# Introduzione all'ambiente

# MATLAB©

*Programmazione.*

*Gestione di "import/export" di dati.*

*Risoluzione di sistemi non lineari.*

*Lezione 4 (5 marzo 2003)* *Lucia Gastaldi*



# Script e Function

Il processo di programmazione in MATLAB funziona nel modo seguente:

**1** Si crea un **M-file** usando un editore di testi (per es. si usa una finestra dell'**Editor/Debug**)

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

⇓

**2** Si chiama l'**M-file** dalla linea di comando oppure da un altro **M-file**

```
>> a = 7.5
>> b = 3.342
>> c = myfile(a,b)
>> c =
        8.2109
```



# Tipi di **M-file**

| **Script** | **Function** |
|---|---|
| • Opera sui dati presenti.<br>• Non accetta variabili in input.<br>• Non ha variabili in output.<br>• Utile per automatizzare una serie di istruzioni che si devono eseguire più volte. | • Le variabili interne sono locali.<br>• Può accettare variabili in input.<br>• Può avere variabili in output.<br>• Utile per estendere il linguaggio MATLAB alle applicazioni personali. |



# M-file di tipo **script**

**Esempio** `alglin.m`

```
%  Risoluzione di un sistema lineare     % Righe di 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.



### 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,errorerel] = errsl(n)   % Riga di definizione
                                         %    della function
%  ERRSL errore per un 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);
errorerel=errore/norm(x);
```



### Elementi comuni in tutte le function MATLAB

- *Riga di definizione*. Questa riga definisce il nome della `funtion`, il numero e l'ordine delle variabili in ingresso e in uscita.
- *Riga H1*. Con il comando `lookfor nome-function` MATLAB scrive questa riga. Con il comando `help` di un'intera cartella, MATLAB scrive questa riga per ogni **M-file** della cartella.
- *Testo per help*. Con il comando `help nome-function` MATLAB scrive il testo per help insieme alla riga H1.
- *Corpo della function*. Contiene le istruzioni per il calcolo e assegna il valore alle variabili di uscita.
- *Commenti*.

## 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)
```



## 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.

**Oss:** si può usare la function `sum` che somma tutti gli elementi di un vettore.



### Variabili locali e globali

- Per le variabili in un M-file valgono le stesse regole delle variabili MATLAB.
- Di solito ogni `function`, definita in un M-file, ha le **sue** variabili, che sono separate da quelle di altre function e da quelle dell'ambiente di lavoro di base.
- Se diverse functions e anche l'ambiente di lavoro di base dichiarano **globale** il nome di una certa variabile, allora essi condividono una copia di quella variabile.

### global

- Dichiarare la variabile `global` in ogni function che la usa.
- Dichiarare la variabile `global` anche dalla riga di comando.
- Il comando `global` deve precedere il primo punto in cui la variabile viene usata. Si raccomanda di metterlo all'inizio dell'M-file.



### Esempio

Si ha l'M-file `lotka.m` per studiare l'effetto dei coefficienti $\alpha$ e $\beta$ nel modello preda-predatore di Lotka-Volterra

```
function yp = lotka(t,y)
% LOTKA modello preda-predatore di Lotka-Volterra
global ALPHA BETA
yp=[y(1) - ALPHA*y(1)*y(2); -y(2) + BETA*y(1)*y(2)];
```

Si danno inoltre i seguenti comandi:

```
>> global ALPHA BETA
>> ALPHA=0.01
>> BETA = 0.02;
>> [t,y]=ode23('lotka',[0,10],[1;1]);
>> plot(t,y)
```



## Valori speciali

Alcune function danno come risposta dei **valori speciali** che possono essere usati in un M-file.

| Function | Risposta |
|---|---|
| `ans` | La risposta più recente. |
| `eps` | Precisione di macchina. |
| `realmax` | Il più grande numero floating point rappresentabile. |
| `realmin` | Il più piccolo numero floating point rappresentabile. |
| `pi` | $\pi$=3.1415926535897... |
| `i,j` | Unità immaginaria. |
| `inf` | Infinito. |
| `NaN` | Not-a-number. |
| `computer` | Tipo di computer. |
| `version` | Versione di MATLAB. |

# Importare dati in MATLAB

Ci sono diversi metodi per importare dei dati in MATLAB.
La scelta dipende dalla quantità e dal formato dei dati.

| **Metodo** | **Uso** |
|---|---|
| Introdurre i dati come una lista esplicita di elementi. | Nel caso di una piccola quantità di dati, si inseriscono i dati usando le parentesi [ ].<br>Non si può correggere l'input se si sono fatti degli errori. |
| Creare i dati in un M-file. | Si usa un editore di testo per creare un M-file che contiene i dati in una lista di elementi.<br>Utile se i dati non sono ancora in formato floating point e devono comunque essere introdotti.<br>Se si fanno degli errori si possono correggere facilmente. |



| | |
|---|---|
| Caricare i dati da un file **ASCII**. | Si legge il file ASCII direttamente in MATLAB usando il comando `load`.<br>Si crea una variabile il cui nome è lo stesso del file ASCII. |
| `fopen`, `fread`, function I/O di MATLAB. | Utile per caricare file di dati provenienti da altre applicazioni che hanno il loro formato. |
| Function specializzate nel leggere file con formati particolari. | `dlmread` legge file ASCII.<br>`textread` legge stringhe e dati numerici da un file in variabili MATLAB usando degli specificatori di conversione.<br>`imread` legge immagini da file grafici.<br>`auread` legge file Sun (.au) di suoni.<br>`wavread` legge file Microsoft WAVE (.wav) di suoni. |



| | |
|---|---|
| Sviluppare un **MEX-file** per leggere i dati. | È il metodo migliore se sono disponibili delle routine in C o in Fortran che leggono i file di dati da altre applicazioni. |
| Sviluppare un programma di traduzione in C o in Fortran. | Si sviluppa un programma che traduce i dati in un **MAT-file** e poi si legge il MAT-file con il comando `load`. |



# Esportare dati da MATLAB

Ci sono diversi metodi per esportare dei dati da MATLAB verso altre applicazioni.

| **Metodo** | **Uso** |
|---|---|
| Usare il comando `diary`. | Il comando `diary` copia quello che compare sullo schermo in un file diary.<br>Nel file diary compaiono anche i comandi usati durante la sessione. |
| Salvare i dati in un file **ASCII**. | `save` *nome_file* X Y Z -ascii salva le variabili X, Y, Z in un file in formato ASCII con 8 cifre decimali. |
| Salvare i dati in un file `.mat` | `save` *nome_file* X Y Z salva le variabili X, Y, Z in un file `nome_file.mat`. |

**Problema (Piano d'investimento)**
Si vuole calcolare il tasso medio di rendita $r$ di un fondo d'investimento su più anni. Supponiamo che si investano nel fondo $v$ euro all'inizio di ogni anno e che dopo $n$ anni si abbia accumulato un montante pari a $M$ euro. Essendo $M$ legato a $r$ dalla seguente relazione

$$M = v \sum_{k=1}^{n} (1+r)^k = v\frac{1+r}{r}\left((1+r)^n - 1\right),$$

si deduce che il tasso di rendimento percentuale $r$ è lo zero dell'equazione non lineare:

$$f(r) = 0, \quad \text{dove } f(r) = M - v\frac{1+r}{r}\left((1+r)^n - 1\right).$$

Poniamo: $v = 1000$ euro, $n = 5$, $M = 6000$ euro.



**Primo tentativo**: plottare la funzione

```
>> f='6000-1000*(1+x).*((1+x).^5-1)./x';
>> fplot(f,[0,0.3])
```

`fzero` - Funzione matlab per la ricerca degli zeri di una funzione in una variabile.

```
>> help fzero ⏎
```



**Esempio** Il file `rendita.m` contiene la funzione di cui si cerca lo zero:

```
function [f]=rendita(x)
f='6000-1000*(1+x).*((1+x).^5-1)./x';
```

usare il comando

```
x = fzero('rendita',0.1)
```

oppure

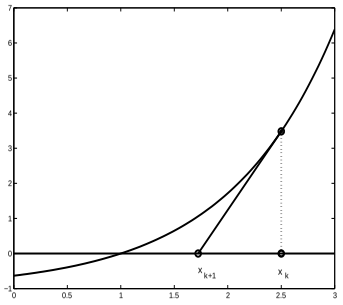
```
[x,feval] = fzero('rendita',0.1)
```

Per avere informazioni complete sul procedimento di calcolo:

```
[x,feval] = fzero('rendita',0.1,'disp','iter')
```



## Il metodo di Newton




Supponiamo che la funzione sia derivabile.
Dato un punto sulla funzione $(x_k, f(x_k))$, scriviamo l'equazione della retta tangente

$$y(x) = f(x_k) + f'(x_k)(x - x_k).$$

Sia $x_{k+1}$ tale che $y(x_{k+1}) = 0$, cioè:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \text{ purché } f'(x_k) \neq 0.$$

**Algoritmo di Newton** (file `newton.m`)

```
function[zero,niter]=newton(f,df,x0,toll,nmax)
niter=0;    diff=toll+1;    x=x0;
fx=eval(f);   dfx=eval(df);
while diff>=toll & niter<=nmax
niter=niter+1;  diff=-fx/dfx;
x=x+diff;       diff=abs(diff);
fx=eval(f);     dfx=eval(df);
end
zero=x;
```

Posto: `x0=0.1; toll=1.e-8; nmax=10;` e

```
>> f='6000-1000*(1+x).*((1+x).^5-1)./x';
>> df='-1000*(-((1+x).^5-1)./x.^2+5*(1+x).^5./x)';
```

usare il comando

```
[x,feval] = newton(f,df,x0,toll,nmax)
```



**$1^o$ modo** `newtonsys.m`

Per usare questa function il comando è:

```
[x, nit] = newtonsys(F, J, x0, toll, nmax, p)
```

dove:

| | |
|---|---|
| `F` | array di tipo carattere che contiene la funzione $\mathbf{f}$ (una componente per ciascuna riga); |
| `J` | array di tipo carattere che contiene la matrice Jacobiana (un elemento per ciascuna riga); |
| `x0` | punto iniziale (vettore colonna); |
| `toll` | precisione richiesta; |
| `nmax` | numero massimo di iterazioni; |
| `p` | numero di iterazioni da eseguire senza aggiornare la matrice Jacobiana (per aggiornarla a tutti i passi `p=1`). |



**Come si passa il nome di una funzione ad una function?**

**Problema** calcolare gli zeri di un sistema di equazioni non lineari con il metodo di Newton-Raphson:

$$\mathbf{f(x)} = \mathbf{0} \text{ ossia } \begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0 \\ f_2(x_1, x_2, \ldots, x_n) = 0 \\ \ldots\ldots\ldots \\ f_n(x_1, x_2, \ldots, x_n) = 0 \end{cases}$$

L'utente deve fornire le espressioni della funzione a valori vettoriali $\mathbf{f}$ e della sua matrice Jacobiana.



Le funzioni possono essere definite dalla riga di comando come array di tipo carattere:

```
>> F=char({'x(1)^3-3*x(1)*x(2)^2-1';'3*x(1)^2*x(2)-x(2)'});
>> J=char({'3*x(1)^2-3*x(2)^2';
    '-6*x(1)*x(2)';
    '6*x(1)*x(2)';
    '3*x(1)^2-3*x(2)^2'});
```

Si noti l'uso del comando `char` che serve a generare un array di tipo carattere in modo che ciascuna riga abbia la stessa lunghezza, altrimenti si potrebbe usare il comando usuale:

```
>> F=['x(1)^3-3*x(1)*x(2)^2-1'; '3*x(1)^2*x(2)-x(2)    '];
>> J=['3*x(1)^2-3*x(2)^2'; '-6*x(1)*x(2)     ';
      '6*x(1)*x(2)      '; '3*x(1)^2-3*x(2)^2'];
```

ma bisogna aggiungere degli spazi bianchi in modo che le stringhe di un array abbiano tutte la stessa lunghezza.

## $2^o$ modo `newtonsys_mat.m`

Per usare questa function il comando è:

```
[x, nit] = newtonsys_mat('F', 'J', x0, toll, nmax, p)
```

dove:

| | |
|---|---|
| `F` | nome della `function` che contiene la funzione $\mathbf{f}$ |
| `J` | nome della `function` che contiene la matrice Jacobiana |
| `x0` | punto iniziale (vettore colonna); |
| `toll` | precisione richiesta; |
| `nmax` | numero massimo di iterazioni; |
| `p` | numero di iterazioni da eseguire senza aggiornare la matrice Jacobiana (per aggiornarla a tutti i passi `p=1`). |



L'utente fornisce le due function di nome `F.m` e `J.m` nel modo seguente:

```
function [f]=F(x)
f=[x(1)^3-3*x(1)*x(2)^2-1;3*x(1)^2*x(2)-x(2)^3];
```

```
function [j]=J(x)
j=[3*x(1)^2-3*x(2)^2,-6*x(1)*x(2);
6*x(1)*x(2),3*x(1)^2-3*x(2)^2];
```