

MATLAB-SIMULINK

Enrico Martini

A.A. 2018/2019

Indice

1	Operazioni su array	2
1.1	Operazioni aritmetiche	2
1.2	Operazioni logiche	2
2	Script	2
2.1	Input	2
2.2	Output	2
2.3	Plot	3
2.3.1	Subplot	3
2.4	Carica e salva	3
2.5	Funzioni	3
3	Strutture	4
3.1	If	4
3.2	For	4
3.3	While	4
3.4	Tic-Toc	4
3.5	Stringhe	5
3.6	Array di celle	5
3.7	Struct	6
4	Ordinamento	6
4.1	Ordinamento stringhe	6
5	Immagini	6

1 Operazioni su array

1.1 Operazioni aritmetiche

1. Somma: $A + B$
2. Sottrazione: $A - B$
3. Moltiplicazione: $A .* B$
4. Divisione: $A ./ B$
5. Esponenziale: $A .^ B$

1.2 Operazioni logiche

Usando operazioni logiche, si viene a creare un array logico (composto solo da 1 e 0):

`[1 4 10] > 3` \rightarrow `[0 1 1]`

Un vettore logico può essere utilizzato come maschera su un altro vettore. Si può creare un vettore logico con ones e zeros:

`ones(1,3,'logical')` \rightarrow `[1 1 1]`

`zeros(1,3,'logical')` \rightarrow `[0 0 0]`

La funzione `any(v)` ritorna `true` se un elemento del vettore logico è `true`.

La funzione `all` ritorna `true` solo se tutti gli elementi del vettore logico sono `true`.

La funzione `find(v>6)` torna gli indici dei valori che rispettano la condizione logica.

La funzione `isequal(v,x)` ritorna `true` solo se tutti gli elementi degli array `x` e `v` sono uguali.

2 Script

2.1 Input

La funzione di input si rappresenta con:

`var = input('prompt string')` *numero*

`var = input('prompt string','s')` *carattere o stringa*

2.2 Output

La funzione di output più semplice è `disp` ma non può essere formattata. Per poter formattare l'output si utilizza `fprintf`

Formato	Variabili
%d	interi
%f	float
%c	char
%s	string

`fprintf('The numbers are %4d and %.1f\n', 3, 24.59)` \rightarrow *The numbers are 3 and 24.6*

2.3 Plot

Per creare un semplice grafico si utilizza la funzione `plot(x)`. Ecco le funzioni per modificare il grafico:

Funzione	Significato
<code>xlabel('string')</code>	<i>nome etichetta asse x</i>
<code>ylabel('string')</code>	<i>nome etichetta asse y</i>
<code>title('string')</code>	<i>titolo plot</i>
<code>axis([xmin xmas ymin ymax])</code>	<i>valore massimo e minimo assi</i>
<code>clf</code>	<i>pulisce la finestra</i>
<code>figure</code>	<i>crea una nuova finestra</i>
<code>hold</code>	<i>blocca il grafico sulla finestra</i>
<code>legend</code>	<i>mostra la legenda</i>
<code>grid</code>	<i>mostra la griglia</i>
<code>bar</code>	<i>crea un grafico a barre</i>

2.3.1 Subplot

La funzione `subplot(m,n,i)` permette di mostrare più plot in una finestra sola. Per una matrice $m * n$ elementi ad esempio, si può vedere l'elemento i

```
for i = 1:4
    subplot(2,2,i)
end
```

2.4 Carica e salva

Per leggere informazioni da un file:

```
load filename.ext
```

Viene creato una variabile "filename" solo se il file ha gli stesso numero di valori in ogni riga.

Per scrivere informazioni su file:

```
save filename variable -ascii (Sovrascrive se già esistente)
```

```
save filename variable -ascii -append (Aggiunge in coda se già esistente)
```

2.5 Funzioni

Per creare una funzione, deve essere salvata in un file "**fname.m**", ad esempio:

```
function area = calcarea(rad)
% This function calculates the area of a circle
area = pi * rad * rad;
end
```

Se una funzione deve ritornare più di un valore, viene specificato dopo function nell'header. Se una funzione non ritorna nulla, si omette la variabile output

e l'uguale. Quando una funzione chiama una sottofunzione, allora entrambe possono essere salvate sullo stesso file chiamato come la funzione primaria. La sottofunzione avrà la dicitura **subfunction** invece di **function**. La sottofunzione ovviamente potrà essere chiamata solo dalla funzione principale.

3 Strutture

3.1 If

Sintassi:

```
if condition1
    action1
elseif condition2
    action2
else
    action3
end
```

Chiamando la funzione **error('string')** si possono lanciare errori all'interno di un determinato branch dell'if.

3.2 For

Sintassi:

```
for loopvar = range
    action
end
```

N.B: la variabile **range** è un vettore.

3.3 While

Sintassi:

```
while condition
    action
end
```

3.4 Tic-Toc

Le funzioni **tic** e **toc** vengono utilizzate per prendere il tempo di una determinata porzione di codice:

```
tic
mysum = 0;
for i = 1:20000000
    mysum = mysum + i;
end
toc
```

Con il comando `fortictoc` viene visualizzato il tempo impiegato per eseguire la porzione di codice.

3.5 Stringhe

Le stringhe sono viste come vettori di caratteri. Per concatenare due stringhe:

`['string1' 'string2'] → string1string2`

`strcat('string1','string2') → string1string2`

Il comando `sprintf` è simile a `fprintf` ma invece di stampare a video, ne crea una stringa, permettendo di creare facilmente stringhe complesse. Altre funzioni utili sono:

1. `strcmp(s1,s2) →` compara due stringhe;
2. `strncmp(s1,s2,n) →` compara due stringhe per i primi `n` caratteri;
3. `strncmpi(s1,s2) →` compara due stringhe ignorando i case;
4. `strncmpi(s1,s2) →` compara due stringhe per i primi `n` caratteri ignorando i case;
5. `strfind(string,substring) →` ritorna le occorrenze delle sottostringa nella stringa;
6. `strrep(string, oldsubstring, newsubstring) →` sostituisce le occorrenze di `oldsubstring` con `newsubstring` in `string`;
7. `strtok(string,limiter) →` divide `string` in due sottostringhe quando compare il carattere `limiter`;
8. `eval(string) →` valuta una stringa come se fosse una funzione;

3.6 Array di celle

E' un tipo di struttura con la quale è possibile salvare valori di tipi diversi. Ogni cella può essere un vettore o una matrice. E' utile per salvare stringhe di lunghezza differente. Per crearlo si utilizza la `{}` invece che `[]`. La funzione `cell(x,y)` è utile per pre-allocare passando le dimensioni della `cell` come argomento.

Esempio: `ca = {2:4 , 'hello'}`

`ca{1} →` 2 3 4

`ca{2} →` hello

Altre funzioni sono:

1. `celldisp(ca) →` mostra il contenuto delle celle;
2. `cellplot(ca) →` mostra il plot dell'array di celle;
3. `cellstr(m) →` converte una matrice di caratteri in array di celle;
4. `iscellstr(ca) →` torna 1 se l'array di celle è composto da tutte stringhe;
5. `strjoin(ca,limiter) →` concatena tutte le stringhe semparandole con un delimitatore;
6. `strsplit(string,limiter) →` divide una stringa in sottostringhe in un array di celle;

3.7 Struct

Sintassi:

```
namestruct = struct('Nome1', valore1, 'Nome2', valore2)
```

Le strutture memorizzano valori di tipi diversi in campi. Ogni campo ha il proprio nome e ci si può riferire tramite esso con la dicitura `namestruct.nomefiled`. La funzione `struct` ammette solo coppie di argomenti nome-valore. Per stampare l'intera struttura si utilizza `disp`, un singolo campo `fprintf`. A differenza degli array di celle, le struct non sono indicizzate e quindi non ci si può iterare. Le funzioni utili per le strutture sono:

1. `rmfield(s,field)` → *rimuove field dalla struttura*;
2. `isstruct(s)` → *ritorna true se l'argomento è una struttura*;
3. `isfield(s,arg)` → *ritorna true se arg è un nome di un campo nella struttura s*;
4. `fieldnames(s)` → *ritorna il nome dei campi come un'array di celle*;

Le strutture possono anche essere annidate e si possono creare array di strutture.

4 Ordinamento

MATLAB ha funzioni già implementate per ordinare array, che possono essere ordinati in modo crescente o decrescente.

```
sort(vec)                % ordine crescente
sort(vec,'descend')      % ordine decrescente
sort(mat)                % ordine per colonne
sort(mat,2)              % ordine per righe
```

4.1 Ordinamento strighe

Utilizzando la funzione `sort(s)`, questo funziona solo con array di celle di stringhe, non con matrici di caratteri. Per poter ordinare una matrice di caratteri, bisogna utilizzare la funzione `sortrows(s)`.

5 Immagini

Le immagini vengono rappresentate da una matrice $m*n*l$, dove $l = 1$ in caso di immagini in scala di grigi, $l = 3$ in caso di immagini RGB. la funzione `image(m)`. Per avere un grafico 3D dell'immagine si utilizza la funzione `colormap(m)`. La funzione `imread(img.jpg)`. Altre funzioni utili:

1. `imshow(m)` → *mostra l'immagine*;
2. `rgb2gray(m)` → *converte un immagine rgb in scala di grigi*;
3. `im2double(m)` → *converte un immagini in matrice di double*;
4. `imhist(m)` → *visualizza l'istogramma dell'immagine*;

COMANDO	OUTPUT
rand	$n \in (0, 1)$
randn	$n \in N$
randi([m,n],1)	$n \in [m, n]$
class(var)	<i>tipo variabile</i>
type(var)	<i>casta la variabile var al tipo type</i>
sin(var)	<i>seno (rad)</i>
cos(var)	<i>coseno (rad)</i>
tan(var)	<i>tangente (rad)</i>
sind(var)	<i>sind (gradi)</i>
cosd(var)	<i>coseno (gradi)</i>
tand(var)	<i>tangente (gradi)</i>
deg2rad(var)	<i>da gradi a rad</i>
rad2deg(var)	<i>da rad a gradi</i>
log(x)	<i>logaritmo base e</i>
log2(x)	<i>logaritmo base e</i>
log10(x)	<i>logaritmo base e</i>
exp(x)	e^x
v = [1 2 3]	<i>vettore riga</i>
v = [1:1:3]	<i>vettore riga con [inizio : step : fine]</i>
linspace(1,10,5)	<i>vettore riga con 5 valori compresi tra 1 e 10</i>
logspace(1,10,5)	<i>vettore riga con 5 valori compresi tra 10^1 e 10^{10}</i>
v[x y]	<i>concatena x e y in v</i>
v(index) = val	<i>sostituisce il valore val in v(index)</i>
v(init-end)= q	<i>sostituisce il vettore q in v nel range init-end</i>
length(v)	<i>numero di elementi di v</i>
abs(v)	<i>valore assoluto di ogni elemento</i>
min(v)	<i>valore minimo in v</i>
max(v)	<i>valore massimo in v</i>
sum(v)	<i>somma degli elementi di v</i>
prod(v)	<i>prodotto degli elementi di v</i>
cumprod(v)	<i>prodotto cumulativo di v</i>
cumsum(v)	<i>somma cumulativa di v</i>
cummin(v)	<i>minimo cumulativo di v</i>
cummax(v)	<i>massimo cumulativo di v</i>
m = [1:3 ; 4:6]	<i>matrice 2 righe e 3 colonne</i>
rand(n)	<i>matrice $n \times n$ con numeri random</i>
rand(n,m)	<i>matrice $n \times m$ con numeri random</i>
randi([range],n,m)	<i>matrice $n \times m$ con numeri random \in range</i>
zeros(n)	<i>matrice $n \times n$ a zero</i>
zeros(n,m)	<i>matrice $n \times m$ a zero</i>
ones(n)	<i>matrice $n \times n$ a uno</i>
ones(n,m)	<i>matrice $n \times m$ a uno</i>
m(2,1)	<i>elemento alla riga 2 e colonna 1</i>
length(m)	<i>dimensione massima tra righe e colonne</i>
[r c] = size(m)	<i>numero righe e numero colonne di m</i>
numel(m)	<i>numero di elementi di m</i>
reshape(m,n)	<i>cambia dimensioni di m in n con lo stesso numero di elementi</i>
rot90(m)	<i>ruota m in senso orario</i>
fliplr(m)	<i>inverte le colonne da dx a sx</i>
flipud(m)	<i>inverte le righe da sopra a sotto</i>
flip(m)	<i>7 inverte righe e colonne</i>
repmat(m,r,c)	<i>replica m r volte a dx e c volte in basso</i>
repelem(m,r,c)	<i>replica ogni elemento r volte a dx e c volte in basso</i>