

# Introduzione al MATLAB © Parte 2

Lucia Gastaldi

Dipartimento di Matematica,  
<http://dm.ing.unibs.it/gastaldi/>

Lezione 2 - 14 Gennaio 2009

# Indice

- 1 M-file di tipo Script e Function
  - Script
  - Function
- 2 Controllo: `if`
- 3 Ciclo: `for`
- 4 Ciclo con controllo: `while`

## Uso di un M-file

Il processo di programmazione in MATLAB funziona nel modo seguente:

1 Si **crea** un **M-file** usando un **editore di testi** (per es. **Editor**)

S\_somma.m

```
c=sqrt(a.^2+b.^2)
```

F\_somma.m

```
function [c] = F_somma(a,b)  
c=sqrt(a.^2+b.^2)
```

2 Si **chiama** l'**M-file** dalla **linea di comando**

```
>> a= 7.5  
>> b= 3.342  
>> S_somma
```

```
      c =  
      8.2109
```

```
>> c = 7.5  
>> d = 3.342  
>> s = F_somma(c,d)  
s =  
s = 8.2109
```

# Script e Function

## Script

- Opera sui dati presenti in Workspace.
- Non accetta variabili in input.
- Non ha variabili di output.
- Utile per automatizzare una serie di istruzioni che si devono eseguire più volte.

## Function

- Le variabili interne sono locali.
- Può accettare variabili in input.
- Può avere variabili in output.
- Utile per estendere il linguaggio MATLAB alle applicazioni personali.

## M-file di tipo **script**

### Esempio: `algin.m`

```
% Risoluzione di un sistema lineare           % Commento
% e calcolo dell'errore relativo
% A Matrice di Hilbert
%
% Inizio istruzioni
A=hilb(n);                                     % Calcolo
x=[1:n]';
b=A*x;
x1=A\b;
errore=norm(x-x1)
errorerel=errore/norm(x)
```

## Caratteristiche di un file di tipo **script**

- È il tipo più semplice di M-file perchè non ha variabili di input e output.
- Serve per automatizzare una serie di comandi MATLAB che devono essere eseguiti più volte.
- Opera sui dati esistenti nell'ambiente di lavoro di base, oppure può creare nuovi dati.
- I dati che vengono generati rimangono nell'ambiente di lavoro di base e possono essere riutilizzati per altri calcoli.

### Come si usa

Basta scrivere il nome del file sulla riga di comando senza l'estensione `.m`.

```
>> miofile
```

```
return
```

## Contenuto di un file di tipo **script**

- Chiamate di un'altra function;
- Cicli `for` oppure `while`;
- `if`, `elseif`, `else`;
- Input/Output interattivi;
- Calcoli;
- Assegnazioni;
- Commenti;
- Linee bianche;
- Comandi per la costruzione di grafici.

## M-file di tipo **function**

### Esempio: `errsl.m`

```
function [errore,errrel] = errsl(n) % Riga di definizione
                                   % della function
% ERRSL errore per sistema lineare % Riga H1
% Risoluzione di un sistema lineare % Testo per help
% e calcolo dell'errore relativo
% A Matrice di Hilbert
%
% Inizio istruzioni della function
A=hilb(n); % Corpo della function
x=[1:n]';
b=A*x;
x1=A\b;
errore=norm(x-x1);
errrel=errore/norm(x);
```

# function

## Riga di definizione

```
function [output] = nome_function(input)
```

**Output** una sola variabile in uscita  $x$ :  $[output] \longrightarrow x$   
più variabili in uscita  $x, y, z$ :  $[output] \longrightarrow [x, y, z]$   
nessuna variabile in uscita:  $[output] \longrightarrow [ ]$

## Input

Le variabili in input possono essere array (scalari, vettori, matrici) ma anche il nome di altre function:

```
function [t,y] = ode23(f,[t0,tf],y0)
```

## Come si usa

```
>> [output]=nome_function(input) return
```

**Esempio** Supponiamo di avere una function di nome `miafunction` con la seguente riga di dichiarazione:

```
[A,z,b]=miafunction(v,Tspan,f,toll)
```

### Input:

|                |                      |                    |                           |
|----------------|----------------------|--------------------|---------------------------|
| <code>v</code> | vettore colonna      | <code>Tspan</code> | vettore di due componenti |
| <code>f</code> | nome di una funzione | <code>toll</code>  | scalare                   |

### Output:

|                |   |                |         |
|----------------|---|----------------|---------|
| <code>A</code> | array di dimensione appropriata                           | <code>z</code> | scalare |
| <code>b</code> | vettore colonna della stessa dimensione di <code>v</code> |                |         |

Allora se `fun` è il nome di una funzione assegnata con l'istruzione `inline`, il comando:

```
>> [A,p,b]=miafunction(v,[t0,tf],fun,1.e-10)
```

asigna alle variabili `A,p,b` il risultato ottenuto dall'esecuzione del programma `miafunction` usando i dati in input assegnati con le seguenti identificazioni:

```
v=v,    Tspan=[t0,tf],    f=fun,    toll=1.e-10
```

# function

## Riga H1

È la prima riga del testo di help.

Siccome è una riga di commento inizia con %

## Testo di help.

Si può creare un aiuto in linea per la propria `function` introducendo una o più righe di commento immediatamente dopo la riga H1.

```
>> help nome_function
```

MATLAB scrive le righe di commento che ci sono fra la riga di definizione della `function` e la prima riga che non è di commento.

## Corpo della **function**

Contiene le istruzioni per il calcolo e l'assegnazione dei valori alle variabili di output.

Le istruzioni possono essere:

- chiamate di un'altra function;
- cicli `for` oppure `while`;
- `if`, `elseif`, `else`;
- input/output interattivi;
- calcoli;
- assegnazioni;
- commenti;
- linee bianche.

## Commenti.

- Le righe di commento iniziano con %
- Si possono inserire righe di commento in qualsiasi punto della function.
- Si possono aggiungere commenti alla fine di una riga del codice.

### Esempio

```
% Somma di tutti gli elementi di un vettore.  
y = sum(x)           % Usa la function sum
```

## Esercizio

È dato un vettore  $x$  di  $n$  elementi. Allora la norma euclidea di  $x$  si calcola nel modo seguente:

$$\|x\|_2 = \sqrt{\sum_{i=1}^n x_i^2}.$$

Realizzare un programma di tipo **script** e uno di tipo **function** per il calcolo della norma euclidea.

Si può usare la function `sum` che somma tutti gli elementi di un vettore.

## Esercizio

### Problema 1:

scrivere un **M-file** di tipo function, per calcolare le radici dell'equazione di secondo grado:

$$ax^2 + bx + c = 0.$$

La function deve iniziare con la seguente riga di dichiarazione:

```
function [x1,x2]=eqsecgrado(a,b,c)
```

essendo

**Input** a, b, c coefficienti

**Output** x1, x2 radici

Tenere conto dei possibili casi degeneri e applicare la formula risolutiva:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

# Esercizio (continua)

## Traccia per la costruzione della function

- Se  $a \neq 0$ , risolvo l'equazione di secondo grado:
  - ▶ calcolo  $\Delta = b^2 - 4ac$ ;
  - ▶ se  $\Delta > 0$ , le radici sono date dalla formula;
  - ▶ se  $\Delta = 0$ , due radici coincidenti date da  $x = -b/2a$ ;
  - ▶ se  $\Delta < 0$ , non ci sono radici reali;
- se  $a = 0$ , ho una riduzione di grado:
  - ▶ se  $b \neq 0$ , l'equazione è di primo grado e la soluzione è  $x = -c/b$ ;
  - ▶ se  $b = 0$ , l'equazione diventa  $c = 0$ ;
    - ★ se  $c \neq 0$ , non ci sono radici;
    - ★ se  $c = 0$ , l'equazione è una identità e tutti i numeri reali sono soluzioni.

## if,else,elseif

**if** valuta una **espressione logica** ed esegue un gruppo di istruzioni a seconda del valore dell'espressione logica.

```
if espressione logica  
    istruzioni  
end
```

```
if espressione logica  
    istruzioni  
elseif espressione logica  
    istruzioni  
else  
    istruzioni  
end
```

# Operatori di relazione

| <b>Operatore</b> | <b>Descrizione</b>     |
|------------------|------------------------|
| <                | Minore di              |
| <=               | Minore di o uguale a   |
| >                | Maggiore di            |
| >=               | Maggiore di o uguale a |
| ==               | Uguale a               |
| ~=               | Diverso da             |

## Esempio

```
>> A=[2 7 6; 9 0 7; 6 3 2];  
>> B=[8 7 2; 8 1 7; 1 2 1];  
>> A==B
```

```
ans =  
     0     1     0  
     0     0     1  
     0     0     0
```

Gli elementi in cui la relazione è **vera** hanno valore 1.

Gli elementi in cui la relazione è **falsa** hanno valore 0.

# Operatori logici

| Operatore | Descrizione |
|-----------|-------------|
| &         | e           |
|           | o           |
| ~         | non         |

- Un'espressione con l'operatore & è vera se sono veri entrambi gli operandi. In termini numerici, l'espressione è vera se entrambi gli operandi sono diversi da zero.

```
>> u=[1 2 0 2 1 0];
```

```
>> v=[0 1 1 0 2 0];
```

```
>> u&v
```

```
ans =
```

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

- Un'espressione con l'operatore  $|$  è vera se almeno uno degli operandi è vero. In termini numerici, l'espressione è falsa se entrambi gli operandi sono uguali a zero.

```
>> u|v
```

```
ans =
```

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

- Un'espressione in cui si usa l'operatore  $\sim$ , si nega l'operando. In termini numerici, ogni elemento  $\neq 0$  diventa 0 e ogni elemento 0 diventa 1.

```
>> ~u
```

```
ans =
```

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

# Esercizio

## Problema

- Usare la function **eqsecgrado** per risolvere l'equazione

$$x^2 - 2bx + c = 0.$$

essendo  $b = \frac{10^\alpha + 1}{2}$  e  $c = 10^\alpha$ . Risolvere l'equazione con  $\alpha = 2, 7, 12, 16.3, 17$ .

- Osservato che le soluzioni sono  $x = 1$  e  $x = 10^\alpha$ , trovare una motivazione per i risultati ottenuti.
- Introdurre nella function **eqsecgrado**, la seguente formula stabile per il calcolo delle radici e verificare che il risultato è sempre corretto.

$$\text{Se } b \geq 0 \quad \begin{aligned} x_1 &= b + \sqrt{b^2 - c} \\ x_2 &= c/x_1 \end{aligned}$$

$$\text{Se } b < 0 \quad \begin{aligned} x_1 &= b - \sqrt{b^2 - c} \\ x_2 &= c/x_1 \end{aligned}$$

## Esercizio

### Algoritmo di sostituzione in avanti

Implementare in una function l'algoritmo di sostituzione in avanti, per risolvere un sistema lineare con matrice dei coefficienti triangolare inferiore.

Usare la seguente riga di dichiarazione della function:

```
function [x]=sost_avanti(L,b)
```

essendo

**Input**    **L** matrice triangolare inferiore;

**b** termine noto;

**Output**   **x** vettore soluzione.

## Algoritmo di sostituzione in avanti

- $x_1 = b_1 / \ell_{11}$

- per  $i = 2, \dots, n$

- ▶  $x_i = \left( b_i - \sum_{j=1}^{i-1} \ell_{ij} x_j \right) / \ell_{ii}$

## Esercizio

### Algoritmo di sostituzione all'indietro

Implementare in una function l'algoritmo di sostituzione all'indietro, per risolvere un sistema lineare con matrice dei coefficienti triangolare superiore.

Usare la seguente riga di dichiarazione della function:

```
function [x]=sost_indietro(U,b)
```

essendo

**Input**    **U** matrice triangolare superiore;  
          **b** termine noto;  
**Output**   **x** vettore soluzione.

## Algoritmo di sostituzione all'indietro

- $x_n = b_n / u_{nn}$
- per  $i = n - 1, \dots, 1$ 
  - ▶  $x_i = \left( b_i - \sum_{j=i+1}^n u_{ij} x_j \right) / u_{ii}$

## for

Il ciclo **for** esegue un gruppo di istruzioni un numero fissato di volte.

```
for indice = inizio : incremento : fine  
    istruzioni  
end
```

Incremento di default: 1.

Se *incremento* > 0, allora il ciclo termina quando la variabile *indice* è maggiore di *fine*.

Se *incremento* < 0, allora il ciclo termina quando la variabile *indice* è minore di *fine*.

## Osservazione

Si ricorda che

$$\sum_{i=1}^n b_i = b_1 + b_2 + b_3 + \cdots + b_n$$

L'operazione  $+$  è un'operazione binaria, cioè opera tra due addendi. Quindi si somma prima  $b_1 + b_2$  al risultato si aggiunge  $b_3$  e così via. Indicata con  $S$  una variabile di accumulo, per realizzare la sommatoria si deve usare un ciclo `for` nel modo seguente:

```
S=0.           inizializzazione della variabile di accumulo
for i=1:n
    S=S+b(i);   b è un array che contiene gli addendi  $b_i$ 
end
```

## Test

Testare il proprio programma per risolvere i seguenti sistemi con matrici triangolari:

$$U = \begin{pmatrix} 1 & 0 & -2 & 3 & 1 \\ 0 & 2 & 5 & -3 & -1 \\ 0 & 0 & 3 & -4 & 1 \\ 0 & 0 & 0 & 4 & -7 \\ 0 & 0 & 0 & 0 & 5 \end{pmatrix} \quad b = \begin{pmatrix} 5 \\ -7 \\ -1 \\ -14 \\ 10 \end{pmatrix}$$

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -3 & 2 & 0 & 0 & 0 \\ 0 & 4 & 3 & 0 & 0 \\ 3 & 2 & -8 & 4 & 0 \\ 2 & -2 & 0 & 3 & 5 \end{pmatrix} \quad c = \begin{pmatrix} 2 \\ -8 \\ -4 \\ 12 \\ 7 \end{pmatrix}$$

# while

Il ciclo **while** esegue un gruppo di istruzioni fintanto che l'**espressione di controllo** rimane vera.

```
while espressione di controllo  
    istruzioni  
end
```

L'**espressione di controllo** è una qualunque **espressione logica**.

# Precisione di macchina

## Definizione

Dicesi **precisione di macchina**, il più piccolo numero che sommato a **1** dà un risultato maggiore da **1**.

## Calcolo della precisione di macchina

Scrivere un file di tipo script per calcolare la precisione di macchina.

### Suggerimento:

- Porre  $x=1$  e  $eps=1$ .
- Fare un ciclo di tipo `while`, dividendo  $eps$  per **2** fintanto che  $x+eps>x$ .
- All'uscita del ciclo porre  $eps=2*eps$ .

## Esercizio

### Problema 3

Sia  $f : I \subseteq \mathbb{R} \rightarrow \mathbb{R}$  una funzione reale e continua sull'intervallo aperto  $I$ . Dato  $x_0 \in I$ , si consideri la successione definita per ricorrenza da

$$x_{n+1} = f(x_n).$$

Trovare il limite della successione.

- Fermare il ciclo quando la differenza fra due iterate successive  $|x_{n+1} - x_n|$  è inferiore a  $\text{toll}=1.e-4$ .
- Fissare un numero massimo di iterazioni.
- $f(x) = \frac{x^2 + 2}{2x}$  per  $0 < x < 2$ ,  $x_0 = 1$ .
- $f(x) = x - x^2 + 2$  per  $-1 < x < 3$ ,  $x_0 = 0.6$ .
- $f(x) = \sqrt{x + 1}$  per  $-1 < x < 3$ ,  $x_0 = 0.6$ .

# Rappresentazione della successione definita per ricorrenza

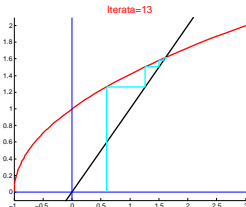
## Osservazione

Supponiamo che la successione  $x_n$  sia convergente, cioè  $\lim_{n \rightarrow \infty} x_n = x$ .

Allora  $x$  soddisfa la seguente equazione  $x = f(x)$  e si dice che  $x$  è **punto fisso di  $f$** .

Scrivere un programma di tipo function che esegua le seguenti operazioni:

- riporta il grafico della funzione di iterazione e della bisettrice del primo e terzo quadrante;
- calcola la soluzione del **problema di punto fisso** mediante il **metodo delle approssimazioni successive**;
- ad ogni iterata riporta sul grafico della funzione l'evoluzione della successione (vedi figura).



# Function **ricorrenza**

```
[xf, iter]=ricorrenza(f, a, b, x0, tol, nmax)
```

## **Input**

f        nome della funzione;  
a, b    estremi dell'intervallo;  
x0      punto iniziale;  
tol     tolleranza desiderata;  
nmax    numero massimo di iterazioni.

## **Output**

xf      valore del limite;  
iter    iterazioni eseguite.

## Traccia dell'esercizio

```
function [xf,iter]=ricorrenza(f,a,b,x0,tol,nmax)
```

1. uso il comando `fplot` per il grafico della funzione;
2. inizializzo le variabili `delta=1` e `i=0` per il controllo;
3. inizio ciclo while: `while delta>tol&i<=nmax`
  - 3.1 salvo il valore vecchio: `x=x0`;
  - 3.2 calcolo il nuovo valore: `x0=f(x0)`;
  - 3.3 calcolo la differenza: `delta=abs(x-x0)`;
  - 3.4 incremento il contatore: `i=i+1`.
4. Assegno i risultati alle variabili di output: `xf=x0` e `iter=i`.

## Successione per ricorrenza: caso lineare

Si consideri la seguente successione per ricorrenza di tipo **lineare**:

$$x_{n+1} = ax_n + b.$$

L'equazione di punto fisso corrispondente è  $x = ax + b$  quindi la soluzione è  $x = b/(1 - a)$  per  $a \neq 1$ .

Usare la function **iterazione** per studiare il comportamento della successione con i seguenti dati:

1.  $a = 2, b = -1; x_0 = 1.1$ , intervallo  $[-0.2, 5]$
2.  $a = 1, b = 3; x_0 = 0.1$ , intervallo  $[0, 20]$
3.  $a = -1, b = 5; x_0 = 1$ , intervallo  $[0, 5]$
4.  $a = 1/2, b = 1; x_0 = 4.5$ , intervallo  $[1, 5]$
5.  $a = -1/2, b = 5; x_0 = 4.5$ , intervallo  $[-5, 5]$
6.  $a = -2, b = 1; x_0 = 0.5$ , intervallo  $[-20, 20]$
7.  $a = 4, b = -1; x_0 = 0.4$ , intervallo  $[0, 10]$

Ripetere la prova negli ultimi due casi scegliendo come dato iniziale  $x_0 = 1/3$ . Osservare il comportamento della successione in 100 iterazioni se si toglie il controllo su  $|x_{n+1} - x_n|$ .