM.

    Example: If X = [0 1 2
                     3 4 5]

    then sum(X,1) is [3 5 7] and sum(X,2) is [ 3
                                              12];

    See also PROD, CUMSUM, DIFF.
```

comando: "**;**" al termine di un'istruzione, prima di battere invio, serve a far eseguire l'operazione senza visualizzare il risultato. Finché si trattano matrici 2x2 o vettori lunghi 10, può essere interessante o utile richiederne la visualizzazione. Quando si tratta un vettore di 10mila elementi, la sua visualizzazione è una perdita di tempo (diversi secondi), ed è pure inutile, perché comunque ne rimane a video una porzione minima e quindi spesso insignificante.

Generazione e visualizzazione di segnali

Noi useremo MATLAB essenzialmente per sperimentare e vedere gli effetti del trattamento dei segnali. Per fare questo dobbiamo rappresentare i segnali come variabili in MATLAB. Avete visto che un segnale è essenzialmente una funzione di una (o più) variabili indipendenti. Per semplificare le cose assumiamo per ora che questa variabile sia unica. Avete anche distinto tra due tipi possibili di segnali: segnali continui e segnali discreti. I primi sono funzioni di una variabile che varia con continuità in un intervallo. I secondi sono funzioni di una variabile che assume valori discreti a intervalli prefissati: una sequenza. Ne deriva che i primi sono anch'essi continui, mentre i secondi sono anch'essi delle sequenze.
Nasce qui ora un'osservazione che ci accompagnerà per tutto il laboratorio: in MATLAB possiamo facilmente rappresentare una sequenza, e quindi un segnale discreto tramite un vettore di elementi. NON possiamo rappresentare un segnale continuo in senso stretto, dal momento che possiamo definire solo vettori costituiti da un numero finito e numerabile di elementi. Useremo "un'approssimazione" di segnali continui basati su variabili indipendenti di nuovo discretizzate, con un passo piccolo "a piacere". Spesso non ci accorgeremo della differenza, se sapremo scegliere un passo di discretizzazione sufficientemente piccolo. Quando invece sarà necessario notare la differenza, ve lo sottolineerò.

Supponiamo dunque di voler rappresentare un segnale nell'intervallo temporale di 1 s, da 0 a 1, con passo di discretizzazione 1 ms. La variabile indipendente (tempo) la rappresentiamo in MATLAB tramite il vettore:

» t=0:0.001:1; (RICORDATEVI il **;** !!! se terminata con **;** l'istruzione viene svolta senza visualizzarne il risultato)

```
» whos
  Name      Size            Bytes  Class

  t         1x1001           8008  double array

Grand total is 1001 elements using 8008 bytes
```

Se si vuole controllare si può per esempio visualizzare i primi 10 elementi

```
» t(1:10)

ans =

  Columns 1 through 7

         0    0.0010    0.0020    0.0030    0.0040    0.0050    0.0060

  Columns 8 through 10

    0.0070    0.0080    0.0090
```

Dato t è possibile creare un segnale y p.e. che consiste di 2 sinusoidi, una a 50 Hz e l'altra a 20 Hz con ampiezza doppia. In MATLAB la costante predefinita **pi** è π.

```
» y = sin(2*pi*50*t) + 2*sin(2*pi*20*t);
```

La nuova variabile y, derivata dal vettore t, è anch'essa di 1001 elementi.

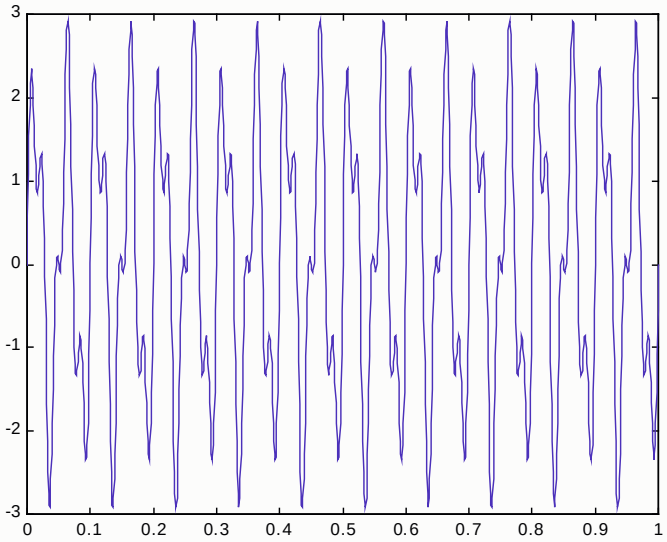
```
» whos
  Name      Size            Bytes  Class

  t         1x1001           8008  double array
  y         1x1001           8008  double array

Grand total is 2002 elements using 16016 bytes
```

Per visualizzarla si usa la funzione MATLAB plot:

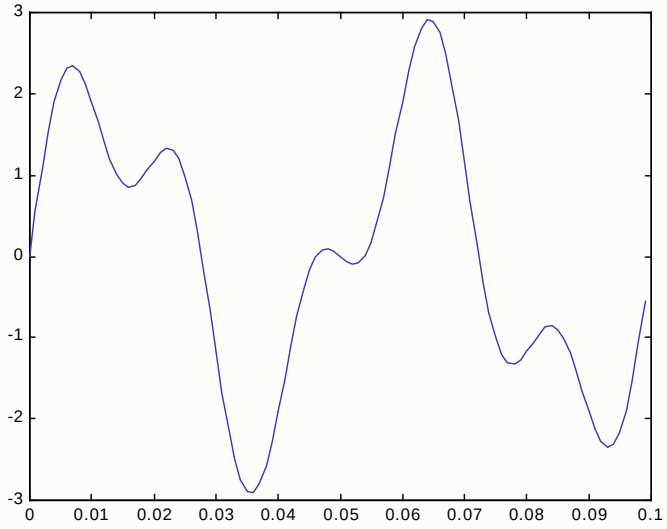
» plot(t,y)



oppure si può vedere più in dettaglio visualizzandone un sottoinsieme:

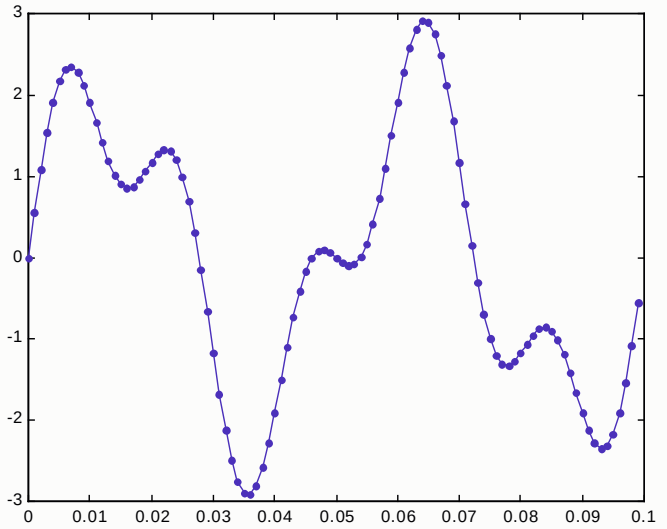
» plot(t(1:100),y(1:100))

Notate che, pur trattandosi di una sequenza, l'onda viene rappresentata in forma continua, interpolando tra i due valori adiacenti per gli istanti di tempo mancanti.



Se aggiungete un formato per il simbolo ad esempio a punto ('.-'), visualizzate i campioni effettivamente presenti, ed il modo con cui la curva viene interpolata tra i due campioni adiacenti:

» plot(t(1:100),y(1:100),'.-')



Funzioni definite dall'utente

Finora abbiamo lavorato solo al prompt dei comandi. Si possono raccogliere insiemi di istruzioni come quelli che abbiamo eseguito, e scriverli in file di testo che si possono poi lanciare in MATLAB, in modo da rendere più comoda e modulare l'esecuzione di elaborazioni più complesse. Questi si chiamano M-files e devono essere nominati nome_file.m. Possono essere scritti con un qualsiasi editor, a patto poi di cambiare l'estensione in .m, ma conviene usare l'editor fornito con MATLAB. Questo si lancia sia dal prompt dei comandi digitando edit sia da da MATLAB entrando in file => new.

Esistono due classi di files:

1. scripts
   - non hanno nessuna intestazione: sono sequenze di comandi
   - producono esattamente lo stesso effetto dell'esecuzione al prompt dei comandi contenuti, riga per riga, per cui p.e. tutte le variabili generate rimangono nel workspace al termine dell'esecuzione
   - si lanciano digitando nome_file al prompt dei comandi
2. funzioni
   - accettano parametri in ingresso e restituiscono dei valori
   - la sintassi di definizione è function [rest1,rest2...] = nome_file(par1,par2...)
   - tutte le variabili generate all'interno della funzione vengono cancellate al termine dell'esecuzione; nel workspace al termine dell'esecuzione rimangono solo le variabili a cui sono stati assegnati i valori restituiti
   - si lanciano dal prompt dei comandi tramite

[rest_attuale1,rest_attuale 2...] = nome_file(par_attuale 1,par_attuale 2...);

Per definire queste funzioni o script si usa un linguaggio la cui sintassi è abbastanza simile ai vari linguaggi di programmazione.
Oltre alle funzioni ed operazioni già viste, aggiungiamo:

- **if** *condizione*, ... **else** ..., **end**
- **while** *condizione*, ... **end**
- **for k**=*vettore_valori*, ... **end**

e gli operatori logici:

- **==** uguale
- **~=** diverso
- **<, >, <=, >=** disuguaglianze
- **&** AND logico
- **|** OR logico

Esempio:
scrivere una funzione rect.m che restituisce l'ordinata di rect(x) (1 tra –0.5 e 0.5, 0 altrove) corrispondente al vettore x passato come parametro.

```
function y=rect(x)
```
(1 solo parametro, 1 solo valore restituito)

```
y=zeros(size(x));
set=find(abs(x)<=0.5);
y(set)=1;
```

La nuova funzione rect.m può essere usata come ogni altra funzione MATLAB. In particolare può essere usata per generare un segnale rettangolare come segue:

```
» t=-1:1/500:1;
» plot(t,rect(t));
```

NOTA: Per salvare su file e poi ricaricare le variabili presenti in memoria si usano i comandi:

```
» save nome_file var1 var2...
» load nome_file
```