Matlab. Vettori, funzioni matematiche e grafici.

Alvise Sommariva

Università degli Studi di Padova
Dipartimento di Matematica

1 aprile 2016

Introduzione

Il proposito di questa terza lezione mostriamo

- Come definire i vettori in Matlab (e alcune operazioni di base).
- Come definire funzioni matematiche, senza utilizzare files ".m".
- Come eseguire il grafico di funzioni matematiche.

Matlab: operazioni con vettori

Un vettore in Matlab lo si rappresenta tramite le sue componenti. A seconda del vettore ci sono vari modi più o meno efficaci. Se ad esempio devo descrivere il vettore (riga)

$$[5, 4, 9]$$

ciò può essere fatto come segue

```
>> % VETTORE (RIGA!) [5 4 9]
>> v=[5 4 9]
v =
     5     4     9
>>
```

in cui tra le componenti numeriche del vettore sono interposti degli spazi vuoti.

Matlab: operazioni con vettori

Si supponga di aver immagazzinato nella variabile v il vettore riga $[5, 4, 9]$ e di voler aggiungere una componente, ad esempio 10, così da avere il vettore

$$[5, 4, 9, 10]$$

A tal proposito si procede come segue

```
>> % VETTORE (RIGA!) [5 4 9]
>> v=[5 4 9];
>> v=[v 10]
v =
     5     4     9    10
>>
```

Matlab: operazioni con vettori

Per selezionare la componente j-sima di un vettore v, si usa il comando `v(j)`. Così se volessimo selezionare la seconda componente del vettore $v = [5, 4, 9, 10]$

```
>> v=[5 4 9 10]
v =
     5     4     9    10
>> v(2)
ans =
     4
>>
```

Un utile comando per selezionare l'ultima componente è `end`. Così ad esempio

```
>> v=[5 4 9 10];  v(end)
ans =
     10
>>
```

Matlab: operazioni con vettori

Per determinare la lunghezza di un vettore, cioè il numero delle sue componenti, si usa il comando `length`, come da esempio.

```
>> v=[5 4 9 10]; l=length(v)
l =
     4
>>
```

Nota.

Un errore comune è scrivere `lenght` *invece di* `length`*.*

Matlab: operazioni con vettori

Se invece di un vettore riga, si vuole descrivere un vettore colonna, si procede in due modi. Nel primo caso, si scrive un vettore riga e lo si traspone con il comando "'"

Così, ad esempio,

```
>> v=[5 4 9 10];
>> v=v'
v =
     5
     4
     9
    10
>> % il simbolo ' fa la trasposizione del vettore.
```

Matlab: operazioni con vettori

Alternativamente lo si descrive, intervallando un ";" tra le varie componenti.

```
>> v=[5; 4; 9; 10]
v =
     5
     4
     9
    10
>> % vettore colonna, descritto direttamente in questa
    forma, utilizzando il ";" .
```

Matlab: operazioni con vettori

Per capire se un vettore è riga o colonna, è bene usare il comando "size" che ne descrive le dimensioni.

```
>> v=[5; 4; 9; 10]; size(v)
ans =
     4     1
>> % Componente 1: n.ro righe. Componente 2: nero
    colonne.
```

Quindi il vettore ha 4 righe e 1 colonna, e di conseguenza è un vettore colonna.

Matlab: operazioni con vettori

Per vettori con componenti equispaziate, cioè del tipo $v = (v_k)_{k=1,\ldots,N}$, $v_k = v_1 + k \cdot h$, con $k = 1, \ldots, N$, cioè

$$v = [v_1 \quad v_1 + h, \quad v_1 + 2h, \ldots, \quad v_1 + Nh]$$

molto utili in matematica, si possono usare due comandi speciali.

Se è nota la spaziatura h, il punto iniziale a e il punto finale b, si scrive

$$(\mathtt{a} : \mathtt{h} : \mathtt{b})$$

Ad esempio:

```
>> a=5; b=7; h=0.5; v=a:h:b

v =

    5.0000    5.5000    6.0000    6.5000    7.0000

>>
```

Matlab: operazioni con vettori

Se è invece noto il numero di punti equispaziati N, il punto iniziale a e il punto finale b, si scrive

$$\texttt{linspace}(\texttt{a}, \texttt{b}, \texttt{N})$$

Ad esempio:

```
>> linspace(5,7,4)
ans =
    5.0000    5.6667    6.3333    7.0000
>>  % i punti sono equispaziati, con spaziatura
    2/3=0.6666 . . . , punto iniziale 5 e finale 7.
```

Operazioni aritmetiche e funzioni elementari predefinite

Le seguenti operazioni tra due vettori dello stesso tipo (riga o colonna) e della stessa dimensione, producono un vettore dello stesso tipo e dimensione.

```
>> linspace(5,7,4)
ans =
    5.0000    5.6667    6.3333    7.0000
>>  % i punti sono equispaziati, con spaziatura
    2/3=0.6666 . . . , punto iniziale 5 e finale 7.
```

Operazioni aritmetiche e funzioni elementari predefinite

Le seguenti operazioni tra due vettori dello stesso tipo (riga o colonna) e della stessa dimensione, producono un vettore dello stesso tipo e dimensione.

$+$	addizione
$-$	sottrazione
$.*$	prodotto
$./$	divisione
$.\wedge$	potenza

Operazioni aritmetiche e funzioni elementari predefinite

```
>> a=[1 2]; b=[5 8];
>> a+b
ans =
     6     10
>> a-b
ans =
    -4     -6
>> a.*b
ans =
     5     16
>> % PRODOTTO PUNTUALE: [1*5 2*8]
>> a./b
ans =
    0.2000    0.2500
>>  % DIVISIONE PUNTUALE: [1/5 2/8]
>> a.^b
ans =
     1   256
>> % POTENZA PUNTUALE: [1^5 2^8]
```

Operazioni aritmetiche e funzioni elementari predefinite

Si sottolinea che i vettori devono essere dello stesso tipo e dimensione.

```
>> a=[2 4]
a =
     2     4
>> b=[5; 7]
b =
     5
     7
>> a+b
Error using  +
Matrix dimensions must agree.
>> % NON POSSO SOMMARE VETTORI RIGA CON VETTORI COLONNA
>> a=[2 4]; b=[1 3 5]; a+b
Error using  +
Matrix dimensions must agree.
>> % NON POSSO SOMMARE VETTORI CON NUMERO DIVERSO DI
    COMPONENTI
```

Operazioni aritmetiche e funzioni elementari predefinite

Unica eccezione a quanto detto, si ha quando uno dei due vettori è un numero.

```
>> a=[2 4]; a+1 % [2+1 4+1]
ans =
     3     5
>> a=[2 4]; 2.*a % [2*2 2*4]
ans =
     4     8
>> a=[2 4]; a.^2 % [2^2 4^2]
ans =
     4    16
>> a=[2 4]; 3.^a % [3^2 3^4]
ans =
     9    81
>> a=[2 4]; 8./a % [8/2 8/4]
ans =
     4     2
```

Operazioni aritmetiche e funzioni elementari predefinite

Importante. Si osservi che se uno dei vettori è un numero, possiamo in alcuni casi evitare il `.`.

```
>> a=[2 4]; 3*a % OK!
ans =
     6    12
>> a=[2 4]; 3/a % KO! NUMERO DIVISO VETTORE.
Error using  /
Matrix dimensions must agree.
>> a=[2 4]; a/3 %OK! VETTORE DIVISO NUMERO.
ans =
    0.6667    1.3333
>> a=[2 4]; a^3 % KO!
Error using  ^
Inputs must be a scalar and a square matrix.
To compute elementwise POWER, use POWER (.^) instead.
>> a=[2 4]; 3^a % KO!
Error using  ^
Inputs must be a scalar and a square matrix.
To compute elementwise POWER, use POWER (.^) instead.
```

Operazioni aritmetiche e funzioni elementari predefinite

Le seguenti operazioni tra due vettori dello stesso tipo (riga o colonna) e della stessa dimensione, producono (puntualmente) un vettore dello stesso tipo e dimensione.

`abs`	valore assoluto
`sin`	seno
`cos`	coseno
`tan`	tangente
`cot`	cotangente
`asin`	arco seno
`acos`	arco coseno
`atan`	arco tangente
`sinh`	seno iperbolico
`cosh`	coseno iperbolico
`tanh`	tangente iperbolica
`asinh`	arco seno iperbolico
`acosh`	arco coseno iperbolico
`atanh`	arco tangente iperbolica
`sqrt`	radice quadrata
`exp`	esponenziale
`log 2`	logaritmo base 2
`log10`	logaritmo base 10
`log`	logaritmo naturale
`fix`	arrotondamento verso 0
`round`	arrotondamento verso l'intero più vicino
`floor`	arrotondamento verso $-\infty$
`ceil`	arrotondamento verso $+\infty$
`sign`	segno
`rem`	resto della divisione

Operazioni aritmetiche e funzioni elementari predefinite

```
>> a=[pi pi/2]; cos(a)
ans =
   -1.0000    0.0000
>>  % [cos(pi) cos(pi/2)]
>> b=[1 exp(1)]; log(b)
ans =
     0      1
>> % [log(1) log(exp(1)]=[0 1]
>> c=exp(log([1 4]))
c =
     1      4
>> % c=exp([log(1) log(4)])=[exp(log(1)) exp(log(4))]
>> sqrt([16 36 64])
ans =
     4      6      8
>> % [sqrt(16) sqrt(36) sqrt(64)]
```

Definizione di funzioni matematiche

Per quanto riguarda le funzioni elementari spesso, qualora necessario, un utente può comporne di proprie, salvandole su file. Così ad esempio, può salvare in `f.m` la funzione

```
function y=f(x)
y=sin(x)+pi;
```

e quindi chiamarla in altri programmi (che fanno parte della stessa cartella di `f.m`).
A volte però la si può definire semplicemente, senza ricorrere a nuovi files, con il comando di definizione di funzione matematica `@`.

```
>> f=@(x) sin(x)+pi;
>> f(0)
ans =
    3.1416
>> % sin(0)+pi
```

Definizione di funzioni matematiche

Talvolta, per valutare le funzioni si usa il comando feval.
Vediamone qualche esempio.

```
>> f=@(x) sin(x)+pi;
>> feval(f,0)
ans =
    3.1416
>> f=@(x) [sin(x)+pi; cos(x)+pi]; % funzione da R in R
    ^2.
>> feval(f,0)
ans =
    3.1416
    4.1416
>>
```

Definizione di funzioni matematiche

Una alternativa a `@` è la definizione tramite `inline` (che Matlab considera desueta)

```
>> g=inline('sin(t)+pi');
>> g(0) % g applicata a numero
ans =
    3.1416
>> g([0 pi/2]) % g applicata a vettore
ans =
    3.1416    4.1416
>>
```

Il comando `plot`

Per effettuare il grafico di funzioni si usa il comando `plot`, avente

- quale primo argomento un vettore che si riferisce alle ascisse,
- quale secondo argomento, un vettore dello stesso tipo e dimensione del primo, che si riferirisce alle ordinate.

Nota.

- *Per sovrapporre più grafici, si usano i comandi hold on e hold off, intervallati dai plot da sovrapporre.*
- *Per cancellare precedenti grafici, si usa il comando clf.*

Il comando plot

Per capire il comando plot digitiamo su workspace

```
>> help plot
 plot   Linear plot.
    plot(X,Y) plots vector Y versus vector X.
. . . . .
Various line types, plot symbols and colors may be
   obtained with plot(X,Y,S) where S is a character
   string made from one element from any or all the
   following 3 columns:
 b     blue          .     point    -      solid
 g     green         o     circle   :      dotted
 r     red           x     x-mark   -.     dashdot
 c     cyan          +     plus     --     dashed
 m     magenta       *     star   (none)   no line
 y     yellow        s     square
 k     black         d     diamond
 w     white         v     triangle (down)
```

Il comando plot

```
        ^      triangle (up)
        <      triangle (left)
        >      triangle (right)
        p      pentagram
        h      hexagram
. . . . .
    Example
       x = -pi:pi/10:pi;
       y = tan(sin(x)) - sin(tan(x));
       plot(x,y,'--rs','LineWidth',2,...
                       'MarkerEdgeColor','k',...
                       'MarkerFaceColor','g',...
                       'MarkerSize',10)
. . . . .
>>
```

Nell'esempio vengono modificati vari pattern di grafica, come tratteggio colore o grossezza delle linee.

Il comando plot

In virtù di quanto detto, digitiamo su workspace

```
>> clf; % cancella, se necessario, precedenti grafici.
>> f=inline('exp(t)-2');
>> x=0:0.001:1; % vettore di ascisse.
>> y=f(x);
>> hold on; % tieni grafici in finestra,
    sovrapponendoli, fino ad hold off
>> plot(x,y); % GRAFICO f (unito i punti di
    campionamento con segmentini in blu).
>> g=@(x) 0*x;
>> yy=g(x);
>> plot(x,yy,'r-'); % ASSE x (in rosso!).
>> hold off;
```

e otteniamo, su una finestra esterna, il grafico della funzione $f(x) = e^x - 2$ (in blu), definita in $[0, 1]$ (mediante campionamenti in punti sufficientemente fitti).

Il comando plot

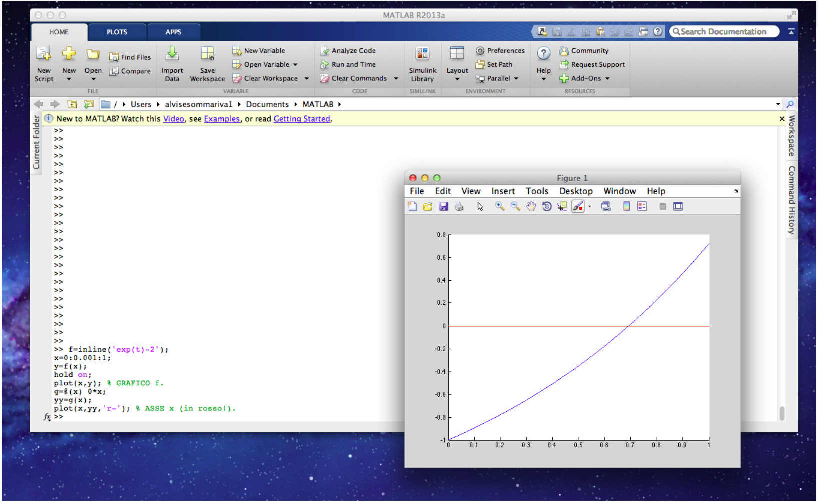

Figura : Finestre del workspace e di plot

Il comando `plot`

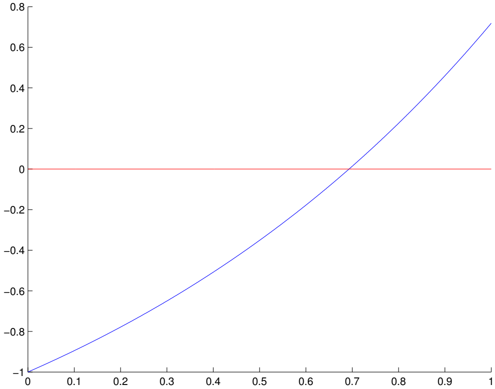

Figura : In blu, grafico della funzione f in $[0, 1]$. In rosso, l'asse delle ascisse.

Il comando `plot`

Per salvare il grafico:

- Da menu File (del plot!), si clicchi su Save as (o sue traduzioni).
- Si dia un nome al file nel campo in alto Save as (o sue traduzioni).
- Per cambiare il formato del file, si clicchi sul menu' a tendina con Format (o sue traduzioni).
- Si clicchi Save (o sue traduzioni) per salvare il file stesso nel formato prescelto.

Il comando `plot`

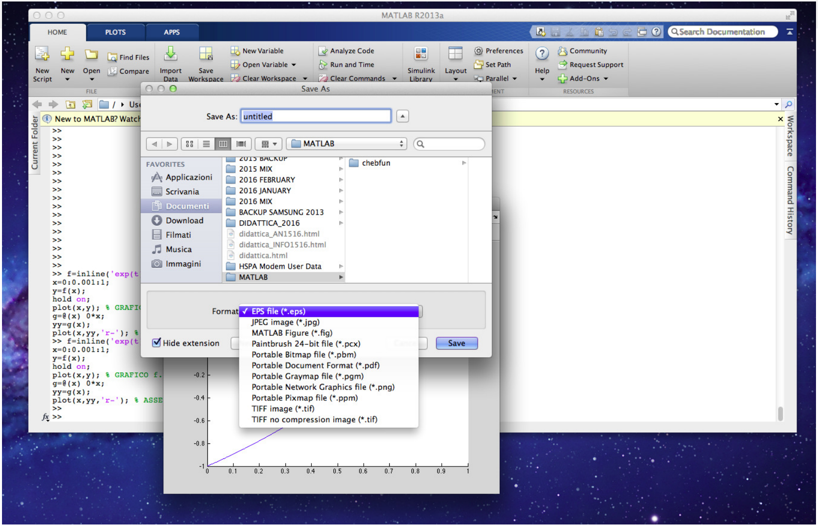

Figura : Finestre del workspace e di plot

Il comando `plot`

Per salvare il grafico:

- Da menu File (del plot!), si clicchi su Save as (o sue traduzioni).
- Si dia un nome al file nel campo in alto Save as (o sue traduzioni).
- Per cambiare il formato del file, si clicchi sul menu' a tendina con Format (o sue traduzioni).
- Si clicchi Save (o sue traduzioni) per salvare il file stesso nel formato prescelto.

Esercizio: algoritmo di bisezione

Si supponga sia $f : [a, b] \to \mathbb{R}$ una funzione continua tale che $f(a) \cdot f(b) < 0$.

E' noto che per il teorema degli zeri (di Bolzano), esiste almeno un punto x^* tale che $f(x^*) = 0$.

Per approssimare x^* utilizziamo l'algoritmo di bisezione che genera una successione di intervalli (a_k, b_k) con

- $f(a_k) \cdot f(b_k) < 0$,
- $[a_k, b_k] \subset [a_{k-1}, b_{k-1}]$,
- $|b_k - a_k| = \frac{1}{2}|b_{k-1} - a_{k-1}|$.

Fissate la tolleranza ϵ si arresta l'algoritmo quando

$$|b_k - a_k| \leq \epsilon.$$

Esercizio: algoritmo di bisezione

Operativamente, dati in input a, b con $a \leq b$ e $f(a) \cdot f(b) < 0$, nonchè la tolleranza ϵ,

- calcola $c = (a+b)/2$;
- se $f(a) \cdot f(c) > 0$ sostituisce "c" ad "a", viceversa sostituisce "c" a "b";
- termina il processo se le condizioni d'arresto sono verificate.

Esercizio

Si implementi in Matlab/Octave l'algoritmo di bisezione.